# A Markdown Interpreter for TEX

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

**Version 3.8.1-0-gbbed72f3**
**2024-11-03**

## Contents



## List of Figures



## 1 Introduction

The Markdown package[1] converts CommonMark[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 converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😉

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

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

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 Starý Novotný, "
             .. "Andrej Genčur",
    copyright = {"2009-2016 John MacFarlane, Hans Hagen",
                 "2016-2024 Vít Starý Novotný, Andrej Genčur"},
    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 (TEX Live $\geqslant$ 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 = require("unicode")
```

**MD5** A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of LuaTEX (TEX Live $\geqslant$ 2008).

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

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

**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
  kpse = require("kpse")
  if should_initialize then
    kpse.set_program_name("luatex")
  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.

```
hard lua-uni-algos
local uni_algos = require("lua-uni-algos")
```

**api7/lua-tinyyaml** A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jekyllData` option is enabled. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

```
# hard lua-tinyyaml  # TODO: Uncomment after TeX Live 2022 deprecation.
```

### 1.1.2 Plain TEX Requirements

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

**expl3** A package that enables the expl3 language from the LATEX3 kernel in TEX Live ⩽ 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
hard l3kernel
\unprotect
```

```
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic
\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.

```
hard lt3luabridge
```

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

**Lua File System** A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TEX code to create the cache directory specified by the `cacheDir` option before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTEX (TEXLive ⩾ 2008).

The plain TEX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTEX engine developers [1, Section 4.2.4].

The Lua File System module is statically linked into the LuaTEX engine [1, Section 4.3].

Unless you convert markdown documents to TEX manually using the Lua command-line interface (see Section 2.1.7), the plain TEX part of the package will require that either the LuaTEX `\directlua` primitive or the shell access file stream 18 is available in your TEX engine. If only the shell access file stream is available in your TEX engine (as is the case with pdfTEX and XeTEX), then unless your TEX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

### 1.1.3 LaTeX Requirements

The LaTeX part of the package requires that the LaTeX 2ε format is loaded, a TEX engine that extends ε-TEX, and all the plain TEX prerequisites (see Section 1.1.2).

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

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or LaTeX themes (see Section 2.3.4) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
soft url
```

**graphicx** A package that provides the `\includegraphics` macro for the typesetting of images. Furthermore, it also provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.

```
soft graphics
```

**enumitem and paralist** Packages that provide macros for the default renderer prototypes for tight and fancy lists.

The package paralist will be used unless the option `experimental` has been enabled, in which case, the package enumitem will be used. Furthermore, enabling any test phase [2] will also cause enumitem to be used. In a future major version, enumitem will replace paralist altogether.

```
soft enumitem
soft paralist
```

**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 2.3.4).

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

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

```
soft fancyvrb
```

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

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

**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.

```
soft gobble
```

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

```
soft amsmath
soft amsfonts
```

**catchfile** A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` LaTeX theme, see Section 2.3.4.

```
soft catchfile
```

**grffile** A package that extends the name processing of the graphics package to support a larger range of file names in 2006 ⩽ TeX Live ⩽ 2019. Since TeX Live ⩾ 2020, the functionality of the package has been integrated in the LaTeX $2_\varepsilon$ kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` LaTeX themes, see Section 2.3.4.

```
soft grffile
```

**etoolbox** A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.9, and also in the default renderer prototype for identifier attributes.

```
soft etoolbox
```

**soulutf8 and xcolor** Packages that are used in the default renderer prototypes for strike-throughs and marked text in pdfTeX.

```
soft soul
soft xcolor
```

**lua-ul and luacolor** Packages that are used in the default renderer prototypes for strike-throughs and marked text in LuaTeX.

```
soft lua-ul
soft luacolor
```

**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.

```
soft ltxcmds
```

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

```
soft verse
```

### 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 2.2.6).

## 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.

## 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 [3] 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 $2_\varepsilon$, 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.

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

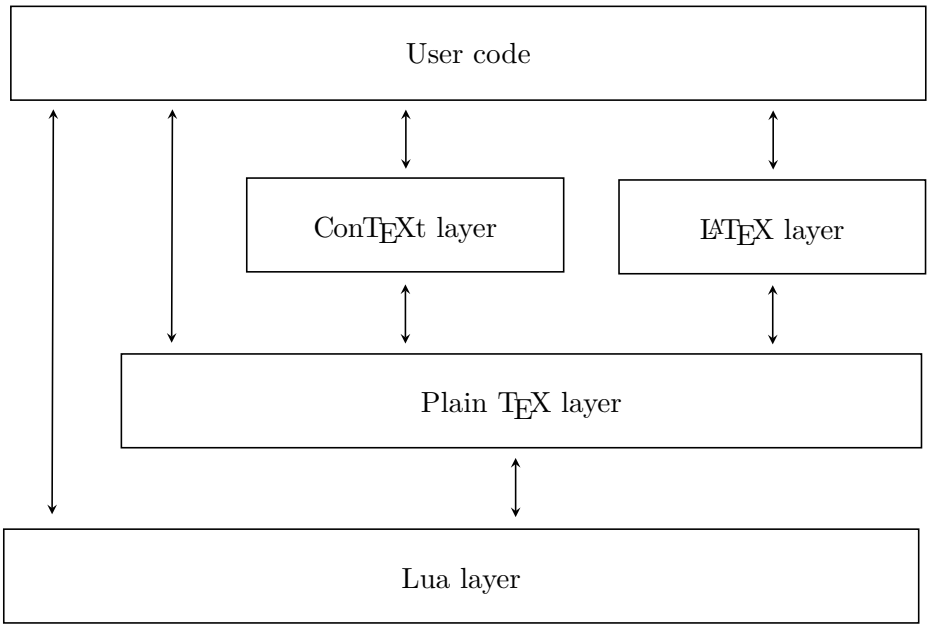


**Figure 1: A block diagram of the Markdown package**

## 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.

```
58 local M = {metadata = metadata}
```

### 2.1.1 Conversion from Markdown to Plain TEX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain TEX according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

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",
    "DisplayHtml",
    "Heading",
  },
  BlockOrParagraph = {
    "Block",
    "Paragraph",
    "Plain",
  },
  Inline = {
    "Str",
    "Space",
    "Endline",
    "EndlineBreak",
    "LinkAndEmph",
    "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 -> LinkAndEmph` and `Inline -> Code` rules, we would call `reader->insert_pattern` with `"Inline after LinkAndEmph"` (or `"Inline before Code"`) and `pattern` as the arguments.

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

### 2.1.3 Options

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

```
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
                  { markdown }
                  { failed-typecheck-for-boolean-option }
                  { #1 }
                  \l_tmpa_tl
              }
```

```
          }
      }
  }
\msg_new:nnn
  { markdown }
  { 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
          { markdown }
          { undefined-option }
          { #1 }
      }
    \seq_if_in:NVF
      \g_@@_option_types_seq
      \l_tmpa_tl
      {
        \msg_error:nnnV
          { markdown }
          { unknown-option-type }
          { #1 }
```

```
          \l_tmpa_tl
      }
    \tl_set_eq:NN
      #2
      \l_tmpa_tl
  }
\msg_new:nnn
  { markdown }
  { unknown-option-type }
  {
    Option~#1~has~unknown~type~#2.
  }
\msg_new:nnn
  { markdown }
  { 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
          { markdown }
          { 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 }
```

```
      }
  }
```

To make it easier to support different coding styles in the interface, engines, we define the `\@@_with_various_cases:nn` function that allows us to generate different variants of a string using different cases.

```
\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 }
  }
```

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

```
\cs_new:Nn \@@_with_various_cases_break:
  {
    \seq_map_break:
  }
```

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

```
\seq_new:N \g_@@_cases_seq
\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 General Behavior

`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. Furthermore, it can also significantly improve the processing speed for documents that require multiple compilation runs, since each markdown document is only converted once. 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. However, it makes it impossible to post-process the converted documents and recover historical versions of the documents from the cache. Furthermore, it can significantly reduce the processing speed for documents that require multiple compilation runs, since each markdown document is converted multiple times needlessly.

This behavior will only be used when the `finalizeCache` option is disabled.

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

`experimental`=true, false default: `false`

`true` Experimental features that are planned to be the new default in the next major release of the Markdown package will be enabled.

At the moment, this just means that the version `experimental` of the theme `witiko/markdown/defaults` will be loaded and warnings for hard-deprecated features will become errors. However, the effects may extend to other areas in the future as well.

`false` Experimental features will be disabled.

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

`singletonCache`=true, false default: `true`

`true` Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time- and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions.

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

`false` Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also #226 (comment)[6].

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

```
\@@_add_lua_option:nnn
  { singletonCache }
  { boolean }
  { true }
defaultOptions.singletonCache = true
local singletonCache = {
  convert = nil,
  options = nil,
}
```

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

`unicodeNormalization`=`true`, `false` default: `true`

`true` Markdown documents will be normalized using one of the four Unicode normalization forms[7] before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

`false` Markdown documents will not be Unicode-normalized before conversion.

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

`unicodeNormalizationForm`=`nfc`, `nfd`, `nfkc`, `nfkd`
default: `nfc`

`nfc` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form C (NFC) before conversion.

`nfd` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form D (NFD) before conversion.

`nfkc` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form KC (NFKC) before conversion.

`nfkd` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form KD (NFKD) before conversion.

```
\@@_add_lua_option:nnn
  { unicodeNormalizationForm }
  { string }
  { nfc }
defaultOptions.unicodeNormalizationForm = "nfc"
```

### 2.1.5 File and Directory Names

[7] See https://unicode.org/faq/normalization.html.

`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 2.2.5.9 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 3.1.7) 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"
```

### 2.1.6 Parser Options

`autoIdentifiers`=`true`, `false` default: `false`

`true` Enable the Pandoc auto identifiers syntax extension[8]:

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

# 123 Sesame Street
```

`false` Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

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

`blankBeforeBlockquote`=`true`, `false` default: `false`

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

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

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

```
\@@_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
```

`blankBeforeList`=true, false default: `false`

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

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

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

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

`true` Enable the Pandoc bracketed span syntax extension[9]:

```
[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: `true`

`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 }
  { true }
defaultOptions.breakableBlockquotes = true
```

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

`citationNbsps`=`true`, `false` default: `false`

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

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

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

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

`true` Enable the Pandoc citation syntax extension[10]:

```
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
```

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

`codeSpans`=true, false default: `true`

`true` Enable the code span syntax:

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

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

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

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

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

true

: Enable the iA Writer content blocks syntax extension [4]:

```
``` 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
```

`contentLevel`=`block`, `inline` default: `block`

`block` Treat content as a sequence of blocks.

```
- this is a list
- it contains two items
```

`inline` Treat all content as inline content.

```
- this is a text
- not a list
```

```
\@@_add_lua_option:nnn
  { contentLevel }
  { string }
  { block }
defaultOptions.contentLevel = "block"
```

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

`true` Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

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

```
\@@_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
```

`ensureJekyllData`=`true`, `false` default: `false`

`false` When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. Otherwise, the markdown document is processed as markdown text.

`true` When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata. Otherwise, an error is produced.

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

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

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

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

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

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

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

```
\@@_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:

```lua
local strike_through = {
  api_version = 2,
  grammar_version = 4,
  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 LinkAndEmph", read_strike_through,
                          "StrikeThrough")
    reader.add_special_character("/")
  end
}

return strike_through
```

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

```
metadata.user_extension_api_version = 2
metadata.grammar_version = 4
```

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

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `reader` object, such as the `reader->insert_pattern` and `reader->add_special_character` methods, see Section 2.1.2.

```
\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[11]:

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

`false` Disable the Pandoc fancy list syntax extension.

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

```
defaultOptions.fancyLists = false
```

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

`true` Enable the commonmark fenced code block extension:

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

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

```
``` 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 }
  { true }
defaultOptions.fencedCode = true
```

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

`true` Enable the Pandoc fenced code attribute syntax extension[12]:

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

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

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

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

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

`true` Enable the Pandoc fenced div syntax extension[13]:

```
::::: {#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.

```
\@@_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*⟩.

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

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

`gfmAutoIdentifiers`=true, false  default: `false`

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension[14]:

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

# 123 Sesame Street
```

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

See also the option `autoIdentifiers`.

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

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

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

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

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

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

[14] See https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers.

`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: `true`

`true` Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.

`false` Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

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

`hybrid`=true, false default: `false`

`true` Disable the escaping of special plain TEX characters, which makes it possible to intersperse your markdown markup with TEX code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix TEX and markdown markup freely.

`false` Enable the escaping of special plain TeX characters outside verbatim environments, so that they are not interpreted by TeX. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

The `hybrid` option makes it difficult to untangle TeX input from markdown text, which makes documents written with the `hybrid` option less interoperable and more difficult to read for authors. Therefore, the option has been soft-deprecated in version 3.7.1 of the Markdown package: It will never be removed but using it prints a warning and is discouraged.

Consider one of the following better alternatives for mixing TeX and markdown:

- With the `contentBlocks` option, authors can move large blocks of TeX code to separate files and include them in their markdown documents as external resources:

```
Here is a mathematical formula:

 /math-formula.tex
```

- With the `rawAttribute` option, authors can denote raw text spans and code blocks that will be interpreted as TeX code:

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

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

- With options `texMathDollars`, `texMathSingleBackslash`, and `texMathDoubleBackslash`, authors can freely type TeX commands between dollar signs or backslash-escaped brackets:

```
$H_2 O$ is a liquid.

Here is a mathematical formula:
\[distance[i] =
```

```
    \begin{dcases}
        a & b \\
        c & d
    \end{dcases}
\]
```

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

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

`true` Enable the Pandoc inline code span attribute extension[15]:

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

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

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

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

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

```
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.

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

[15] See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.
[16] See https://pandoc.org/MANUAL.html#extension-inline_notes.

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

`true` Enable the Pandoc YAML metadata block syntax extension[17] for entering metadata in YAML:

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

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

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

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

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

`true` Enable the Pandoc link and image attribute syntax extension[18]:

```
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
```

[17] See https://pandoc.org/MANUAL.html#extension-yaml_metadata_block.
[18] See https://pandoc.org/MANUAL.html#extension-link_attributes.

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

`true` Enable the Pandoc line block syntax extension[19]:

```
| 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
```

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

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

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

`false` Disable the Pandoc mark syntax extension.

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

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

`true` Enable the Pandoc note syntax extension[21]:

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

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

[19] See https://pandoc.org/MANUAL.html#extension-line_blocks.
[20] See https://pandoc.org/MANUAL.html#extension-mark.
[21] See https://pandoc.org/MANUAL.html#extension-footnotes.

```
[^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.

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

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

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

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

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

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

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

`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 }
  { true }
defaultOptions.preserveTabs = true
```

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

`true` Enable the Pandoc raw attribute syntax extension[22]:

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

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

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

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

`false` Disable the Pandoc raw attribute syntax extension.

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

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

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

`true` Enable relative references[23] in autolinks:

```
I conclude in Section <#conclusion>.

Conclusion {#conclusion}
==========
In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
```

`false` Disable relative references in autolinks.

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

`shiftHeadings`=⟨*shift amount*⟩ default: `0`

All headings will be shifted by ⟨*shift amount*⟩, which can be both positive and negative. Headings will not be shifted beyond level 6 or below level 1. Instead, those headings will be shifted to level 6, when ⟨*shift amount*⟩ is positive, and to level 1, when ⟨*shift amount*⟩ is negative.

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

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

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

- The circumflex (`^`) selects the beginning of a document.
- The dollar sign (`$`) selects the end of a document.

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

- `^`⟨*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[24]:

```
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
```

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

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

`true` Enable the Pandoc subscript syntax extension[25]:

```
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[26]:

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

`false` Disable the Pandoc superscript syntax extension.

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

`tableAttributes`=true, false default: `false`

true

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

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

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

[25] See https://pandoc.org/MANUAL.html#extension-superscript-subscript.
[26] See https://pandoc.org/MANUAL.html#extension-superscript-subscript.

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

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

`tableCaptions`=true, `false` default: `false`

true

: Enable the Pandoc table caption syntax extension[27] for pipe tables (see the `pipeTables` option).

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

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

`false` Disable the Pandoc table caption syntax extension.

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

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

`true` Enable the Pandoc task list syntax extension[28]:

```
- [ ] 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.

[27] See https://pandoc.org/MANUAL.html#extension-table_captions.
[28] See https://pandoc.org/MANUAL.html#extension-task_lists.

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

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

`true` Strip TEX-style comments.

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

Always enabled when `hybrid` is enabled.

`false` Do not strip TEX-style comments.

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

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

`true` Enable the Pandoc dollar math syntax extension[29]:

```
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
```

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

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

`true` Enable the Pandoc double backslash math syntax extension[30]:

```
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
```

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

`true` Enable the Pandoc single backslash math syntax extension[31]:

```
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:

[30] See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.
[31] See https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash.

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

- This is

  not a tight

- unordered list.
```

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

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

`underscores`=`true`, `false` default: `true`

`true` Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```
*single asterisks*
_single underscores_
**double asterisks**
__double underscores__
```

`false` Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the `hybrid` option without the need to constantly escape subscripts.

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

### 2.1.7 Command-Line Interface

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

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

A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to TEX is also provided, see Figure 3.

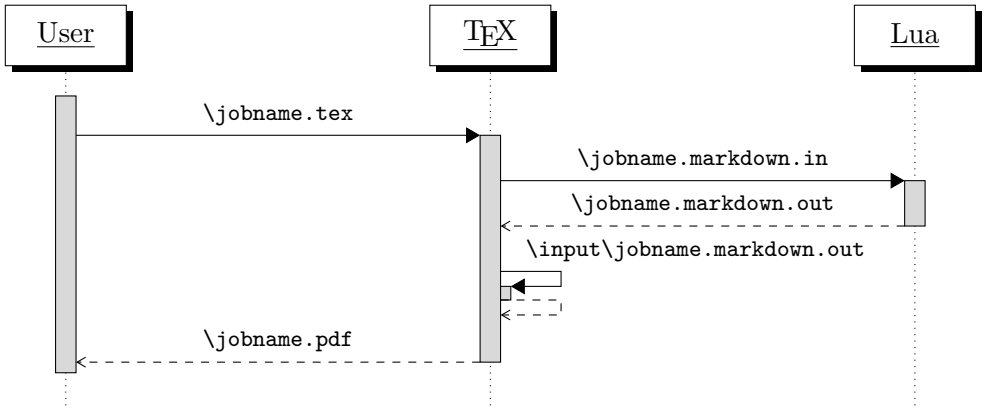


**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface**

```
738
739 local HELP_STRING = [[
740 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
741 where OPTIONS are documented in the Lua interface section of the
742 technical Markdown package documentation.
743
744 When OUTPUT_FILE is unspecified, the result of the conversion will be
745 written to the standard output. When INPUT_FILE is also unspecified, the
746 result of the conversion will be read from the standard input.
747
748 Report bugs to: witiko@mail.muni.cz
749 Markdown package home page: <https://github.com/witiko/markdown>]]
750
```

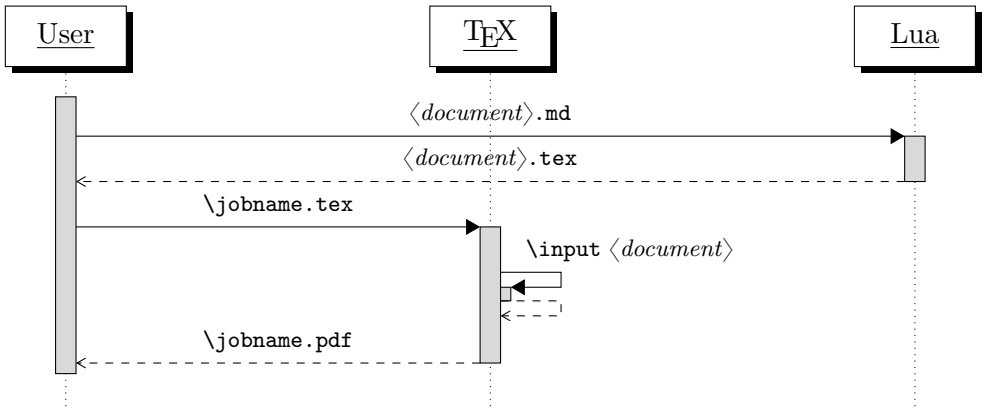


**Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface**

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

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

License: ]] .. metadata.license

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

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

To make it easier to copy-and-paste options from Pandoc [5] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [6] 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)
```

```
776     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
777   end)
778   return cased_option_name
779 end
780
781 local cases = {camel_case, snake_case}
782 local various_case_options = {}
783 for option_name, _ in pairs(defaultOptions) do
784   for _, case in ipairs(cases) do
785     various_case_options[case(option_name)] = option_name
786   end
787 end
788
789 local process_options = true
790 local options = {}
791 local input_filename
792 local output_filename
793 for i = 1, #arg do
794   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.

```
795     if arg[i] == "--" then
796       process_options = false
797       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.

```
798     elseif arg[i]:match("=") then
799       local key, value = arg[i]:match("(.-)=(.*)")
800       if defaultOptions[key] == nil and
801          various_case_options[key] ~= nil then
802         key = various_case_options[key]
803       end
```

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

```
804       local default_type = type(defaultOptions[key])
805       if default_type == "boolean" then
806         options[key] = (value == "true")
807       elseif default_type == "number" then
808         options[key] = tonumber(value)
809       elseif default_type == "table" then
810         options[key] = {}
811         for item in value:gmatch("[^ ,]+") do
```

```
812           table.insert(options[key], item)
813         end
814       else
815         if default_type ~= "string" then
816           if default_type == "nil" then
817             warn('Option "' .. key .. '" not recognized.')
818           else
819             warn('Option "' .. key .. '" type not recognized, ' ..
820                  'please file a report to the package maintainer.')
821           end
822           warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
823                key .. '" as a string.')
824         end
825         options[key] = value
826       end
827       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.

```
828     elseif arg[i] == "--help" or arg[i] == "-h" then
829       print(HELP_STRING)
830       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.

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

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

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

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

```
838   elseif output_filename == nil then
839     output_filename = arg[i]
840   else
841     error('Unexpected argument: "' .. arg[i] .. '".')
842   end
843   ::continue::
```

```
end
```

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

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

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

```
\input hello
```

## 2.2 Plain TEX Interface

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

```
\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 and YAML

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

#### 2.2.1.1 Typesetting Markdown and YAML directly

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 [7, 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 `\yamlBegin` macro marks the beginning of an YAML document fragment and the `\yamlEnd` macro marks its end.

```
\let\yamlBegin\relax
\def\yamlEnd{\markdownEnd\endgroup}
```

The `\yamlBegin` and `\yamlEnd` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```
\input markdown
\yamlBegin
title: _Hello_ **world** ...
```

```
author: John Doe
\yamlEnd
\bye
```

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

```
\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownBegin
title: _Hello_ **world** ...
author: John Doe
\markdownEnd
\bye
```

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

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

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

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

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

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

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

#### 2.2.1.2 Typesetting Markdown and YAML from external documents

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

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

The macro \markdownInput is not subject to the limitations of the \markdownBegin and \markdownEnd macros.

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

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

You can use the \yamlInput macro to include YAML documents. similarly to how you might use the \input TEX primitive to include TEX documents. The \yamlInput macro accepts a single parameter with the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain TEX.

```
\def\yamlInput#1{%
  \begingroup
  \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
  \markdownInput{#1}%
  \endgroup
}%
```

The macro \yamlInput is also not subject to the limitations of the \markdownBegin and \markdownEnd macros.

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

```
\input markdown
\yamlInput{hello.yml}
\bye
```

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

```
\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownInput{hello.yml}
\bye
```

#### 2.2.1.3 Typesetting TeX from inside Markdown and YAML documents

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

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

### 2.2.2 Options

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

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

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_plain_tex_tl
  }
```

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

```
\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
\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 }
  }
```

The plain TEX options may be also be specified via the `\markdownSetup` macro. Here, the plain TEX options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted

as ⟨*key*⟩`=true` if the `=`⟨*value*⟩ part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```
\cs_new:Nn
  \@@_setup:n
  {
    \keys_set:nn
      { markdown/options }
      { #1 }
  }
\cs_gset_eq:NN
  \markdownSetup
  \@@_setup:n
```

The command `\yamlSetup` is also available as an alias for the command `\markdownSetup`.

```
\cs_gset_eq:NN
  \yamlSetup
  \markdownSetup
```

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.

```
\prg_new_conditional:Nnn
  \@@_if_option:n
  { TF, T, F }
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \msg_error:nnxx
          { markdown }
          { 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
      { \prg_return_true: }
      { \prg_return_false: }
```

```
  }
\msg_new:nnn
  { markdown }
  { expected-boolean-option }
  {
    Option~#1~has~type~#2,~
    but~a~boolean~was~expected.
  }
\let\markdownIfOption=\@@_if_option:nTF
```

#### 2.2.2.1 Finalizing and Freezing the Cache

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

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

```
\@@_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 `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a TEX source. It defaults to `\jobname.markdown.in`.

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

```
\@@_add_plain_tex_option:nnn
  { inputTempFileName }
  { path }
  { \jobname.markdown.in }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain TEX implementation. The option defaults to `.` or, since TEX Live 2024, to the value of the `-output-directory` option of your TEX engine.

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

The `\markdownOptionOutputDir` macro has been deprecated and will be removed in the next major version of the Markdown package.

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

#### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level TEX formats such as LATEX and ConTEXt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

The `\markdownOptionPlain` macro specifies whether higher-level TEX formats should only use the plain TEX default definitions or whether they should also use the format-specific default definitions. Whereas plain TEX default definitions only provide definitions for simple elements such as emphasis, strong emphasis, and paragraph separators, format-specific default definitions add support for more complex elements such as lists, tables, and citations. On the flip side, plain TEX default definitions load no extra resources and are rather stable, whereas format-specific default definitions load extra resources and are subject to a more rapid change.

Here is how you would enable the macro in a LATEX document:

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

Here is how you would enable the macro in a ConTEXt document:

```
\def\markdownOptionPlain{true}
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
\@@_add_plain_tex_option:nnn
  { plain }
  { boolean }
  { false }
```

The `\markdownOptionNoDefaults` macro specifies whether we should prevent the loading of default definitions or not. This is useful in contexts, where we want to have total control over how all elements are rendered.

Here is how you would enable the macro in a LaTeX document:

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

Here is how you would enable the macro in a ConTeXt document:

```
\def\markdownOptionNoDefaults{true}
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
\@@_add_plain_tex_option:nnn
  { noDefaults }
  { boolean }
  { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (`%`) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing TeX package documentation using the Doc LaTeX package [8] 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 }
```

#### 2.2.2.5 Generating Plain TEX Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain TEX macros and the key-value interface of the `\markdownSetup` macro for the above plain TEX options.

The command also defines macros and key-values that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain TEX implementation, only passed along to Lua.

Furthermore, the command also defines options and key-values for subsequently loaded layers that correspond to higher-level TEX formats such as LATEX and ConTEXt.

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

```
\cs_new:Nn
  \@@_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
              \@@_define_option_command:n
                { ####1 }
```

To make it easier to copy-and-paste options from Pandoc [5] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [6] also show that snake_case is faster to read than camelCase.

```
              \@@_with_various_cases:nn
                { ####1 }
                {
                  \@@_define_option_keyval:nnn
                    { ##1 }
                    { ####1 }
                    { ########1 }
                }
          }
      }
  }
\cs_new:Nn
  \@@_define_option_command:n
  {
```

Use the lt3luabridge library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since TeX Live 2024.

```
    \str_if_eq:nnTF
      { #1 }
      { outputDir }
      { \@@_define_option_command_output_dir: }
      {
```

Do not override options defined before loading the package.

```
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \cs_if_exist:cF
          { \l_tmpa_tl }
          {
            \@@_get_default_option_value:nN
              { #1 }
              \l_tmpa_tl
            \@@_set_option_value:nV
              { #1 }
              \l_tmpa_tl
          }
      }
  }
\ExplSyntaxOff
\input lt3luabridge.tex
```

Use the lt3luabridge library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since TeX Live 2024.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_define_option_command_output_dir:
  {
    \cs_if_free:NT
      \markdownOptionOutputDir
      {
        \bool_if:nTF
          {
            \cs_if_exist_p:N
              \luabridge_tl_set:Nn &&
            (
              \int_compare_p:nNn
                { \g_luabridge_method_int }
                =
                { \c_luabridge_method_directlua_int } ||
```

```
            \sys_if_shell_unrestricted_p:
          )
        }
        {
```

Set most catcodes to category 12 (other) to ensure that special characters in TEXMF_OUTPUT_DIRECTORY such as backslashes (\) are not interpreted as control sequences.

```
          \group_begin:
          \cctab_select:N
            \c_str_cctab
          \luabridge_tl_set:Nn
            \l_tmpa_tl
            { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
          \tl_gset:NV
            \markdownOptionOutputDir
            \l_tmpa_tl
          \group_end:
        }
        {
          \tl_gset:Nn
            \markdownOptionOutputDir
            { . }
        }
    }
  }
\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 }
          }
      }
  }
\cs_new:Nn
  \@@_define_option_keyval:nnn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \str_if_eq:VVTF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \keys_define:nn
          { markdown/options }
```

```
        {
```

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

```
          #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
          },
          #3 .default:n = { true },
        }
    }
    {
      \keys_define:nn
        { markdown/options }
        {
          #3 .code:n = {
            \@@_set_option_value:nn
              { #2 }
              { ##1 }
          },
        }
    }
```

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
              { markdown }
              { 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/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 }
  { p, F }
\msg_new:nnn
  { markdown }
  { malformed-name-for-clist-option }
  {
    Clist~option~name~#1~does~not~end~with~-s.
  }
```

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

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \@@_define_option_commands_and_keyvals:
  }
\ExplSyntaxOff
```

### 2.2.3 Themes

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

The key-values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩, optionally followed by `@`⟨*theme version*⟩, load a TEX document (further referred to as *a theme*) named `markdowntheme`⟨*munged theme name*⟩`.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (`/`) for an underscore (`_`). The theme name must be *qualified* and contain no underscores or at signs (`@`). Themes are inspired by the Beamer LATEX package, which provides similar functionality with its `\usetheme` macro [9, Section 15.1].

A theme name is qualified if and only if it contains at least one forward slash. Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is ⟨*theme author*⟩`/`⟨*theme purpose*⟩`/`⟨*private naming scheme*⟩, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the TEX directory structure. For example, loading a theme named `witiko/beamer/MU` would load a TEX document package named `markdownthemewitiko_beamer_MU.tex`.

If `@`⟨*theme version*⟩ is specified after ⟨*theme name*⟩, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If `@`⟨*theme version*⟩ is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [10].

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/options }
  {
    theme .code:n = {
      \@@_set_theme:n
        { #1 }
    },
    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/options/import }
        \l_tmpa_tl
    },
  }
```

To keep track of the current theme when themes are nested, we will maintain the stacks `\g_@@_theme_names_seq` and `\g_@@_theme_versions_seq` stack of theme names and versions, respectively. For convenience, the name of the current theme and version is also available in the macros `\g_@@_current_theme_tl` and `\markdownThemeVersion`, respectively.

```
\seq_new:N
  \g_@@_theme_names_seq
\seq_new:N
  \g_@@_theme_versions_seq
\tl_new:N
```

```
  \g_@@_current_theme_tl
\tl_gset:Nn
  \g_@@_current_theme_tl
  { }
\seq_gput_right:NV
  \g_@@_theme_names_seq
  \g_@@_current_theme_tl
\cs_new:Npn
  \markdownThemeVersion
  { }
\seq_gput_right:NV
  \g_@@_theme_versions_seq
  \g_@@_current_theme_tl
\cs_new:Nn
  \@@_set_theme:n
  {
```

First, we validate the theme name.

```
    \str_if_in:nnF
      { #1 }
      { / }
      {
        \msg_error:nnn
          { markdown }
          { unqualified-theme-name }
          { #1 }
      }
    \str_if_in:nnT
      { #1 }
      { _ }
      {
        \msg_error:nnn
          { markdown }
          { underscores-in-theme-name }
          { #1 }
      }
```

Next, we extract the theme version.

```
    \str_if_in:nnTF
      { #1 }
      { @ }
      {
        \regex_extract_once:nnN
          { (.*) @ (.*) }
          { #1 }
          \l_tmpa_seq
        \seq_gpop_left:NN
          \l_tmpa_seq
```

```
      \l_tmpa_tl
    \seq_gpop_left:NN
      \l_tmpa_seq
      \l_tmpa_tl
    \tl_gset:NV
      \g_@@_current_theme_tl
      \l_tmpa_tl
    \seq_gpop_left:NN
      \l_tmpa_seq
      \l_tmpa_tl
    \cs_gset:Npe
      \markdownThemeVersion
      {
        \tl_use:N
          \l_tmpa_tl
      }
  }
  {
    \tl_gset:Nn
      \g_@@_current_theme_tl
      { #1 }
    \cs_gset:Npn
      \markdownThemeVersion
      { latest }
  }
```

Next, we munge the theme name.

```
  \str_set:NV
    \l_tmpa_str
    \g_@@_current_theme_tl
  \str_replace_all:Nnn
    \l_tmpa_str
    { / }
    { _ }
```

Finally, we load the theme. Before loading the theme, we push down the current name and version of the theme on the stack.

```
  \tl_set:NV
    \l_tmpa_tl
    \g_@@_current_theme_tl
  \tl_put_right:Nn
    \g_@@_current_theme_tl
    { / }
  \seq_gput_right:NV
    \g_@@_theme_names_seq
    \g_@@_current_theme_tl
  \seq_gput_right:NV
    \g_@@_theme_versions_seq
```

```
      \markdownThemeVersion
    \@@_load_theme:VeV
      \l_tmpa_tl
      { \markdownThemeVersion }
      \l_tmpa_str
```

After the theme has been loaded, we recover the name and version of the previous theme from the stack.

```
    \seq_gpop_right:NN
      \g_@@_theme_names_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_theme_names_seq
      \l_tmpa_tl
    \tl_gset:NV
      \g_@@_current_theme_tl
      \l_tmpa_tl
    \seq_gpop_right:NN
      \g_@@_theme_versions_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_theme_versions_seq
      \l_tmpa_tl
    \cs_gset:Npe
      \markdownThemeVersion
      {
        \tl_use:N
          \l_tmpa_tl
      }
  }
\msg_new:nnnn
  { markdown }
  { unqualified-theme-name }
  { Won't~load~theme~with~unqualified~name~#1 }
  { Theme~names~must~contain~at~least~one~forward~slash }
\msg_new:nnnn
  { markdown }
  { underscores-in-theme-name }
  { Won't~load~theme~with~an~underscore~in~its~name~#1 }
  { Theme~names~must~not~contain~underscores~in~their~names }
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NnV }
\cs_generate_variant:Nn
  \cs_gset:Npn
  { Npe }
\ExplSyntaxOff
```

Built-in plain TEX themes provided with the Markdown package include:

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

```
\input markdown
\markdownSetup{import=witiko/tilde}
\markdownBegin
Bartel~Leendert van~der~Waerden
\markdownEnd
\bye
```

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

**witiko/markdown/defaults** A plain TEX theme with the default definitions of token renderer prototypes for plain TEX. This theme is loaded automatically together with the package and explicitly loading it has no effect.

Please, see Section 3.2.2 for implementation details of the built-in plain TEX themes.

### 2.2.4 Snippets

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

```
\ExplSyntaxOn
\prop_new:N
  \g_@@_snippets_prop
\cs_new:Nn
  \@@_setup_snippet:nn
  {
    \tl_if_empty:nT
      { #1 }
      {
        \msg_error:nnn
          { markdown }
          { empty-snippet-name }
          { #1 }
      }
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_put_right:Nn
```

```
      \l_tmpa_tl
      { #1 }
    \@@_if_snippet_exists:nT
      { #1 }
      {
        \msg_warning:nnV
          { markdown }
          { redefined-snippet }
          \l_tmpa_tl
      }
    \keys_precompile:nnN
      { markdown/options }
      { #2 }
      \l_tmpb_tl
    \prop_gput:NVV
      \g_@@_snippets_prop
      \l_tmpa_tl
      \l_tmpb_tl
  }
\cs_gset_eq:NN
  \markdownSetupSnippet
  \@@_setup_snippet:nn
\msg_new:nnnn
  { markdown }
  { empty-snippet-name }
  { Empty~snippet~name~#1 }
  { Pick~a~non-empty~name~for~your~snippet }
\msg_new:nnn
  { markdown }
  { redefined-snippet }
  { Redefined~snippet~#1 }
```

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

```
\tl_new:N
  \l_@@_current_snippet_tl
\prg_new_conditional:Nnn
  \@@_if_snippet_exists:n
  { TF, T, F }
  {
    \tl_set:NV
      \l_@@_current_snippet_tl
      \g_@@_current_theme_tl
    \tl_put_right:Nn
      \l_@@_current_snippet_tl
      { #1 }
    \prop_if_in:NVTF
      \g_@@_snippets_prop
```

```
      \l_@@_current_snippet_tl
      { \prg_return_true: }
      { \prg_return_false: }
  }
\cs_gset_eq:NN
  \markdownIfSnippetExists
  \@@_if_snippet_exists:nTF
```

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

```
\keys_define:nn
  { markdown/options }
  {
    snippet .code:n = {
      \tl_set:NV
        \l_tmpa_tl
        \g_@@_current_theme_tl
      \tl_put_right:Nn
        \l_tmpa_tl
        { #1 }
      \@@_if_snippet_exists:nTF
        { #1 }
        {
          \prg_get:NVN
            \g_@@_snippets_prop
            \l_tmpa_tl
            \l_tmpb_tl
          \tl_use:N
            \l_tmpb_tl
        }
        {
          \msg_error:nnV
            { markdown }
            { undefined-snippet }
            \l_tmpa_tl
        }
    }
  }
\msg_new:nnn
  { markdown }
  { undefined-snippet }
  { Can't~invoke~undefined~snippet~#1 }
\ExplSyntaxOff
```

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

```
\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 could import the `jdoe/lists` theme and use the qualified name `jdoe/lists/romanNumerals` to invoke 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 for easier access:

```
\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 can be different from the name of the snippet in the `jdoe/lists` theme. For example, we can make the snippet `jdoe/lists/romanNumerals` available under the name `roman`.

```
\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:

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

```
    jdoe/yetanotherlongpackagename,
  },
}
```

```
\ExplSyntaxOn
\tl_new:N
  \l_@@_import_current_theme_tl
\keys_define:nn
  { markdown/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_@@_import_current_theme_tl
        \l_keys_key_str
      \tl_replace_all:NVn
        \l_@@_import_current_theme_tl
        \c_backslash_str
        { / }
```

Here, we import the 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_@@_import_current_theme_tl
          \@@_setup_snippet:Vx
            \l_tmpb_tl
            { snippet = { \l_tmpa_tl } }
        }
```

Here, we load the theme.

```
      \@@_set_theme:V
        \l_@@_import_current_theme_tl
    },
  }
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NVn }
\cs_generate_variant:Nn
  \@@_set_theme:n
  { V }
\cs_generate_variant:Nn
  \@@_setup_snippet:nn
  { Vx }
```

### 2.2.5 Token Renderers

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

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

```
\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
```

#### 2.2.5.1 Attribute Renderers

The following macros are only produced, when at least one of the following options for markdown attributes on different elements is enabled:

- `autoIdentifiers`
- `fencedCodeAttributes`
- `gfmAutoIdentifiers`
- `headerAttributes`
- `inlineCodeAttributes`
- `linkAttributes`

`\markdownRendererAttributeIdentifier` represents the ⟨*identifier*⟩ of a markdown element (`id="`⟨*identifier*⟩`"` in HTML and `#`⟨*identifier*⟩ in markdown attributes). 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 attributes). 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
```

#### 2.2.5.2 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
```

#### 2.2.5.3 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 a context in which the attributes of an inline bracketed 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
```

#### 2.2.5.4 Bullet List Renderers

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

```
\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
```

#### 2.2.5.5 Citation Renderers

The \markdownRendererCite macro represents a string of one or more parenthetical citations. This macro will only be produced, when the citations option is enabled. The macro receives the parameter {⟨*number of citations*⟩} followed by ⟨*suppress author*⟩ {⟨*prenote*⟩}{⟨*postnote*⟩}{⟨*name*⟩} repeated ⟨*number of citations*⟩ times. The ⟨*suppress author*⟩ parameter is either the token -, when the author's name is to be suppressed, or + otherwise.

```
\def\markdownRendererCite{%
  \markdownRendererCitePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { cite }
```

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

The `\markdownRendererTextCite` macro represents a string of one or more text citations. This macro will only be produced, when the `citations` option is enabled. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

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

#### 2.2.5.6 Code Block Renderers

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

```
\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 three arguments that correspond to the filename of a file containing the code block contents, the fully escaped code fence infostring that can be directly typeset, and the raw code fence infostring that can be used outside typesetting.

```
\def\markdownRendererInputFencedCode{%
  \markdownRendererInputFencedCodePrototype}%
```

```
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputFencedCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputFencedCode }
  { 3 }
\ExplSyntaxOff
```

#### 2.2.5.7 Code Span Renderer

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

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

#### 2.2.5.8 Code Span Attribute Context Renderers

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

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

```
\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
```

#### 2.2.5.9 Content Block Renderers

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

```
\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[32] 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 [4] 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
```

#### 2.2.5.10 Definition List Renderers

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

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

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

[32] 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 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\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
```

#### 2.2.5.11 Ellipsis Renderer

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

```
\def\markdownRendererEllipsis{%
  \markdownRendererEllipsisPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ellipsis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ellipsis }
  { 0 }
\ExplSyntaxOff
```

#### 2.2.5.12 Emphasis Renderers

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

```
\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
```

#### 2.2.5.13 Fenced Code Attribute Context Renderers

The following macros are only produced, when the fencedCode and fencedCodeAttributes options are 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
```

#### 2.2.5.14 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 context in which the attributes of a 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
```

#### 2.2.5.15 Header Attribute Context Renderers

The following macros are only produced, when the `autoIdentifiers`, `gfmAutoIdentifiers`, or `headerAttributes` options are enabled.

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

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

```
  \markdownRendererHeaderAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

#### 2.2.5.16 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
```

#### 2.2.5.17 Inline HTML Comment Renderer

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

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

#### 2.2.5.18 HTML Tag and Element Renderers

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

The `\markdownRendererInputBlockHtmlElement` macro represents a block HTML element. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that filename of a file containing the contents of the HTML element.

```
\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
```

#### 2.2.5.19 Image Renderer

The `\markdownRendererImage` macro represents an image. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

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

#### 2.2.5.20 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 a context in which the attributes of an 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
```

#### 2.2.5.21 Interblock Separator Renderers

The `\markdownRendererInterblockSeparator` macro represents an interblock 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
```

Users can use more than one blank line to delimit two block to indicate the end of a series of blocks that make up a logical paragraph. This produces a paragraph separator instead of an interblock separator. Between some blocks, such as markdown paragraphs, a paragraph separator is always produced.

The `\markdownRendererParagraphSeparator` macro represents a paragraph separator. The macro receives no arguments.

```
\def\markdownRendererParagraphSeparator{%
  \markdownRendererParagraphSeparatorPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { paragraphSeparator }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { paragraphSeparator }
  { 0 }
\ExplSyntaxOff
```

#### 2.2.5.22 Line Block Renderers

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

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

```
\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
```

#### 2.2.5.23 Line Break Renderers

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

```
\def\markdownRendererSoftLineBreak{%
  \markdownRendererSoftLineBreakPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { softLineBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { softLineBreak }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererHardLineBreak{%
  \markdownRendererHardLineBreakPrototype}%
```

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

#### 2.2.5.24 Link Renderer

The `\markdownRendererLink` macro represents a hyperlink. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```
\def\markdownRendererLink{%
  \markdownRendererLinkPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { link }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { link }
  { 4 }
\ExplSyntaxOff
```

#### 2.2.5.25 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 context in which the attributes of a 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
```

#### 2.2.5.26 Marked Text Renderer

The following macro is only produced, when the `mark` option is enabled.

The `\markdownRendererMark` macro represents a span of marked or highlighted text. The macro receives a single argument that corresponds to the marked text.

```
\def\markdownRendererMark{%
  \markdownRendererMarkPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { mark }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { mark }
  { 1 }
\ExplSyntaxOff
```

#### 2.2.5.27 Markdown Document Renderers

The `\markdownRendererDocumentBegin` and `\markdownRendererDocumentEnd` macros represent the beginning and the end of a *markdown* document. The macros receive no arguments.

A TEX document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```
\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
```

#### 2.2.5.28 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
```

#### 2.2.5.29 Note Renderer

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

```
\def\markdownRendererNote{%
  \markdownRendererNotePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { note }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { note }
  { 1 }
\ExplSyntaxOff
```

#### 2.2.5.30 Ordered List Renderers

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

```
\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
```

#### 2.2.5.31 Raw Content Renderers

The `\markdownRendererInputRawInline` macro represents an inline raw span. The macro receives two arguments: the filename of a file containing the inline raw

span contents and the raw attribute that designates the format of the inline raw span. This macro will only be produced, when the `rawAttribute` option is enabled.

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

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

#### 2.2.5.32 Section Renderers

The `\markdownRendererSectionBegin` and `\markdownRendererSectionEnd` 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
```

#### 2.2.5.33 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
```

#### 2.2.5.34 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
```

#### 2.2.5.35 Strike-Through Renderer

The `\markdownRendererStrikeThrough` macro represents a strike-through span of text. The macro receives a single argument that corresponds to the striked-out span of text. This macro will only be produced, when the `strikeThrough` option is enabled.

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

#### 2.2.5.36 Subscript Renderer

The `\markdownRendererSubscript` macro represents a subscript span of text. The macro receives a single argument that corresponds to the subscript span of text. This macro will only be produced, when the `subscripts` option is enabled.

```
\def\markdownRendererSubscript{%
  \markdownRendererSubscriptPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { subscript }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { subscript }
  { 1 }
```

#### 2.2.5.37 Superscript Renderer

The `\markdownRendererSuperscript` macro represents a superscript span of text. The macro receives a single argument that corresponds to the superscript span of text. This macro will only be produced, when the `superscripts` option is enabled.

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

#### 2.2.5.38 Table Attribute Context Renderers

The following macros are only produced, when the `tableCaptions` and `tableAttributes` options are enabled.

The `\markdownRendererTableAttributeContextBegin` and `\markdownRendererTableAttribu` macros represent the beginning and the end of a context in which the attributes of a table apply. The macros receive no arguments.

```
\def\markdownRendererTableAttributeContextBegin{%
  \markdownRendererTableAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tableAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tableAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererTableAttributeContextEnd{%
  \markdownRendererTableAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tableAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tableAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

#### 2.2.5.39 Table Renderer

The `\markdownRendererTable` macro represents a table. This macro will only be produced, when the `pipeTables` option is enabled. The macro receives the parameters `{`⟨*caption*⟩`}{`⟨*number of rows*⟩`}{`⟨*number of columns*⟩`}` followed by `{`⟨*alignments*⟩`}` and then by `{`⟨*row*⟩`}` repeated ⟨*number of rows*⟩ times, where ⟨*row*⟩ is `{`⟨*column*⟩`}` repeated ⟨*number of columns*⟩ times, ⟨*alignments*⟩ is ⟨*alignment*⟩ repeated ⟨*number of columns*⟩ times, and ⟨*alignment*⟩ is one of the following:

- `d` – The corresponding column has an unspecified (default) alignment.
- `l` – The corresponding column is left-aligned.
- `c` – The corresponding column is centered.
- `r` – The corresponding column is right-aligned.

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

#### 2.2.5.40 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
```

#### 2.2.5.41 Thematic Break Renderer

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

```
\def\markdownRendererThematicBreak{%
  \markdownRendererThematicBreakPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { thematicBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { thematicBreak }
  { 0 }
\ExplSyntaxOff
```

#### 2.2.5.42 Tickbox Renderers

The macros named `\markdownRendererTickedBox`, `\markdownRendererHalfTickedBox`, and `\markdownRendererUntickedBox` represent ticked and unticked boxes, respectively. These macros will either be produced, when the `taskLists` option is enabled, or when the Ballot Box with X (☒, U+2612), Hourglass (⊡, U+231B) or Ballot Box (▫, U+2610) Unicode characters are encountered in the markdown input, respectively.

```
\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
```

#### 2.2.5.43 Warning and Error Renderers

The `\markdownRendererWarning` and `\markdownRendererError` macros represent warnings and errors produced by the markdown parser. Both macros receive four parameters:

1. The fully escaped text of the warning or error that can be directly typeset
2. The raw text of the warning or error that can be used outside typesetting for e.g. logging the warning or error.
3. The fully escaped text with more details about the warning or error that can be directly typeset. Can be empty, unlike the first two parameters.
4. The raw text with more details about the warning or error that can be used outside typesetting for e.g. logging the warning or error. Can be empty, unlike the first two parameters.

```
\def\markdownRendererWarning{%
  \markdownRendererWarningPrototype}%
\def\markdownRendererError{%
  \markdownRendererErrorPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { warning }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { warning }
```

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

#### 2.2.5.44 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 \markdownRendererJekyllDataTypographicString and \markdownRendererJekyllDataP macros represent string scalar values 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.

For each string scalar value, both macros are produced. Whereas \markdownRendererJekyllDataT receives the scalar value after all markdown markup and special TEX characters in the string have been replaced by TEX macros, \markdownRendererJekyllDataProgrammaticString receives the raw scalar value. Therefore, whereas the \markdownRendererJekyllDataTypographicSt macro is more appropriate for texts that are supposed to be typeset with TEX, such as document titles, author names, or exam questions, the \markdownRendererJekyllDataProgrammaticString macro is more appropriate for identifiers and other programmatic text that won't be typeset by TEX.

```
\def\markdownRendererJekyllDataTypographicString{%
  \markdownRendererJekyllDataTypographicStringPrototype}%
\def\markdownRendererJekyllDataProgrammaticString{%
  \markdownRendererJekyllDataProgrammaticStringPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataTypographicString }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataTypographicString }
  { 2 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataProgrammaticString }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataProgrammaticString }
  { 2 }
\ExplSyntaxOff
```

Before Markdown 3.7.0, the \markdownRendererJekyllDataTypographicString macro was named \markdownRendererJekyllDataString and the \markdownRendererJekyllData macro was not produced. The \markdownRendererJekyllDataString has been deprecated and will be removed in Markdown 4.0.0.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererJekyllDataTypographicString
```

```
{
  \cs_if_exist:NTF
    \markdownRendererJekyllDataString
    {
      \@@_if_option:nTF
        { experimental }
        {
          \markdownError
            {
              The~jekyllDataString~renderer~has~been~deprecated,~
              to~be~removed~in~Markdown~4.0.0
            }
        }
        {
          \markdownWarning
            {
              The~jekyllDataString~renderer~has~been~deprecated,~
              to~be~removed~in~Markdown~4.0.0
            }
          \markdownRendererJekyllDataString
        }
    }
    {
      \cs_if_exist:NTF
        \markdownRendererJekyllDataStringPrototype
        {
          \@@_if_option:nTF
            { experimental }
            {
              \markdownError
                {
                  The~jekyllDataString~renderer~prototype~
                  has~been~deprecated,~
                  to~be~removed~in~Markdown~4.0.0
                }
            }
            {
              \markdownWarning
                {
                  The~jekyllDataString~renderer~prototype~
                  has~been~deprecated,~
                  to~be~removed~in~Markdown~4.0.0
                }
              \markdownRendererJekyllDataStringPrototype
            }
        }
        {
```

```
            \markdownRendererJekyllDataTypographicStringPrototype
          }
      }
  }
\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 2.2.6.1 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
```

#### 2.2.5.45 Generating Plain TEX Token Renderer Macros and Key-Values

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

```
\ExplSyntaxOn
\cs_new:Nn \@@_define_renderers:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
        \@@_define_renderer:n
          { ##1 }
```

```
      }
  }
\cs_new:Nn \@@_define_renderer:n
  {
    \@@_renderer_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_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 }
      }
  }
\tl_new:N
  \l_@@_renderer_definition_tl
\bool_new:N
  \g_@@_appending_renderer_bool
\cs_new:Nn \@@_define_renderer:nNn
  {
    \keys_define:nn
      { markdown/options/renderers }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_@@_renderer_definition_tl
            { ##1 }
          \regex_replace_all:nnN
            { \cP\#0 }
            { #1 }
            \l_@@_renderer_definition_tl
          \bool_if:NT
```

```
            \g_@@_appending_renderer_bool
            {
              \@@_tl_set_from_cs:NNn
                \l_tmpa_tl
                #2
                { #3 }
              \tl_put_left:NV
                \l_@@_renderer_definition_tl
                \l_tmpa_tl
            }
          \cs_generate_from_arg_count:NNnV
            #2
            \cs_set:Npn
            { #3 }
            \l_@@_renderer_definition_tl
        },
    }
```

If the token renderer macro has been deprecated, we undefine it.

The `\markdownRendererJekyllDataString` macro has been deprecated and will be removed in Markdown 4.0.0.

```
  \str_if_eq:nnT
    { #1 }
    { jekyllDataString }
    {
      \cs_undefine:N
        #2
    }
}
```

We define the function `\@@_tl_set_from_cs:NNn` [11]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```
\cs_new_protected:Nn
  \@@_tl_set_from_cs:NNn
  {
    \tl_set:Nn
      \l_tmpa_tl
      { #2 }
    \int_step_inline:nn
      { #3 }
      {
        \exp_args:NNc
          \tl_put_right:Nn
          \l_tmpa_tl
          { @@_tl_set_from_cs_parameter_ ##1 }
```

```
      }
    \exp_args:NNV
      \tl_set:No
      \l_tmpb_tl
      \l_tmpa_tl
    \regex_replace_all:nnN
      { \cP. }
      { \0\0 }
      \l_tmpb_tl
    \int_step_inline:nn
      { #3 }
      {
        \regex_replace_all:nnN
          { \c { @@_tl_set_from_cs_parameter_ ##1 } }
          { \cP\# ##1 }
          \l_tmpb_tl
      }
    \tl_set:NV
      #1
      \l_tmpb_tl
  }
\cs_generate_variant:Nn
  \@@_define_renderer:nNn
  { ncV }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { NNnV }
\cs_generate_variant:Nn
  \tl_put_left:Nn
  { Nv }
\keys_define:nn
  { markdown/options }
  {
    renderers .code:n = {
      \keys_set:nn
        { markdown/options/renderers }
        { #1 }
    },
  }
```

The following example code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` token renderer macros.

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

```
  }
}
```

```
\tl_new:N
  \l_@@_renderer_glob_definition_tl
\seq_new:N
  \l_@@_renderer_glob_results_seq
\regex_const:Nn
  \c_@@_appending_key_regex
  { \s*+$ }
\keys_define:nn
  { markdown/options/renderers }
  {
    unknown .code:n = {
```

Besides defining renderers at once, we can also define them incrementally using the appending operator (`+=`). This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
  renderers = {
    % Start with empty renderers.
    headerAttributeContextBegin = {},
    attributeClassName = {},
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
    headerAttributeContextBegin += {
      \markdownSetup{
        renderers = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
        renderers = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
      \regex_match:NVTF
        \c_@@_appending_key_regex
        \l_keys_key_str
        {
          \bool_gset_true:N
            \g_@@_appending_renderer_bool
          \tl_set:NV
            \l_tmpa_tl
            \l_keys_key_str
          \regex_replace_once:NnN
            \c_@@_appending_key_regex
            { }
            \l_tmpa_tl
          \tl_set:Nx
            \l_tmpb_tl
            { { \l_tmpa_tl } = }
          \tl_put_right:Nn
            \l_tmpb_tl
            { { #1 } }
          \keys_set:nV
            { markdown/options/renderers }
            \l_tmpb_tl
          \bool_gset_false:N
            \g_@@_appending_renderer_bool
        }
```

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

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

Wildcards and enumerations can be combined:

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

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

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

```
3108          {
3109            \@@_glob_seq:VnN
3110              \l_keys_key_str
3111              { g_@@_renderers_seq }
3112              \l_@@_renderer_glob_results_seq
3113            \seq_if_empty:NTF
3114              \l_@@_renderer_glob_results_seq
3115              {
3116                \msg_error:nnV
3117                  { markdown }
3118                  { undefined-renderer }
3119                  \l_keys_key_str
3120              }
3121              {
3122                \tl_set:Nn
3123                  \l_@@_renderer_glob_definition_tl
3124                  { \exp_not:n { #1 } }
3125                \seq_map_inline:Nn
3126                  \l_@@_renderer_glob_results_seq
3127                  {
3128                    \tl_set:Nn
3129                      \l_tmpa_tl
3130                      { { ##1 } = }
3131                    \tl_put_right:Nx
3132                      \l_tmpa_tl
3133                      { { \l_@@_renderer_glob_definition_tl } }
3134                    \keys_set:nV
3135                      { markdown/options/renderers }
3136                      \l_tmpa_tl
3137                  }
3138              }
3139          }
3140      },
3141    }
3142  \msg_new:nnn
3143    { markdown }
3144    { undefined-renderer }
3145    {
```

```
    Renderer~#1~is~undefined.
  }
\cs_generate_variant:Nn
  \@@_glob_seq:nnN
  { VnN }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { cNVV }
\cs_generate_variant:Nn
  \msg_error:nnn
  { nnV }
\prg_generate_conditional_variant:Nnn
  \regex_match:Nn
  { NV }
  { TF }
\prop_new:N
  \g_@@_glob_cache_prop
\tl_new:N
  \l_@@_current_glob_tl
\cs_new:Nn
  \@@_glob_seq:nnN
  {
    \tl_set:Nn
      \l_@@_current_glob_tl
      { ^ #1 $ }
    \prop_get:NeNTF
      \g_@@_glob_cache_prop
      { #2 / \l_@@_current_glob_tl }
      \l_tmpa_clist
      {
        \seq_set_from_clist:NN
          #3
          \l_tmpa_clist
      }
      {
        \seq_clear:N
          #3
        \regex_replace_all:nnN
          { \* }
          { .* }
          \l_@@_current_glob_tl
        \regex_set:NV
          \l_tmpa_regex
          \l_@@_current_glob_tl
        \seq_map_inline:cn
          { #2 }
          {
```

```
            \regex_match:NnT
              \l_tmpa_regex
              { ##1 }
              {
                \seq_put_right:Nn
                  #3
                  { ##1 }
              }
          }
        \clist_set_from_seq:NN
          \l_tmpa_clist
          #3
        \prop_gput:NeV
          \g_@@_glob_cache_prop
          { #2 / \l_@@_current_glob_tl }
          \l_tmpa_clist
      }
  }
% TODO: Remove in TeX Live 2023.
\prg_generate_conditional_variant:Nnn
  \prop_get:NnN
  { NeN }
  { TF }
\cs_generate_variant:Nn
  \regex_set:Nn
  { NV }
\cs_generate_variant:Nn
  \prop_gput:Nnn
  { NeV }
```

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

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \@@_define_renderers:
  }
\ExplSyntaxOff
```

### 2.2.6 Token Renderer Prototypes

#### 2.2.6.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key–values from the l3keys module of the LaTeX3 kernel.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  { }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\@@_with_various_cases:nn
  { jekyllDataRenderers }
  {
    \keys_define:nn
      { markdown/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/options/jekyll-data-renderers }
            \l_tmpa_tl
        },
      }
  }
\keys_define:nn
  { markdown/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
```

#### 2.2.6.2 Generating Plain TEX Token Renderer Prototype Macros and Key-Values

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

```
\ExplSyntaxOn
\cs_new:Nn \@@_define_renderer_prototypes:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
        \@@_define_renderer_prototype:n
          { ##1 }
      }
  }
\cs_new:Nn \@@_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
    \@@_define_renderer_prototype:ncV
      { #1 }
      { \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
      }
  }
\tl_new:N
  \l_@@_renderer_prototype_definition_tl
\bool_new:N
  \g_@@_appending_renderer_prototype_bool
\cs_new:Nn \@@_define_renderer_prototype:nNn
  {
    \keys_define:nn
      { markdown/options/renderer-prototypes }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_@@_renderer_prototype_definition_tl
            { ##1 }
          \regex_replace_all:nnN
            { \cP\#0 }
            { #1 }
            \l_@@_renderer_prototype_definition_tl
          \bool_if:NT
            \g_@@_appending_renderer_prototype_bool
            {
              \@@_tl_set_from_cs:NNn
                \l_tmpa_tl
                #2
                { #3 }
              \tl_put_left:NV
                \l_@@_renderer_prototype_definition_tl
                \l_tmpa_tl
            }
          \cs_generate_from_arg_count:NNnV
            #2
            \cs_set:Npn
```

```
            { #3 }
            \l_@@_renderer_prototype_definition_tl
        },
    }
```

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

The \markdownRendererJekyllDataStringPrototype macro has been deprecated and will be removed in Markdown 4.0.0.

```
    \str_if_eq:nnF
      { #1 }
      { jekyllDataString }
      {
        \cs_if_free:NT
          #2
          {
            \cs_generate_from_arg_count:NNnn
              #2
              \cs_set:Npn
              { #3 }
              { }
          }
      }
  }
\cs_generate_variant:Nn
  \@@_define_renderer_prototype:nNn
  { ncV }
```

The following example code showcases a possible configuration of the \markdownRendererImagePrototype and \markdownRendererCodeSpanPrototype token renderer prototype macros.

```
\markdownSetup{
  rendererPrototypes = {
    image = {\pdfximage{#2}},    % Embed PDF images in the document.
    codeSpan = {{\tt #1}},     % Render inline code using monospace.
  }
}
```

```
\keys_define:nn
  { markdown/options/renderer-prototypes }
  {
    unknown .code:n = {
```

Besides defining renderer prototypes at once, we can also define them incrementally using the appending operator (+=). This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = {},
    attributeClassName = {},
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
3375        \regex_match:NVTF
3376          \c_@@_appending_key_regex
3377          \l_keys_key_str
3378          {
3379            \bool_gset_true:N
3380              \g_@@_appending_renderer_prototype_bool
3381            \tl_set:NV
3382              \l_tmpa_tl
3383              \l_keys_key_str
3384            \regex_replace_once:NnN
3385              \c_@@_appending_key_regex
3386              { }
3387              \l_tmpa_tl
3388            \tl_set:Nx
3389              \l_tmpb_tl
3390              { { \l_tmpa_tl } = }
3391            \tl_put_right:Nn
3392              \l_tmpb_tl
```

```
            { { #1 } }
          \keys_set:nV
            { markdown/options/renderer-prototypes }
            \l_tmpb_tl
          \bool_gset_false:N
            \g_@@_appending_renderer_prototype_bool
        }
```

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

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

Wildcards and enumerations can be combined:

```
\markdownSetup{
  rendererPrototypes = {
    *lItem(|End) = {"},           % Quote ordered/bullet list items.
  }
}
```

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

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

```
        {
          \@@_glob_seq:VnN
            \l_keys_key_str
            { g_@@_renderers_seq }
            \l_@@_renderer_glob_results_seq
          \seq_if_empty:NTF
```

```
            \l_@@_renderer_glob_results_seq
            {
              \msg_error:nnV
                { markdown }
                { undefined-renderer-prototype }
                \l_keys_key_str
            }
            {
              \tl_set:Nn
                \l_@@_renderer_glob_definition_tl
                { \exp_not:n { #1 } }
              \seq_map_inline:Nn
                \l_@@_renderer_glob_results_seq
                {
                  \tl_set:Nn
                    \l_tmpa_tl
                    { { ##1 } = }
                  \tl_put_right:Nx
                    \l_tmpa_tl
                    { { \l_@@_renderer_glob_definition_tl } }
                  \keys_set:nV
                    { markdown/options/renderer-prototypes }
                    \l_tmpa_tl
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer-prototype }
  {
    Renderer~prototype~#1~is~undefined.
  }
\@@_with_various_cases:nn
  { rendererPrototypes }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \keys_set:nn
            { markdown/options/renderer-prototypes }
            { ##1 }
        },
      }
  }
```

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

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \@@_define_renderer_prototypes:
  }
\ExplSyntaxOff
```

### 2.2.7 Logging Facilities

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

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

### 2.2.8 Miscellanea

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

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

The `\markdownReadAndConvert` macro implements the `\markdownBegin` and `\yamlBegin` macros. The first argument specifies the token sequence that will terminate the markdown input when the plain TEX special characters have had their category changed to *other*: `\markdownEnd` for the `\markdownBegin` macro and `\yamlEnd` for the `\yamlBegin` macro. 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}}%
  |gdef|yamlBegin{%
    |begingroup
    |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
    |markdownReadAndConvert{\yamlEnd}%
                           {|yamlEnd}}%
|endgroup
```

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

The `code` key, which can be used to immediately expand and execute code.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/options }
  {
    code .code:n = { #1 },
  }
\ExplSyntaxOff
```

This can be especially useful in snippets.

## 2.3 LaTeX Interface

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

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

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_latex_tl { latex }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_latex_tl
  }
\ExplSyntaxOff
```

```
3493 \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 2.3.3). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single LaTeX theme (see Section 2.3.4) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way LaTeX $2_\varepsilon$ parses package options.

### 2.3.1 Typesetting Markdown

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

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*` LaTeX environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain TeX interface.

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

```
3494 \newenvironment{markdown}\relax\relax
3495 \newenvironment{markdown*}[1]\relax\relax
```

Furthermore, both environments accept LaTeX interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown` and `markdown*` environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

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

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

You can't directly extend the `markdown` LaTeX environment by using it in other environments as follows:

```
\newenvironment{foo}%
               {code before \begin{markdown}[some, options]}%
               {\end{markdown} code after}
```

This is because the implementation looks for the literal string `\end{markdown}` to stop scanning the markdown text. However, you can work around this limitation by using the `\markdown` and `\markdownEnd` macros directly in the definition as follows:

```
\newenvironment{foo}%
               {code before \markdown[some, options]}%
               {\markdownEnd code after}
```

Specifically, the `\markdown` macro must appear at the end of the replacement before-text and must be followed by text that has not yet been ingested by TeX's input processor.

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown` LaTeX environment.

Lastly, you can't nest the other environments. For example, the following definition would be incorrect:

```
\newenvironment{bar}{\begin{foo}}{\end{foo}}
```

In this example, you should use the `\markdown` macro directly in the definition of the environment `bar`:

```
\newenvironment{bar}{\markdown[some, options]}{\markdownEnd}
```

The `yaml` LaTeX environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```
3496 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts LaTeX interface options (see Section 2.3.3) as the only optional argument.

The `yaml` environment is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example LaTeX code showcases the usage of the `yaml` environment:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{yaml}[smartEllipses]
title: _Hello_ **world** ...
author: John Doe
\end{yaml}
\end{document}
```

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}[
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]
title: _Hello_ **world** ...
author: John Doe
\end{markdown}
\end{document}
```

You can't directly extend the `yaml` LaTeX environment by using it in other environments. However, you can work around this limitation by using the `\yaml` and `\yamlEnd` macros directly in the definition, similarly to the `\markdown` and `\markdownEnd` macros described previously. Unlike with the `\markdown` and `\markdownEnd` macros, The `\yamlEnd` macro _must_ be used in or after the replacement after-text.

The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain TeX. Unlike the `\markinline` macro provided by the plain TeX interface, this macro also accepts LaTeX interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

#### 2.3.1.2 Typesetting Markdown and YAML from external documents

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 2.3.3) as its optional argument. These options will only influence 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}
```

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain TEX. Unlike the `\yamlInput` macro provided by the plain TEX interface, this macro also accepts LATEX interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

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

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]{hello.yml}
\end{document}
```

### 2.3.2 Using LaTeX hooks with the Markdown package

LaTeX provides an intricate hook management system that allows users to insert extra material before and after certain TeX macros and LaTeX environments, among other things. [12, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before TeX commands and before/after LaTeX environments without restriction:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}
```

Processing the above example with LaTeX will produce the text "markdownfoo emphasis: _bar_ baz!/markdown", as expected.

However, using hooks to insert extra material after TeX commands only works for commands with a fixed number of parameters that don't use currying.

If, in the above example, you explicitly defined the renderer for emphasis using `\markdownSetup` or another method that does not use currying, then you would be able to insert extra material even after the renderer:

```
\documentclass{article}
\usepackage{markdown}
\markdownSetup{renderers={emphasis={\emph{#1}}}}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{<emphasis>}
\AddToHook{cmd/markdownRendererEmphasis/after}{</emphasis>}
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}
```

Processing the above example with LaTeX will produce the text "markdownfoo emphasis_bar_/emphasis baz!/markdown", as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.12. In such a case, you would need to redefine the renderer in a way that does not use currying before you would be able to use hooks to insert extra material after it.

Hooks also cannot be used to insert extra material after renderers with a variable number of parameters such as the renderer for tables, see Section 2.2.5.39.

### 2.3.3 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.

LaTeX options map directly to the options recognized by the plain TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain TeX interface (see Sections 2.2.5 and 2.2.6).

The LaTeX options may be specified when loading the LaTeX package, when using the `markdown*` LaTeX environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

#### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [13, Section 5.1], which supports the `finalizecache` and `frozencache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain TeX options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `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}}
```

#### 2.3.3.2 Generating Plain TeX Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

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

```
\ExplSyntaxOn
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_latex_tl
  {
    \@@_define_option_commands_and_keyvals:
    \@@_define_renderers:
    \@@_define_renderer_prototypes:
  }
\ExplSyntaxOff
```

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

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

### 2.3.4 Themes

In Section 2.2.3, we described the concept of themes. In LaTeX, we expand on the concept of themes by allowing a theme to be a full-blown LaTeX package. Specifically, the key-values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩ load a LaTeX package named `markdowntheme`⟨*munged theme name*⟩`.sty` if it exists and a TeX document named `markdowntheme`⟨*munged theme name*⟩`.tex` otherwise.

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

If the LaTeX option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown LaTeX package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named

`witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` LaTeX package, and finally the `markdownthemewitiko_dot.sty` LaTeX package:

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

```
\newif\ifmarkdownLaTeXLoaded
  \markdownLaTeXLoadedfalse
```

Due to limitations of LaTeX, themes may not be loaded after the beginning of a LaTeX document.

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

**witiko/dot** A theme that typesets fenced code blocks with the `dot …` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

````
\documentclass{article}
\usepackage[import=witiko/dot]{markdown}
\setkeys{Gin}{
  width = \columnwidth,
  height = 0.65\paperheight,
  keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

  latex -> pmml;
  latex -> cmml;
  pmml -> slt;
  cmml -> opt;
  cmml -> prefix;
  cmml -> infix;
  pmml -> mterms [style=dashed];
  cmml -> mterms;
````
````

```
  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];
  infix [label = "Infix"];
  mterms [label = "M-Terms"];
}
```
```
\end{markdown}
\end{document}
```

Typesetting the above document produces the output shown in Figure 4.

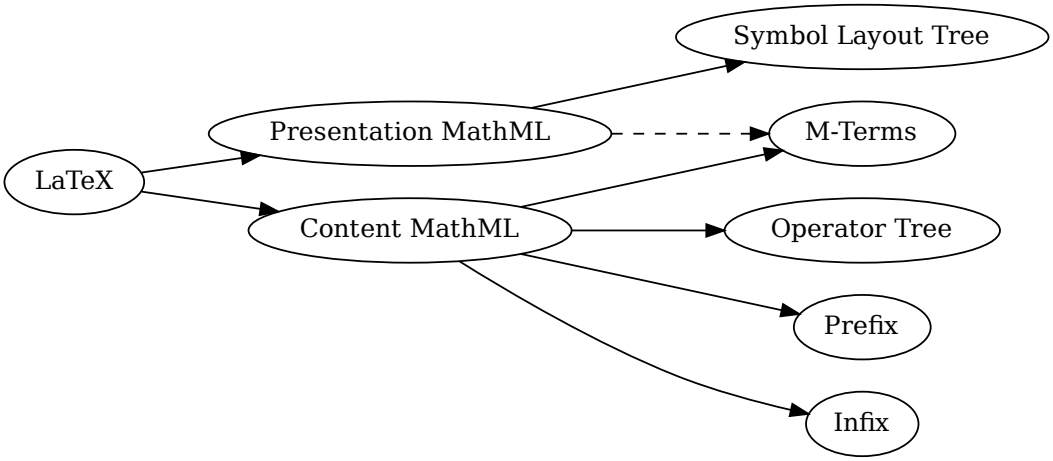


**Figure 4: Various formats of mathemathical formulae**

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.

```
3511 \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

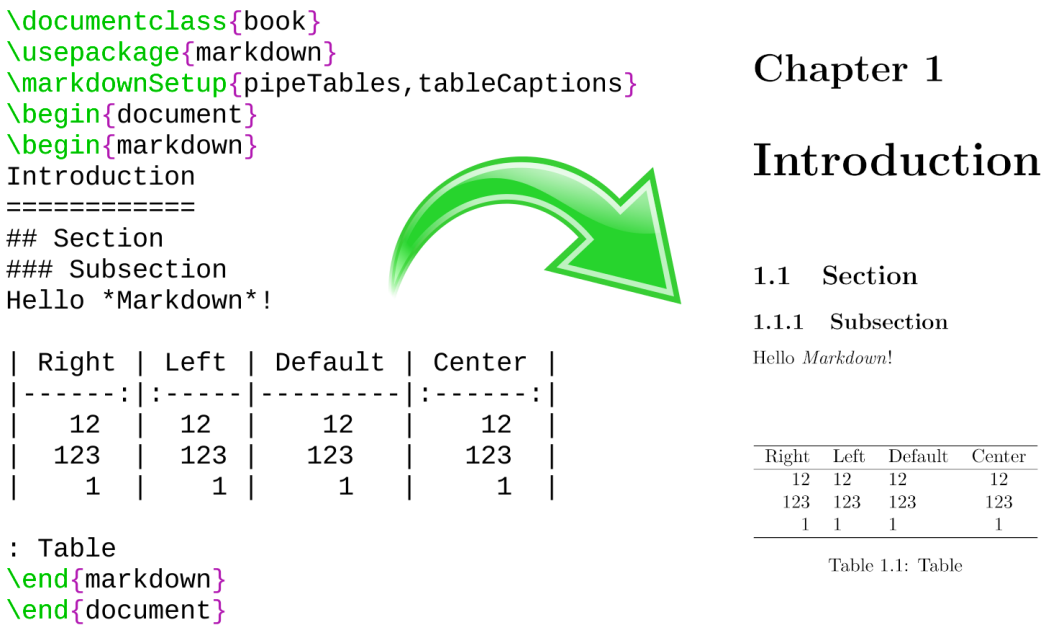


**Figure 5: The banner of the Markdown package**

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/markdown/defaults** A LaTeX theme with the default definitions of token renderer prototypes for plain TeX. This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
\AtEndOfPackage{
  \markdownLaTeXLoadedtrue
}
```

At the end of the LaTeX module, we load the witiko/markdown/defaults LaTeX theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option noDefaults has been enabled (see Section 2.2.2.3).

```
\ExplSyntaxOn
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_latex_tl
  {
    \ExplSyntaxOff
    \AtEndOfPackage
      {
        \@@_if_option:nF
          { noDefaults }
          {
            \@@_if_option:nTF
              { experimental }
              {
                \@@_setup:n
                  { theme = witiko/markdown/defaults@experimental }
              }
              {
                \@@_setup:n
                  { theme = witiko/markdown/defaults }
              }
          }
      }
    \ExplSyntaxOn
  }
\ExplSyntaxOff

\ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/10/29]%
```

Please, see Section 3.3.2 for implementation details of the built-in LaTeX themes.

## 2.4 ConTeXt Interface

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

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_context_tl { context }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
```

```
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_context_tl
  }
\ExplSyntaxOff
```

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

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

The interface exposes the `\startmarkdown`, `\stopmarkdown`, `\startyaml`, `\stopyaml`, `\inputmarkdown`, and `\inputyaml` macros.

#### 2.4.1.1 Typesetting Markdown and YAML directly

The `\startmarkdown` and `\stopmarkdown` macros are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain TEX interface.

```
\let\startmarkdown\relax
\let\stopmarkdown\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.

The macros `\startmarkdown` and `\stopmarkdown` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```
\usemodule[t][markdown]
\starttext
\startmarkdown
```

```
_Hello_ **world** ...
\stopmarkdown
\stoptext
```

The \startyaml and \stopyaml macros are aliases for the macros \yamlBegin and \yamlEnd exposed by the plain TEX interface.

```
\let\startyaml\relax
\let\stopyaml\relax
```

You may prepend your own code to the \startyaml macro and append your own code to the \stopyaml macro to produce special effects before and after the YAML document.

The macros \startyaml and \stopyaml are subject to the same limitations as the \markdownBegin and \markdownEnd macros.

The following example ConTEXt code showcases the usage of the \startyaml and \stopyaml macros:

```
\usemodule[t][markdown]
\starttext
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

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

```
\usemodule[t][markdown]
\starttext
\setupyaml[jekyllData, expectJekyllData, ensureJekyllData]
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

#### 2.4.1.2 Typesetting Markdown and YAML from external documents

The \inputmarkdown macro aliases the macro \markdownInput exposed by the plain TEX interface.

```
\let\inputmarkdown\relax
```

Furthermore, the `\inputmarkdown` macro also accepts ConTEXt interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example ConTEXt code showcases the usage of the `\inputmarkdown` macro:

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

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

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

The `\inputyaml` macro aliases the macro `\yamlInput` exposed by the plain TEX interface.

```
3566 \let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts ConTEXt interface options (see Section 2.4.2) as its optional argument. These options will only influence this YAML document.

The following example ConTEXt code showcases the usage of the `\inputyaml` macro:

```
\usemodule[t][markdown]
\starttext
\inputyaml[smartEllipses]{hello.yml}
\stoptext
```

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

```
\usemodule[t][markdown]
\starttext
\setupyaml[smartEllipses]
\inputyaml{hello.yml}
\stoptext
```

### 2.4.2 Options

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

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

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

```
\ExplSyntaxOn
\cs_new:Npn
  \setupmarkdown
  [ #1 ]
  {
    \@@_setup:n
      { #1 }
  }
```

The command `\setupyaml` is also available as an alias for the command `\setupmarkdown`.

```
\cs_gset_eq:NN
  \setupyaml
  \setupmarkdown
```

#### 2.4.2.1 Generating Plain TEX Option Macros and Key-Values

Unlike plain TEX, 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 }
```

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

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_context_tl
  {
    \@@_define_option_commands_and_keyvals:
    \@@_define_renderers:
    \@@_define_renderer_prototypes:
  }
\ExplSyntaxOff
```

### 2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConTEXt, we expand on the concept of themes by allowing a theme to be a full-blown ConTEXt module. Specifically, the key-values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩ load a ConTEXt module named `t-markdowntheme`⟨*munged theme name*⟩`.tex` if it exists and a TEX document named `markdowntheme`⟨*munged theme name*⟩`.tex` otherwise.

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

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

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

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

**witiko/markdown/defaults** A ConTEXt theme with the default definitions of token renderer prototypes for plain TEX. This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
\startmodule[markdownthemewitiko_markdown_defaults]
\unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConTEXt themes.

# 3 Implementation

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

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to 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, Cp, 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.Cp, 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 used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exists, it gets created with `transform(string)` as its content. Regardless, the pathname 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, create a new one.
    file = assert(io.open(name, "w"),
```

```
3616        [[Could not open file "]] .. name .. [[" for writing]])
3617      local result = string
3618      if transform ~= nil then
3619        result = transform(result)
3620      end
3621      assert(file:write(result))
3622      assert(file:close())
3623    end
3624    return name
3625  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.

```
3626  function util.cache_verbatim(dir, string)
3627    local name = util.cache(dir, string, nil, nil, ".verbatim")
3628    return name
3629  end
```

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

```
3630  function util.table_copy(t)
3631    local u = { }
3632    for k, v in pairs(t) do u[k] = v end
3633    return setmetatable(u, getmetatable(t))
3634  end
```

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

```
3635  function util.encode_json_string(s)
3636    s = s:gsub([[\]], [[\\]])
3637    s = s:gsub([["]], [[\"]])
3638    return [["]] .. s .. [["]]
3639  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 [14, Chapter 21].

```
3640  function util.expand_tabs_in_line(s, tabstop)
3641    local tab = tabstop or 4
3642    local corr = 0
3643    return (s:gsub("()\t", function(p)
3644              local sp = tab - (p - 1 + corr) % tab
3645              corr = corr - 1 + sp
3646              return string.rep(" ", sp)
3647            end))
3648  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 occurrences of escapable strings and characters (in this order).

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

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

Build a string of escapable characters.

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

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

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

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

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

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

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

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

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

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

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

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

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

The `util.salt` method produces cryptographic salt out of a table of options `options`.

```
function util.salt(options)
  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
```

The `cacheDir` option is disregarded.

```
    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
  return salt
end
```

The `util.warning` method produces a warning `s` that is unrelated to any specific markdown text being processed. For warnings that are specific to a markdown text, use `writer->warning` function.

```
function util.warning(s)
  io.stderr:write("Warning: " .. s .. "\n")
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,
```

```
3776   ["apos"] = 39,
3777   ["lpar"] = 40,
3778   ["rpar"] = 41,
3779   ["ast"] = 42,
3780   ["midast"] = 42,
3781   ["plus"] = 43,
3782   ["comma"] = 44,
3783   ["period"] = 46,
3784   ["sol"] = 47,
3785   ["colon"] = 58,
3786   ["semi"] = 59,
3787   ["LT"] = 60,
3788   ["lt"] = 60,
3789   ["nvlt"] = {60, 8402},
3790   ["bne"] = {61, 8421},
3791   ["equals"] = 61,
3792   ["GT"] = 62,
3793   ["gt"] = 62,
3794   ["nvgt"] = {62, 8402},
3795   ["quest"] = 63,
3796   ["commat"] = 64,
3797   ["lbrack"] = 91,
3798   ["lsqb"] = 91,
3799   ["bsol"] = 92,
3800   ["rbrack"] = 93,
3801   ["rsqb"] = 93,
3802   ["Hat"] = 94,
3803   ["UnderBar"] = 95,
3804   ["lowbar"] = 95,
3805   ["DiacriticalGrave"] = 96,
3806   ["grave"] = 96,
3807   ["fjlig"] = {102, 106},
3808   ["lbrace"] = 123,
3809   ["lcub"] = 123,
3810   ["VerticalLine"] = 124,
3811   ["verbar"] = 124,
3812   ["vert"] = 124,
3813   ["rbrace"] = 125,
3814   ["rcub"] = 125,
3815   ["NonBreakingSpace"] = 160,
3816   ["nbsp"] = 160,
3817   ["iexcl"] = 161,
3818   ["cent"] = 162,
3819   ["pound"] = 163,
3820   ["curren"] = 164,
3821   ["yen"] = 165,
3822   ["brvbar"] = 166,
```

```
3823   ["sect"] = 167,
3824   ["Dot"] = 168,
3825   ["DoubleDot"] = 168,
3826   ["die"] = 168,
3827   ["uml"] = 168,
3828   ["COPY"] = 169,
3829   ["copy"] = 169,
3830   ["ordf"] = 170,
3831   ["laquo"] = 171,
3832   ["not"] = 172,
3833   ["shy"] = 173,
3834   ["REG"] = 174,
3835   ["circledR"] = 174,
3836   ["reg"] = 174,
3837   ["macr"] = 175,
3838   ["strns"] = 175,
3839   ["deg"] = 176,
3840   ["PlusMinus"] = 177,
3841   ["plusmn"] = 177,
3842   ["pm"] = 177,
3843   ["sup2"] = 178,
3844   ["sup3"] = 179,
3845   ["DiacriticalAcute"] = 180,
3846   ["acute"] = 180,
3847   ["micro"] = 181,
3848   ["para"] = 182,
3849   ["CenterDot"] = 183,
3850   ["centerdot"] = 183,
3851   ["middot"] = 183,
3852   ["Cedilla"] = 184,
3853   ["cedil"] = 184,
3854   ["sup1"] = 185,
3855   ["ordm"] = 186,
3856   ["raquo"] = 187,
3857   ["frac14"] = 188,
3858   ["frac12"] = 189,
3859   ["half"] = 189,
3860   ["frac34"] = 190,
3861   ["iquest"] = 191,
3862   ["Agrave"] = 192,
3863   ["Aacute"] = 193,
3864   ["Acirc"] = 194,
3865   ["Atilde"] = 195,
3866   ["Auml"] = 196,
3867   ["Aring"] = 197,
3868   ["angst"] = 197,
3869   ["AElig"] = 198,
```

```
3870    ["Ccedil"] = 199,
3871    ["Egrave"] = 200,
3872    ["Eacute"] = 201,
3873    ["Ecirc"] = 202,
3874    ["Euml"] = 203,
3875    ["Igrave"] = 204,
3876    ["Iacute"] = 205,
3877    ["Icirc"] = 206,
3878    ["Iuml"] = 207,
3879    ["ETH"] = 208,
3880    ["Ntilde"] = 209,
3881    ["Ograve"] = 210,
3882    ["Oacute"] = 211,
3883    ["Ocirc"] = 212,
3884    ["Otilde"] = 213,
3885    ["Ouml"] = 214,
3886    ["times"] = 215,
3887    ["Oslash"] = 216,
3888    ["Ugrave"] = 217,
3889    ["Uacute"] = 218,
3890    ["Ucirc"] = 219,
3891    ["Uuml"] = 220,
3892    ["Yacute"] = 221,
3893    ["THORN"] = 222,
3894    ["szlig"] = 223,
3895    ["agrave"] = 224,
3896    ["aacute"] = 225,
3897    ["acirc"] = 226,
3898    ["atilde"] = 227,
3899    ["auml"] = 228,
3900    ["aring"] = 229,
3901    ["aelig"] = 230,
3902    ["ccedil"] = 231,
3903    ["egrave"] = 232,
3904    ["eacute"] = 233,
3905    ["ecirc"] = 234,
3906    ["euml"] = 235,
3907    ["igrave"] = 236,
3908    ["iacute"] = 237,
3909    ["icirc"] = 238,
3910    ["iuml"] = 239,
3911    ["eth"] = 240,
3912    ["ntilde"] = 241,
3913    ["ograve"] = 242,
3914    ["oacute"] = 243,
3915    ["ocirc"] = 244,
3916    ["otilde"] = 245,
```

```
3917   ["ouml"] = 246,
3918   ["div"] = 247,
3919   ["divide"] = 247,
3920   ["oslash"] = 248,
3921   ["ugrave"] = 249,
3922   ["uacute"] = 250,
3923   ["ucirc"] = 251,
3924   ["uuml"] = 252,
3925   ["yacute"] = 253,
3926   ["thorn"] = 254,
3927   ["yuml"] = 255,
3928   ["Amacr"] = 256,
3929   ["amacr"] = 257,
3930   ["Abreve"] = 258,
3931   ["abreve"] = 259,
3932   ["Aogon"] = 260,
3933   ["aogon"] = 261,
3934   ["Cacute"] = 262,
3935   ["cacute"] = 263,
3936   ["Ccirc"] = 264,
3937   ["ccirc"] = 265,
3938   ["Cdot"] = 266,
3939   ["cdot"] = 267,
3940   ["Ccaron"] = 268,
3941   ["ccaron"] = 269,
3942   ["Dcaron"] = 270,
3943   ["dcaron"] = 271,
3944   ["Dstrok"] = 272,
3945   ["dstrok"] = 273,
3946   ["Emacr"] = 274,
3947   ["emacr"] = 275,
3948   ["Edot"] = 278,
3949   ["edot"] = 279,
3950   ["Eogon"] = 280,
3951   ["eogon"] = 281,
3952   ["Ecaron"] = 282,
3953   ["ecaron"] = 283,
3954   ["Gcirc"] = 284,
3955   ["gcirc"] = 285,
3956   ["Gbreve"] = 286,
3957   ["gbreve"] = 287,
3958   ["Gdot"] = 288,
3959   ["gdot"] = 289,
3960   ["Gcedil"] = 290,
3961   ["Hcirc"] = 292,
3962   ["hcirc"] = 293,
3963   ["Hstrok"] = 294,
```

```
3964   ["hstrok"] = 295,
3965   ["Itilde"] = 296,
3966   ["itilde"] = 297,
3967   ["Imacr"] = 298,
3968   ["imacr"] = 299,
3969   ["Iogon"] = 302,
3970   ["iogon"] = 303,
3971   ["Idot"] = 304,
3972   ["imath"] = 305,
3973   ["inodot"] = 305,
3974   ["IJlig"] = 306,
3975   ["ijlig"] = 307,
3976   ["Jcirc"] = 308,
3977   ["jcirc"] = 309,
3978   ["Kcedil"] = 310,
3979   ["kcedil"] = 311,
3980   ["kgreen"] = 312,
3981   ["Lacute"] = 313,
3982   ["lacute"] = 314,
3983   ["Lcedil"] = 315,
3984   ["lcedil"] = 316,
3985   ["Lcaron"] = 317,
3986   ["lcaron"] = 318,
3987   ["Lmidot"] = 319,
3988   ["lmidot"] = 320,
3989   ["Lstrok"] = 321,
3990   ["lstrok"] = 322,
3991   ["Nacute"] = 323,
3992   ["nacute"] = 324,
3993   ["Ncedil"] = 325,
3994   ["ncedil"] = 326,
3995   ["Ncaron"] = 327,
3996   ["ncaron"] = 328,
3997   ["napos"] = 329,
3998   ["ENG"] = 330,
3999   ["eng"] = 331,
4000   ["Omacr"] = 332,
4001   ["omacr"] = 333,
4002   ["Odblac"] = 336,
4003   ["odblac"] = 337,
4004   ["OElig"] = 338,
4005   ["oelig"] = 339,
4006   ["Racute"] = 340,
4007   ["racute"] = 341,
4008   ["Rcedil"] = 342,
4009   ["rcedil"] = 343,
4010   ["Rcaron"] = 344,
```

```
4011    ["rcaron"] = 345,
4012    ["Sacute"] = 346,
4013    ["sacute"] = 347,
4014    ["Scirc"] = 348,
4015    ["scirc"] = 349,
4016    ["Scedil"] = 350,
4017    ["scedil"] = 351,
4018    ["Scaron"] = 352,
4019    ["scaron"] = 353,
4020    ["Tcedil"] = 354,
4021    ["tcedil"] = 355,
4022    ["Tcaron"] = 356,
4023    ["tcaron"] = 357,
4024    ["Tstrok"] = 358,
4025    ["tstrok"] = 359,
4026    ["Utilde"] = 360,
4027    ["utilde"] = 361,
4028    ["Umacr"] = 362,
4029    ["umacr"] = 363,
4030    ["Ubreve"] = 364,
4031    ["ubreve"] = 365,
4032    ["Uring"] = 366,
4033    ["uring"] = 367,
4034    ["Udblac"] = 368,
4035    ["udblac"] = 369,
4036    ["Uogon"] = 370,
4037    ["uogon"] = 371,
4038    ["Wcirc"] = 372,
4039    ["wcirc"] = 373,
4040    ["Ycirc"] = 374,
4041    ["ycirc"] = 375,
4042    ["Yuml"] = 376,
4043    ["Zacute"] = 377,
4044    ["zacute"] = 378,
4045    ["Zdot"] = 379,
4046    ["zdot"] = 380,
4047    ["Zcaron"] = 381,
4048    ["zcaron"] = 382,
4049    ["fnof"] = 402,
4050    ["imped"] = 437,
4051    ["gacute"] = 501,
4052    ["jmath"] = 567,
4053    ["circ"] = 710,
4054    ["Hacek"] = 711,
4055    ["caron"] = 711,
4056    ["Breve"] = 728,
4057    ["breve"] = 728,
```

```
4058   ["DiacriticalDot"] = 729,
4059   ["dot"] = 729,
4060   ["ring"] = 730,
4061   ["ogon"] = 731,
4062   ["DiacriticalTilde"] = 732,
4063   ["tilde"] = 732,
4064   ["DiacriticalDoubleAcute"] = 733,
4065   ["dblac"] = 733,
4066   ["DownBreve"] = 785,
4067   ["Alpha"] = 913,
4068   ["Beta"] = 914,
4069   ["Gamma"] = 915,
4070   ["Delta"] = 916,
4071   ["Epsilon"] = 917,
4072   ["Zeta"] = 918,
4073   ["Eta"] = 919,
4074   ["Theta"] = 920,
4075   ["Iota"] = 921,
4076   ["Kappa"] = 922,
4077   ["Lambda"] = 923,
4078   ["Mu"] = 924,
4079   ["Nu"] = 925,
4080   ["Xi"] = 926,
4081   ["Omicron"] = 927,
4082   ["Pi"] = 928,
4083   ["Rho"] = 929,
4084   ["Sigma"] = 931,
4085   ["Tau"] = 932,
4086   ["Upsilon"] = 933,
4087   ["Phi"] = 934,
4088   ["Chi"] = 935,
4089   ["Psi"] = 936,
4090   ["Omega"] = 937,
4091   ["ohm"] = 937,
4092   ["alpha"] = 945,
4093   ["beta"] = 946,
4094   ["gamma"] = 947,
4095   ["delta"] = 948,
4096   ["epsi"] = 949,
4097   ["epsilon"] = 949,
4098   ["zeta"] = 950,
4099   ["eta"] = 951,
4100   ["theta"] = 952,
4101   ["iota"] = 953,
4102   ["kappa"] = 954,
4103   ["lambda"] = 955,
4104   ["mu"] = 956,
```

```
4105  ["nu"] = 957,
4106  ["xi"] = 958,
4107  ["omicron"] = 959,
4108  ["pi"] = 960,
4109  ["rho"] = 961,
4110  ["sigmaf"] = 962,
4111  ["sigmav"] = 962,
4112  ["varsigma"] = 962,
4113  ["sigma"] = 963,
4114  ["tau"] = 964,
4115  ["upsi"] = 965,
4116  ["upsilon"] = 965,
4117  ["phi"] = 966,
4118  ["chi"] = 967,
4119  ["psi"] = 968,
4120  ["omega"] = 969,
4121  ["thetasym"] = 977,
4122  ["thetav"] = 977,
4123  ["vartheta"] = 977,
4124  ["Upsi"] = 978,
4125  ["upsih"] = 978,
4126  ["phiv"] = 981,
4127  ["straightphi"] = 981,
4128  ["varphi"] = 981,
4129  ["piv"] = 982,
4130  ["varpi"] = 982,
4131  ["Gammad"] = 988,
4132  ["digamma"] = 989,
4133  ["gammad"] = 989,
4134  ["kappav"] = 1008,
4135  ["varkappa"] = 1008,
4136  ["rhov"] = 1009,
4137  ["varrho"] = 1009,
4138  ["epsiv"] = 1013,
4139  ["straightepsilon"] = 1013,
4140  ["varepsilon"] = 1013,
4141  ["backepsilon"] = 1014,
4142  ["bepsi"] = 1014,
4143  ["IOcy"] = 1025,
4144  ["DJcy"] = 1026,
4145  ["GJcy"] = 1027,
4146  ["Jukcy"] = 1028,
4147  ["DScy"] = 1029,
4148  ["Iukcy"] = 1030,
4149  ["YIcy"] = 1031,
4150  ["Jsercy"] = 1032,
4151  ["LJcy"] = 1033,
```

```
4152   ["NJcy"] = 1034,
4153   ["TSHcy"] = 1035,
4154   ["KJcy"] = 1036,
4155   ["Ubrcy"] = 1038,
4156   ["DZcy"] = 1039,
4157   ["Acy"] = 1040,
4158   ["Bcy"] = 1041,
4159   ["Vcy"] = 1042,
4160   ["Gcy"] = 1043,
4161   ["Dcy"] = 1044,
4162   ["IEcy"] = 1045,
4163   ["ZHcy"] = 1046,
4164   ["Zcy"] = 1047,
4165   ["Icy"] = 1048,
4166   ["Jcy"] = 1049,
4167   ["Kcy"] = 1050,
4168   ["Lcy"] = 1051,
4169   ["Mcy"] = 1052,
4170   ["Ncy"] = 1053,
4171   ["Ocy"] = 1054,
4172   ["Pcy"] = 1055,
4173   ["Rcy"] = 1056,
4174   ["Scy"] = 1057,
4175   ["Tcy"] = 1058,
4176   ["Ucy"] = 1059,
4177   ["Fcy"] = 1060,
4178   ["KHcy"] = 1061,
4179   ["TScy"] = 1062,
4180   ["CHcy"] = 1063,
4181   ["SHcy"] = 1064,
4182   ["SHCHcy"] = 1065,
4183   ["HARDcy"] = 1066,
4184   ["Ycy"] = 1067,
4185   ["SOFTcy"] = 1068,
4186   ["Ecy"] = 1069,
4187   ["YUcy"] = 1070,
4188   ["YAcy"] = 1071,
4189   ["acy"] = 1072,
4190   ["bcy"] = 1073,
4191   ["vcy"] = 1074,
4192   ["gcy"] = 1075,
4193   ["dcy"] = 1076,
4194   ["iecy"] = 1077,
4195   ["zhcy"] = 1078,
4196   ["zcy"] = 1079,
4197   ["icy"] = 1080,
4198   ["jcy"] = 1081,
```

```
4199  ["kcy"] = 1082,
4200  ["lcy"] = 1083,
4201  ["mcy"] = 1084,
4202  ["ncy"] = 1085,
4203  ["ocy"] = 1086,
4204  ["pcy"] = 1087,
4205  ["rcy"] = 1088,
4206  ["scy"] = 1089,
4207  ["tcy"] = 1090,
4208  ["ucy"] = 1091,
4209  ["fcy"] = 1092,
4210  ["khcy"] = 1093,
4211  ["tscy"] = 1094,
4212  ["chcy"] = 1095,
4213  ["shcy"] = 1096,
4214  ["shchcy"] = 1097,
4215  ["hardcy"] = 1098,
4216  ["ycy"] = 1099,
4217  ["softcy"] = 1100,
4218  ["ecy"] = 1101,
4219  ["yucy"] = 1102,
4220  ["yacy"] = 1103,
4221  ["iocy"] = 1105,
4222  ["djcy"] = 1106,
4223  ["gjcy"] = 1107,
4224  ["jukcy"] = 1108,
4225  ["dscy"] = 1109,
4226  ["iukcy"] = 1110,
4227  ["yicy"] = 1111,
4228  ["jsercy"] = 1112,
4229  ["ljcy"] = 1113,
4230  ["njcy"] = 1114,
4231  ["tshcy"] = 1115,
4232  ["kjcy"] = 1116,
4233  ["ubrcy"] = 1118,
4234  ["dzcy"] = 1119,
4235  ["ensp"] = 8194,
4236  ["emsp"] = 8195,
4237  ["emsp13"] = 8196,
4238  ["emsp14"] = 8197,
4239  ["numsp"] = 8199,
4240  ["puncsp"] = 8200,
4241  ["ThinSpace"] = 8201,
4242  ["thinsp"] = 8201,
4243  ["VeryThinSpace"] = 8202,
4244  ["hairsp"] = 8202,
4245  ["NegativeMediumSpace"] = 8203,
```

```
4246  ["NegativeThickSpace"] = 8203,
4247  ["NegativeThinSpace"] = 8203,
4248  ["NegativeVeryThinSpace"] = 8203,
4249  ["ZeroWidthSpace"] = 8203,
4250  ["zwnj"] = 8204,
4251  ["zwj"] = 8205,
4252  ["lrm"] = 8206,
4253  ["rlm"] = 8207,
4254  ["dash"] = 8208,
4255  ["hyphen"] = 8208,
4256  ["ndash"] = 8211,
4257  ["mdash"] = 8212,
4258  ["horbar"] = 8213,
4259  ["Verbar"] = 8214,
4260  ["Vert"] = 8214,
4261  ["OpenCurlyQuote"] = 8216,
4262  ["lsquo"] = 8216,
4263  ["CloseCurlyQuote"] = 8217,
4264  ["rsquo"] = 8217,
4265  ["rsquor"] = 8217,
4266  ["lsquor"] = 8218,
4267  ["sbquo"] = 8218,
4268  ["OpenCurlyDoubleQuote"] = 8220,
4269  ["ldquo"] = 8220,
4270  ["CloseCurlyDoubleQuote"] = 8221,
4271  ["rdquo"] = 8221,
4272  ["rdquor"] = 8221,
4273  ["bdquo"] = 8222,
4274  ["ldquor"] = 8222,
4275  ["dagger"] = 8224,
4276  ["Dagger"] = 8225,
4277  ["ddagger"] = 8225,
4278  ["bull"] = 8226,
4279  ["bullet"] = 8226,
4280  ["nldr"] = 8229,
4281  ["hellip"] = 8230,
4282  ["mldr"] = 8230,
4283  ["permil"] = 8240,
4284  ["pertenk"] = 8241,
4285  ["prime"] = 8242,
4286  ["Prime"] = 8243,
4287  ["tprime"] = 8244,
4288  ["backprime"] = 8245,
4289  ["bprime"] = 8245,
4290  ["lsaquo"] = 8249,
4291  ["rsaquo"] = 8250,
4292  ["OverBar"] = 8254,
```

```
  ["oline"] = 8254,
  ["caret"] = 8257,
  ["hybull"] = 8259,
  ["frasl"] = 8260,
  ["bsemi"] = 8271,
  ["qprime"] = 8279,
  ["MediumSpace"] = 8287,
  ["ThickSpace"] = {8287, 8202},
  ["NoBreak"] = 8288,
  ["ApplyFunction"] = 8289,
  ["af"] = 8289,
  ["InvisibleTimes"] = 8290,
  ["it"] = 8290,
  ["InvisibleComma"] = 8291,
  ["ic"] = 8291,
  ["euro"] = 8364,
  ["TripleDot"] = 8411,
  ["tdot"] = 8411,
  ["DotDot"] = 8412,
  ["Copf"] = 8450,
  ["complexes"] = 8450,
  ["incare"] = 8453,
  ["gscr"] = 8458,
  ["HilbertSpace"] = 8459,
  ["Hscr"] = 8459,
  ["hamilt"] = 8459,
  ["Hfr"] = 8460,
  ["Poincareplane"] = 8460,
  ["Hopf"] = 8461,
  ["quaternions"] = 8461,
  ["planckh"] = 8462,
  ["hbar"] = 8463,
  ["hslash"] = 8463,
  ["planck"] = 8463,
  ["plankv"] = 8463,
  ["Iscr"] = 8464,
  ["imagline"] = 8464,
  ["Ifr"] = 8465,
  ["Im"] = 8465,
  ["image"] = 8465,
  ["imagpart"] = 8465,
  ["Laplacetrf"] = 8466,
  ["Lscr"] = 8466,
  ["lagran"] = 8466,
  ["ell"] = 8467,
  ["Nopf"] = 8469,
  ["naturals"] = 8469,
```

```
4340    ["numero"] = 8470,
4341    ["copysr"] = 8471,
4342    ["weierp"] = 8472,
4343    ["wp"] = 8472,
4344    ["Popf"] = 8473,
4345    ["primes"] = 8473,
4346    ["Qopf"] = 8474,
4347    ["rationals"] = 8474,
4348    ["Rscr"] = 8475,
4349    ["realine"] = 8475,
4350    ["Re"] = 8476,
4351    ["Rfr"] = 8476,
4352    ["real"] = 8476,
4353    ["realpart"] = 8476,
4354    ["Ropf"] = 8477,
4355    ["reals"] = 8477,
4356    ["rx"] = 8478,
4357    ["TRADE"] = 8482,
4358    ["trade"] = 8482,
4359    ["Zopf"] = 8484,
4360    ["integers"] = 8484,
4361    ["mho"] = 8487,
4362    ["Zfr"] = 8488,
4363    ["zeetrf"] = 8488,
4364    ["iiota"] = 8489,
4365    ["Bernoullis"] = 8492,
4366    ["Bscr"] = 8492,
4367    ["bernou"] = 8492,
4368    ["Cayleys"] = 8493,
4369    ["Cfr"] = 8493,
4370    ["escr"] = 8495,
4371    ["Escr"] = 8496,
4372    ["expectation"] = 8496,
4373    ["Fouriertrf"] = 8497,
4374    ["Fscr"] = 8497,
4375    ["Mellintrf"] = 8499,
4376    ["Mscr"] = 8499,
4377    ["phmmat"] = 8499,
4378    ["order"] = 8500,
4379    ["orderof"] = 8500,
4380    ["oscr"] = 8500,
4381    ["alefsym"] = 8501,
4382    ["aleph"] = 8501,
4383    ["beth"] = 8502,
4384    ["gimel"] = 8503,
4385    ["daleth"] = 8504,
4386    ["CapitalDifferentialD"] = 8517,
```

```
4387  ["DD"] = 8517,
4388  ["DifferentialD"] = 8518,
4389  ["dd"] = 8518,
4390  ["ExponentialE"] = 8519,
4391  ["ee"] = 8519,
4392  ["exponentiale"] = 8519,
4393  ["ImaginaryI"] = 8520,
4394  ["ii"] = 8520,
4395  ["frac13"] = 8531,
4396  ["frac23"] = 8532,
4397  ["frac15"] = 8533,
4398  ["frac25"] = 8534,
4399  ["frac35"] = 8535,
4400  ["frac45"] = 8536,
4401  ["frac16"] = 8537,
4402  ["frac56"] = 8538,
4403  ["frac18"] = 8539,
4404  ["frac38"] = 8540,
4405  ["frac58"] = 8541,
4406  ["frac78"] = 8542,
4407  ["LeftArrow"] = 8592,
4408  ["ShortLeftArrow"] = 8592,
4409  ["larr"] = 8592,
4410  ["leftarrow"] = 8592,
4411  ["slarr"] = 8592,
4412  ["ShortUpArrow"] = 8593,
4413  ["UpArrow"] = 8593,
4414  ["uarr"] = 8593,
4415  ["uparrow"] = 8593,
4416  ["RightArrow"] = 8594,
4417  ["ShortRightArrow"] = 8594,
4418  ["rarr"] = 8594,
4419  ["rightarrow"] = 8594,
4420  ["srarr"] = 8594,
4421  ["DownArrow"] = 8595,
4422  ["ShortDownArrow"] = 8595,
4423  ["darr"] = 8595,
4424  ["downarrow"] = 8595,
4425  ["LeftRightArrow"] = 8596,
4426  ["harr"] = 8596,
4427  ["leftrightarrow"] = 8596,
4428  ["UpDownArrow"] = 8597,
4429  ["updownarrow"] = 8597,
4430  ["varr"] = 8597,
4431  ["UpperLeftArrow"] = 8598,
4432  ["nwarr"] = 8598,
4433  ["nwarrow"] = 8598,
```

```
4434   ["UpperRightArrow"] = 8599,
4435   ["nearr"] = 8599,
4436   ["nearrow"] = 8599,
4437   ["LowerRightArrow"] = 8600,
4438   ["searr"] = 8600,
4439   ["searrow"] = 8600,
4440   ["LowerLeftArrow"] = 8601,
4441   ["swarr"] = 8601,
4442   ["swarrow"] = 8601,
4443   ["nlarr"] = 8602,
4444   ["nleftarrow"] = 8602,
4445   ["nrarr"] = 8603,
4446   ["nrightarrow"] = 8603,
4447   ["nrarrw"] = {8605, 824},
4448   ["rarrw"] = 8605,
4449   ["rightsquigarrow"] = 8605,
4450   ["Larr"] = 8606,
4451   ["twoheadleftarrow"] = 8606,
4452   ["Uarr"] = 8607,
4453   ["Rarr"] = 8608,
4454   ["twoheadrightarrow"] = 8608,
4455   ["Darr"] = 8609,
4456   ["larrtl"] = 8610,
4457   ["leftarrowtail"] = 8610,
4458   ["rarrtl"] = 8611,
4459   ["rightarrowtail"] = 8611,
4460   ["LeftTeeArrow"] = 8612,
4461   ["mapstoleft"] = 8612,
4462   ["UpTeeArrow"] = 8613,
4463   ["mapstoup"] = 8613,
4464   ["RightTeeArrow"] = 8614,
4465   ["map"] = 8614,
4466   ["mapsto"] = 8614,
4467   ["DownTeeArrow"] = 8615,
4468   ["mapstodown"] = 8615,
4469   ["hookleftarrow"] = 8617,
4470   ["larrhk"] = 8617,
4471   ["hookrightarrow"] = 8618,
4472   ["rarrhk"] = 8618,
4473   ["larrlp"] = 8619,
4474   ["looparrowleft"] = 8619,
4475   ["looparrowright"] = 8620,
4476   ["rarrlp"] = 8620,
4477   ["harrw"] = 8621,
4478   ["leftrightsquigarrow"] = 8621,
4479   ["nharr"] = 8622,
4480   ["nleftrightarrow"] = 8622,
```

```
["Lsh"] = 8624,
["lsh"] = 8624,
["Rsh"] = 8625,
["rsh"] = 8625,
["ldsh"] = 8626,
["rdsh"] = 8627,
["crarr"] = 8629,
["cularr"] = 8630,
["curvearrowleft"] = 8630,
["curarr"] = 8631,
["curvearrowright"] = 8631,
["circlearrowleft"] = 8634,
["olarr"] = 8634,
["circlearrowright"] = 8635,
["orarr"] = 8635,
["LeftVector"] = 8636,
["leftharpoonup"] = 8636,
["lharu"] = 8636,
["DownLeftVector"] = 8637,
["leftharpoondown"] = 8637,
["lhard"] = 8637,
["RightUpVector"] = 8638,
["uharr"] = 8638,
["upharpoonright"] = 8638,
["LeftUpVector"] = 8639,
["uharl"] = 8639,
["upharpoonleft"] = 8639,
["RightVector"] = 8640,
["rharu"] = 8640,
["rightharpoonup"] = 8640,
["DownRightVector"] = 8641,
["rhard"] = 8641,
["rightharpoondown"] = 8641,
["RightDownVector"] = 8642,
["dharr"] = 8642,
["downharpoonright"] = 8642,
["LeftDownVector"] = 8643,
["dharl"] = 8643,
["downharpoonleft"] = 8643,
["RightArrowLeftArrow"] = 8644,
["rightleftarrows"] = 8644,
["rlarr"] = 8644,
["UpArrowDownArrow"] = 8645,
["udarr"] = 8645,
["LeftArrowRightArrow"] = 8646,
["leftrightarrows"] = 8646,
["lrarr"] = 8646,
```

```
4528  ["leftleftarrows"] = 8647,
4529  ["llarr"] = 8647,
4530  ["upuparrows"] = 8648,
4531  ["uuarr"] = 8648,
4532  ["rightrightarrows"] = 8649,
4533  ["rrarr"] = 8649,
4534  ["ddarr"] = 8650,
4535  ["downdownarrows"] = 8650,
4536  ["ReverseEquilibrium"] = 8651,
4537  ["leftrightharpoons"] = 8651,
4538  ["lrhar"] = 8651,
4539  ["Equilibrium"] = 8652,
4540  ["rightleftharpoons"] = 8652,
4541  ["rlhar"] = 8652,
4542  ["nLeftarrow"] = 8653,
4543  ["nlArr"] = 8653,
4544  ["nLeftrightarrow"] = 8654,
4545  ["nhArr"] = 8654,
4546  ["nRightarrow"] = 8655,
4547  ["nrArr"] = 8655,
4548  ["DoubleLeftArrow"] = 8656,
4549  ["Leftarrow"] = 8656,
4550  ["lArr"] = 8656,
4551  ["DoubleUpArrow"] = 8657,
4552  ["Uparrow"] = 8657,
4553  ["uArr"] = 8657,
4554  ["DoubleRightArrow"] = 8658,
4555  ["Implies"] = 8658,
4556  ["Rightarrow"] = 8658,
4557  ["rArr"] = 8658,
4558  ["DoubleDownArrow"] = 8659,
4559  ["Downarrow"] = 8659,
4560  ["dArr"] = 8659,
4561  ["DoubleLeftRightArrow"] = 8660,
4562  ["Leftrightarrow"] = 8660,
4563  ["hArr"] = 8660,
4564  ["iff"] = 8660,
4565  ["DoubleUpDownArrow"] = 8661,
4566  ["Updownarrow"] = 8661,
4567  ["vArr"] = 8661,
4568  ["nwArr"] = 8662,
4569  ["neArr"] = 8663,
4570  ["seArr"] = 8664,
4571  ["swArr"] = 8665,
4572  ["Lleftarrow"] = 8666,
4573  ["lAarr"] = 8666,
4574  ["Rrightarrow"] = 8667,
```

```
  ["rAarr"] = 8667,
  ["zigrarr"] = 8669,
  ["LeftArrowBar"] = 8676,
  ["larrb"] = 8676,
  ["RightArrowBar"] = 8677,
  ["rarrb"] = 8677,
  ["DownArrowUpArrow"] = 8693,
  ["duarr"] = 8693,
  ["loarr"] = 8701,
  ["roarr"] = 8702,
  ["hoarr"] = 8703,
  ["ForAll"] = 8704,
  ["forall"] = 8704,
  ["comp"] = 8705,
  ["complement"] = 8705,
  ["PartialD"] = 8706,
  ["npart"] = {8706, 824},
  ["part"] = 8706,
  ["Exists"] = 8707,
  ["exist"] = 8707,
  ["NotExists"] = 8708,
  ["nexist"] = 8708,
  ["nexists"] = 8708,
  ["empty"] = 8709,
  ["emptyset"] = 8709,
  ["emptyv"] = 8709,
  ["varnothing"] = 8709,
  ["Del"] = 8711,
  ["nabla"] = 8711,
  ["Element"] = 8712,
  ["in"] = 8712,
  ["isin"] = 8712,
  ["isinv"] = 8712,
  ["NotElement"] = 8713,
  ["notin"] = 8713,
  ["notinva"] = 8713,
  ["ReverseElement"] = 8715,
  ["SuchThat"] = 8715,
  ["ni"] = 8715,
  ["niv"] = 8715,
  ["NotReverseElement"] = 8716,
  ["notni"] = 8716,
  ["notniva"] = 8716,
  ["Product"] = 8719,
  ["prod"] = 8719,
  ["Coproduct"] = 8720,
  ["coprod"] = 8720,
```

```
4622  ["Sum"] = 8721,
4623  ["sum"] = 8721,
4624  ["minus"] = 8722,
4625  ["MinusPlus"] = 8723,
4626  ["mnplus"] = 8723,
4627  ["mp"] = 8723,
4628  ["dotplus"] = 8724,
4629  ["plusdo"] = 8724,
4630  ["Backslash"] = 8726,
4631  ["setminus"] = 8726,
4632  ["setmn"] = 8726,
4633  ["smallsetminus"] = 8726,
4634  ["ssetmn"] = 8726,
4635  ["lowast"] = 8727,
4636  ["SmallCircle"] = 8728,
4637  ["compfn"] = 8728,
4638  ["Sqrt"] = 8730,
4639  ["radic"] = 8730,
4640  ["Proportional"] = 8733,
4641  ["prop"] = 8733,
4642  ["propto"] = 8733,
4643  ["varpropto"] = 8733,
4644  ["vprop"] = 8733,
4645  ["infin"] = 8734,
4646  ["angrt"] = 8735,
4647  ["ang"] = 8736,
4648  ["angle"] = 8736,
4649  ["nang"] = {8736, 8402},
4650  ["angmsd"] = 8737,
4651  ["measuredangle"] = 8737,
4652  ["angsph"] = 8738,
4653  ["VerticalBar"] = 8739,
4654  ["mid"] = 8739,
4655  ["shortmid"] = 8739,
4656  ["smid"] = 8739,
4657  ["NotVerticalBar"] = 8740,
4658  ["nmid"] = 8740,
4659  ["nshortmid"] = 8740,
4660  ["nsmid"] = 8740,
4661  ["DoubleVerticalBar"] = 8741,
4662  ["par"] = 8741,
4663  ["parallel"] = 8741,
4664  ["shortparallel"] = 8741,
4665  ["spar"] = 8741,
4666  ["NotDoubleVerticalBar"] = 8742,
4667  ["npar"] = 8742,
4668  ["nparallel"] = 8742,
```

```
["nshortparallel"] = 8742,
["nspar"] = 8742,
["and"] = 8743,
["wedge"] = 8743,
["or"] = 8744,
["vee"] = 8744,
["cap"] = 8745,
["caps"] = {8745, 65024},
["cup"] = 8746,
["cups"] = {8746, 65024},
["Integral"] = 8747,
["int"] = 8747,
["Int"] = 8748,
["iiint"] = 8749,
["tint"] = 8749,
["ContourIntegral"] = 8750,
["conint"] = 8750,
["oint"] = 8750,
["Conint"] = 8751,
["DoubleContourIntegral"] = 8751,
["Cconint"] = 8752,
["cwint"] = 8753,
["ClockwiseContourIntegral"] = 8754,
["cwconint"] = 8754,
["CounterClockwiseContourIntegral"] = 8755,
["awconint"] = 8755,
["Therefore"] = 8756,
["there4"] = 8756,
["therefore"] = 8756,
["Because"] = 8757,
["becaus"] = 8757,
["because"] = 8757,
["ratio"] = 8758,
["Colon"] = 8759,
["Proportion"] = 8759,
["dotminus"] = 8760,
["minusd"] = 8760,
["mDDot"] = 8762,
["homtht"] = 8763,
["Tilde"] = 8764,
["nvsim"] = {8764, 8402},
["sim"] = 8764,
["thicksim"] = 8764,
["thksim"] = 8764,
["backsim"] = 8765,
["bsim"] = 8765,
["race"] = {8765, 817},
```

```
4716  ["ac"] = 8766,
4717  ["acE"] = {8766, 819},
4718  ["mstpos"] = 8766,
4719  ["acd"] = 8767,
4720  ["VerticalTilde"] = 8768,
4721  ["wr"] = 8768,
4722  ["wreath"] = 8768,
4723  ["NotTilde"] = 8769,
4724  ["nsim"] = 8769,
4725  ["EqualTilde"] = 8770,
4726  ["NotEqualTilde"] = {8770, 824},
4727  ["eqsim"] = 8770,
4728  ["esim"] = 8770,
4729  ["nesim"] = {8770, 824},
4730  ["TildeEqual"] = 8771,
4731  ["sime"] = 8771,
4732  ["simeq"] = 8771,
4733  ["NotTildeEqual"] = 8772,
4734  ["nsime"] = 8772,
4735  ["nsimeq"] = 8772,
4736  ["TildeFullEqual"] = 8773,
4737  ["cong"] = 8773,
4738  ["simne"] = 8774,
4739  ["NotTildeFullEqual"] = 8775,
4740  ["ncong"] = 8775,
4741  ["TildeTilde"] = 8776,
4742  ["ap"] = 8776,
4743  ["approx"] = 8776,
4744  ["asymp"] = 8776,
4745  ["thickapprox"] = 8776,
4746  ["thkap"] = 8776,
4747  ["NotTildeTilde"] = 8777,
4748  ["nap"] = 8777,
4749  ["napprox"] = 8777,
4750  ["ape"] = 8778,
4751  ["approxeq"] = 8778,
4752  ["apid"] = 8779,
4753  ["napid"] = {8779, 824},
4754  ["backcong"] = 8780,
4755  ["bcong"] = 8780,
4756  ["CupCap"] = 8781,
4757  ["asympeq"] = 8781,
4758  ["nvap"] = {8781, 8402},
4759  ["Bumpeq"] = 8782,
4760  ["HumpDownHump"] = 8782,
4761  ["NotHumpDownHump"] = {8782, 824},
4762  ["bump"] = 8782,
```

```
["nbump"] = {8782, 824},
["HumpEqual"] = 8783,
["NotHumpEqual"] = {8783, 824},
["bumpe"] = 8783,
["bumpeq"] = 8783,
["nbumpe"] = {8783, 824},
["DotEqual"] = 8784,
["doteq"] = 8784,
["esdot"] = 8784,
["nedot"] = {8784, 824},
["doteqdot"] = 8785,
["eDot"] = 8785,
["efDot"] = 8786,
["fallingdotseq"] = 8786,
["erDot"] = 8787,
["risingdotseq"] = 8787,
["Assign"] = 8788,
["colone"] = 8788,
["coloneq"] = 8788,
["ecolon"] = 8789,
["eqcolon"] = 8789,
["ecir"] = 8790,
["eqcirc"] = 8790,
["circeq"] = 8791,
["cire"] = 8791,
["wedgeq"] = 8793,
["veeeq"] = 8794,
["triangleq"] = 8796,
["trie"] = 8796,
["equest"] = 8799,
["questeq"] = 8799,
["NotEqual"] = 8800,
["ne"] = 8800,
["Congruent"] = 8801,
["bnequiv"] = {8801, 8421},
["equiv"] = 8801,
["NotCongruent"] = 8802,
["nequiv"] = 8802,
["le"] = 8804,
["leq"] = 8804,
["nvle"] = {8804, 8402},
["GreaterEqual"] = 8805,
["ge"] = 8805,
["geq"] = 8805,
["nvge"] = {8805, 8402},
["LessFullEqual"] = 8806,
["lE"] = 8806,
```

```
["leqq"] = 8806,
["nlE"] = {8806, 824},
["nleqq"] = {8806, 824},
["GreaterFullEqual"] = 8807,
["NotGreaterFullEqual"] = {8807, 824},
["gE"] = 8807,
["geqq"] = 8807,
["ngE"] = {8807, 824},
["ngeqq"] = {8807, 824},
["lnE"] = 8808,
["lneqq"] = 8808,
["lvertneqq"] = {8808, 65024},
["lvnE"] = {8808, 65024},
["gnE"] = 8809,
["gneqq"] = 8809,
["gvertneqq"] = {8809, 65024},
["gvnE"] = {8809, 65024},
["Lt"] = 8810,
["NestedLessLess"] = 8810,
["NotLessLess"] = {8810, 824},
["ll"] = 8810,
["nLt"] = {8810, 8402},
["nLtv"] = {8810, 824},
["Gt"] = 8811,
["NestedGreaterGreater"] = 8811,
["NotGreaterGreater"] = {8811, 824},
["gg"] = 8811,
["nGt"] = {8811, 8402},
["nGtv"] = {8811, 824},
["between"] = 8812,
["twixt"] = 8812,
["NotCupCap"] = 8813,
["NotLess"] = 8814,
["nless"] = 8814,
["nlt"] = 8814,
["NotGreater"] = 8815,
["ngt"] = 8815,
["ngtr"] = 8815,
["NotLessEqual"] = 8816,
["nle"] = 8816,
["nleq"] = 8816,
["NotGreaterEqual"] = 8817,
["nge"] = 8817,
["ngeq"] = 8817,
["LessTilde"] = 8818,
["lesssim"] = 8818,
["lsim"] = 8818,
```

```
4857    ["GreaterTilde"] = 8819,
4858    ["gsim"] = 8819,
4859    ["gtrsim"] = 8819,
4860    ["NotLessTilde"] = 8820,
4861    ["nlsim"] = 8820,
4862    ["NotGreaterTilde"] = 8821,
4863    ["ngsim"] = 8821,
4864    ["LessGreater"] = 8822,
4865    ["lessgtr"] = 8822,
4866    ["lg"] = 8822,
4867    ["GreaterLess"] = 8823,
4868    ["gl"] = 8823,
4869    ["gtrless"] = 8823,
4870    ["NotLessGreater"] = 8824,
4871    ["ntlg"] = 8824,
4872    ["NotGreaterLess"] = 8825,
4873    ["ntgl"] = 8825,
4874    ["Precedes"] = 8826,
4875    ["pr"] = 8826,
4876    ["prec"] = 8826,
4877    ["Succeeds"] = 8827,
4878    ["sc"] = 8827,
4879    ["succ"] = 8827,
4880    ["PrecedesSlantEqual"] = 8828,
4881    ["prcue"] = 8828,
4882    ["preccurlyeq"] = 8828,
4883    ["SucceedsSlantEqual"] = 8829,
4884    ["sccue"] = 8829,
4885    ["succcurlyeq"] = 8829,
4886    ["PrecedesTilde"] = 8830,
4887    ["precsim"] = 8830,
4888    ["prsim"] = 8830,
4889    ["NotSucceedsTilde"] = {8831, 824},
4890    ["SucceedsTilde"] = 8831,
4891    ["scsim"] = 8831,
4892    ["succsim"] = 8831,
4893    ["NotPrecedes"] = 8832,
4894    ["npr"] = 8832,
4895    ["nprec"] = 8832,
4896    ["NotSucceeds"] = 8833,
4897    ["nsc"] = 8833,
4898    ["nsucc"] = 8833,
4899    ["NotSubset"] = {8834, 8402},
4900    ["nsubset"] = {8834, 8402},
4901    ["sub"] = 8834,
4902    ["subset"] = 8834,
4903    ["vnsub"] = {8834, 8402},
```

```
4904    ["NotSuperset"] = {8835, 8402},
4905    ["Superset"] = 8835,
4906    ["nsupset"] = {8835, 8402},
4907    ["sup"] = 8835,
4908    ["supset"] = 8835,
4909    ["vnsup"] = {8835, 8402},
4910    ["nsub"] = 8836,
4911    ["nsup"] = 8837,
4912    ["SubsetEqual"] = 8838,
4913    ["sube"] = 8838,
4914    ["subseteq"] = 8838,
4915    ["SupersetEqual"] = 8839,
4916    ["supe"] = 8839,
4917    ["supseteq"] = 8839,
4918    ["NotSubsetEqual"] = 8840,
4919    ["nsube"] = 8840,
4920    ["nsubseteq"] = 8840,
4921    ["NotSupersetEqual"] = 8841,
4922    ["nsupe"] = 8841,
4923    ["nsupseteq"] = 8841,
4924    ["subne"] = 8842,
4925    ["subsetneq"] = 8842,
4926    ["varsubsetneq"] = {8842, 65024},
4927    ["vsubne"] = {8842, 65024},
4928    ["supne"] = 8843,
4929    ["supsetneq"] = 8843,
4930    ["varsupsetneq"] = {8843, 65024},
4931    ["vsupne"] = {8843, 65024},
4932    ["cupdot"] = 8845,
4933    ["UnionPlus"] = 8846,
4934    ["uplus"] = 8846,
4935    ["NotSquareSubset"] = {8847, 824},
4936    ["SquareSubset"] = 8847,
4937    ["sqsub"] = 8847,
4938    ["sqsubset"] = 8847,
4939    ["NotSquareSuperset"] = {8848, 824},
4940    ["SquareSuperset"] = 8848,
4941    ["sqsup"] = 8848,
4942    ["sqsupset"] = 8848,
4943    ["SquareSubsetEqual"] = 8849,
4944    ["sqsube"] = 8849,
4945    ["sqsubseteq"] = 8849,
4946    ["SquareSupersetEqual"] = 8850,
4947    ["sqsupe"] = 8850,
4948    ["sqsupseteq"] = 8850,
4949    ["SquareIntersection"] = 8851,
4950    ["sqcap"] = 8851,
```

```
4951   ["sqcaps"] = {8851, 65024},
4952   ["SquareUnion"] = 8852,
4953   ["sqcup"] = 8852,
4954   ["sqcups"] = {8852, 65024},
4955   ["CirclePlus"] = 8853,
4956   ["oplus"] = 8853,
4957   ["CircleMinus"] = 8854,
4958   ["ominus"] = 8854,
4959   ["CircleTimes"] = 8855,
4960   ["otimes"] = 8855,
4961   ["osol"] = 8856,
4962   ["CircleDot"] = 8857,
4963   ["odot"] = 8857,
4964   ["circledcirc"] = 8858,
4965   ["ocir"] = 8858,
4966   ["circledast"] = 8859,
4967   ["oast"] = 8859,
4968   ["circleddash"] = 8861,
4969   ["odash"] = 8861,
4970   ["boxplus"] = 8862,
4971   ["plusb"] = 8862,
4972   ["boxminus"] = 8863,
4973   ["minusb"] = 8863,
4974   ["boxtimes"] = 8864,
4975   ["timesb"] = 8864,
4976   ["dotsquare"] = 8865,
4977   ["sdotb"] = 8865,
4978   ["RightTee"] = 8866,
4979   ["vdash"] = 8866,
4980   ["LeftTee"] = 8867,
4981   ["dashv"] = 8867,
4982   ["DownTee"] = 8868,
4983   ["top"] = 8868,
4984   ["UpTee"] = 8869,
4985   ["bot"] = 8869,
4986   ["bottom"] = 8869,
4987   ["perp"] = 8869,
4988   ["models"] = 8871,
4989   ["DoubleRightTee"] = 8872,
4990   ["vDash"] = 8872,
4991   ["Vdash"] = 8873,
4992   ["Vvdash"] = 8874,
4993   ["VDash"] = 8875,
4994   ["nvdash"] = 8876,
4995   ["nvDash"] = 8877,
4996   ["nVdash"] = 8878,
4997   ["nVDash"] = 8879,
```

```
4998  ["prurel"] = 8880,
4999  ["LeftTriangle"] = 8882,
5000  ["vartriangleleft"] = 8882,
5001  ["vltri"] = 8882,
5002  ["RightTriangle"] = 8883,
5003  ["vartriangleright"] = 8883,
5004  ["vrtri"] = 8883,
5005  ["LeftTriangleEqual"] = 8884,
5006  ["ltrie"] = 8884,
5007  ["nvltrie"] = {8884, 8402},
5008  ["trianglelefteq"] = 8884,
5009  ["RightTriangleEqual"] = 8885,
5010  ["nvrtrie"] = {8885, 8402},
5011  ["rtrie"] = 8885,
5012  ["trianglerighteq"] = 8885,
5013  ["origof"] = 8886,
5014  ["imof"] = 8887,
5015  ["multimap"] = 8888,
5016  ["mumap"] = 8888,
5017  ["hercon"] = 8889,
5018  ["intcal"] = 8890,
5019  ["intercal"] = 8890,
5020  ["veebar"] = 8891,
5021  ["barvee"] = 8893,
5022  ["angrtvb"] = 8894,
5023  ["lrtri"] = 8895,
5024  ["Wedge"] = 8896,
5025  ["bigwedge"] = 8896,
5026  ["xwedge"] = 8896,
5027  ["Vee"] = 8897,
5028  ["bigvee"] = 8897,
5029  ["xvee"] = 8897,
5030  ["Intersection"] = 8898,
5031  ["bigcap"] = 8898,
5032  ["xcap"] = 8898,
5033  ["Union"] = 8899,
5034  ["bigcup"] = 8899,
5035  ["xcup"] = 8899,
5036  ["Diamond"] = 8900,
5037  ["diam"] = 8900,
5038  ["diamond"] = 8900,
5039  ["sdot"] = 8901,
5040  ["Star"] = 8902,
5041  ["sstarf"] = 8902,
5042  ["divideontimes"] = 8903,
5043  ["divonx"] = 8903,
5044  ["bowtie"] = 8904,
```

```
["ltimes"] = 8905,
["rtimes"] = 8906,
["leftthreetimes"] = 8907,
["lthree"] = 8907,
["rightthreetimes"] = 8908,
["rthree"] = 8908,
["backsimeq"] = 8909,
["bsime"] = 8909,
["curlyvee"] = 8910,
["cuvee"] = 8910,
["curlywedge"] = 8911,
["cuwed"] = 8911,
["Sub"] = 8912,
["Subset"] = 8912,
["Sup"] = 8913,
["Supset"] = 8913,
["Cap"] = 8914,
["Cup"] = 8915,
["fork"] = 8916,
["pitchfork"] = 8916,
["epar"] = 8917,
["lessdot"] = 8918,
["ltdot"] = 8918,
["gtdot"] = 8919,
["gtrdot"] = 8919,
["Ll"] = 8920,
["nLl"] = {8920, 824},
["Gg"] = 8921,
["ggg"] = 8921,
["nGg"] = {8921, 824},
["LessEqualGreater"] = 8922,
["leg"] = 8922,
["lesg"] = {8922, 65024},
["lesseqgtr"] = 8922,
["GreaterEqualLess"] = 8923,
["gel"] = 8923,
["gesl"] = {8923, 65024},
["gtreqless"] = 8923,
["cuepr"] = 8926,
["curlyeqprec"] = 8926,
["cuesc"] = 8927,
["curlyeqsucc"] = 8927,
["NotPrecedesSlantEqual"] = 8928,
["nprcue"] = 8928,
["NotSucceedsSlantEqual"] = 8929,
["nsccue"] = 8929,
["NotSquareSubsetEqual"] = 8930,
```

```
  ["nsqsube"] = 8930,
  ["NotSquareSupersetEqual"] = 8931,
  ["nsqsupe"] = 8931,
  ["lnsim"] = 8934,
  ["gnsim"] = 8935,
  ["precnsim"] = 8936,
  ["prnsim"] = 8936,
  ["scnsim"] = 8937,
  ["succnsim"] = 8937,
  ["NotLeftTriangle"] = 8938,
  ["nltri"] = 8938,
  ["ntriangleleft"] = 8938,
  ["NotRightTriangle"] = 8939,
  ["nrtri"] = 8939,
  ["ntriangleright"] = 8939,
  ["NotLeftTriangleEqual"] = 8940,
  ["nltrie"] = 8940,
  ["ntrianglelefteq"] = 8940,
  ["NotRightTriangleEqual"] = 8941,
  ["nrtrie"] = 8941,
  ["ntrianglerighteq"] = 8941,
  ["vellip"] = 8942,
  ["ctdot"] = 8943,
  ["utdot"] = 8944,
  ["dtdot"] = 8945,
  ["disin"] = 8946,
  ["isinsv"] = 8947,
  ["isins"] = 8948,
  ["isindot"] = 8949,
  ["notindot"] = {8949, 824},
  ["notinvc"] = 8950,
  ["notinvb"] = 8951,
  ["isinE"] = 8953,
  ["notinE"] = {8953, 824},
  ["nisd"] = 8954,
  ["xnis"] = 8955,
  ["nis"] = 8956,
  ["notnivc"] = 8957,
  ["notnivb"] = 8958,
  ["barwed"] = 8965,
  ["barwedge"] = 8965,
  ["Barwed"] = 8966,
  ["doublebarwedge"] = 8966,
  ["LeftCeiling"] = 8968,
  ["lceil"] = 8968,
  ["RightCeiling"] = 8969,
  ["rceil"] = 8969,
```

```
5139   ["LeftFloor"] = 8970,
5140   ["lfloor"] = 8970,
5141   ["RightFloor"] = 8971,
5142   ["rfloor"] = 8971,
5143   ["drcrop"] = 8972,
5144   ["dlcrop"] = 8973,
5145   ["urcrop"] = 8974,
5146   ["ulcrop"] = 8975,
5147   ["bnot"] = 8976,
5148   ["profline"] = 8978,
5149   ["profsurf"] = 8979,
5150   ["telrec"] = 8981,
5151   ["target"] = 8982,
5152   ["ulcorn"] = 8988,
5153   ["ulcorner"] = 8988,
5154   ["urcorn"] = 8989,
5155   ["urcorner"] = 8989,
5156   ["dlcorn"] = 8990,
5157   ["llcorner"] = 8990,
5158   ["drcorn"] = 8991,
5159   ["lrcorner"] = 8991,
5160   ["frown"] = 8994,
5161   ["sfrown"] = 8994,
5162   ["smile"] = 8995,
5163   ["ssmile"] = 8995,
5164   ["cylcty"] = 9005,
5165   ["profalar"] = 9006,
5166   ["topbot"] = 9014,
5167   ["ovbar"] = 9021,
5168   ["solbar"] = 9023,
5169   ["angzarr"] = 9084,
5170   ["lmoust"] = 9136,
5171   ["lmoustache"] = 9136,
5172   ["rmoust"] = 9137,
5173   ["rmoustache"] = 9137,
5174   ["OverBracket"] = 9140,
5175   ["tbrk"] = 9140,
5176   ["UnderBracket"] = 9141,
5177   ["bbrk"] = 9141,
5178   ["bbrktbrk"] = 9142,
5179   ["OverParenthesis"] = 9180,
5180   ["UnderParenthesis"] = 9181,
5181   ["OverBrace"] = 9182,
5182   ["UnderBrace"] = 9183,
5183   ["trpezium"] = 9186,
5184   ["elinters"] = 9191,
5185   ["blank"] = 9251,
```

```
5186   ["circledS"] = 9416,
5187   ["oS"] = 9416,
5188   ["HorizontalLine"] = 9472,
5189   ["boxh"] = 9472,
5190   ["boxv"] = 9474,
5191   ["boxdr"] = 9484,
5192   ["boxdl"] = 9488,
5193   ["boxur"] = 9492,
5194   ["boxul"] = 9496,
5195   ["boxvr"] = 9500,
5196   ["boxvl"] = 9508,
5197   ["boxhd"] = 9516,
5198   ["boxhu"] = 9524,
5199   ["boxvh"] = 9532,
5200   ["boxH"] = 9552,
5201   ["boxV"] = 9553,
5202   ["boxdR"] = 9554,
5203   ["boxDr"] = 9555,
5204   ["boxDR"] = 9556,
5205   ["boxdL"] = 9557,
5206   ["boxDl"] = 9558,
5207   ["boxDL"] = 9559,
5208   ["boxuR"] = 9560,
5209   ["boxUr"] = 9561,
5210   ["boxUR"] = 9562,
5211   ["boxuL"] = 9563,
5212   ["boxUl"] = 9564,
5213   ["boxUL"] = 9565,
5214   ["boxvR"] = 9566,
5215   ["boxVr"] = 9567,
5216   ["boxVR"] = 9568,
5217   ["boxvL"] = 9569,
5218   ["boxVl"] = 9570,
5219   ["boxVL"] = 9571,
5220   ["boxHd"] = 9572,
5221   ["boxhD"] = 9573,
5222   ["boxHD"] = 9574,
5223   ["boxHu"] = 9575,
5224   ["boxhU"] = 9576,
5225   ["boxHU"] = 9577,
5226   ["boxvH"] = 9578,
5227   ["boxVh"] = 9579,
5228   ["boxVH"] = 9580,
5229   ["uhblk"] = 9600,
5230   ["lhblk"] = 9604,
5231   ["block"] = 9608,
5232   ["blk14"] = 9617,
```

```
  ["blk12"] = 9618,
  ["blk34"] = 9619,
  ["Square"] = 9633,
  ["squ"] = 9633,
  ["square"] = 9633,
  ["FilledVerySmallSquare"] = 9642,
  ["blacksquare"] = 9642,
  ["squarf"] = 9642,
  ["squf"] = 9642,
  ["EmptyVerySmallSquare"] = 9643,
  ["rect"] = 9645,
  ["marker"] = 9646,
  ["fltns"] = 9649,
  ["bigtriangleup"] = 9651,
  ["xutri"] = 9651,
  ["blacktriangle"] = 9652,
  ["utrif"] = 9652,
  ["triangle"] = 9653,
  ["utri"] = 9653,
  ["blacktriangleright"] = 9656,
  ["rtrif"] = 9656,
  ["rtri"] = 9657,
  ["triangleright"] = 9657,
  ["bigtriangledown"] = 9661,
  ["xdtri"] = 9661,
  ["blacktriangledown"] = 9662,
  ["dtrif"] = 9662,
  ["dtri"] = 9663,
  ["triangledown"] = 9663,
  ["blacktriangleleft"] = 9666,
  ["ltrif"] = 9666,
  ["ltri"] = 9667,
  ["triangleleft"] = 9667,
  ["loz"] = 9674,
  ["lozenge"] = 9674,
  ["cir"] = 9675,
  ["tridot"] = 9708,
  ["bigcirc"] = 9711,
  ["xcirc"] = 9711,
  ["ultri"] = 9720,
  ["urtri"] = 9721,
  ["lltri"] = 9722,
  ["EmptySmallSquare"] = 9723,
  ["FilledSmallSquare"] = 9724,
  ["bigstar"] = 9733,
  ["starf"] = 9733,
  ["star"] = 9734,
```

```
  ["phone"] = 9742,
  ["female"] = 9792,
  ["male"] = 9794,
  ["spades"] = 9824,
  ["spadesuit"] = 9824,
  ["clubs"] = 9827,
  ["clubsuit"] = 9827,
  ["hearts"] = 9829,
  ["heartsuit"] = 9829,
  ["diamondsuit"] = 9830,
  ["diams"] = 9830,
  ["sung"] = 9834,
  ["flat"] = 9837,
  ["natur"] = 9838,
  ["natural"] = 9838,
  ["sharp"] = 9839,
  ["check"] = 10003,
  ["checkmark"] = 10003,
  ["cross"] = 10007,
  ["malt"] = 10016,
  ["maltese"] = 10016,
  ["sext"] = 10038,
  ["VerticalSeparator"] = 10072,
  ["lbbrk"] = 10098,
  ["rbbrk"] = 10099,
  ["bsolhsub"] = 10184,
  ["suphsol"] = 10185,
  ["LeftDoubleBracket"] = 10214,
  ["lobrk"] = 10214,
  ["RightDoubleBracket"] = 10215,
  ["robrk"] = 10215,
  ["LeftAngleBracket"] = 10216,
  ["lang"] = 10216,
  ["langle"] = 10216,
  ["RightAngleBracket"] = 10217,
  ["rang"] = 10217,
  ["rangle"] = 10217,
  ["Lang"] = 10218,
  ["Rang"] = 10219,
  ["loang"] = 10220,
  ["roang"] = 10221,
  ["LongLeftArrow"] = 10229,
  ["longleftarrow"] = 10229,
  ["xlarr"] = 10229,
  ["LongRightArrow"] = 10230,
  ["longrightarrow"] = 10230,
  ["xrarr"] = 10230,
```

```
5327   ["LongLeftRightArrow"] = 10231,
5328   ["longleftrightarrow"] = 10231,
5329   ["xharr"] = 10231,
5330   ["DoubleLongLeftArrow"] = 10232,
5331   ["Longleftarrow"] = 10232,
5332   ["xlArr"] = 10232,
5333   ["DoubleLongRightArrow"] = 10233,
5334   ["Longrightarrow"] = 10233,
5335   ["xrArr"] = 10233,
5336   ["DoubleLongLeftRightArrow"] = 10234,
5337   ["Longleftrightarrow"] = 10234,
5338   ["xhArr"] = 10234,
5339   ["longmapsto"] = 10236,
5340   ["xmap"] = 10236,
5341   ["dzigrarr"] = 10239,
5342   ["nvlArr"] = 10498,
5343   ["nvrArr"] = 10499,
5344   ["nvHarr"] = 10500,
5345   ["Map"] = 10501,
5346   ["lbarr"] = 10508,
5347   ["bkarow"] = 10509,
5348   ["rbarr"] = 10509,
5349   ["lBarr"] = 10510,
5350   ["dbkarow"] = 10511,
5351   ["rBarr"] = 10511,
5352   ["RBarr"] = 10512,
5353   ["drbkarow"] = 10512,
5354   ["DDotrahd"] = 10513,
5355   ["UpArrowBar"] = 10514,
5356   ["DownArrowBar"] = 10515,
5357   ["Rarrtl"] = 10518,
5358   ["latail"] = 10521,
5359   ["ratail"] = 10522,
5360   ["lAtail"] = 10523,
5361   ["rAtail"] = 10524,
5362   ["larrfs"] = 10525,
5363   ["rarrfs"] = 10526,
5364   ["larrbfs"] = 10527,
5365   ["rarrbfs"] = 10528,
5366   ["nwarhk"] = 10531,
5367   ["nearhk"] = 10532,
5368   ["hksearow"] = 10533,
5369   ["searhk"] = 10533,
5370   ["hkswarow"] = 10534,
5371   ["swarhk"] = 10534,
5372   ["nwnear"] = 10535,
5373   ["nesear"] = 10536,
```

```
5374  ["toea"] = 10536,
5375  ["seswar"] = 10537,
5376  ["tosa"] = 10537,
5377  ["swnwar"] = 10538,
5378  ["nrarrc"] = {10547, 824},
5379  ["rarrc"] = 10547,
5380  ["cudarrr"] = 10549,
5381  ["ldca"] = 10550,
5382  ["rdca"] = 10551,
5383  ["cudarrl"] = 10552,
5384  ["larrpl"] = 10553,
5385  ["curarrm"] = 10556,
5386  ["cularrp"] = 10557,
5387  ["rarrpl"] = 10565,
5388  ["harrcir"] = 10568,
5389  ["Uarrocir"] = 10569,
5390  ["lurdshar"] = 10570,
5391  ["ldrushar"] = 10571,
5392  ["LeftRightVector"] = 10574,
5393  ["RightUpDownVector"] = 10575,
5394  ["DownLeftRightVector"] = 10576,
5395  ["LeftUpDownVector"] = 10577,
5396  ["LeftVectorBar"] = 10578,
5397  ["RightVectorBar"] = 10579,
5398  ["RightUpVectorBar"] = 10580,
5399  ["RightDownVectorBar"] = 10581,
5400  ["DownLeftVectorBar"] = 10582,
5401  ["DownRightVectorBar"] = 10583,
5402  ["LeftUpVectorBar"] = 10584,
5403  ["LeftDownVectorBar"] = 10585,
5404  ["LeftTeeVector"] = 10586,
5405  ["RightTeeVector"] = 10587,
5406  ["RightUpTeeVector"] = 10588,
5407  ["RightDownTeeVector"] = 10589,
5408  ["DownLeftTeeVector"] = 10590,
5409  ["DownRightTeeVector"] = 10591,
5410  ["LeftUpTeeVector"] = 10592,
5411  ["LeftDownTeeVector"] = 10593,
5412  ["lHar"] = 10594,
5413  ["uHar"] = 10595,
5414  ["rHar"] = 10596,
5415  ["dHar"] = 10597,
5416  ["luruhar"] = 10598,
5417  ["ldrdhar"] = 10599,
5418  ["ruluhar"] = 10600,
5419  ["rdldhar"] = 10601,
5420  ["lharul"] = 10602,
```

```
5421   ["llhard"] = 10603,
5422   ["rharul"] = 10604,
5423   ["lrhard"] = 10605,
5424   ["UpEquilibrium"] = 10606,
5425   ["udhar"] = 10606,
5426   ["ReverseUpEquilibrium"] = 10607,
5427   ["duhar"] = 10607,
5428   ["RoundImplies"] = 10608,
5429   ["erarr"] = 10609,
5430   ["simrarr"] = 10610,
5431   ["larrsim"] = 10611,
5432   ["rarrsim"] = 10612,
5433   ["rarrap"] = 10613,
5434   ["ltlarr"] = 10614,
5435   ["gtrarr"] = 10616,
5436   ["subrarr"] = 10617,
5437   ["suplarr"] = 10619,
5438   ["lfisht"] = 10620,
5439   ["rfisht"] = 10621,
5440   ["ufisht"] = 10622,
5441   ["dfisht"] = 10623,
5442   ["lopar"] = 10629,
5443   ["ropar"] = 10630,
5444   ["lbrke"] = 10635,
5445   ["rbrke"] = 10636,
5446   ["lbrkslu"] = 10637,
5447   ["rbrksld"] = 10638,
5448   ["lbrksld"] = 10639,
5449   ["rbrkslu"] = 10640,
5450   ["langd"] = 10641,
5451   ["rangd"] = 10642,
5452   ["lparlt"] = 10643,
5453   ["rpargt"] = 10644,
5454   ["gtlPar"] = 10645,
5455   ["ltrPar"] = 10646,
5456   ["vzigzag"] = 10650,
5457   ["vangrt"] = 10652,
5458   ["angrtvbd"] = 10653,
5459   ["ange"] = 10660,
5460   ["range"] = 10661,
5461   ["dwangle"] = 10662,
5462   ["uwangle"] = 10663,
5463   ["angmsdaa"] = 10664,
5464   ["angmsdab"] = 10665,
5465   ["angmsdac"] = 10666,
5466   ["angmsdad"] = 10667,
5467   ["angmsdae"] = 10668,
```

```
5468   ["angmsdaf"] = 10669,
5469   ["angmsdag"] = 10670,
5470   ["angmsdah"] = 10671,
5471   ["bemptyv"] = 10672,
5472   ["demptyv"] = 10673,
5473   ["cemptyv"] = 10674,
5474   ["raemptyv"] = 10675,
5475   ["laemptyv"] = 10676,
5476   ["ohbar"] = 10677,
5477   ["omid"] = 10678,
5478   ["opar"] = 10679,
5479   ["operp"] = 10681,
5480   ["olcross"] = 10683,
5481   ["odsold"] = 10684,
5482   ["olcir"] = 10686,
5483   ["ofcir"] = 10687,
5484   ["olt"] = 10688,
5485   ["ogt"] = 10689,
5486   ["cirscir"] = 10690,
5487   ["cirE"] = 10691,
5488   ["solb"] = 10692,
5489   ["bsolb"] = 10693,
5490   ["boxbox"] = 10697,
5491   ["trisb"] = 10701,
5492   ["rtriltri"] = 10702,
5493   ["LeftTriangleBar"] = 10703,
5494   ["NotLeftTriangleBar"] = {10703, 824},
5495   ["NotRightTriangleBar"] = {10704, 824},
5496   ["RightTriangleBar"] = 10704,
5497   ["iinfin"] = 10716,
5498   ["infintie"] = 10717,
5499   ["nvinfin"] = 10718,
5500   ["eparsl"] = 10723,
5501   ["smeparsl"] = 10724,
5502   ["eqvparsl"] = 10725,
5503   ["blacklozenge"] = 10731,
5504   ["lozf"] = 10731,
5505   ["RuleDelayed"] = 10740,
5506   ["dsol"] = 10742,
5507   ["bigodot"] = 10752,
5508   ["xodot"] = 10752,
5509   ["bigoplus"] = 10753,
5510   ["xoplus"] = 10753,
5511   ["bigotimes"] = 10754,
5512   ["xotime"] = 10754,
5513   ["biguplus"] = 10756,
5514   ["xuplus"] = 10756,
```

```
5515    ["bigsqcup"] = 10758,
5516    ["xsqcup"] = 10758,
5517    ["iiiint"] = 10764,
5518    ["qint"] = 10764,
5519    ["fpartint"] = 10765,
5520    ["cirfnint"] = 10768,
5521    ["awint"] = 10769,
5522    ["rppolint"] = 10770,
5523    ["scpolint"] = 10771,
5524    ["npolint"] = 10772,
5525    ["pointint"] = 10773,
5526    ["quatint"] = 10774,
5527    ["intlarhk"] = 10775,
5528    ["pluscir"] = 10786,
5529    ["plusacir"] = 10787,
5530    ["simplus"] = 10788,
5531    ["plusdu"] = 10789,
5532    ["plussim"] = 10790,
5533    ["plustwo"] = 10791,
5534    ["mcomma"] = 10793,
5535    ["minusdu"] = 10794,
5536    ["loplus"] = 10797,
5537    ["roplus"] = 10798,
5538    ["Cross"] = 10799,
5539    ["timesd"] = 10800,
5540    ["timesbar"] = 10801,
5541    ["smashp"] = 10803,
5542    ["lotimes"] = 10804,
5543    ["rotimes"] = 10805,
5544    ["otimesas"] = 10806,
5545    ["Otimes"] = 10807,
5546    ["odiv"] = 10808,
5547    ["triplus"] = 10809,
5548    ["triminus"] = 10810,
5549    ["tritime"] = 10811,
5550    ["intprod"] = 10812,
5551    ["iprod"] = 10812,
5552    ["amalg"] = 10815,
5553    ["capdot"] = 10816,
5554    ["ncup"] = 10818,
5555    ["ncap"] = 10819,
5556    ["capand"] = 10820,
5557    ["cupor"] = 10821,
5558    ["cupcap"] = 10822,
5559    ["capcup"] = 10823,
5560    ["cupbrcap"] = 10824,
5561    ["capbrcup"] = 10825,
```

```
  ["cupcup"] = 10826,
  ["capcap"] = 10827,
  ["ccups"] = 10828,
  ["ccaps"] = 10829,
  ["ccupssm"] = 10832,
  ["And"] = 10835,
  ["Or"] = 10836,
  ["andand"] = 10837,
  ["oror"] = 10838,
  ["orslope"] = 10839,
  ["andslope"] = 10840,
  ["andv"] = 10842,
  ["orv"] = 10843,
  ["andd"] = 10844,
  ["ord"] = 10845,
  ["wedbar"] = 10847,
  ["sdote"] = 10854,
  ["simdot"] = 10858,
  ["congdot"] = 10861,
  ["ncongdot"] = {10861, 824},
  ["easter"] = 10862,
  ["apacir"] = 10863,
  ["apE"] = 10864,
  ["napE"] = {10864, 824},
  ["eplus"] = 10865,
  ["pluse"] = 10866,
  ["Esim"] = 10867,
  ["Colone"] = 10868,
  ["Equal"] = 10869,
  ["ddotseq"] = 10871,
  ["eDDot"] = 10871,
  ["equivDD"] = 10872,
  ["ltcir"] = 10873,
  ["gtcir"] = 10874,
  ["ltquest"] = 10875,
  ["gtquest"] = 10876,
  ["LessSlantEqual"] = 10877,
  ["NotLessSlantEqual"] = {10877, 824},
  ["leqslant"] = 10877,
  ["les"] = 10877,
  ["nleqslant"] = {10877, 824},
  ["nles"] = {10877, 824},
  ["GreaterSlantEqual"] = 10878,
  ["NotGreaterSlantEqual"] = {10878, 824},
  ["geqslant"] = 10878,
  ["ges"] = 10878,
  ["ngeqslant"] = {10878, 824},
```

```
5609    ["nges"] = {10878, 824},
5610    ["lesdot"] = 10879,
5611    ["gesdot"] = 10880,
5612    ["lesdoto"] = 10881,
5613    ["gesdoto"] = 10882,
5614    ["lesdotor"] = 10883,
5615    ["gesdotol"] = 10884,
5616    ["lap"] = 10885,
5617    ["lessapprox"] = 10885,
5618    ["gap"] = 10886,
5619    ["gtrapprox"] = 10886,
5620    ["lne"] = 10887,
5621    ["lneq"] = 10887,
5622    ["gne"] = 10888,
5623    ["gneq"] = 10888,
5624    ["lnap"] = 10889,
5625    ["lnapprox"] = 10889,
5626    ["gnap"] = 10890,
5627    ["gnapprox"] = 10890,
5628    ["lEg"] = 10891,
5629    ["lesseqqgtr"] = 10891,
5630    ["gEl"] = 10892,
5631    ["gtreqqless"] = 10892,
5632    ["lsime"] = 10893,
5633    ["gsime"] = 10894,
5634    ["lsimg"] = 10895,
5635    ["gsiml"] = 10896,
5636    ["lgE"] = 10897,
5637    ["glE"] = 10898,
5638    ["lesges"] = 10899,
5639    ["gesles"] = 10900,
5640    ["els"] = 10901,
5641    ["eqslantless"] = 10901,
5642    ["egs"] = 10902,
5643    ["eqslantgtr"] = 10902,
5644    ["elsdot"] = 10903,
5645    ["egsdot"] = 10904,
5646    ["el"] = 10905,
5647    ["eg"] = 10906,
5648    ["siml"] = 10909,
5649    ["simg"] = 10910,
5650    ["simlE"] = 10911,
5651    ["simgE"] = 10912,
5652    ["LessLess"] = 10913,
5653    ["NotNestedLessLess"] = {10913, 824},
5654    ["GreaterGreater"] = 10914,
5655    ["NotNestedGreaterGreater"] = {10914, 824},
```

```
  ["glj"] = 10916,
  ["gla"] = 10917,
  ["ltcc"] = 10918,
  ["gtcc"] = 10919,
  ["lescc"] = 10920,
  ["gescc"] = 10921,
  ["smt"] = 10922,
  ["lat"] = 10923,
  ["smte"] = 10924,
  ["smtes"] = {10924, 65024},
  ["late"] = 10925,
  ["lates"] = {10925, 65024},
  ["bumpE"] = 10926,
  ["NotPrecedesEqual"] = {10927, 824},
  ["PrecedesEqual"] = 10927,
  ["npre"] = {10927, 824},
  ["npreceq"] = {10927, 824},
  ["pre"] = 10927,
  ["preceq"] = 10927,
  ["NotSucceedsEqual"] = {10928, 824},
  ["SucceedsEqual"] = 10928,
  ["nsce"] = {10928, 824},
  ["nsucceq"] = {10928, 824},
  ["sce"] = 10928,
  ["succeq"] = 10928,
  ["prE"] = 10931,
  ["scE"] = 10932,
  ["precneqq"] = 10933,
  ["prnE"] = 10933,
  ["scnE"] = 10934,
  ["succneqq"] = 10934,
  ["prap"] = 10935,
  ["precapprox"] = 10935,
  ["scap"] = 10936,
  ["succapprox"] = 10936,
  ["precnapprox"] = 10937,
  ["prnap"] = 10937,
  ["scnap"] = 10938,
  ["succnapprox"] = 10938,
  ["Pr"] = 10939,
  ["Sc"] = 10940,
  ["subdot"] = 10941,
  ["supdot"] = 10942,
  ["subplus"] = 10943,
  ["supplus"] = 10944,
  ["submult"] = 10945,
  ["supmult"] = 10946,
```

```
5703   ["subedot"] = 10947,
5704   ["supedot"] = 10948,
5705   ["nsubE"] = {10949, 824},
5706   ["nsubseteqq"] = {10949, 824},
5707   ["subE"] = 10949,
5708   ["subseteqq"] = 10949,
5709   ["nsupE"] = {10950, 824},
5710   ["nsupseteqq"] = {10950, 824},
5711   ["supE"] = 10950,
5712   ["supseteqq"] = 10950,
5713   ["subsim"] = 10951,
5714   ["supsim"] = 10952,
5715   ["subnE"] = 10955,
5716   ["subsetneqq"] = 10955,
5717   ["varsubsetneqq"] = {10955, 65024},
5718   ["vsubnE"] = {10955, 65024},
5719   ["supnE"] = 10956,
5720   ["supsetneqq"] = 10956,
5721   ["varsupsetneqq"] = {10956, 65024},
5722   ["vsupnE"] = {10956, 65024},
5723   ["csub"] = 10959,
5724   ["csup"] = 10960,
5725   ["csube"] = 10961,
5726   ["csupe"] = 10962,
5727   ["subsup"] = 10963,
5728   ["supsub"] = 10964,
5729   ["subsub"] = 10965,
5730   ["supsup"] = 10966,
5731   ["suphsub"] = 10967,
5732   ["supdsub"] = 10968,
5733   ["forkv"] = 10969,
5734   ["topfork"] = 10970,
5735   ["mlcp"] = 10971,
5736   ["Dashv"] = 10980,
5737   ["DoubleLeftTee"] = 10980,
5738   ["Vdashl"] = 10982,
5739   ["Barv"] = 10983,
5740   ["vBar"] = 10984,
5741   ["vBarv"] = 10985,
5742   ["Vbar"] = 10987,
5743   ["Not"] = 10988,
5744   ["bNot"] = 10989,
5745   ["rnmid"] = 10990,
5746   ["cirmid"] = 10991,
5747   ["midcir"] = 10992,
5748   ["topcir"] = 10993,
5749   ["nhpar"] = 10994,
```

```
  ["parsim"] = 10995,
  ["nparsl"] = {11005, 8421},
  ["parsl"] = 11005,
  ["fflig"] = 64256,
  ["filig"] = 64257,
  ["fllig"] = 64258,
  ["ffilig"] = 64259,
  ["ffllig"] = 64260,
  ["Ascr"] = 119964,
  ["Cscr"] = 119966,
  ["Dscr"] = 119967,
  ["Gscr"] = 119970,
  ["Jscr"] = 119973,
  ["Kscr"] = 119974,
  ["Nscr"] = 119977,
  ["Oscr"] = 119978,
  ["Pscr"] = 119979,
  ["Qscr"] = 119980,
  ["Sscr"] = 119982,
  ["Tscr"] = 119983,
  ["Uscr"] = 119984,
  ["Vscr"] = 119985,
  ["Wscr"] = 119986,
  ["Xscr"] = 119987,
  ["Yscr"] = 119988,
  ["Zscr"] = 119989,
  ["ascr"] = 119990,
  ["bscr"] = 119991,
  ["cscr"] = 119992,
  ["dscr"] = 119993,
  ["fscr"] = 119995,
  ["hscr"] = 119997,
  ["iscr"] = 119998,
  ["jscr"] = 119999,
  ["kscr"] = 120000,
  ["lscr"] = 120001,
  ["mscr"] = 120002,
  ["nscr"] = 120003,
  ["pscr"] = 120005,
  ["qscr"] = 120006,
  ["rscr"] = 120007,
  ["sscr"] = 120008,
  ["tscr"] = 120009,
  ["uscr"] = 120010,
  ["vscr"] = 120011,
  ["wscr"] = 120012,
  ["xscr"] = 120013,
```

```
  ["yscr"] = 120014,
  ["zscr"] = 120015,
  ["Afr"] = 120068,
  ["Bfr"] = 120069,
  ["Dfr"] = 120071,
  ["Efr"] = 120072,
  ["Ffr"] = 120073,
  ["Gfr"] = 120074,
  ["Jfr"] = 120077,
  ["Kfr"] = 120078,
  ["Lfr"] = 120079,
  ["Mfr"] = 120080,
  ["Nfr"] = 120081,
  ["Ofr"] = 120082,
  ["Pfr"] = 120083,
  ["Qfr"] = 120084,
  ["Sfr"] = 120086,
  ["Tfr"] = 120087,
  ["Ufr"] = 120088,
  ["Vfr"] = 120089,
  ["Wfr"] = 120090,
  ["Xfr"] = 120091,
  ["Yfr"] = 120092,
  ["afr"] = 120094,
  ["bfr"] = 120095,
  ["cfr"] = 120096,
  ["dfr"] = 120097,
  ["efr"] = 120098,
  ["ffr"] = 120099,
  ["gfr"] = 120100,
  ["hfr"] = 120101,
  ["ifr"] = 120102,
  ["jfr"] = 120103,
  ["kfr"] = 120104,
  ["lfr"] = 120105,
  ["mfr"] = 120106,
  ["nfr"] = 120107,
  ["ofr"] = 120108,
  ["pfr"] = 120109,
  ["qfr"] = 120110,
  ["rfr"] = 120111,
  ["sfr"] = 120112,
  ["tfr"] = 120113,
  ["ufr"] = 120114,
  ["vfr"] = 120115,
  ["wfr"] = 120116,
  ["xfr"] = 120117,
```

```
  ["yfr"] = 120118,
  ["zfr"] = 120119,
  ["Aopf"] = 120120,
  ["Bopf"] = 120121,
  ["Dopf"] = 120123,
  ["Eopf"] = 120124,
  ["Fopf"] = 120125,
  ["Gopf"] = 120126,
  ["Iopf"] = 120128,
  ["Jopf"] = 120129,
  ["Kopf"] = 120130,
  ["Lopf"] = 120131,
  ["Mopf"] = 120132,
  ["Oopf"] = 120134,
  ["Sopf"] = 120138,
  ["Topf"] = 120139,
  ["Uopf"] = 120140,
  ["Vopf"] = 120141,
  ["Wopf"] = 120142,
  ["Xopf"] = 120143,
  ["Yopf"] = 120144,
  ["aopf"] = 120146,
  ["bopf"] = 120147,
  ["copf"] = 120148,
  ["dopf"] = 120149,
  ["eopf"] = 120150,
  ["fopf"] = 120151,
  ["gopf"] = 120152,
  ["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)
  local n = tonumber(s)
  if n == nil then
    return "&#" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.hex_entity(s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#x" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```
function entities.hex_entity_with_x_char(x, s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#" .. x .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
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 code_points = character_entities[s]
  if code_points == nil then
    return "&" .. s .. ";"
  end
  if type(code_points) ~= 'table' then
    code_points = {code_points}
  end
  local char_table = {}
    for _, code_point in ipairs(code_points) do
      table.insert(char_table, unicode.utf8.char(code_point))
    end
  return table.concat(char_table)
```

```
5926 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.

```
5927 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.

```
5928 function M.writer.new(options)
5929   local self = {}
```

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

```
5930   self.options = options
```

Define `writer->flatten_inlines`, which indicates whether or not the writer should produce raw text rather than text in the output format for inline elements. The `writer->flatten_inlines` member variable is mutable.

```
5931   self.flatten_inlines = false
```

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

```
5932   local slice_specifiers = {}
5933   for specifier in options.slice:gmatch("[^%s]+") do
5934     table.insert(slice_specifiers, specifier)
5935   end
5936
5937   if #slice_specifiers == 2 then
5938     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5939     local slice_begin_type = self.slice_begin:sub(1, 1)
5940     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5941       self.slice_begin = "^" .. self.slice_begin
```

```
5942      end
5943      local slice_end_type = self.slice_end:sub(1, 1)
5944      if slice_end_type ~= "^" and slice_end_type ~= "$" then
5945        self.slice_end = "$" .. self.slice_end
5946      end
5947    elseif #slice_specifiers == 1 then
5948      self.slice_begin = "^" .. slice_specifiers[1]
5949      self.slice_end = "$" .. slice_specifiers[1]
5950    end
5951
5952    self.slice_begin_type = self.slice_begin:sub(1, 1)
5953    self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5954    self.slice_end_type = self.slice_end:sub(1, 1)
5955    self.slice_end_identifier = self.slice_end:sub(2) or ""
5956
5957    if self.slice_begin == "^" and self.slice_end ~= "^" then
5958      self.is_writing = true
5959    else
5960      self.is_writing = false
5961    end
```

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

```
5962    self.space = " "
```

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

```
5963    self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5964    function self.plain(s)
5965      return s
5966    end
```

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

```
5967    function self.paragraph(s)
5968      if not self.is_writing then return "" end
5969      return s
5970    end
```

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

```
5971    self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
5972    function self.interblocksep()
5973      if not self.is_writing then return "" end
5974      return self.interblocksep_text
5975    end
```

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of

a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

```
5976  self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
5977  function self.paragraphsep()
5978    if not self.is_writing then return "" end
5979    return self.paragraphsep_text
5980  end
```

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

```
5981  self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
5982  function self.undosep()
5983    if not self.is_writing then return "" end
5984    return self.undosep_text
5985  end
```

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

```
5986  self.soft_line_break = function()
5987    if self.flatten_inlines then return "\n" end
5988    return "\\markdownRendererSoftLineBreak\n{}"
5989  end
```

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

```
5990  self.hard_line_break = function()
5991    if self.flatten_inlines then return "\n" end
5992    return "\\markdownRendererHardLineBreak\n{}"
5993  end
```

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

```
5994  self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
5995  function self.thematic_break()
5996    if not self.is_writing then return "" end
5997    return "\\markdownRendererThematicBreak{}"
5998  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.

```
5999  self.escaped_uri_chars = {
6000    ["{"] = "\\markdownRendererLeftBrace{}",
6001    ["}"] = "\\markdownRendererRightBrace{}",
6002    ["\\"] = "\\markdownRendererBackslash{}",
6003    ["\r"] = " ",
6004    ["\n"] = " ",
6005  }
6006  self.escaped_minimal_strings = {
6007    ["^^"] = "\\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 `escape_typographic_text`, `escape_programmatic_text`, and `escape_minimal` local escaper functions.

```
  local function create_escaper(char_escapes, string_escapes)
    local escape = util.escaper(char_escapes, string_escapes)
    return function(s)
      if self.flatten_inlines then return s end
      return escape(s)
    end
  end
  local escape_typographic_text = create_escaper(
    self.escaped_chars, self.escaped_strings)
  local escape_programmatic_text = create_escaper(
    self.escaped_uri_chars, self.escaped_minimal_strings)
  local escape_minimal = create_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.
- `writer->infostring` transforms a fence code infostring.

```
6045   self.escape = escape_typographic_text
6046   self.math = escape_minimal
6047   if options.hybrid then
6048     self.identifier = escape_minimal
6049     self.string = escape_minimal
6050     self.uri = escape_minimal
6051     self.infostring = escape_minimal
6052   else
6053     self.identifier = escape_programmatic_text
6054     self.string = escape_typographic_text
6055     self.uri = escape_programmatic_text
6056     self.infostring = escape_programmatic_text
6057   end
```

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```
6058   function self.warning(t, m)
6059     return {"\\markdownRendererWarning{", self.escape(t), "}{",
6060             escape_minimal(t), "}{", self.escape(m or ""), "}{",
6061             escape_minimal(m or ""), "}"}
6062   end
```

Define `writer->error` as a function that will transform an input error text `t` with optional more error text `m` to the output format.

```
6063   function self.error(t, m)
6064     return {"\\markdownRendererError{", self.escape(t), "}{",
6065             escape_minimal(t), "}{", self.escape(m or ""), "}{",
6066             escape_minimal(m or ""), "}"}
6067   end
```

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

```
6068   function self.code(s, attributes)
6069     if self.flatten_inlines then return s end
6070     local buf = {}
6071     if attributes ~= nil then
```

```
6072        table.insert(buf,
6073                     "\\markdownRendererCodeSpanAttributeContextBegin\n")
6074        table.insert(buf, self.attributes(attributes))
6075      end
6076      table.insert(buf,
6077                   {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
6078      if attributes ~= nil then
6079        table.insert(buf,
6080                     "\\markdownRendererCodeSpanAttributeContextEnd{}")
6081      end
6082      return buf
6083    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.

```
6084    function self.link(lab, src, tit, attributes)
6085      if self.flatten_inlines then return lab end
6086      local buf = {}
6087      if attributes ~= nil then
6088        table.insert(buf,
6089                     "\\markdownRendererLinkAttributeContextBegin\n")
6090        table.insert(buf, self.attributes(attributes))
6091      end
6092      table.insert(buf, {"\\markdownRendererLink{",lab,"}",
6093                         "{",self.escape(src),"}",
6094                         "{",self.uri(src),"}",
6095                         "{",self.string(tit or ""),"}"})
6096      if attributes ~= nil then
6097        table.insert(buf,
6098                     "\\markdownRendererLinkAttributeContextEnd{}")
6099      end
6100      return buf
6101    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.

```
6102    function self.image(lab, src, tit, attributes)
6103      if self.flatten_inlines then return lab end
6104      local buf = {}
6105      if attributes ~= nil then
6106        table.insert(buf,
6107                     "\\markdownRendererImageAttributeContextBegin\n")
6108        table.insert(buf, self.attributes(attributes))
6109      end
6110      table.insert(buf, {"\\markdownRendererImage{",lab,"}",
```

```
                        "{",self.string(src),"}",
                        "{",self.uri(src),"}",
                        "{",self.string(tit or ""),"}"})
    if attributes ~= nil then
      table.insert(buf,
                   "\\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
    local buffer = {}
    for _,item in ipairs(items) do
      if item ~= "" then
        buffer[#buffer + 1] = self.bulletitem(item)
      end
    end
    local contents = util.intersperse(buffer,"\n")
    if tight and options.tightLists then
      return {"\\markdownRendererUlBeginTight\n",contents,
        "\n\\markdownRendererUlEndTight "}
    else
      return {"\\markdownRendererUlBegin\n",contents,
        "\n\\markdownRendererUlEnd "}
    end
  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.

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

```
  function self.orderedlist(items,tight,startnum)
    if not self.is_writing then return "" end
    local buffer = {}
    local num = startnum
    for _,item in ipairs(items) do
      if item ~= "" then
```

```
6147          buffer[#buffer + 1] = self.ordereditem(item,num)
6148        end
6149        if num ~= nil and item ~= "" then
6150          num = num + 1
6151        end
6152      end
6153      local contents = util.intersperse(buffer,"\n")
6154      if tight and options.tightLists then
6155        return {"\\markdownRendererOlBeginTight\n",contents,
6156                "\n\\markdownRendererOlEndTight "}
6157      else
6158        return {"\\markdownRendererOlBegin\n",contents,
6159                "\n\\markdownRendererOlEnd "}
6160      end
6161    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.

```
6162    function self.ordereditem(s,num)
6163      if num ~= nil then
6164        return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
6165                "\\markdownRendererOlItemEnd "}
6166      else
6167        return {"\\markdownRendererOlItem ",s,
6168                "\\markdownRendererOlItemEnd "}
6169      end
6170    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.

```
6171    function self.inline_html_comment(contents)
6172      if self.flatten_inlines then return contents end
6173      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
6174    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.

```
6175    function self.inline_html_tag(contents)
6176      if self.flatten_inlines then return contents end
6177      return {"\\markdownRendererInlineHtmlTag{",
6178              self.string(contents),"}"}
6179    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)
  if self.flatten_inlines then return s end
  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)
  if self.flatten_inlines then return s end
  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 not self.is_writing then return "" end
  return {"\\markdownRendererBlockQuoteBegin\n",s,
    "\\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$", "")
```

```
6210    local name = util.cache_verbatim(options.cacheDir, s)
6211    return {"\\markdownRendererInputVerbatim{",name,"}"}
6212  end
```

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

```
6213  function self.document(d)
6214    local buf = {"\\markdownRendererDocumentBegin\n"}
6215
6216    -- warn against the `hybrid` option
6217    if options.hybrid then
6218      local text = "The `hybrid` option has been soft-deprecated."
6219      local more = "Consider using one of the following better options "
6220                .. "for mixing TeX and markdown: `contentBlocks`, "
6221                .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6222                .. "`texMathSingleBackslash`, and "
6223                .. "`texMathDoubleBackslash`. "
6224                .. "For more information, see the user manual at "
6225                .. "<https://witiko.github.io/markdown/>."
6226      table.insert(buf, self.warning(text, more))
6227    end
6228
6229    -- insert the text of the document
6230    table.insert(buf, d)
6231
6232    -- pop all attributes
6233    table.insert(buf, self.pop_attributes())
6234
6235    table.insert(buf, "\\markdownRendererDocumentEnd")
6236
6237    return buf
6238  end
```

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

```
6239  local seen_identifiers = {}
6240  local key_value_regex = "([^= ]+)%s*=%s*(.*)"
6241  local function normalize_attributes(attributes, auto_identifiers)
6242    -- normalize attributes
6243    local normalized_attributes = {}
6244    local has_explicit_identifiers = false
6245    local key, value
6246    for _, attribute in ipairs(attributes or {}) do
6247      if attribute:sub(1, 1) == "#" then
6248        table.insert(normalized_attributes, attribute)
6249        has_explicit_identifiers = true
6250        seen_identifiers[attribute:sub(2)] = true
6251      elseif attribute:sub(1, 1) == "." then
```

```
      table.insert(normalized_attributes, attribute)
    else
      key, value = attribute:match(key_value_regex)
      if key:lower() == "id" then
        table.insert(normalized_attributes, "#" .. value)
      elseif key:lower() == "class" then
        local classes = {}
        for class in value:gmatch("%S+") do
          table.insert(classes, class)
        end
        table.sort(classes)
        for _, class in ipairs(classes) do
          table.insert(normalized_attributes, "." .. class)
        end
      else
        table.insert(normalized_attributes, attribute)
      end
    end
  end

  -- if no explicit identifiers exist, add auto identifiers
  if not has_explicit_identifiers and auto_identifiers ~= nil then
    local seen_auto_identifiers = {}
    for _, auto_identifier in ipairs(auto_identifiers) do
      if seen_auto_identifiers[auto_identifier] == nil then
        seen_auto_identifiers[auto_identifier] = true
        if seen_identifiers[auto_identifier] == nil then
          seen_identifiers[auto_identifier] = true
          table.insert(normalized_attributes,
                       "#" .. auto_identifier)
        else
          local auto_identifier_number = 1
          while true do
            local numbered_auto_identifier = auto_identifier .. "-"
                                             .. auto_identifier_number
            if seen_identifiers[numbered_auto_identifier] == nil then
              seen_identifiers[numbered_auto_identifier] = true
              table.insert(normalized_attributes,
                           "#" .. numbered_auto_identifier)
              break
            end
            auto_identifier_number = auto_identifier_number + 1
          end
        end
      end
    end
  end
```

```
6299
6300      -- sort and deduplicate normalized attributes
6301      table.sort(normalized_attributes)
6302      local seen_normalized_attributes = {}
6303      local deduplicated_normalized_attributes = {}
6304      for _, attribute in ipairs(normalized_attributes) do
6305        if seen_normalized_attributes[attribute] == nil then
6306          seen_normalized_attributes[attribute] = true
6307          table.insert(deduplicated_normalized_attributes, attribute)
6308        end
6309      end
6310
6311      return deduplicated_normalized_attributes
6312    end
6313
6314    function self.attributes(attributes, should_normalize_attributes)
6315      local normalized_attributes
6316      if should_normalize_attributes == false then
6317        normalized_attributes = attributes
6318      else
6319        normalized_attributes = normalize_attributes(attributes)
6320      end
6321
6322      local buf = {}
6323      local key, value
6324      for _, attribute in ipairs(normalized_attributes) do
6325        if attribute:sub(1, 1) == "#" then
6326          table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
6327                             attribute:sub(2), "}"})
6328        elseif attribute:sub(1, 1) == "." then
6329          table.insert(buf, {"\\markdownRendererAttributeClassName{",
6330                             attribute:sub(2), "}"})
6331        else
6332          key, value = attribute:match(key_value_regex)
6333          table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
6334                             key, "}{", value, "}"})
6335        end
6336      end
6337
6338      return buf
6339    end
```

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

```
6340    self.active_attributes = {}
```

Define `writer->attribute_type_levels` as a hash table that maps attribute types to the number of attributes of said type in `writer->active_attributes`.

```
6341  self.attribute_type_levels = {}
6342  setmetatable(self.attribute_type_levels,
6343               { __index = function() return 0 end })
```

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

```
6344  local function apply_attributes()
6345    local buf = {}
6346    for i = 1, #self.active_attributes do
6347      local start_output = self.active_attributes[i][3]
6348      if start_output ~= nil then
6349        table.insert(buf, start_output)
6350      end
6351    end
6352    return buf
6353  end
6354
6355  local function tear_down_attributes()
6356    local buf = {}
6357    for i = #self.active_attributes, 1, -1 do
6358      local end_output = self.active_attributes[i][4]
6359      if end_output ~= nil then
6360        table.insert(buf, end_output)
6361      end
6362    end
6363    return buf
6364  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.

```
6365  function self.push_attributes(attribute_type, attributes,
6366                                start_output, end_output)
6367    local attribute_type_level
6368      = self.attribute_type_levels[attribute_type]
6369    self.attribute_type_levels[attribute_type]
6370      = attribute_type_level + 1
6371
6372    -- index attributes in a hash table for easy lookup
6373    attributes = attributes or {}
6374    for i = 1, #attributes do
6375      attributes[attributes[i]] = true
6376    end
6377
```

```
6378    local buf = {}
6379    -- handle slicing
6380    if attributes["#" .. self.slice_end_identifier] ~= nil and
6381       self.slice_end_type == "^" then
6382      if self.is_writing then
6383        table.insert(buf, self.undosep())
6384        table.insert(buf, tear_down_attributes())
6385      end
6386      self.is_writing = false
6387    end
6388    if attributes["#" .. self.slice_begin_identifier] ~= nil and
6389       self.slice_begin_type == "^" then
6390      table.insert(buf, apply_attributes())
6391      self.is_writing = true
6392    end
6393    if self.is_writing and start_output ~= nil then
6394      table.insert(buf, start_output)
6395    end
6396    table.insert(self.active_attributes,
6397                 {attribute_type, attributes,
6398                  start_output, end_output})
6399    return buf
6400  end
6401
```

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).

```
6402  function self.pop_attributes(attribute_type)
6403    local buf = {}
6404    -- pop attributes until we find attributes of correct type
6405    -- or until no attributes remain
6406    local current_attribute_type = false
6407    while current_attribute_type ~= attribute_type and
6408          #self.active_attributes > 0 do
6409      local attributes, _, end_output
6410      current_attribute_type, attributes, _, end_output = table.unpack(
6411        self.active_attributes[#self.active_attributes])
6412      local attribute_type_level
6413        = self.attribute_type_levels[current_attribute_type]
6414      self.attribute_type_levels[current_attribute_type]
6415        = attribute_type_level - 1
6416      if self.is_writing and end_output ~= nil then
6417        table.insert(buf, end_output)
```

```
      end
      table.remove(self.active_attributes, #self.active_attributes)
      -- handle slicing
      if attributes["#" .. self.slice_end_identifier] ~= nil
         and self.slice_end_type == "$" then
        if self.is_writing then
          table.insert(buf, self.undosep())
          table.insert(buf, tear_down_attributes())
        end
        self.is_writing = false
      end
      if attributes["#" .. self.slice_begin_identifier] ~= nil and
         self.slice_begin_type == "$" then
        self.is_writing = true
        table.insert(buf, apply_attributes())
      end
    end
    return buf
  end
```

Create an auto identifier string by stripping and converting characters from string `s`.

```
  local function create_auto_identifier(s)
    local buffer = {}
    local prev_space = false
    local letter_found = false
    local normalized_s = s
    if not options.unicodeNormalization
       or options.unicodeNormalizationForm ~= "nfc" then
      normalized_s = uni_algos.normalize.NFC(normalized_s)
    end

    for _, code in utf8.codes(normalized_s) do
      local char = utf8.char(code)

      -- Remove everything up to the first letter.
      if not letter_found then
        local is_letter = unicode.utf8.match(char, "%a")
        if is_letter then
          letter_found = true
        else
          goto continue
        end
      end

      -- Remove all non-alphanumeric characters, except underscores,
      -- hyphens, and periods.
      if not unicode.utf8.match(char, "[%w_%-%.%s]") then
        goto continue
```

```
      end

      -- Replace all spaces and newlines with hyphens.
      if unicode.utf8.match(char, "[%s\n]") then
        char = "-"
        if prev_space then
          goto continue
        else
          prev_space = true
        end
      else
        -- Convert all alphabetic characters to lowercase.
        char = unicode.utf8.lower(char)
        prev_space = false
      end

      table.insert(buffer, char)

      ::continue::
    end

    if prev_space then
      table.remove(buffer)
    end

    local identifier = #buffer == 0 and "section"
                    or table.concat(buffer, "")
    return identifier
  end
```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string s.

```
  local function create_gfm_auto_identifier(s)
    local buffer = {}
    local prev_space = false
    local letter_found = false
    local normalized_s = s
    if not options.unicodeNormalization
       or options.unicodeNormalizationForm ~= "nfc" then
      normalized_s = uni_algos.normalize.NFC(normalized_s)
    end

    for _, code in utf8.codes(normalized_s) do
      local char = utf8.char(code)

      -- Remove everything up to the first non-space.
      if not letter_found then
```

```
6508          local is_letter = unicode.utf8.match(char, "%S")
6509          if is_letter then
6510            letter_found = true
6511          else
6512            goto continue
6513          end
6514        end
6515
6516        -- Remove all non-alphanumeric characters, except underscores
6517        -- and hyphens.
6518        if not unicode.utf8.match(char, "[%w_%-%s]") then
6519          prev_space = false
6520          goto continue
6521        end
6522
6523        -- Replace all spaces and newlines with hyphens.
6524        if unicode.utf8.match(char, "[%s\n]") then
6525          char = "-"
6526          if prev_space then
6527            goto continue
6528          else
6529            prev_space = true
6530          end
6531        else
6532          -- Convert all alphabetic characters to lowercase.
6533          char = unicode.utf8.lower(char)
6534          prev_space = false
6535        end
6536
6537        table.insert(buffer, char)
6538
6539        ::continue::
6540      end
6541
6542      if prev_space then
6543        table.remove(buffer)
6544      end
6545
6546      local identifier = #buffer == 0 and "section"
6547                      or table.concat(buffer, "")
6548      return identifier
6549    end
```

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

```
6550    self.secbegin_text = "\\markdownRendererSectionBegin\n"
6551    self.secend_text = "\n\\markdownRendererSectionEnd "
```

```
function self.heading(s, level, attributes)
  local buf = {}
  local flat_text, inlines = table.unpack(s)

  -- push empty attributes for implied sections
  while self.attribute_type_levels["heading"] < level - 1 do
    table.insert(buf,
                 self.push_attributes("heading",
                                      nil,
                                      self.secbegin_text,
                                      self.secend_text))
  end

  -- pop attributes for sections that have ended
  while self.attribute_type_levels["heading"] >= level do
    table.insert(buf, self.pop_attributes("heading"))
  end

  -- construct attributes for the new section
  local auto_identifiers = {}
  if self.options.autoIdentifiers then
    table.insert(auto_identifiers, create_auto_identifier(flat_text))
  end
  if self.options.gfmAutoIdentifiers then
    table.insert(auto_identifiers,
                 create_gfm_auto_identifier(flat_text))
  end
  local normalized_attributes = normalize_attributes(attributes,
                                                     auto_identifiers)

  -- push attributes for the new section
  local start_output = {}
  local end_output = {}
  table.insert(start_output, self.secbegin_text)
  table.insert(end_output, self.secend_text)

  table.insert(buf, self.push_attributes("heading",
                                         normalized_attributes,
                                         start_output,
                                         end_output))
  assert(self.attribute_type_levels["heading"] == level)

  -- render the heading and its attributes
  if self.is_writing and #normalized_attributes > 0 then
    table.insert(buf,
                 "\\markdownRendererHeaderAttributeContextBegin\n")
    table.insert(buf, self.attributes(normalized_attributes, false))
```

```
    end

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

    if self.is_writing and #normalized_attributes > 0 then
      table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{}")
    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,
      flatten_inlines=self.flatten_inlines,
      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.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 + parsers.dash)^0

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

```

### 3.1.5 Unicode punctuation

This section documents the Unicode punctuation[33] recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

| (CommonMark Spec, Version 0.31.2 (2024-01-28)) |
|---|

[33] See https://spec.commonmark.org/0.31.2/#unicode-punctuation-character.

All code from this section will be executed during the compilation of the Markdown package and the standard output will be stored in a file named markdown-unicode-data.lua with the precompiled parser of Unicode punctuation.

```
;(function()
  local pathname = assert(kpse.find_file("UnicodeData.txt"),
    [[Could not locate file "UnicodeData.txt"]])
  local file = assert(io.open(pathname, "r"),
    [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct a prefix tree of UTF-8 encodings for all codepoints of a given code length.

```
  local prefix_trees = {}
  for line in file:lines() do
    local codepoint, major_category = line:match("^(%x+);[^;]*;(%a)")
    if major_category == "P" or major_category == "S" then
      local code = unicode.utf8.char(tonumber(codepoint, 16))
      if prefix_trees[#code] == nil then
        prefix_trees[#code] = {}
      end
      local node = prefix_trees[#code]
      for i = 1, #code do
        local byte = code:sub(i, i)
        if i < #code then
          if node[byte] == nil then
            node[byte] = {}
          end
          node = node[byte]
        else
          table.insert(node, byte)
        end
      end
    end
  end
  assert(file:close())

```

Next, we will construct a parser out of the prefix tree.

```
  local function depth_first_search(node, path, visit, leave)
    visit(node, path)
    for label, child in pairs(node) do
      if type(child) == "table" then
        depth_first_search(child, path .. label, visit, leave)
      else
        visit(child, path)
      end
    end
    leave(node, path)
```

```
  end

  print("M.punctuation = {}")
  print("local S = lpeg.S")
  print("-- luacheck: push no max line length")
  for length, prefix_tree in pairs(prefix_trees) do
    local subparsers = {}
    depth_first_search(prefix_tree, "", function(node, path)
      if type(node) == "string" then
        local suffix
        if node == "]" then
          suffix = "S('" .. node .. "')"
        else
          suffix = "S([[" .. node .. "]])"
        end
        if subparsers[path] ~= nil then
          subparsers[path] = subparsers[path] .. " + " .. suffix
        else
          subparsers[path] = suffix
        end
      end
    end, function(_, path)
      if #path > 0 then
        local byte = path:sub(#path, #path)
        local parent_path = path:sub(1, #path-1)
        if subparsers[path] ~= nil then
          local suffix
          if byte == "]" then
            suffix = "S('" .. byte .. "')"
          else
            suffix = "S([[" .. byte .. "]])"
          end
          suffix = suffix .. " * (" .. subparsers[path] .. ")"
          if subparsers[parent_path] ~= nil then
            subparsers[parent_path] = subparsers[parent_path]
                                      .. " + " .. suffix
          else
            subparsers[parent_path] = suffix
          end
        end
      else
        print("M.punctuation[" .. length .. "] = " .. subparsers[path])
      end
    end)
  end
  print("-- luacheck: pop")
end)()
```

```
print("return M")
```

Back in the Markdown package, we will load the precompiled parser of Unicode punctuation.

```
local unicode_data = require("markdown-unicode-data")
if metadata.version ~= unicode_data.metadata.version then
  util.warning(
    "markdown.lua " .. metadata.version .. " used with " ..
    "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
  )
end
parsers.punctuation = unicode_data.punctuation

parsers.escapable               = parsers.ascii_punctuation
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.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
```

#### 3.1.5.1 Parsers Used for Indentation

```

parsers.leader      = parsers.space^-3

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```
local function has_trail(indent_table)
  return indent_table ~= nil and
    indent_table.trail ~= nil and
    next(indent_table.trail) ~= nil
end

```

Check if indent table `indent_table` has any indents.

```
local function has_indents(indent_table)
  return indent_table ~= nil and
    indent_table.indents ~= nil and
    next(indent_table.indents) ~= nil
end

```

Add a trail `trail_info` to the indent table `indent_table`.

```
local function add_trail(indent_table, trail_info)
  indent_table.trail = trail_info
  return indent_table
end

```

Remove a trail `trail_info` from the indent table `indent_table`.

```
local function remove_trail(indent_table)
  indent_table.trail = nil
  return indent_table
end

```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
local function update_indent_table(indent_table, new_indent, add)
  indent_table = remove_trail(indent_table)

  if not has_indents(indent_table) then
    indent_table.indents = {}
  end


  if add then
    indent_table.indents[#indent_table.indents + 1] = new_indent
```

```
  else
    if indent_table.indents[#indent_table.indents].name
       == new_indent.name then
      indent_table.indents[#indent_table.indents] = nil
    end
  end

  return indent_table
end

```

Remove an indent by its name `name`.

```
local function remove_indent(name)
  local remove_indent_level =
    function(s, i, indent_table) -- luacheck: ignore s i
      indent_table = update_indent_table(indent_table, {name=name},
                                         false)
      return true, indent_table
    end

  return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
end

```

Process the spacing of a string of spaces and tabs `spacing` with preceding indent width from the start of the line `indent` and strip up to `left_strip_length` spaces. Return the remainder `remainder` and whether there is enough spaces to produce a code `is_code`. Return how many spaces were stripped, as well as if the minimum was met `is_minimum` and what remainder it left `minimum_remainder`.

```
local function process_starter_spacing(indent, spacing,
                                       minimum, left_strip_length)
  left_strip_length = left_strip_length or 0

  local count = 0
  local tab_value = 4 - (indent) % 4

  local code_started, minimum_found = false, false
  local code_start, minimum_remainder = "", ""

  local left_total_stripped = 0
  local full_remainder = ""

  if spacing ~= nil then
    for i = 1, #spacing do
      local character = spacing:sub(i, i)

      if character == "\t" then
```

```
        count = count + tab_value
        tab_value = 4
      elseif character == " " then
        count = count + 1
        tab_value = 4 - (1 - tab_value) % 4
      end

      if (left_strip_length ~= 0) then
        local possible_to_strip = math.min(count, left_strip_length)
        count = count - possible_to_strip
        left_strip_length = left_strip_length - possible_to_strip
        left_total_stripped = left_total_stripped + possible_to_strip
      else
        full_remainder =  full_remainder .. character
      end

      if (minimum_found) then
        minimum_remainder = minimum_remainder .. character
      elseif (count >= minimum) then
        minimum_found = true
        minimum_remainder = minimum_remainder
                          .. string.rep(" ", count - minimum)
      end

      if (code_started) then
        code_start = code_start .. character
      elseif (count >= minimum + 4) then
        code_started = true
        code_start = code_start
                  .. string.rep(" ", count - (minimum + 4))
      end
    end
  end

  local remainder
  if (code_started) then
    remainder = code_start
  else
    remainder = string.rep(" ", count - minimum)
  end

  local is_minimum = count >= minimum
  return {
    is_code = code_started,
    remainder = remainder,
    left_total_stripped = left_total_stripped,
    is_minimum = is_minimum,
```

```
    minimum_remainder = minimum_remainder,
    total_length = count,
    full_remainder = full_remainder
  }
end
```

Count the total width of all indents in the indent table `indent_table`.

```
local function count_indent_tab_level(indent_table)
  local count = 0
  if not has_indents(indent_table) then
    return count
  end

  for i=1, #indent_table.indents do
    count = count + indent_table.indents[i].length
  end
  return count
end
```

Count the total width of a delimiter `delimiter`.

```
local function total_delimiter_length(delimiter)
  local count = 0
  if type(delimiter) == "string" then return #delimiter end
  for _, value in pairs(delimiter) do
    count = count + total_delimiter_length(value)
  end
  return count
end
```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```
local function process_starter_indent(_, _, indent_table, starter,
                                       is_blank, indent_type, breakable)
  local last_trail = starter[1]
  local delimiter = starter[2]
  local raw_new_trail = starter[3]

  if indent_type == "bq" and not breakable then
    indent_table.ignore_blockquote_blank = true
  end

  if has_trail(indent_table) then
    local trail = indent_table.trail
    if trail.is_code then
      return false
```

```
    end
    last_trail = trail.remainder
  else
    local sp = process_starter_spacing(0, last_trail, 0, 0)

    if sp.is_code then
      return false
    end
    last_trail = sp.remainder
  end

  local preceding_indentation = count_indent_tab_level(indent_table) % 4
  local last_trail_length = #last_trail
  local delimiter_length = total_delimiter_length(delimiter)

  local total_indent_level = preceding_indentation + last_trail_length
                           + delimiter_length

  local sp = {}
  if not is_blank then
    sp = process_starter_spacing(total_indent_level, raw_new_trail,
                                 0, 1)
  end

  local del_trail_length = sp.left_total_stripped
  if is_blank then
    del_trail_length = 1
  elseif not sp.is_code then
    del_trail_length = del_trail_length + #sp.remainder
  end

  local indent_length = last_trail_length + delimiter_length
                      + del_trail_length
  local new_indent_info = {name=indent_type, length=indent_length}

  indent_table = update_indent_table(indent_table, new_indent_info,
                                     true)
  indent_table = add_trail(indent_table,
                           {is_code=sp.is_code,
                            remainder=sp.remainder,
                            total_length=sp.total_length,
                            full_remainder=sp.full_remainder})

  return true, indent_table
end

```

Return the pattern corresponding with the indent name `name`.

```
local function decode_pattern(name)
  local delimeter = parsers.succeed
  if name == "bq" then
    delimeter = parsers.more
  end

  return C(parsers.optionalspace) * C(delimeter)
       * C(parsers.optionalspace) * Cp()
end

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```
local function left_blank_starter(indent_table)
  local blank_starter_index

  if not has_indents(indent_table) then
    return
  end

  for i = #indent_table.indents,1,-1 do
    local value = indent_table.indents[i]
    if value.name == "li" then
      blank_starter_index = i
    else
      break
    end
  end

  return blank_starter_index
end

```

Apply the patterns decoded from the indents of the indent table `indent_table` iteratively starting at position `index` of the string `s`. If the `is_optional` mode is selected, match as many patterns as possible, else match all or fail. With the option `is_blank`, the parsing behaves as optional after the position of a blank-only indent has been surpassed.

```
local function traverse_indent(s, i, indent_table, is_optional,
                               is_blank, current_line_indents)
  local new_index = i

  local preceding_indentation = 0
  local current_trail = {}

  local blank_starter = left_blank_starter(indent_table)
```

```
7085
7086  if current_line_indents == nil then
7087    current_line_indents = {}
7088  end
7089
7090  for index = 1,#indent_table.indents do
7091    local value = indent_table.indents[index]
7092    local pattern = decode_pattern(value.name)
7093
7094    -- match decoded pattern
7095    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7096    if new_indent_info == nil then
7097      local blankline_end = lpeg.match(
7098        Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7099      if is_optional or not indent_table.ignore_blockquote_blank
7100         or not blankline_end then
7101        return is_optional, new_index, current_trail,
7102               current_line_indents
7103      end
7104
7105      return traverse_indent(s, tonumber(blankline_end.pos),
7106                             indent_table, is_optional, is_blank,
7107                             current_line_indents)
7108    end
7109
7110    local raw_last_trail = new_indent_info[1]
7111    local delimiter = new_indent_info[2]
7112    local raw_new_trail = new_indent_info[3]
7113    local next_index = new_indent_info[4]
7114
7115    local space_only = delimiter == ""
7116
7117    -- check previous trail
7118    if not space_only and next(current_trail) == nil then
7119      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7120      current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7121                       total_length=sp.total_length,
7122                       full_remainder=sp.full_remainder}
7123    end
7124
7125    if next(current_trail) ~= nil then
7126      if not space_only and current_trail.is_code then
7127        return is_optional, new_index, current_trail,
7128               current_line_indents
7129      end
7130      if current_trail.internal_remainder ~= nil then
7131        raw_last_trail = current_trail.internal_remainder
```

```
      end
    end

    local raw_last_trail_length = 0
    local delimiter_length = 0

    if not space_only then
      delimiter_length = #delimiter
      raw_last_trail_length = #raw_last_trail
    end

    local total_indent_level = preceding_indentation
                             + raw_last_trail_length + delimiter_length

    local spacing_to_process
    local minimum = 0
    local left_strip_length = 0

    if not space_only then
      spacing_to_process = raw_new_trail
      left_strip_length = 1
    else
      spacing_to_process = raw_last_trail
      minimum = value.length
    end

    local sp = process_starter_spacing(total_indent_level,
                                       spacing_to_process, minimum,
                                       left_strip_length)

    if space_only and not sp.is_minimum then
      return is_optional or (is_blank and blank_starter <= index),
             new_index, current_trail, current_line_indents
    end

    local indent_length = raw_last_trail_length + delimiter_length
                        + sp.left_total_stripped

    -- update info for the next pattern
    if not space_only then
      preceding_indentation = preceding_indentation + indent_length
    else
      preceding_indentation = preceding_indentation + value.length
    end

    current_trail = {is_code=sp.is_code, remainder=sp.remainder,
                     internal_remainder=sp.minimum_remainder,
```

```
                    total_length=sp.total_length,
                    full_remainder=sp.full_remainder}

    current_line_indents[#current_line_indents + 1] = new_indent_info
    new_index = next_index
  end

  return true, new_index, current_trail, current_line_indents
end

```

Check if a code trail is expected.

```
local function check_trail(expect_code, is_code)
  return (expect_code and is_code) or (not expect_code and not is_code)
end

```

Check if the current trail of the `indent_table` would produce code if it is expected `expect_code` or it would not if it is not. If there is no trail, process and check the current spacing `spacing`.

```
local check_trail_joined =
  function(s, i, indent_table, -- luacheck: ignore s i
           spacing, expect_code, omit_remainder)
    local is_code
    local remainder

    if has_trail(indent_table) then
      local trail = indent_table.trail
      is_code = trail.is_code
      if is_code then
        remainder = trail.remainder
      else
        remainder = trail.full_remainder
      end
    else
      local sp = process_starter_spacing(0, spacing, 0, 0)
      is_code = sp.is_code
      if is_code then
        remainder = sp.remainder
      else
        remainder = sp.full_remainder
      end
    end

    local result = check_trail(expect_code, is_code)
    if omit_remainder then
      return result
    end
```

```
7221      return result, remainder
7222    end
7223
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
7224 local check_trail_length =
7225    function(s, i, indent_table, -- luacheck: ignore s i
7226             spacing, min, max)
7227      local trail
7228
7229      if has_trail(indent_table) then
7230        trail = indent_table.trail
7231      else
7232        trail = process_starter_spacing(0, spacing, 0, 0)
7233      end
7234
7235      local total_length = trail.total_length
7236      if total_length == nil then
7237        return false
7238      end
7239
7240      return min <= total_length and total_length <= max
7241    end
7242
```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```
7243 local function check_continuation_indentation(s, i, indent_table,
7244                                                 is_optional, is_blank)
7245    if not has_indents(indent_table) then
7246      return true
7247    end
7248
7249    local passes, new_index, current_trail, current_line_indents =
7250      traverse_indent(s, i, indent_table, is_optional, is_blank)
7251
7252    if passes then
7253      indent_table.current_line_indents = current_line_indents
7254      indent_table = add_trail(indent_table, current_trail)
7255      return new_index, indent_table
7256    end
7257    return false
7258 end
7259
```

Get name of the last indent from the `indent_table`.

```
7260 local function get_last_indent_name(indent_table)
7261    if has_indents(indent_table) then
```

```
7262      return indent_table.indents[#indent_table.indents].name
7263    end
7264  end
7265
```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
7266  local function remove_remainder_if_blank(indent_table, remainder)
7267    if get_last_indent_name(indent_table) == "li" then
7268      return ""
7269    end
7270    return remainder
7271  end
7272
```

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```
7273  local check_trail_type =
7274    function(s, i, -- luacheck: ignore s i
7275             trail, spacing, trail_type)
7276      if trail == nil then
7277        trail = process_starter_spacing(0, spacing, 0, 0)
7278      end
7279
7280      if trail_type == "non-code" then
7281        return check_trail(false, trail.is_code)
7282      end
7283      if trail_type == "code" then
7284        return check_trail(true, trail.is_code)
7285      end
7286      if trail_type == "full-code" then
7287        if (trail.is_code) then
7288          return i, trail.remainder
7289        end
7290        return i, ""
7291      end
7292      if trail_type == "full-any" then
7293        return i, trail.internal_remainder
7294      end
7295    end
7296
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
7297  local trail_freezing =
7298    function(s, i, -- luacheck: ignore s i
7299             indent_table, is_freezing)
7300      if is_freezing then
7301        if indent_table.is_trail_frozen then
```

```
7302          indent_table.trail = indent_table.frozen_trail
7303        else
7304          indent_table.frozen_trail = indent_table.trail
7305          indent_table.is_trail_frozen = true
7306        end
7307      else
7308        indent_table.frozen_trail = nil
7309        indent_table.is_trail_frozen = false
7310      end
7311      return true, indent_table
7312    end
7313
```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```
7314 local check_continuation_indentation_and_trail =
7315   function (s, i, indent_table, is_optional, is_blank, trail_type,
7316             reset_rem, omit_remainder)
7317     if not has_indents(indent_table) then
7318       local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
7319                                              * Cp(), s, i)
7320       local result, remainder = check_trail_type(s, i,
7321         indent_table.trail, spacing, trail_type)
7322       if remainder == nil then
7323         if result then
7324           return new_index
7325         end
7326         return false
7327       end
7328       if result then
7329         return new_index, remainder
7330       end
7331       return false
7332     end
7333
7334     local passes, new_index, current_trail = traverse_indent(s, i,
7335       indent_table, is_optional, is_blank)
7336
7337     if passes then
7338       local spacing
7339       if current_trail == nil then
7340         local newer_spacing, newer_index = lpeg.match(
7341           C(parsers.spacechar^0) * Cp(), s, i)
7342         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7343         new_index = newer_index
7344         spacing = newer_spacing
```

```
7345       else
7346         spacing = current_trail.remainder
7347       end
7348       local result, remainder = check_trail_type(s, new_index,
7349         current_trail, spacing, trail_type)
7350       if remainder == nil or omit_remainder then
7351         if result then
7352           return new_index
7353         end
7354         return false
7355       end
7356
7357       if is_blank and reset_rem then
7358         remainder = remove_remainder_if_blank(indent_table, remainder)
7359       end
7360       if result then
7361         return new_index, remainder
7362       end
7363       return false
7364     end
7365     return false
7366   end
7367
```

The following patterns check whitespace indentation at the start of a block.

```
7368 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
7369                          * Cc(false), check_trail_joined)
7370
7371 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7372                                 * C(parsers.spacechar^0) * Cc(false)
7373                                 * Cc(true), check_trail_joined)
7374
7375 parsers.check_code_trail  = Cmt( Cb("indent_info")
7376                                * C(parsers.spacechar^0)
7377                                * Cc(true), check_trail_joined)
7378
7379 parsers.check_trail_length_range  = function(min, max)
7380   return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7381             * Cc(max), check_trail_length)
7382 end
7383
7384 parsers.check_trail_length = function(n)
7385   return parsers.check_trail_length_range(n, n)
7386 end
7387
```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```
parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
                         * Cc(true), trail_freezing), "indent_info")

parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
                            trail_freezing), "indent_info")

```

The following patterns check indentation in continuation lines as defined by the container start.

```
parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
                                   check_continuation_indentation)

parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
                                    check_continuation_indentation)

parsers.check_minimal_blank_indent
  = Cmt( Cb("indent_info") * Cc(false)
       * Cc(true)
       , check_continuation_indentation)

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

parsers.check_minimal_indent_and_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_indent_and_code_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("code") * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_full_code_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)
```

```
7430
7431 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7432   Cmt( Cb("indent_info")
7433      * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7434      , check_continuation_indentation_and_trail)
7435
7436 parsers.check_optional_indent_and_any_trail =
7437   Cmt( Cb("indent_info")
7438      * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7439      , check_continuation_indentation_and_trail)
7440
7441 parsers.check_optional_blank_indent_and_any_trail =
7442   Cmt( Cb("indent_info")
7443      * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7444      , check_continuation_indentation_and_trail)
7445
```

The following patterns specify behaviour around newlines.

```
7446
7447 parsers.spnlc_noexc = parsers.optionalspace
7448                     * ( parsers.newline
7449                       * parsers.check_minimal_indent_and_any_trail)^-1
7450
7451 parsers.spnlc = parsers.optionalspace
7452               * (V("EndlineNoSub"))^-1
7453
7454 parsers.spnlc_sep  = parsers.optionalspace * V("EndlineNoSub")
7455                    + parsers.spacechar^1
7456
7457 parsers.only_blank = parsers.spacechar^0
7458                    * (parsers.newline + parsers.eof)
7459
7460 %   \end{macrocode}
7461 % \begin{figure}
7462 % \hspace*{-0.1\textwidth}
7463 % \begin{minipage}{1.2\textwidth}
7464 % \centering
7465 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
7466 % \node[state, initial by diamond, accepting] (noop) {initial};
7467 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
7468 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
7469 % \node[state] (comment) [below=of noop] {comment};
7470 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs
7471 % \node[state] (blank_line) [below right=of comment] {blank line};
7472 % \path[->]
7473 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^\
7474 %        edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
7475 %        edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
```

```
% (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge$$\drsh
%           edge [bend left=10] node {match $\drsh$} (leading_spaces)
% (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$
%                  edge [bend right=90] node [right] {match \textbackslash} (odd_back
%                  edge [bend left=10] node {match \%} (comment)
%                  edge [bend right=10] node {$\epsilon$} (blank_line)
%                  edge [bend left=10] node [align=center, right=0.3cm] {match [$^\we
% (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
%              edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\\
% (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
%                 edge [bend right=10] node [align=center, above left=-
0.3cm, xshift=0.1cm] {match [$^\wedge$\textbackslash]\\for \%, capture \textbackslash
% (even_backslash) edge [bend left=10] node {$\epsilon$} (noop);
% \end{tikzpicture}
% \caption{A pushdown automaton that recognizes \TeX{} comments}
% \label{fig:commented_line}
% \end{minipage}
% \end{figure}
% \begin{markdown}
%
% The \luamdef{parsers.commented_line}`^1` parser recognizes the regular
% language of \TeX{} comments, see an equivalent finite automaton in Figure
% <#fig:commented_line>.
%
% \end{markdown}
%  \begin{macrocode}
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 + parsers.newline))
                         * Cg((parsers.backslash  -- even backslash
                              * ( parsers.backslash
                                + #parsers.newline))^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 spaces
                          + #(parsers.newline)
                          * Cb("backslashes")
                          * C(parsers.newline))

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

parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
parsers.attribute_raw_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.squote / "")
                            * (parsers.anyescaped - parsers.squote)^0
                            * (parsers.squote / ""))
                          + ( parsers.anyescaped
                            - parsers.dquote
                            - parsers.rbrace
                            - parsers.space)^0
parsers.attribute_identifier = parsers.attribute_key_char^1
parsers.attribute_classname = parsers.letter
                              * parsers.attribute_key_char^0
parsers.attribute_raw = parsers.attribute_raw_char^1

parsers.attribute = (parsers.dash * Cc(".unnumbered"))
```

```
                    + C( parsers.hash
                       * parsers.attribute_identifier)
                    + C( parsers.period
                       * parsers.attribute_classname)
                    + 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_raw)
                        * 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
```

#### 3.1.5.2 Parsers Used for HTML Entities

```
local function repeat_between(pattern, min, max)
  return -pattern^(max + 1) * pattern^min
end

parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
                    * C(repeat_between(parsers.hexdigit, 1, 6))
                    * parsers.semicolon
parsers.decentity = parsers.ampersand * parsers.hash
                    * C(repeat_between(parsers.digit, 1, 7))
                    * parsers.semicolon
```

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

parsers.html_entities
  = parsers.hexentity / entities.hex_entity_with_x_char
  + parsers.decentity / entities.dec_entity
  + parsers.tagentity / entities.char_entity
```

#### 3.1.5.3 Parsers Used for Markdown Lists

```
parsers.bullet = function(bullet_char, interrupting)
  local allowed_end
  if interrupting then
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
    allowed_end = C(parsers.spacechar^1)
                + #(parsers.newline + parsers.eof)
  end
  return parsers.check_trail
        * Ct(C(bullet_char) * Cc(""))
        * allowed_end
end

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)

```

#### 3.1.5.4 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  = Cmt(C(parsers.backtick^1)
                          * Cb("ticks"), captures_equal_length)

parsers.intickschar = (parsers.any - S("\n\r`"))
                    + V("NoSoftLineBreakEndline")
                    + (parsers.backtick^1 - parsers.closeticks)

```

```
local function process_inticks(s)
  s = s:gsub("\n", " ")
  s = s:gsub("^ (.*) $", "%1")
  return s
end

parsers.inticks = parsers.openticks
                * C(parsers.space^0)
                * parsers.closeticks
                + parsers.openticks
                * Cs(Cs(parsers.intickschar^0) / process_inticks)
                * parsers.closeticks

```

#### 3.1.5.5 Parsers Used for HTML

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

parsers.special_block_keyword =
    parsers.keyword_exact("pre") +
    parsers.keyword_exact("script") +
    parsers.keyword_exact("style") +
    parsers.keyword_exact("textarea")

parsers.block_keyword =
    parsers.keyword_exact("address") +
    parsers.keyword_exact("article") +
    parsers.keyword_exact("aside") +
    parsers.keyword_exact("base") +
    parsers.keyword_exact("basefont") +
    parsers.keyword_exact("blockquote") +
    parsers.keyword_exact("body") +
    parsers.keyword_exact("caption") +
    parsers.keyword_exact("center") +
    parsers.keyword_exact("col") +
    parsers.keyword_exact("colgroup") +
    parsers.keyword_exact("dd") +
```

```
    parsers.keyword_exact("details") +
    parsers.keyword_exact("dialog") +
    parsers.keyword_exact("dir") +
    parsers.keyword_exact("div") +
    parsers.keyword_exact("dl") +
    parsers.keyword_exact("dt") +
    parsers.keyword_exact("fieldset") +
    parsers.keyword_exact("figcaption") +
    parsers.keyword_exact("figure") +
    parsers.keyword_exact("footer") +
    parsers.keyword_exact("form") +
    parsers.keyword_exact("frame") +
    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("head") +
    parsers.keyword_exact("header") +
    parsers.keyword_exact("hr") +
    parsers.keyword_exact("html") +
    parsers.keyword_exact("iframe") +
    parsers.keyword_exact("legend") +
    parsers.keyword_exact("li") +
    parsers.keyword_exact("link") +
    parsers.keyword_exact("main") +
    parsers.keyword_exact("menu") +
    parsers.keyword_exact("menuitem") +
    parsers.keyword_exact("nav") +
    parsers.keyword_exact("noframes") +
    parsers.keyword_exact("ol") +
    parsers.keyword_exact("optgroup") +
    parsers.keyword_exact("option") +
    parsers.keyword_exact("p") +
    parsers.keyword_exact("param") +
    parsers.keyword_exact("section") +
    parsers.keyword_exact("source") +
    parsers.keyword_exact("summary") +
    parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") +
    parsers.keyword_exact("td") +
    parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("th") +
    parsers.keyword_exact("thead") +
    parsers.keyword_exact("title") +
```

```
    parsers.keyword_exact("tr") +
    parsers.keyword_exact("track") +
    parsers.keyword_exact("ul")

-- end conditions
parsers.html_blankline_end_condition
  = parsers.linechar^0
  * ( parsers.newline
    * (parsers.check_minimal_blank_indent_and_any_trail
      * #parsers.blankline
      + parsers.check_minimal_indent_and_any_trail)
    * parsers.linechar^1)^0
  * (parsers.newline^-1 / "")

local function remove_trailing_blank_lines(s)
  return s:gsub("[\n\r]+%s*$", "")
end

parsers.html_until_end = function(end_marker)
  return Cs(Cs((parsers.newline
           * (parsers.check_minimal_blank_indent_and_any_trail
             * #parsers.blankline
             + parsers.check_minimal_indent_and_any_trail)
           + parsers.linechar - end_marker)^0
           * parsers.linechar^0 * parsers.newline^-1)
         / remove_trailing_blank_lines)
end

-- attributes
parsers.html_attribute_spacing  = parsers.optionalspace
                                * V("NoSoftLineBreakEndline")
                                * parsers.optionalspace
                                + parsers.spacechar^1

parsers.html_attribute_name = ( parsers.letter
                              + parsers.colon
                              + parsers.underscore)
                            * ( parsers.alphanumeric
                              + parsers.colon
                              + parsers.underscore
                            + parsers.period
                            + parsers.dash)^0

parsers.html_attribute_value  = parsers.squote
                              * (parsers.linechar - parsers.squote)^0
                              * parsers.squote
                              + parsers.dquote
```

```
                                  * (parsers.linechar - parsers.dquote)^0
                                  * parsers.dquote
                                  + ( parsers.any
                                    - parsers.spacechar
                                    - parsers.newline
                                    - parsers.dquote
                                    - parsers.squote
                                    - parsers.backtick
                                    - parsers.equal
                                    - parsers.less
                                    - parsers.more)^1

parsers.html_inline_attribute_value = parsers.squote
                                      * (V("NoSoftLineBreakEndline")
                                        + parsers.any
                                        - parsers.blankline^2
                                        - parsers.squote)^0
                                      * parsers.squote
                                      + parsers.dquote
                                      * (V("NoSoftLineBreakEndline")
                                        + parsers.any
                                        - parsers.blankline^2
                                        - parsers.dquote)^0
                                      * parsers.dquote
                                      + (parsers.any
                                        - parsers.spacechar
                                        - parsers.newline
                                        - parsers.dquote
                                        - parsers.squote
                                        - parsers.backtick
                                        - parsers.equal
                                        - parsers.less
                                        - parsers.more)^1

parsers.html_attribute_value_specification
  = parsers.optionalspace
  * parsers.equal
  * parsers.optionalspace
  * parsers.html_attribute_value

parsers.html_spnl = parsers.optionalspace
                  * (V("NoSoftLineBreakEndline")
                  * parsers.optionalspace)^-1

parsers.html_inline_attribute_value_specification
  = parsers.html_spnl
  * parsers.equal
```

```
  * parsers.html_spnl
  * parsers.html_inline_attribute_value

parsers.html_attribute
  = parsers.html_attribute_spacing
  * parsers.html_attribute_name
  * parsers.html_inline_attribute_value_specification^-1

parsers.html_non_newline_attribute
  = parsers.spacechar^1
  * parsers.html_attribute_name
  * parsers.html_attribute_value_specification^-1

parsers.nested_breaking_blank = parsers.newline
                              * parsers.check_minimal_blank_indent
                              * parsers.blankline

parsers.html_comment_start = P("<!--")

parsers.html_comment_end = P("-->")

parsers.html_comment
  = Cs( parsers.html_comment_start
      * parsers.html_until_end(parsers.html_comment_end))

parsers.html_inline_comment = (parsers.html_comment_start / "")
                            * -P(">") * -P("->")
                            * Cs(( V("NoSoftLineBreakEndline")
                                 + parsers.any
                                 - parsers.nested_breaking_blank
                                 - parsers.html_comment_end)^0)
                            * (parsers.html_comment_end / "")

parsers.html_cdatasection_start = P("<![CDATA[")

parsers.html_cdatasection_end = P("]]>")

parsers.html_cdatasection
  = Cs( parsers.html_cdatasection_start
      * parsers.html_until_end(parsers.html_cdatasection_end))

parsers.html_inline_cdatasection
  = parsers.html_cdatasection_start
  * Cs(V("NoSoftLineBreakEndline") + parsers.any
      - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
  * parsers.html_cdatasection_end

```

```
parsers.html_declaration_start = P("<!") * parsers.letter

parsers.html_declaration_end = P(">")

parsers.html_declaration
  = Cs( parsers.html_declaration_start
      * parsers.html_until_end(parsers.html_declaration_end))

parsers.html_inline_declaration
  = parsers.html_declaration_start
  * Cs(V("NoSoftLineBreakEndline") + parsers.any
      - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
  * parsers.html_declaration_end

parsers.html_instruction_start = P("<?")

parsers.html_instruction_end = P("?>")

parsers.html_instruction
  = Cs( parsers.html_instruction_start
      * parsers.html_until_end(parsers.html_instruction_end))

parsers.html_inline_instruction = parsers.html_instruction_start
                                * Cs( V("NoSoftLineBreakEndline")
                                    + parsers.any
                                    - parsers.nested_breaking_blank
                                    - parsers.html_instruction_end)^0
                                * parsers.html_instruction_end

parsers.html_blankline  = parsers.newline
                        * parsers.optionalspace
                        * parsers.newline

parsers.html_tag_start = parsers.less

parsers.html_tag_closing_start  = parsers.less
                                * parsers.slash

parsers.html_tag_end  = parsers.html_spnl
                      * parsers.more

parsers.html_empty_tag_end  = parsers.html_spnl
                            * parsers.slash
                            * parsers.more

-- opening tags
parsers.html_any_open_inline_tag  = parsers.html_tag_start
```

```
                                  * parsers.keyword
                                  * parsers.html_attribute^0
                                  * parsers.html_tag_end

parsers.html_any_open_tag = parsers.html_tag_start
                          * parsers.keyword
                          * parsers.html_non_newline_attribute^0
                          * parsers.html_tag_end

parsers.html_open_tag = parsers.html_tag_start
                      * parsers.block_keyword
                      * parsers.html_attribute^0
                      * parsers.html_tag_end

parsers.html_open_special_tag = parsers.html_tag_start
                              * parsers.special_block_keyword
                              * parsers.html_attribute^0
                              * parsers.html_tag_end

-- incomplete tags
parsers.incomplete_tag_following  = parsers.spacechar
                                  + parsers.more
                                  + parsers.slash * parsers.more
                                  + #(parsers.newline + parsers.eof)

parsers.incomplete_special_tag_following = parsers.spacechar
                                         + parsers.more
                                         + #( parsers.newline
                                            + parsers.eof)

parsers.html_incomplete_open_tag  = parsers.html_tag_start
                                  * parsers.block_keyword
                                  * parsers.incomplete_tag_following

parsers.html_incomplete_open_special_tag
  = parsers.html_tag_start
  * parsers.special_block_keyword
  * parsers.incomplete_special_tag_following

parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
                                  * parsers.block_keyword
                                  * parsers.incomplete_tag_following

parsers.html_incomplete_close_special_tag
  = parsers.html_tag_closing_start
  * parsers.special_block_keyword
  * parsers.incomplete_tag_following
```

```
7981
7982 -- closing tags
7983 parsers.html_close_tag  = parsers.html_tag_closing_start
7984                        * parsers.block_keyword
7985                        * parsers.html_tag_end
7986
7987 parsers.html_any_close_tag  = parsers.html_tag_closing_start
7988                             * parsers.keyword
7989                             * parsers.html_tag_end
7990
7991 parsers.html_close_special_tag = parsers.html_tag_closing_start
7992                                * parsers.special_block_keyword
7993                                * parsers.html_tag_end
7994
7995 -- empty tags
7996 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7997                                   * parsers.keyword
7998                                   * parsers.html_attribute^0
7999                                   * parsers.html_empty_tag_end
8000
8001 parsers.html_any_empty_tag  = parsers.html_tag_start
8002                             * parsers.keyword
8003                             * parsers.html_non_newline_attribute^0
8004                             * parsers.optionalspace
8005                             * parsers.slash
8006                             * parsers.more
8007
8008 parsers.html_empty_tag  = parsers.html_tag_start
8009                        * parsers.block_keyword
8010                        * parsers.html_attribute^0
8011                        * parsers.html_empty_tag_end
8012
8013 parsers.html_empty_special_tag  = parsers.html_tag_start
8014                                 * parsers.special_block_keyword
8015                                 * parsers.html_attribute^0
8016                                 * parsers.html_empty_tag_end
8017
8018 parsers.html_incomplete_blocks
8019   = parsers.html_incomplete_open_tag
8020   + parsers.html_incomplete_open_special_tag
8021   + parsers.html_incomplete_close_tag
8022
8023 -- parse special html blocks
8024 parsers.html_blankline_ending_special_block_opening
8025   = ( parsers.html_close_special_tag
8026     + parsers.html_empty_special_tag)
8027   * #( parsers.optionalspace
```

```
8028      * (parsers.newline + parsers.eof))
8029
8030 parsers.html_blankline_ending_special_block
8031   = parsers.html_blankline_ending_special_block_opening
8032   * parsers.html_blankline_end_condition
8033
8034 parsers.html_special_block_opening
8035   = parsers.html_incomplete_open_special_tag
8036   - parsers.html_empty_special_tag
8037
8038 parsers.html_closing_special_block
8039   = parsers.html_special_block_opening
8040   * parsers.html_until_end(parsers.html_close_special_tag)
8041
8042 parsers.html_special_block
8043   = parsers.html_blankline_ending_special_block
8044   + parsers.html_closing_special_block
8045
8046 -- parse html blocks
8047 parsers.html_block_opening  = parsers.html_incomplete_open_tag
8048                             + parsers.html_incomplete_close_tag
8049
8050 parsers.html_block  = parsers.html_block_opening
8051                     * parsers.html_blankline_end_condition
8052
8053 -- parse any html blocks
8054 parsers.html_any_block_opening
8055   = ( parsers.html_any_open_tag
8056     + parsers.html_any_close_tag
8057     + parsers.html_any_empty_tag)
8058   * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8059
8060 parsers.html_any_block  = parsers.html_any_block_opening
8061                         * parsers.html_blankline_end_condition
8062
8063 parsers.html_inline_comment_full  = parsers.html_comment_start
8064                                   * -P(">") * -P("->")
8065                                   * Cs(( V("NoSoftLineBreakEndline")
8066                                        + parsers.any - P("--")
8067                                        - parsers.nested_breaking_blank
8068                                        - parsers.html_comment_end)^0)
8069                                   * parsers.html_comment_end
8070
8071 parsers.html_inline_tags  = parsers.html_inline_comment_full
8072                           + parsers.html_any_empty_inline_tag
8073                           + parsers.html_inline_instruction
8074                           + parsers.html_inline_cdatasection
```

```
                            + parsers.html_inline_declaration
                            + parsers.html_any_open_inline_tag
                            + parsers.html_any_close_tag

```

#### 3.1.5.6 Parsers Used for Markdown Tags and Links

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

parsers.auto_link_scheme_part = parsers.alphanumeric
                              + parsers.plus
                              + parsers.period
                              + parsers.dash

parsers.auto_link_scheme  = parsers.letter
                          * parsers.auto_link_scheme_part
                          * parsers.auto_link_scheme_part^-30

parsers.absolute_uri  = parsers.auto_link_scheme * parsers.colon
                      * ( parsers.any - parsers.spacing
                        - parsers.less - parsers.more)^0

parsers.printable_characters = S(".!#$%&'*+/=?^_`{|}~-")

parsers.email_address_local_part_char = parsers.alphanumeric
                                      + parsers.printable_characters

parsers.email_address_local_part
  = parsers.email_address_local_part_char^1

parsers.email_address_dns_label = parsers.alphanumeric
                                * ( parsers.alphanumeric
                                  + parsers.dash)^-62
                                * B(parsers.alphanumeric)

parsers.email_address_domain  = parsers.email_address_dns_label
                              * ( parsers.period
                                * parsers.email_address_dns_label)^0

parsers.email_address = parsers.email_address_local_part
                      * parsers.at
                      * parsers.email_address_domain

parsers.auto_link_url = parsers.less
                      * C(parsers.absolute_uri)
```

```
                      * parsers.more

parsers.auto_link_email = parsers.less
                        * C(parsers.email_address)
                        * parsers.more

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

parsers.autolink  = parsers.auto_link_url
                  + parsers.auto_link_email

-- 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 }

parsers.link_text  = parsers.lbracket
                   * Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + parsers.autolink
                       + V("InlineHtml")
```

```
                          + ( parsers.backslash * parsers.backslash)
                          + ( parsers.backslash
                            * ( parsers.lbracket
                              + parsers.rbracket)
                            + V("NoSoftLineBreakSpace")
                            + V("NoSoftLineBreakEndline")
                            + (parsers.any
                              - ( parsers.newline
                                + parsers.lbracket
                                + parsers.rbracket
                                + parsers.blankline^2))))^0)
                      * parsers.rbracket

parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
                          * #( ( parsers.any
                               - parsers.rbracket)^-999
                             * parsers.rbracket)
                          * Cs((parsers.alphanumeric^1
                              + parsers.inticks
                              + parsers.autolink
                              + V("InlineHtml")
                              + ( parsers.backslash * parsers.backslash)
                              + ( parsers.backslash
                                * ( parsers.lbracket
                                  + parsers.rbracket)
                                + V("NoSoftLineBreakSpace")
                                + V("NoSoftLineBreakEndline")
                                + (parsers.any
                                  - ( parsers.newline
                                    + parsers.lbracket
                                    + parsers.rbracket
                                    + parsers.blankline^2))))^1)

parsers.link_label  = parsers.lbracket
                      * parsers.link_label_body
                      * parsers.rbracket

parsers.inparens_url  = P{ parsers.lparent
                           * ((parsers.anyescaped - (parsers.lparent
                                                   + parsers.rparent
                                                   + parsers.spacing)
                              ) + V(1))^0
                           * parsers.rparent }

-- url for markdown links, allowing nested brackets:
parsers.url         = parsers.less * Cs((parsers.anyescaped
                                       - parsers.newline
```

```
                                        - parsers.less
                                        - parsers.more)^0)
                                   * parsers.more
                     + -parsers.less
                     * Cs((parsers.inparens_url + (parsers.anyescaped
                                                   - parsers.spacing
                                                   - parsers.lparent
                                                   - parsers.rparent))^1)

-- quoted text:
parsers.title_s     = parsers.squote
                     * Cs((parsers.html_entities
                          + V("NoSoftLineBreakSpace")
                          + V("NoSoftLineBreakEndline")
                          + ( parsers.anyescaped
                            - parsers.newline
                            - parsers.squote
                            - parsers.blankline^2))^0)
                     * parsers.squote

parsers.title_d     = parsers.dquote
                     * Cs((parsers.html_entities
                          + V("NoSoftLineBreakSpace")
                          + V("NoSoftLineBreakEndline")
                          + ( parsers.anyescaped
                            - parsers.newline
                            - parsers.dquote
                            - parsers.blankline^2))^0)
                     * parsers.dquote

parsers.title_p     = parsers.lparent
                     * Cs((parsers.html_entities
                          + V("NoSoftLineBreakSpace")
                          + V("NoSoftLineBreakEndline")
                          + ( parsers.anyescaped
                            - parsers.newline
                            - parsers.lparent
                            - parsers.rparent
                            - parsers.blankline^2))^0)
                     * parsers.rparent

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

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

```

#### 3.1.5.7 Helpers for Links and Link Reference Definitions

```
-- parse a reference definition:  [foo]: /bar "title"
parsers.define_reference_parser = (parsers.check_trail / "")
                                * parsers.link_label * parsers.colon
                                * parsers.spnlc * parsers.url
                                * ( parsers.spnlc_sep * parsers.title
                                  * parsers.only_blank
                                  + Cc("") * parsers.only_blank)
```

#### 3.1.5.8 Inline Elements

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

-- 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

```

#### 3.1.5.9 Block Elements

```
parsers.lineof = function(c)
    return ( parsers.check_trail_no_rem
           * (P(c) * parsers.optionalspace)^3
           * (parsers.newline + parsers.eof))
end

parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
                             + parsers.lineof(parsers.dash)
                             + parsers.lineof(parsers.underscore)
```

#### 3.1.5.10 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.nonindentspace * parsers.equal^1
  * parsers.optionalspace * #parsers.newline * Cc(1)
  + parsers.nonindentspace * parsers.dash^1
  * parsers.optionalspace * #parsers.newline * Cc(2)

```

```
local function strip_atx_end(s)
  return s:gsub("%s+#*%s*\n$","")
end

parsers.atx_heading = parsers.check_trail_no_rem
                    * Cg(parsers.heading_start, "level")
                    * (C( parsers.optionalspace
                        * parsers.hash^0
                        * parsers.optionalspace
                        * parsers.newline)
                      + parsers.spacechar^1
                      * C(parsers.line))
```

### 3.1.6 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

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

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these ⟨*member*⟩s as `reader->`⟨*member*⟩.

```
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.

```
8323  local parsers = self.parsers
```

#### 3.1.6.1 Top-Level Helper Functions

Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
8324  function self.normalize_tag(tag)
8325    tag = util.rope_to_string(tag)
8326    tag = tag:gsub("[ \n\r\t]+", " ")
8327    tag = tag:gsub("^ ", ""):gsub(" $", "")
8328    tag = uni_algos.case.casefold(tag, true, false)
8329    return tag
8330  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.

```
8331  local function iterlines(s, f)
8332    local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8333    return util.rope_to_string(rope)
8334  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.

```
8335  if options.preserveTabs then
8336    self.expandtabs = function(s) return s end
8337  else
8338    self.expandtabs = function(s)
8339                        if s:find("\t") then
8340                          return iterlines(s, util.expand_tabs_in_line)
8341                        else
8342                          return s
8343                        end
8344                      end
8345  end
```

#### 3.1.6.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
8346  self.parser_functions = {}
```

```
  self.create_parser = function(name, grammar, toplevel)
    self.parser_functions[name] = function(str)
```

If the parser function is top-level and the `stripIndent` Lua option is enabled, we will first expand tabs in the input string `str` into spaces and then we will count the minimum indent across all lines, skipping blank lines. Next, we will remove the minimum indent from all lines.

```
      if toplevel and options.stripIndent then
          local min_prefix_length, min_prefix = nil, ''
          str = iterlines(str, function(line)
              if lpeg.match(parsers.nonemptyline, line) == nil then
                  return line
              end
              line = util.expand_tabs_in_line(line)
              local prefix = lpeg.match(C(parsers.optionalspace), line)
              local prefix_length = #prefix
              local is_shorter = min_prefix_length == nil
              if not is_shorter then
                is_shorter = prefix_length < min_prefix_length
              end
              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
        return writer.error(
          format("Parser `%s` failed to process the input text.", name),
          format("Here are the first 20 characters of the remaining "
              .. "unprocessed text: `%s`.", str:sub(1,20))
        )
      else
        return res
      end
    end
  end
```

```
8385
8386  self.create_parser("parse_blocks",
8387                     function()
8388                       return parsers.blocks
8389                     end, true)
8390
8391  self.create_parser("parse_blocks_nested",
8392                     function()
8393                       return parsers.blocks_nested
8394                     end, false)
8395
8396  self.create_parser("parse_inlines",
8397                     function()
8398                       return parsers.inlines
8399                     end, false)
8400
8401  self.create_parser("parse_inlines_no_inline_note",
8402                     function()
8403                       return parsers.inlines_no_inline_note
8404                     end, false)
8405
8406  self.create_parser("parse_inlines_no_html",
8407                     function()
8408                       return parsers.inlines_no_html
8409                     end, false)
8410
8411  self.create_parser("parse_inlines_nbsp",
8412                     function()
8413                       return parsers.inlines_nbsp
8414                     end, false)
8415  self.create_parser("parse_inlines_no_link_or_emphasis",
8416                    function()
8417                      return parsers.inlines_no_link_or_emphasis
8418                    end, false)
```

#### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
8419  parsers.minimally_indented_blankline
8420    = parsers.check_minimal_indent * (parsers.blankline / "")
8421
8422  parsers.minimally_indented_block
8423    = parsers.check_minimal_indent * V("Block")
8424
8425  parsers.minimally_indented_block_or_paragraph
8426    = parsers.check_minimal_indent * V("BlockOrParagraph")
8427
```

```
parsers.minimally_indented_paragraph
  = parsers.check_minimal_indent * V("Paragraph")

parsers.minimally_indented_plain
  = parsers.check_minimal_indent * V("Plain")

parsers.minimally_indented_par_or_plain
  = parsers.minimally_indented_paragraph
  + parsers.minimally_indented_plain

parsers.minimally_indented_par_or_plain_no_blank
  = parsers.minimally_indented_par_or_plain
  - parsers.minimally_indented_blankline

parsers.minimally_indented_ref
  = parsers.check_minimal_indent * V("Reference")

parsers.minimally_indented_blank
  = parsers.check_minimal_indent * V("Blank")

parsers.conditionally_indented_blankline
  = parsers.check_minimal_blank_indent * (parsers.blankline / "")

parsers.minimally_indented_ref_or_block
  = parsers.minimally_indented_ref
  + parsers.minimally_indented_block
  - parsers.minimally_indented_blankline

parsers.minimally_indented_ref_or_block_or_par
  = parsers.minimally_indented_ref
  + parsers.minimally_indented_block_or_paragraph
  - parsers.minimally_indented_blankline

```

The following pattern parses the properly indented content that follows the initial container start.

```

function parsers.separator_loop(separated_block, paragraph,
                                block_separator, paragraph_separator)
  return  separated_block
        + block_separator
          * paragraph
          * separated_block
        + paragraph_separator
        * paragraph
end

```

```
function parsers.create_loop_body_pair(separated_block, paragraph,
                                       block_separator,
                                       paragraph_separator)
  return {
    block = parsers.separator_loop(separated_block, paragraph,
                                   block_separator, block_separator),
    par = parsers.separator_loop(separated_block, paragraph,
                                 block_separator, paragraph_separator)
  }
end

parsers.block_sep_group = function(blank)
  return  blank^0 * parsers.eof
        + ( blank^2 / writer.paragraphsep
          + blank^0 / writer.interblocksep
          )
end

parsers.par_sep_group = function(blank)
  return  blank^0 * parsers.eof
        + blank^0 / writer.paragraphsep
end

parsers.sep_group_no_output = function(blank)
  return  blank^0 * parsers.eof
        + blank^0
end

parsers.content_blank = parsers.minimally_indented_blankline

parsers.ref_or_block_separated
  = parsers.sep_group_no_output(parsers.content_blank)
  * ( parsers.minimally_indented_ref
    - parsers.content_blank)
  + parsers.block_sep_group(parsers.content_blank)
  * ( parsers.minimally_indented_block
    - parsers.content_blank)

parsers.loop_body_pair  =
  parsers.create_loop_body_pair(
    parsers.ref_or_block_separated,
    parsers.minimally_indented_par_or_plain_no_blank,
    parsers.block_sep_group(parsers.content_blank),
    parsers.par_sep_group(parsers.content_blank))

parsers.content_loop  = ( V("Block")
                        * parsers.loop_body_pair.block^0
```

```
                          + (V("Paragraph") + V("Plain"))
                          * parsers.ref_or_block_separated
                          * parsers.loop_body_pair.block^0
                          + (V("Paragraph") + V("Plain"))
                          * parsers.loop_body_pair.par^0)
                        * parsers.content_blank^0

parsers.indented_content = function()
  return  Ct( (V("Reference") + (parsers.blankline / ""))
            * parsers.content_blank^0
            * parsers.check_minimal_indent
            * parsers.content_loop
            + (V("Reference") + (parsers.blankline / ""))
            * parsers.content_blank^0
            + parsers.content_loop)
end

parsers.add_indent = function(pattern, name, breakable)
  return  Cg(Cmt( Cb("indent_info")
                * Ct(pattern)
                * ( #parsers.linechar  -- check if starter is blank
                  * Cc(false) + Cc(true))
                * Cc(name)
                * Cc(breakable),
              process_starter_indent), "indent_info")
end

```

#### 3.1.6.4 Parsers Used for Markdown Lists (local)

```
if options.hashEnumerators then
  parsers.dig = parsers.digit + parsers.hash
else
  parsers.dig = parsers.digit
end

parsers.enumerator = function(delimiter_type, interrupting)
  local delimiter_range
  local allowed_end
  if interrupting then
    delimiter_range = P("1")
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
    delimiter_range = parsers.dig * parsers.dig^-8
    allowed_end = C(parsers.spacechar^1)
                + #(parsers.newline + parsers.eof)
  end
```

```
8563
8564      return parsers.check_trail
8565               * Ct(C(delimiter_range) * C(delimiter_type))
8566               * allowed_end
8567    end
8568
8569    parsers.starter = parsers.bullet(parsers.dash)
8570                    + parsers.bullet(parsers.asterisk)
8571                    + parsers.bullet(parsers.plus)
8572                    + parsers.enumerator(parsers.period)
8573                    + parsers.enumerator(parsers.rparent)
8574
```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```
8575    parsers.blockquote_start
8576      = parsers.check_trail
8577      * C(parsers.more)
8578      * C(parsers.spacechar^0)
8579
8580    parsers.blockquote_body
8581      = parsers.add_indent(parsers.blockquote_start, "bq", true)
8582      * parsers.indented_content()
8583      * remove_indent("bq")
8584
8585    if not options.breakableBlockquotes then
8586      parsers.blockquote_body
8587        = parsers.add_indent(parsers.blockquote_start, "bq", false)
8588        * parsers.indented_content()
8589        * remove_indent("bq")
8590    end
```

#### 3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```
8591    local function parse_content_part(content_part)
8592      local rope = util.rope_to_string(content_part)
8593      local parsed
8594        = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8595      parsed.indent_info = nil
8596      return parsed
8597    end
8598
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8599    local collect_emphasis_content =
```

```
    function(t, opening_index, closing_index)
      local content = {}

      local content_part = {}
      for i = opening_index, closing_index do
        local value = t[i]

        if value.rendered ~= nil then
          content[#content + 1] = parse_content_part(content_part)
          content_part = {}
          content[#content + 1] = value.rendered
          value.rendered = nil
        else
          if value.type == "delimiter"
              and value.element == "emphasis" then
            if value.is_active then
              content_part[#content_part + 1]
                = string.rep(value.character, value.current_count)
            end
          else
            content_part[#content_part + 1] = value.content
          end
          value.content = ''
          value.is_active = false
        end
      end

      if next(content_part) ~= nil then
        content[#content + 1] = parse_content_part(content_part)
      end

      return content
    end

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```
  local function fill_emph(t, opening_index, closing_index)
    local content
      = collect_emphasis_content(t, opening_index + 1,
                                 closing_index - 1)
    t[opening_index + 1].is_active = true
    t[opening_index + 1].rendered = writer.emphasis(content)
  end

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
8642  local function fill_strong(t, opening_index, closing_index)
8643    local content
8644      = collect_emphasis_content(t, opening_index + 1,
8645                                 closing_index - 1)
8646    t[opening_index + 1].is_active = true
8647    t[opening_index + 1].rendered = writer.strong(content)
8648  end
8649
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
8650  local function breaks_three_rule(opening_delimiter, closing_delimiter)
8651    return ( opening_delimiter.is_closing
8652          or closing_delimiter.is_opening)
8653       and (( opening_delimiter.original_count
8654            + closing_delimiter.original_count) % 3 == 0)
8655       and ( opening_delimiter.original_count % 3 ~= 0
8656          or closing_delimiter.original_count % 3 ~= 0)
8657  end
8658
```

Look for the first potential emphasis opener in the delimiter table `t` in the range from `bottom_index` to `latest_index` that has the same character `character` as the closing delimiter `closing_delimiter`.

```
8659  local find_emphasis_opener = function(t, bottom_index, latest_index,
8660                                        character, closing_delimiter)
8661    for i = latest_index, bottom_index, -1 do
8662      local value = t[i]
8663      if value.is_active and
8664         value.is_opening and
8665         value.type == "delimiter" and
8666         value.element == "emphasis" and
8667         (value.character == character) and
8668         (value.current_count > 0) then
8669        if not breaks_three_rule(value, closing_delimiter) then
8670          return i
8671        end
8672      end
8673    end
8674  end
8675
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```
8676  local function process_emphasis(t, opening_index, closing_index)
8677    for i = opening_index, closing_index do
8678      local value = t[i]
8679      if value.type == "delimiter" and value.element == "emphasis" then
```

```
        local delimiter_length = string.len(value.content)
        value.character = string.sub(value.content, 1, 1)
        value.current_count = delimiter_length
        value.original_count = delimiter_length
    end
  end

  local openers_bottom = {
    ['*'] = {
      [true] = {opening_index, opening_index, opening_index},
      [false] = {opening_index, opening_index, opening_index}
    },
    ['_'] = {
      [true] = {opening_index, opening_index, opening_index},
      [false] = {opening_index, opening_index, opening_index}
    }
  }

  local current_position = opening_index
  local max_position = closing_index

  while current_position <= max_position do
    local value = t[current_position]

    if value.type ~= "delimiter" or
      value.element ~= "emphasis" or
      not value.is_active or
      not value.is_closing or
      (value.current_count <= 0) then
      current_position = current_position + 1
      goto continue
    end

    local character = value.character
    local is_opening = value.is_opening
    local closing_length_modulo_three = value.original_count % 3

    local current_openers_bottom
      = openers_bottom[character][is_opening]
                      [closing_length_modulo_three + 1]

    local opener_position
      = find_emphasis_opener(t, current_openers_bottom,
                             current_position - 1, character, value)

    if (opener_position == nil) then
      openers_bottom[character][is_opening]
```

```
                      [closing_length_modulo_three + 1]
          = current_position
        current_position = current_position + 1
        goto continue
      end

      local opening_delimiter = t[opener_position]

      local current_opening_count = opening_delimiter.current_count
      local current_closing_count = t[current_position].current_count

      if (current_opening_count >= 2)
        and (current_closing_count >= 2) then
        opening_delimiter.current_count = current_opening_count - 2
        t[current_position].current_count = current_closing_count - 2
        fill_strong(t, opener_position, current_position)
      else
        opening_delimiter.current_count = current_opening_count - 1
        t[current_position].current_count = current_closing_count - 1
        fill_emph(t, opener_position, current_position)
      end

      ::continue::
    end
  end

  local cont = lpeg.R("\128\191") -- continuation byte

```

Match a UTF-8 character of byte length `n`.

```
  local function utf8_by_byte_count(n)
    if (n == 1) then
      return lpeg.R("\0\127")
    end
    if (n == 2) then
      return lpeg.R("\194\223") * cont
    end
    if (n == 3) then
      return lpeg.R("\224\239") * cont * cont
    end
    if (n == 4) then
      return lpeg.R("\240\244") * cont * cont * cont
    end
  end
```

Check if a there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```
  local function check_unicode_type(s, i, start_pos, end_pos, chartype)
```

```
8770    local c
8771    local char_length
8772    for pos = start_pos, end_pos, 1 do
8773      if (start_pos < 0) then
8774        char_length = -pos
8775      else
8776        char_length = pos + 1
8777      end
8778
8779      if (chartype == "punctuation") then
8780        if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8781          return i
8782        end
8783      else
8784        c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
8785        if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8786          return i
8787        end
8788      end
8789    end
8790  end
8791
8792  local function check_preceding_unicode_punctuation(s, i)
8793    return check_unicode_type(s, i, -4, -1, "punctuation")
8794  end
8795
8796  local function check_preceding_unicode_whitespace(s, i)
8797    return check_unicode_type(s, i, -4, -1, "%s")
8798  end
8799
8800  local function check_following_unicode_punctuation(s, i)
8801    return check_unicode_type(s, i, 0, 3, "punctuation")
8802  end
8803
8804  local function check_following_unicode_whitespace(s, i)
8805    return check_unicode_type(s, i, 0, 3, "%s")
8806  end
8807
8808  parsers.unicode_preceding_punctuation
8809    = B(parsers.escapable)
8810    + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8811
8812  parsers.unicode_preceding_whitespace
8813    = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8814
8815  parsers.unicode_following_punctuation
8816    = #parsers.escapable
```

```
8817    + Cmt(parsers.succeed, check_following_unicode_punctuation)
8818
8819  parsers.unicode_following_whitespace
8820    = Cmt(parsers.succeed, check_following_unicode_whitespace)
8821
8822  parsers.delimiter_run = function(character)
8823    return  (B(parsers.backslash * character) + -B(character))
8824          * character^1
8825          * -#character
8826  end
8827
8828  parsers.left_flanking_delimiter_run = function(character)
8829    return  (B( parsers.any)
8830              * ( parsers.unicode_preceding_punctuation
8831                + parsers.unicode_preceding_whitespace)
8832             + -B(parsers.any))
8833            * parsers.delimiter_run(character)
8834            * parsers.unicode_following_punctuation
8835            + parsers.delimiter_run(character)
8836            * -#( parsers.unicode_following_punctuation
8837                + parsers.unicode_following_whitespace
8838                + parsers.eof)
8839  end
8840
8841  parsers.right_flanking_delimiter_run = function(character)
8842    return  parsers.unicode_preceding_punctuation
8843          * parsers.delimiter_run(character)
8844          * ( parsers.unicode_following_punctuation
8845            + parsers.unicode_following_whitespace
8846            + parsers.eof)
8847          + (B(parsers.any)
8848            * -( parsers.unicode_preceding_punctuation
8849               + parsers.unicode_preceding_whitespace))
8850          * parsers.delimiter_run(character)
8851  end
8852
8853  if options.underscores then
8854    parsers.emph_start
8855      = parsers.left_flanking_delimiter_run(parsers.asterisk)
8856      + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
8857        + ( parsers.unicode_preceding_punctuation
8858          * #parsers.right_flanking_delimiter_run(parsers.underscore)))
8859      * parsers.left_flanking_delimiter_run(parsers.underscore)
8860
8861    parsers.emph_end
8862      = parsers.right_flanking_delimiter_run(parsers.asterisk)
8863      + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
```

```
      + #( parsers.left_flanking_delimiter_run(parsers.underscore)
         * parsers.unicode_following_punctuation))
    * parsers.right_flanking_delimiter_run(parsers.underscore)
else
  parsers.emph_start
    = parsers.left_flanking_delimiter_run(parsers.asterisk)

  parsers.emph_end
    = parsers.right_flanking_delimiter_run(parsers.asterisk)
end

parsers.emph_capturing_open_and_close
  = #parsers.emph_start * #parsers.emph_end
  * Ct( Cg(Cc("delimiter"), "type")
      * Cg(Cc("emphasis"), "element")
      * Cg(C(parsers.emph_start), "content")
      * Cg(Cc(true), "is_opening")
      * Cg(Cc(true), "is_closing"))

parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
                                * Cg(Cc("emphasis"), "element")
                                * Cg(C(parsers.emph_start), "content")
                                * Cg(Cc(true), "is_opening")
                                * Cg(Cc(false), "is_closing"))

parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
                                 * Cg(Cc("emphasis"), "element")
                                 * Cg(C(parsers.emph_end), "content")
                                 * Cg(Cc(false), "is_opening")
                                 * Cg(Cc(true), "is_closing"))

parsers.emph_open_or_close  = parsers.emph_capturing_open_and_close
                            + parsers.emph_capturing_open
                            + parsers.emph_capturing_close

parsers.emph_open = parsers.emph_capturing_open_and_close
                  + parsers.emph_capturing_open

parsers.emph_close  = parsers.emph_capturing_open_and_close
                    + parsers.emph_capturing_close
```

#### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

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

```
8908  -- List of note references defined in the document
8909  parsers.rawnotes = {}
8910
```

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.

```
8911  function self.register_link(_, tag, url, title,
8912                              attributes)
8913    local normalized_tag = self.normalize_tag(tag)
8914      if references[normalized_tag] == nil then
8915        references[normalized_tag] = {
8916          url = url,
8917          title = title,
8918          attributes = attributes
8919        }
8920      end
8921    return ""
8922  end
8923
```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
8924  function self.lookup_reference(tag)
8925    return references[self.normalize_tag(tag)]
8926  end
8927
```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
8928  function self.lookup_note_reference(tag)
8929    return parsers.rawnotes[self.normalize_tag(tag)]
8930  end
8931
8932  parsers.title_s_direct_ref  = parsers.squote
8933                              * Cs((parsers.html_entities
8934                                  + ( parsers.anyescaped
8935                                    - parsers.squote
8936                                    - parsers.blankline^2))^0)
8937                              * parsers.squote
8938
8939  parsers.title_d_direct_ref  = parsers.dquote
8940                              * Cs((parsers.html_entities
8941                                  + ( parsers.anyescaped
8942                                    - parsers.dquote
8943                                    - parsers.blankline^2))^0)
8944                              * parsers.dquote
8945
8946  parsers.title_p_direct_ref  = parsers.lparent
```

```
                                * Cs((parsers.html_entities
                                    + ( parsers.anyescaped
                                      - parsers.lparent
                                      - parsers.rparent
                                      - parsers.blankline^2))^0)
                                * parsers.rparent

parsers.title_direct_ref  = parsers.title_s_direct_ref
                          + parsers.title_d_direct_ref
                          + parsers.title_p_direct_ref

parsers.inline_direct_ref_inside  = parsers.lparent * parsers.spnl
                                  * Cg(parsers.url + Cc(""), "url")
                                  * parsers.spnl
                                  * Cg( parsers.title_direct_ref
                                      + Cc(""), "title")
                                  * parsers.spnl * parsers.rparent

parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
                          * Cg(parsers.url + Cc(""), "url")
                          * parsers.spnlc
                          * Cg(parsers.title + Cc(""), "title")
                          * parsers.spnlc * parsers.rparent

parsers.empty_link  = parsers.lbracket
                    * parsers.rbracket

parsers.inline_link = parsers.link_text
                    * parsers.inline_direct_ref

parsers.full_link = parsers.link_text
                  * parsers.link_label

parsers.shortcut_link = parsers.link_label
                      * -(parsers.empty_link + parsers.link_label)

parsers.collapsed_link  = parsers.link_label
                        * parsers.empty_link

parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
                      * Cg(Cc("inline"), "link_type")
                      + #(parsers.exclamation * parsers.full_link)
                      * Cg(Cc("full"), "link_type")
                      + #( parsers.exclamation
                         * parsers.collapsed_link)
                      * Cg(Cc("collapsed"), "link_type")
                      + #(parsers.exclamation * parsers.shortcut_link)
```

```
                        * Cg(Cc("shortcut"), "link_type")
                        + #(parsers.exclamation * parsers.empty_link)
                        * Cg(Cc("empty"), "link_type")

parsers.link_opening  = #parsers.inline_link
                        * Cg(Cc("inline"), "link_type")
                        + #parsers.full_link
                        * Cg(Cc("full"), "link_type")
                        + #parsers.collapsed_link
                        * Cg(Cc("collapsed"), "link_type")
                        + #parsers.shortcut_link
                        * Cg(Cc("shortcut"), "link_type")
                        + #parsers.empty_link
                        * Cg(Cc("empty_link"), "link_type")
                        + #parsers.link_text
                        * Cg(Cc("link_text"), "link_type")

parsers.note_opening  = #(parsers.circumflex * parsers.link_text)
                        * Cg(Cc("note_inline"), "link_type")

parsers.raw_note_opening  = #( parsers.lbracket
                             * parsers.circumflex
                             * parsers.link_label_body
                             * parsers.rbracket)
                           * Cg(Cc("raw_note"), "link_type")

local inline_note_element = Cg(Cc("note"), "element")
                            * parsers.note_opening
                            * Cg( parsers.circumflex
                                * parsers.lbracket, "content")

local image_element = Cg(Cc("image"), "element")
                      * parsers.image_opening
                      * Cg( parsers.exclamation
                          * parsers.lbracket, "content")

local note_element  = Cg(Cc("note"), "element")
                      * parsers.raw_note_opening
                      * Cg( parsers.lbracket
                          * parsers.circumflex, "content")

local link_element  = Cg(Cc("link"), "element")
                      * parsers.link_opening
                      * Cg(parsers.lbracket, "content")

local opening_elements = parsers.fail

```

```
9041  if options.inlineNotes then
9042    opening_elements = opening_elements + inline_note_element
9043  end
9044
9045  opening_elements = opening_elements + image_element
9046
9047  if options.notes then
9048    opening_elements = opening_elements + note_element
9049  end
9050
9051  opening_elements = opening_elements + link_element
9052
9053  parsers.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
9054                                  * Cg(Cc(true), "is_opening")
9055                                  * Cg(Cc(false), "is_closing")
9056                                  * opening_elements)
9057
9058  parsers.link_image_closing  = Ct( Cg(Cc("delimiter"), "type")
9059                                  * Cg(Cc("link"), "element")
9060                                  * Cg(Cc(false), "is_opening")
9061                                  * Cg(Cc(true), "is_closing")
9062                                  * ( Cg(Cc(true), "is_direct")
9063                                    * Cg( parsers.rbracket
9064                                        * #parsers.inline_direct_ref,
9065                                         "content")
9066                                    + Cg(Cc(false), "is_direct")
9067                                    * Cg(parsers.rbracket, "content")))
9068
9069  parsers.link_image_open_or_close  = parsers.link_image_opening
9070                                    + parsers.link_image_closing
9071
9072  if options.html then
9073    parsers.link_emph_precedence  = parsers.inticks
9074                                  + parsers.autolink
9075                                  + parsers.html_inline_tags
9076  else
9077    parsers.link_emph_precedence  = parsers.inticks
9078                                  + parsers.autolink
9079  end
9080
9081  parsers.link_and_emph_endline = parsers.newline
9082                                * ((parsers.check_minimal_indent
9083                                  * -V("EndlineExceptions")
9084                                  + parsers.check_optional_indent
9085                                  * -V("EndlineExceptions")
9086                                  * -V("ListStarter")) / "")
9087                                * parsers.spacechar^0 / "\n"
```

```
9088
9089  parsers.link_and_emph_content
9090    = Ct( Cg(Cc("content"), "type")
9091        * Cg(Cs(( parsers.link_emph_precedence
9092                + parsers.backslash * parsers.linechar
9093                + parsers.link_and_emph_endline
9094                + (parsers.linechar
9095                  - parsers.blankline^2
9096                  - parsers.link_image_open_or_close
9097                  - parsers.emph_open_or_close))^0), "content"))
9098
9099  parsers.link_and_emph_table
9100    = (parsers.link_image_opening + parsers.emph_open)
9101    * parsers.link_and_emph_content
9102    * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9103      * parsers.link_and_emph_content)^1
9104
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
9105  local function collect_link_content(t, opening_index, closing_index)
9106    local content = {}
9107    for i = opening_index, closing_index do
9108      content[#content + 1] = t[i].content
9109    end
9110    return util.rope_to_string(content)
9111  end
9112
```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
9113  local function find_link_opener(t, bottom_index, latest_index)
9114    for i = latest_index, bottom_index, -1 do
9115      local value = t[i]
9116      if value.type == "delimiter" and
9117         value.is_opening and
9118         ( value.element == "link"
9119        or value.element == "image"
9120        or value.element == "note")
9121         and not value.removed then
9122        if value.is_active then
9123          return i
9124        end
9125        value.removed = true
9126        return nil
9127      end
9128    end
9129  end
```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
local function find_next_link_closing_index(t, latest_index)
  for i = latest_index, #t do
    local value = t[i]
    if value.is_closing and
       value.element == "link" and
       not value.removed then
      return i
    end
  end
end
```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```
local function disable_previous_link_openers(t, opening_index)
  if t[opening_index].element == "image" then
    return
  end

  for i = opening_index, 1, -1 do
    local value = t[i]
    if value.is_active and
       value.type == "delimiter" and
       value.is_opening and
       value.element == "link" then
      value.is_active = false
    end
  end
end
```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
local function disable_range(t, opening_index, closing_index)
  for i = opening_index, closing_index do
    local value = t[i]
    if value.is_active then
      value.is_active = false
      if value.type == "delimiter" then
        value.removed = true
      end
    end
  end
end
```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
local delete_parsed_content_in_range =
  function(t, opening_index, closing_index)
    for i = opening_index, closing_index do
      t[i].rendered = nil
    end
  end

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
local function empty_content_in_range(t, opening_index, closing_index)
  for i = opening_index, closing_index do
    t[i].content = ''
  end
end

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
local function join_attributes(reference_attributes, own_attributes)
  local merged_attributes = {}
  for _, attribute in ipairs(reference_attributes or {}) do
    table.insert(merged_attributes, attribute)
  end
  for _, attribute in ipairs(own_attributes or {}) do
    table.insert(merged_attributes, attribute)
  end
  if next(merged_attributes) == nil then
    merged_attributes = nil
  end
  return merged_attributes
end

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
local render_link_or_image =
  function(t, opening_index, closing_index, content_end_index,
           reference)
    process_emphasis(t, opening_index, content_end_index)
    local mapped = collect_emphasis_content(t, opening_index + 1,
                                            content_end_index - 1)

    local rendered = {}
```

```
9205        if (t[opening_index].element == "link") then
9206          rendered = writer.link(mapped, reference.url,
9207                                 reference.title, reference.attributes)
9208        end
9209
9210        if (t[opening_index].element == "image") then
9211          rendered = writer.image(mapped, reference.url, reference.title,
9212                                  reference.attributes)
9213        end
9214
9215        if (t[opening_index].element == "note") then
9216          if (t[opening_index].link_type == "note_inline") then
9217            rendered = writer.note(mapped)
9218          end
9219          if (t[opening_index].link_type == "raw_note") then
9220            rendered = writer.note(reference)
9221          end
9222        end
9223
9224        t[opening_index].rendered = rendered
9225        delete_parsed_content_in_range(t, opening_index + 1,
9226                                       closing_index)
9227        empty_content_in_range(t, opening_index, closing_index)
9228        disable_previous_link_openers(t, opening_index)
9229        disable_range(t, opening_index, closing_index)
9230      end
9231
```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```
9232    local resolve_inline_following_content =
9233      function(t, closing_index, match_reference, match_link_attributes)
9234        local content = ""
9235        for i = closing_index + 1, #t do
9236          content = content .. t[i].content
9237        end
9238
9239        local matching_content = parsers.succeed
9240
9241        if match_reference then
9242          matching_content = matching_content
9243                           * parsers.inline_direct_ref_inside
9244        end
9245
9246        if match_link_attributes then
9247          matching_content = matching_content
```

```
                          * Cg(Ct(parsers.attributes^-1), "attributes")
    end

    local matched = lpeg.match(Ct( matching_content
                                 * Cg(Cp(), "end_position")), content)

    local matched_count = matched.end_position - 1
    for i = closing_index + 1, #t do
      local value = t[i]

      local chars_left = matched_count
      matched_count = matched_count - #value.content

      if matched_count <= 0 then
        value.content = value.content:sub(chars_left + 1)
        break
      end

      value.content = ''
      value.is_active = false
    end

    local attributes = matched.attributes
    if attributes == nil or next(attributes) == nil then
      attributes = nil
    end

    return {
      url = matched.url or "",
      title = matched.title or "",
      attributes = attributes
    }
  end

```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```
local function resolve_inline_link(t, opening_index, closing_index)
  local inline_content
    = resolve_inline_following_content(t, closing_index, true,
                                       t.match_link_attributes)
  render_link_or_image(t, opening_index, closing_index,
                       closing_index, inline_content)
end

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
local resolve_note_inline_link =
  function(t, opening_index, closing_index)
    local inline_content
      = resolve_inline_following_content(t, closing_index,
                                         false, false)
    render_link_or_image(t, opening_index, closing_index,
                         closing_index, inline_content)
  end

```

Resolve a shortcut link `[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```
local function resolve_shortcut_link(t, opening_index, closing_index)
  local content
    = collect_link_content(t, opening_index + 1, closing_index - 1)
  local r = self.lookup_reference(content)

  if r then
    local inline_content
      = resolve_inline_following_content(t, closing_index, false,
                                         t.match_link_attributes)
    r.attributes
      = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, closing_index,
                         closing_index, r)
  end
end

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```
local function resolve_raw_note_link(t, opening_index, closing_index)
  local content
    = collect_link_content(t, opening_index + 1, closing_index - 1)
  local r = self.lookup_note_reference(content)

  if r then
    local parsed_ref = self.parser_functions.parse_blocks_nested(r)
    render_link_or_image(t, opening_index, closing_index,
                         closing_index, parsed_ref)
  end
end

```

Resolve a full link `[a][b]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `b` is not found in the references.

```
local function resolve_full_link(t, opening_index, closing_index)
  local next_link_closing_index
    = find_next_link_closing_index(t, closing_index + 4)
  local next_link_content
    = collect_link_content(t, closing_index + 3,
                           next_link_closing_index - 1)
  local r = self.lookup_reference(next_link_content)

  if r then
    local inline_content
      = resolve_inline_following_content(t, next_link_closing_index,
                                         false,
                                         t.match_link_attributes)
    r.attributes
      = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, next_link_closing_index,
                         closing_index, r)
  end
end

```

Resolve a collapsed link `[a][]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```
local function resolve_collapsed_link(t, opening_index, closing_index)
  local next_link_closing_index
    = find_next_link_closing_index(t, closing_index + 4)
  local content
    = collect_link_content(t, opening_index + 1, closing_index - 1)
  local r = self.lookup_reference(content)

  if r then
    local inline_content
      = resolve_inline_following_content(t, closing_index, false,
                                         t.match_link_attributes)
    r.attributes
      = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, next_link_closing_index,
                         closing_index, r)
  end
end

```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the

entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```
local function process_links_and_emphasis(t)
  for _,value in ipairs(t) do
    value.is_active = true
  end

  for i,value in ipairs(t) do
    if not value.is_closing
        or value.type ~= "delimiter"
        or not ( value.element == "link"
              or value.element == "image"
              or value.element == "note")
        or value.removed then
      goto continue
    end

    local opener_position = find_link_opener(t, 1, i - 1)
    if (opener_position == nil) then
      goto continue
    end

    local opening_delimiter = t[opener_position]
    opening_delimiter.removed = true

    local link_type = opening_delimiter.link_type

    if (link_type == "inline") then
      resolve_inline_link(t, opener_position, i)
    end
    if (link_type == "shortcut") then
      resolve_shortcut_link(t, opener_position, i)
    end
    if (link_type == "full") then
      resolve_full_link(t, opener_position, i)
    end
    if (link_type == "collapsed") then
      resolve_collapsed_link(t, opener_position, i)
    end
    if (link_type == "note_inline") then
      resolve_note_inline_link(t, opener_position, i)
    end
    if (link_type == "raw_note") then
      resolve_raw_note_link(t, opener_position, i)
    end

    ::continue::
```

```
    end

    t[#t].content = t[#t].content:gsub("%s*$","")

    process_emphasis(t, 1, #t)
    local final_result = collect_emphasis_content(t, 1, #t)
    return final_result
  end

  function self.defer_link_and_emphasis_processing(delimiter_table)
    return writer.defer_call(function()
      return process_links_and_emphasis(delimiter_table)
    end)
  end

```

#### 3.1.6.8 Inline Elements (local)

```
  parsers.Str      = ( parsers.normalchar
                     * (parsers.normalchar + parsers.at)^0)
                   / writer.string

  parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
                   / writer.string

  parsers.Ellipsis = P("...") / writer.ellipsis

  parsers.Smart    = parsers.Ellipsis

  parsers.Code     = parsers.inticks / writer.code

  if options.blankBeforeBlockquote then
    parsers.bqstart = parsers.fail
  else
    parsers.bqstart = parsers.blockquote_start
  end

  if options.blankBeforeHeading then
    parsers.headerstart = parsers.fail
  else
    parsers.headerstart = parsers.atx_heading
  end

  if options.blankBeforeList then
    parsers.interrupting_bullets = parsers.fail
    parsers.interrupting_enumerators = parsers.fail
  else
```

```
  parsers.interrupting_bullets
    = parsers.bullet(parsers.dash, true)
    + parsers.bullet(parsers.asterisk, true)
    + parsers.bullet(parsers.plus, true)

  parsers.interrupting_enumerators
    = parsers.enumerator(parsers.period, true)
    + parsers.enumerator(parsers.rparent, true)
end

if options.html then
  parsers.html_interrupting
    = parsers.check_trail
    * ( parsers.html_incomplete_open_tag
      + parsers.html_incomplete_close_tag
      + parsers.html_incomplete_open_special_tag
      + parsers.html_comment_start
      + parsers.html_cdatasection_start
      + parsers.html_declaration_start
      + parsers.html_instruction_start
      - parsers.html_close_special_tag
      - parsers.html_empty_special_tag)
else
  parsers.html_interrupting = parsers.fail
end

parsers.ListStarter = parsers.starter

parsers.EndlineExceptions
                   = parsers.blankline -- paragraph break
                   + parsers.eof       -- end of document
                   + parsers.bqstart
                   + parsers.thematic_break_lines
                   + parsers.interrupting_bullets
                   + parsers.interrupting_enumerators
                   + parsers.headerstart
                   + parsers.html_interrupting

parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions

parsers.endline = parsers.newline
                * (parsers.check_minimal_indent
                  * -V("EndlineExceptions")
                  + parsers.check_optional_indent
                  * -V("EndlineExceptions")
                  * -V("ListStarter")) / function(_) return end
                * parsers.spacechar^0
```

```
9501
9502  parsers.Endline = parsers.endline
9503                  / writer.soft_line_break
9504
9505  parsers.EndlineNoSub = parsers.endline
9506
9507  parsers.NoSoftLineBreakEndline
9508                        = parsers.newline
9509                        * (parsers.check_minimal_indent
9510                          * -V("NoSoftLineBreakEndlineExceptions")
9511                          + parsers.check_optional_indent
9512                          * -V("NoSoftLineBreakEndlineExceptions")
9513                          * -V("ListStarter"))
9514                        * parsers.spacechar^0
9515                        / writer.space
9516
9517  parsers.EndlineBreak = parsers.backslash * parsers.endline
9518                                           / writer.hard_line_break
9519
9520  parsers.OptionalIndent
9521                      = parsers.spacechar^1 / writer.space
9522
9523  parsers.Space       = parsers.spacechar^2 * parsers.endline
9524                                           / writer.hard_line_break
9525                      + parsers.spacechar^1
9526                      * parsers.endline^-1
9527                      * parsers.eof / self.expandtabs
9528                      + parsers.spacechar^1 * parsers.endline
9529                                           / writer.soft_line_break
9530                      + parsers.spacechar^1
9531                      * -parsers.newline / self.expandtabs
9532                      + parsers.spacechar^1
9533
9534  parsers.NoSoftLineBreakSpace
9535                      = parsers.spacechar^2 * parsers.endline
9536                                           / writer.hard_line_break
9537                      + parsers.spacechar^1
9538                      * parsers.endline^-1
9539                      * parsers.eof / self.expandtabs
9540                      + parsers.spacechar^1 * parsers.endline
9541                                           / writer.soft_line_break
9542                      + parsers.spacechar^1
9543                      * -parsers.newline / self.expandtabs
9544                      + parsers.spacechar^1
9545
9546  parsers.NonbreakingEndline
9547                     = parsers.endline
```

```
                    / writer.nbsp

  parsers.NonbreakingSpace
                  = parsers.spacechar^2 * parsers.endline
                                        / writer.nbsp
                  + parsers.spacechar^1
                  * parsers.endline^-1 * parsers.eof / ""
                  + parsers.spacechar^1 * parsers.endline
                                        * parsers.optionalspace
                                        / writer.nbsp
                  + parsers.spacechar^1 * parsers.optionalspace
                                        / writer.nbsp

```

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

  parsers.AutoLinkUrl = parsers.auto_link_url
                      / self.auto_link_url

  parsers.AutoLinkEmail
                      = parsers.auto_link_email
                      / self.auto_link_email

  parsers.AutoLinkRelativeReference
                      = parsers.auto_link_relative_reference
                      / self.auto_link_url

  parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
                      / self.defer_link_and_emphasis_processing

  parsers.EscapedChar = parsers.backslash
                      * C(parsers.escapable) / writer.string
```

```
9587
9588  parsers.InlineHtml = Cs(parsers.html_inline_comment)
9589                     / writer.inline_html_comment
9590                     + Cs(parsers.html_any_empty_inline_tag
9591                         + parsers.html_inline_instruction
9592                         + parsers.html_inline_cdatasection
9593                         + parsers.html_inline_declaration
9594                         + parsers.html_any_open_inline_tag
9595                         + parsers.html_any_close_tag)
9596                       / writer.inline_html_tag
9597
9598  parsers.HtmlEntity = parsers.html_entities / writer.string
```

#### 3.1.6.9 Block Elements (local)

```
9599  parsers.DisplayHtml = Cs(parsers.check_trail
9600                           * ( parsers.html_comment
9601                             + parsers.html_special_block
9602                             + parsers.html_block
9603                             + parsers.html_any_block
9604                             + parsers.html_instruction
9605                             + parsers.html_cdatasection
9606                             + parsers.html_declaration))
9607                         / writer.block_html_element
9608
9609  parsers.indented_non_blank_line = parsers.indentedline
9610                                  - parsers.blankline
9611
9612  parsers.Verbatim
9613    = Cs( parsers.check_code_trail
9614        * (parsers.line - parsers.blankline)
9615        * (( parsers.check_minimal_blank_indent_and_full_code_trail
9616           * parsers.blankline)^0
9617          * ( (parsers.check_minimal_indent / "")
9618            * parsers.check_code_trail
9619            * (parsers.line - parsers.blankline))^1)^0)
9620    / self.expandtabs / writer.verbatim
9621
9622  parsers.Blockquote   = parsers.blockquote_body
9623                       / writer.blockquote
9624
9625  parsers.ThematicBreak = parsers.thematic_break_lines
9626                        / writer.thematic_break
9627
9628  parsers.Reference    = parsers.define_reference_parser
9629                       / self.register_link
9630
```

```
parsers.Paragraph   = parsers.freeze_trail
                    * (Ct((parsers.Inline)^1)
                    * (parsers.newline + parsers.eof)
                    * parsers.unfreeze_trail
                    / writer.paragraph)

parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
                     / writer.plain
```

#### 3.1.6.10 Lists (local)

```
if options.taskLists then
  parsers.tickbox = ( parsers.ticked_box
                    + parsers.halfticked_box
                    + parsers.unticked_box
                    ) / writer.tickbox
else
   parsers.tickbox = parsers.fail
end

parsers.list_blank = parsers.conditionally_indented_blankline

parsers.ref_or_block_list_separated
  = parsers.sep_group_no_output(parsers.list_blank)
  * parsers.minimally_indented_ref
  + parsers.block_sep_group(parsers.list_blank)
  * parsers.minimally_indented_block

parsers.ref_or_block_non_separated
  = parsers.minimally_indented_ref
  + (parsers.succeed / writer.interblocksep)
  * parsers.minimally_indented_block
  - parsers.minimally_indented_blankline

parsers.tight_list_loop_body_pair =
  parsers.create_loop_body_pair(
    parsers.ref_or_block_non_separated,
    parsers.minimally_indented_par_or_plain_no_blank,
    (parsers.succeed / writer.interblocksep),
    (parsers.succeed / writer.paragraphsep))

parsers.loose_list_loop_body_pair =
  parsers.create_loop_body_pair(
    parsers.ref_or_block_list_separated,
    parsers.minimally_indented_par_or_plain,
    parsers.block_sep_group(parsers.list_blank),
```

```
9675      parsers.par_sep_group(parsers.list_blank))
9676
9677  parsers.tight_list_content_loop
9678    = V("Block")
9679    * parsers.tight_list_loop_body_pair.block^0
9680    + (V("Paragraph") + V("Plain"))
9681    * parsers.ref_or_block_non_separated
9682    * parsers.tight_list_loop_body_pair.block^0
9683    +  (V("Paragraph") + V("Plain"))
9684    * parsers.tight_list_loop_body_pair.par^0
9685
9686  parsers.loose_list_content_loop
9687    = V("Block")
9688    * parsers.loose_list_loop_body_pair.block^0
9689    + (V("Paragraph") + V("Plain"))
9690    * parsers.ref_or_block_list_separated
9691    * parsers.loose_list_loop_body_pair.block^0
9692    + (V("Paragraph") + V("Plain"))
9693    * parsers.loose_list_loop_body_pair.par^0
9694
9695  parsers.list_item_tightness_condition
9696    = -#( parsers.list_blank^0
9697        * parsers.minimally_indented_ref_or_block_or_par)
9698    * remove_indent("li")
9699    + remove_indent("li")
9700    * parsers.fail
9701
9702  parsers.indented_content_tight
9703    = Ct( (parsers.blankline / "")
9704        * #parsers.list_blank
9705        * remove_indent("li")
9706        + ( (V("Reference") + (parsers.blankline / ""))
9707          * parsers.check_minimal_indent
9708          * parsers.tight_list_content_loop
9709          + (V("Reference") + (parsers.blankline / ""))
9710          + (parsers.tickbox^-1 / writer.escape)
9711          * parsers.tight_list_content_loop
9712          )
9713        * parsers.list_item_tightness_condition)
9714
9715  parsers.indented_content_loose
9716    = Ct( (parsers.blankline / "")
9717        * #parsers.list_blank
9718        + ( (V("Reference") + (parsers.blankline / ""))
9719          * parsers.check_minimal_indent
9720          * parsers.loose_list_content_loop
9721          + (V("Reference") + (parsers.blankline / ""))
```

```
9722          + (parsers.tickbox^-1 / writer.escape)
9723          * parsers.loose_list_content_loop))
9724
9725  parsers.TightListItem = function(starter)
9726    return  -parsers.ThematicBreak
9727            * parsers.add_indent(starter, "li")
9728            * parsers.indented_content_tight
9729  end
9730
9731  parsers.LooseListItem = function(starter)
9732    return  -parsers.ThematicBreak
9733            * parsers.add_indent(starter, "li")
9734            * parsers.indented_content_loose
9735            * remove_indent("li")
9736  end
9737
9738  parsers.BulletListOfType = function(bullet_type)
9739    local bullet = parsers.bullet(bullet_type)
9740    return  ( Ct( parsers.TightListItem(bullet)
9741                * ( (parsers.check_minimal_indent / "")
9742                  * parsers.TightListItem(bullet)
9743                  )^0
9744              )
9745              * Cc(true)
9746              * -#( (parsers.list_blank^0 / "")
9747                  * parsers.check_minimal_indent
9748                  * (bullet - parsers.ThematicBreak)
9749              )
9750              + Ct( parsers.LooseListItem(bullet)
9751                  * ( (parsers.list_blank^0 / "")
9752                    * (parsers.check_minimal_indent / "")
9753                    * parsers.LooseListItem(bullet)
9754                    )^0
9755              )
9756              * Cc(false)
9757            ) / writer.bulletlist
9758  end
9759
9760  parsers.BulletList = parsers.BulletListOfType(parsers.dash)
9761                     + parsers.BulletListOfType(parsers.asterisk)
9762                     + parsers.BulletListOfType(parsers.plus)
9763
9764  local function ordered_list(items,tight,starter)
9765    local startnum = starter[2][1]
9766    if options.startNumber then
9767      startnum = tonumber(startnum) or 1  -- fallback for '#'
9768      if startnum ~= nil then
```

```
9769          startnum = math.floor(startnum)
9770        end
9771      else
9772        startnum = nil
9773      end
9774      return writer.orderedlist(items,tight,startnum)
9775    end
9776
9777    parsers.OrderedListOfType = function(delimiter_type)
9778      local enumerator = parsers.enumerator(delimiter_type)
9779      return  Cg(enumerator, "listtype")
9780            * (Ct( parsers.TightListItem(Cb("listtype"))
9781                 * ( (parsers.check_minimal_indent / "")
9782                   * parsers.TightListItem(enumerator))^0)
9783            * Cc(true)
9784            * -#((parsers.list_blank^0 / "")
9785                * parsers.check_minimal_indent * enumerator)
9786            + Ct( parsers.LooseListItem(Cb("listtype"))
9787                * ((parsers.list_blank^0 / "")
9788                  * (parsers.check_minimal_indent / "")
9789                  * parsers.LooseListItem(enumerator))^0)
9790            * Cc(false)
9791            ) * Ct(Cb("listtype")) / ordered_list
9792    end
9793
9794    parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
9795                        + parsers.OrderedListOfType(parsers.rparent)
```

#### 3.1.6.11 Blank (local)

```
9796    parsers.Blank        = parsers.blankline / ""
9797                         + V("Reference")
```

#### 3.1.6.12 Headings (local)

```
9798    function parsers.parse_heading_text(s)
9799      local inlines = self.parser_functions.parse_inlines(s)
9800      local flatten_inlines = self.writer.flatten_inlines
9801      self.writer.flatten_inlines = true
9802      local flat_text = self.parser_functions.parse_inlines(s)
9803      flat_text = util.rope_to_string(flat_text)
9804      self.writer.flatten_inlines = flatten_inlines
9805      return {flat_text, inlines}
9806    end
9807
9808    -- parse atx header
9809    parsers.AtxHeading = parsers.check_trail_no_rem
9810                       * Cg(parsers.heading_start, "level")
```

```
                      * ((C( parsers.optionalspace
                           * parsers.hash^0
                           * parsers.optionalspace
                           * parsers.newline)
                         + parsers.spacechar^1
                         * C(parsers.line))
                        / strip_atx_end
                        / parsers.parse_heading_text)
                      * Cb("level")
                      / writer.heading

parsers.heading_line  = parsers.linechar^1
                      - parsers.thematic_break_lines

parsers.heading_text = parsers.heading_line
                     * ( (V("Endline") / "\n")
                       * ( parsers.heading_line
                         - parsers.heading_level))^0
                     * parsers.newline^-1

parsers.SetextHeading = parsers.freeze_trail
                      * parsers.check_trail_no_rem
                      * #( parsers.heading_text
                         * parsers.check_minimal_indent
                         * parsers.check_trail
                         * parsers.heading_level)
                      * Cs(parsers.heading_text)
                      / parsers.parse_heading_text
                      * parsers.check_minimal_indent_and_trail
                      * parsers.heading_level
                      * parsers.newline
                      * parsers.unfreeze_trail
                      / writer.heading

parsers.Heading = parsers.AtxHeading + parsers.SetextHeading
```

#### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain 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
```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```
               * ( V("Block")
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
                     )
                   )
                 + ( V("Paragraph") + V("Plain") )
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
                     )
                   )
                 * V("Block")
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
                     )
                   )
                 + ( V("Paragraph") + V("Plain") )
                 * ( V("Blank")^0 * parsers.eof
                   + V("Blank")^0 / writer.paragraphsep
                   )
```

```
                )^0,

        ExpectedJekyllData = parsers.succeed,

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

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

        ListStarter        = parsers.ListStarter,
        EndlineExceptions  = parsers.EndlineExceptions,
        NoSoftLineBreakEndlineExceptions
                           = parsers.NoSoftLineBreakEndlineExceptions,

        Str                = parsers.Str,
        Space              = parsers.Space,
        NoSoftLineBreakSpace
                           = parsers.NoSoftLineBreakSpace,
        OptionalIndent     = parsers.OptionalIndent,
        Endline            = parsers.Endline,
        EndlineNoSub       = parsers.EndlineNoSub,
        NoSoftLineBreakEndline
                           = parsers.NoSoftLineBreakEndline,
        EndlineBreak       = parsers.EndlineBreak,
        LinkAndEmph        = parsers.LinkAndEmph,
        Code               = parsers.Code,
        AutoLinkUrl        = parsers.AutoLinkUrl,
        AutoLinkEmail      = parsers.AutoLinkEmail,
        AutoLinkRelativeReference
                           = parsers.AutoLinkRelativeReference,
        InlineHtml         = parsers.InlineHtml,
        HtmlEntity         = parsers.HtmlEntity,
        EscapedChar        = parsers.EscapedChar,
        Smart              = parsers.Smart,
        Symbol             = parsers.Symbol,
        SpecialChar        = parsers.fail,
        InitializeState    = parsers.succeed,
    }
```

Define `reader->update_rule` as a function that receives two arguments: a left-

hand side terminal symbol and a function that accepts the current PEG pattern in walkable_syntax[left-hand side terminal symbol] if defined or nil otherwise and returns a PEG pattern that will (re)define walkable_syntax[left-hand side terminal symbol].

```
    self.update_rule = function(rule_name, get_pattern)
      assert(current_extension_name ~= nil)
      assert(syntax[rule_name] ~= nil,
        [[Rule ]] .. rule_name
        .. [[ -> ... does not exist in markdown grammar]])
      local previous_pattern
      local extension_name
      if walkable_syntax[rule_name] then
        local previous_accountable_pattern
          = walkable_syntax[rule_name][1]
        previous_pattern = previous_accountable_pattern[1]
        extension_name
          = previous_accountable_pattern[2]
          .. ", " .. current_extension_name
      else
        previous_pattern = nil
        extension_name = current_extension_name
      end
      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
```

```
      if type(get_pattern) == "function" then
        pattern = get_pattern(previous_pattern)
      else
        assert(previous_pattern == nil,
               [[Rule ]] .. rule_name ..
               [[ has already been updated by ]] .. extension_name)
        pattern = get_pattern
      end
      local accountable_pattern = { pattern, extension_name, rule_name }
      walkable_syntax[rule_name] = { accountable_pattern }
    end
```

Define a hash table of all characters with special meaning and add method `reader->add_special_character` that extends the hash table and updates the PEG grammar of markdown.

```
10005     local special_characters = {}
10006     self.add_special_character = function(c)
10007       table.insert(special_characters, c)
10008       syntax.SpecialChar = S(table.concat(special_characters, ""))
10009     end
10010
10011     self.add_special_character("*")
10012     self.add_special_character("[")
10013     self.add_special_character("]")
10014     self.add_special_character("<")
10015     self.add_special_character("!")
10016     self.add_special_character("\\")
```

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

```
10017     self.initialize_named_group = function(name, value)
10018       local pattern = Ct("")
10019       if value ~= nil then
10020         pattern = pattern / value
10021       end
10022       syntax.InitializeState = syntax.InitializeState
10023                              * Cg(pattern, name)
10024     end
```

Add a named group for indentation.

```
10025     self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
10026     for _, extension in ipairs(extensions) do
10027       current_extension_name = extension.name
10028       extension.extend_writer(writer)
10029       extension.extend_reader(self)
10030     end
10031     current_extension_name = nil
```

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

```
10032     if options.debugExtensions then
10033       local sorted_lhs = {}
10034       for lhs, _ in pairs(walkable_syntax) do
10035         table.insert(sorted_lhs, lhs)
10036       end
10037       table.sort(sorted_lhs)
10038
10039       local output_lines = {"{"}
```

```
10040       for lhs_index, lhs in ipairs(sorted_lhs) do
10041         local encoded_lhs = util.encode_json_string(lhs)
10042         table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
10043         local rule = walkable_syntax[lhs]
10044         for rhs_index, rhs in ipairs(rule) do
10045           local human_readable_rhs
10046           if type(rhs) == "string" then
10047             human_readable_rhs = rhs
10048           else
10049             local pattern_name
10050             if rhs[3] then
10051               pattern_name = rhs[3]
10052             else
10053               pattern_name = "Anonymous Pattern"
10054             end
10055             local extension_name = rhs[2]
10056             human_readable_rhs = pattern_name .. [[ (]]
10057                                  .. extension_name .. [[)]]
10058           end
10059           local encoded_rhs
10060             = util.encode_json_string(human_readable_rhs)
10061           local output_line = [[        ]] .. encoded_rhs
10062           if rhs_index < #rule then
10063             output_line = output_line .. ","
10064           end
10065           table.insert(output_lines, output_line)
10066         end
10067         local output_line = "    ]"
10068         if lhs_index < #sorted_lhs then
10069           output_line = output_line .. ","
10070         end
10071         table.insert(output_lines, output_line)
10072       end
10073       table.insert(output_lines, "}")
10074
10075       local output = table.concat(output_lines, "\n")
10076       local output_filename = options.debugExtensionsFileName
10077       local output_file = assert(io.open(output_filename, "w"),
10078         [[Could not open file "]] .. output_filename
10079         .. [[" for writing]])
10080       assert(output_file:write(output))
10081       assert(output_file:close())
10082     end
```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```
10083     for lhs, rule in pairs(walkable_syntax) do
10084       syntax[lhs] = parsers.fail
10085       for _, rhs in ipairs(rule) do
10086         local pattern
```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```
10087         if type(rhs) == "string" then
10088           pattern = V(rhs)
10089         else
10090           pattern = rhs[1]
10091           if type(pattern) == "string" then
10092             pattern = V(pattern)
10093           end
10094         end
10095         syntax[lhs] = syntax[lhs] + pattern
10096       end
10097     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.

```
10098     if options.underscores then
10099       self.add_special_character("_")
10100     end
10101
10102     if not options.codeSpans then
10103       syntax.Code = parsers.fail
10104     else
10105       self.add_special_character("`")
10106     end
10107
10108     if not options.html then
10109       syntax.DisplayHtml = parsers.fail
10110       syntax.InlineHtml = parsers.fail
10111       syntax.HtmlEntity  = parsers.fail
10112     else
10113       self.add_special_character("&")
10114     end
10115
10116     if options.preserveTabs then
10117       options.stripIndent = false
10118     end
10119
10120     if not options.smartEllipses then
```

```
10121       syntax.Smart = parsers.fail
10122     else
10123       self.add_special_character(".")
10124     end
10125
10126     if not options.relativeReferences then
10127       syntax.AutoLinkRelativeReference = parsers.fail
10128     end
10129
10130     if options.contentLevel == "inline" then
10131       syntax[1] = "Inlines"
10132       syntax.Inlines = V("InitializeState")
10133                        * parsers.Inline^0
10134                        * ( parsers.spacing^0
10135                          * parsers.eof / "")
10136       syntax.Space = parsers.Space + parsers.blankline / writer.space
10137     end
10138
10139     local blocks_nested_t = util.table_copy(syntax)
10140     blocks_nested_t.ExpectedJekyllData = parsers.succeed
10141     parsers.blocks_nested = Ct(blocks_nested_t)
10142
10143     parsers.blocks = Ct(syntax)
10144
10145     local inlines_t = util.table_copy(syntax)
10146     inlines_t[1] = "Inlines"
10147     inlines_t.Inlines = V("InitializeState")
10148                         * parsers.Inline^0
10149                         * ( parsers.spacing^0
10150                           * parsers.eof / "")
10151     parsers.inlines = Ct(inlines_t)
10152
10153     local inlines_no_inline_note_t = util.table_copy(inlines_t)
10154     inlines_no_inline_note_t.InlineNote = parsers.fail
10155     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10156
10157     local inlines_no_html_t = util.table_copy(inlines_t)
10158     inlines_no_html_t.DisplayHtml = parsers.fail
10159     inlines_no_html_t.InlineHtml = parsers.fail
10160     inlines_no_html_t.HtmlEntity = parsers.fail
10161     parsers.inlines_no_html = Ct(inlines_no_html_t)
10162
10163     local inlines_nbsp_t = util.table_copy(inlines_t)
10164     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10165     inlines_nbsp_t.Space = parsers.NonbreakingSpace
10166     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10167
```

```
10168     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10169     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10170     inlines_no_link_or_emphasis_t.EndlineExceptions
10171       = parsers.EndlineExceptions - parsers.eof
10172     parsers.inlines_no_link_or_emphasis
10173       = Ct(inlines_no_link_or_emphasis_t)
```

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

```
10174     return function(input)
```

Unicode-normalize the input.

```
10175       if options.unicodeNormalization then
10176         local form = options.unicodeNormalizationForm
10177         if form == "nfc" then
10178           input = uni_algos.normalize.NFC(input)
10179         elseif form == "nfd" then
10180           input = uni_algos.normalize.NFD(input)
10181         elseif form == "nfkc" then
10182           input = uni_algos.normalize.NFKC(input)
10183         elseif form == "nfkd" then
10184           input = uni_algos.normalize.NFKD(input)
10185         else
10186           return writer.error(
10187             format("Unknown normalization form %s.", form))
10188         end
10189       end
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
10190       input = input:gsub("\r\n?", "\n")
10191       if input:sub(-1) ~= "\n" then
10192         input = input .. "\n"
10193       end
```

Clear the table of references.

```
10194       references = {}
10195       local document = self.parser_functions.parse_blocks(input)
10196       local output = util.rope_to_string(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
10197       local undosep_start, undosep_end
10198       local potential_secend_start, secend_start
10199       local potential_sep_start, sep_start
10200       while true do
10201         -- find a `writer->undosep`
10202         undosep_start, undosep_end
```

```
10203           = output:find(writer.undosep_text, 1, true)
10204         if undosep_start == nil then break end
10205         -- skip any preceding section ends
10206         secend_start = undosep_start
10207         while true do
10208           potential_secend_start = secend_start - #writer.secend_text
10209           if potential_secend_start < 1
10210              or output:sub(potential_secend_start,
10211                            secend_start - 1) ~= writer.secend_text
10212              then
10213             break
10214           end
10215           secend_start = potential_secend_start
10216         end
10217         -- find an immediately preceding
10218         -- block element / paragraph separator
10219         sep_start = secend_start
10220         potential_sep_start = sep_start - #writer.interblocksep_text
10221         if potential_sep_start >= 1
10222            and output:sub(potential_sep_start,
10223                           sep_start - 1) == writer.interblocksep_text
10224            then
10225           sep_start = potential_sep_start
10226         else
10227           potential_sep_start = sep_start - #writer.paragraphsep_text
10228           if potential_sep_start >= 1
10229              and output:sub(potential_sep_start,
10230                             sep_start - 1) == writer.paragraphsep_text
10231              then
10232             sep_start = potential_sep_start
10233           end
10234         end
10235         -- remove `writer->undosep` and immediately preceding
10236         -- block element / paragraph separator
10237         output = output:sub(1, sep_start - 1)
10238               .. output:sub(secend_start, undosep_start - 1)
10239               .. output:sub(undosep_end + 1)
10240       end
10241       return output
10242     end
10243   end
10244   return self
10245 end
```

### 3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax

extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

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

#### 3.1.7.1 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)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererBracketedSpanAttributeContextBegin",
                self.attributes(attr),
                s,
                "\\markdownRendererBracketedSpanAttributeContextEnd{}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local span_label  = parsers.lbracket
                          * (Cs((parsers.alphanumeric^1
                              + parsers.inticks
                              + parsers.autolink
                              + V("InlineHtml")
                              + ( parsers.backslash * parsers.backslash)
                              + ( parsers.backslash
                                * (parsers.lbracket + parsers.rbracket)
                                + V("Space") + V("Endline")
                                + (parsers.any
                                  - ( parsers.newline
                                    + parsers.lbracket
                                    + parsers.rbracket
                                    + parsers.blankline^2))))^1)
                          / self.parser_functions.parse_inlines)
                          * parsers.rbracket

      local Span = span_label
                   * Ct(parsers.attributes)
```

```
                  / writer.span

      self.insert_pattern("Inline before LinkAndEmph",
                          Span, "Span")
    end
  }
end
```

#### 3.1.7.2 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```
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.

```
      function self.citations(text_cites, cites)
        local buffer = {}
        if self.flatten_inlines then
          for _,cite in ipairs(cites) do
            if cite.prenote then
              table.insert(buffer, {cite.prenote, " "})
            end
            table.insert(buffer, cite.name)
            if cite.postnote then
              table.insert(buffer, {" ", cite.postnote})
            end
          end
        else
```

```
        table.insert(buffer,
                     {"\\markdownRenderer",
                      text_cites and "TextCite" or "Cite",
                      "{", #cites, "}"})
        for _,cite in ipairs(cites) do
          table.insert(buffer,
                       {cite.suppress_author and "-" or "+", "{",
                        cite.prenote or "", "}{",
                        cite.postnote or "", "}{", cite.name, "}"})
        end
      end
      return buffer
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local citation_chars
                = parsers.alphanumeric
                + S("#$%&-+<>~/_")

    local citation_name
                = Cs(parsers.dash^-1) * parsers.at
                * Cs(citation_chars
                    * ((( citation_chars
                        + parsers.internal_punctuation
                        - parsers.comma - parsers.semicolon)
                       * -#(( parsers.internal_punctuation
                            - parsers.comma
                            - parsers.semicolon)^0
                           * -( citation_chars
                              + parsers.internal_punctuation
                              - parsers.comma
                              - parsers.semicolon)))^0
                      * citation_chars)^-1)

    local citation_body_prenote
                = Cs((parsers.alphanumeric^1
                     + parsers.bracketed
                     + parsers.inticks
                     + parsers.autolink
                     + V("InlineHtml")
                     + V("Space") + V("EndlineNoSub")
                     + (parsers.anyescaped
                       - ( parsers.newline
                         + parsers.rbracket
                         + parsers.blankline^2))
```

```
                      - ( parsers.spnl
                        * parsers.dash^-1
                        * parsers.at))^1)

    local citation_body_postnote
                  = Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + parsers.autolink
                       + V("InlineHtml")
                       + V("Space") + V("EndlineNoSub")
                       + (parsers.anyescaped
                         - ( parsers.newline
                           + parsers.rbracket
                           + parsers.semicolon
                           + parsers.blankline^2))
                       - (parsers.spnl * parsers.rbracket))^1)

    local citation_body_chunk
                  = ( citation_body_prenote
                    * parsers.spnlc_sep
                    + Cc("")
                    * parsers.spnlc
                  )
                  * citation_name
                  * ( parsers.internal_punctuation
                    - parsers.semicolon)^-1
                  * ( parsers.spnlc / function(_) return end
                    * citation_body_postnote
                    + Cc("")
                    * parsers.spnlc
                  )

    local citation_body
                  = citation_body_chunk
                  * ( parsers.semicolon
                    * parsers.spnlc
                    * citation_body_chunk
                  )^0

    local citation_headless_body_postnote
                  = Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + parsers.autolink
                       + V("InlineHtml")
                       + V("Space") + V("Endline")
```

```
                      + (parsers.anyescaped
                        - ( parsers.newline
                          + parsers.rbracket
                          + parsers.at
                          + parsers.semicolon + parsers.blankline^2))
                      - (parsers.spnl * parsers.rbracket))^0)

    local citation_headless_body
                  = citation_headless_body_postnote
                  * ( parsers.semicolon
                    * parsers.spnlc
                    * 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.spnlc
                  * Cc("")
                  * citation_name
                  * ((parsers.spnlc
                      * parsers.lbracket
                      * citation_headless_body
                      * parsers.rbracket) + Cc("")))^1)
```

```
                    / function(raw_cites)
                        return citations(true, raw_cites)
                      end

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

      local Citations = TextCitations + ParenthesizedCitations

      self.insert_pattern("Inline before LinkAndEmph",
                          Citations, "Citations")

      self.add_special_character("@")
      self.add_special_character("-")
    end
  }
end
```

#### 3.1.7.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```
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{kpse.lookup(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
```

```
10482           base = curr
10483         else
10484           setmetatable(prev, { __index = curr })
10485         end
10486         prev = curr
10487       end
10488       ::continue::
10489     end
10490     return base or {}
10491   end)()
10492
10493   return {
10494     name = "built-in content_blocks syntax extension",
10495     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.

```
10496       function self.contentblock(src,suf,type,tit)
10497         if not self.is_writing then return "" end
10498         src = src.."."..suf
10499         suf = suf:lower()
10500         if type == "onlineimage" then
10501           return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
10502                   "{",self.string(src),"}",
10503                   "{",self.uri(src),"}",
10504                   "{",self.string(tit or ""),"}"}
10505         elseif languages_json[suf] then
10506           return {"\\markdownRendererContentBlockCode{",suf,"}",
10507                   "{",self.string(languages_json[suf]),"}",
10508                   "{",self.string(src),"}",
10509                   "{",self.uri(src),"}",
10510                   "{",self.string(tit or ""),"}"}
10511         else
10512           return {"\\markdownRendererContentBlock{",suf,"}",
10513                   "{",self.string(src),"}",
10514                   "{",self.uri(src),"}",
10515                   "{",self.string(tit or ""),"}"}
10516         end
10517       end
10518     end, extend_reader = function(self)
10519       local parsers = self.parsers
10520       local writer = self.writer
10521
10522       local contentblock_tail
10523                       = 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.spacing
                       - #(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.check_trail_no_rem
                    * (localfilepath + onlineimageurl)
                    * contentblock_tail
                    / writer.contentblock

      self.insert_pattern("Block before Blockquote",
                          ContentBlock, "ContentBlock")
    end
  }
end
```

#### 3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

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

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1]
```

```
            = {"\\markdownRendererDlDefinitionBegin ",def,
               "\\markdownRendererDlDefinitionEnd "}
        end
        retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
        return retVal
      end

      function self.definitionlist(items,tight)
        if not self.is_writing then return "" end
        local buffer = {}
        for _,item in ipairs(items) do
          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
        end
        if tight and tight_lists then
          return {"\\markdownRendererDlBeginTight\n", buffer,
            "\n\\markdownRendererDlEndTight"}
        else
          return {"\\markdownRendererDlBegin\n", buffer,
            "\n\\markdownRendererDlEnd"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local defstartchar = S("~:")

      local defstart
        = parsers.check_trail_length(0) * defstartchar
        * #parsers.spacing
        * (parsers.tab + parsers.space^-3)
        + parsers.check_trail_length(1)
        * defstartchar * #parsers.spacing
        * (parsers.tab + parsers.space^-2)
        + parsers.check_trail_length(2)
        * defstartchar * #parsers.spacing
        * (parsers.tab + parsers.space^-1)
        + parsers.check_trail_length(3)
        * defstartchar * #parsers.spacing

      local indented_line
        = (parsers.check_minimal_indent / "")
        * parsers.check_code_trail * parsers.line

      local blank
        = parsers.check_minimal_blank_indent_and_any_trail
        * parsers.optionalspace * parsers.newline
```

```
10654
10655       local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10656
10657       local indented_blocks = function(bl)
10658         return Cs( bl
10659               * (blank^1 * (parsers.check_minimal_indent / "")
10660                 * parsers.check_code_trail * -parsers.blankline * bl)^0
10661               * (blank^1 + parsers.eof))
10662       end
10663
10664       local function definition_list_item(term, defs, _)
10665         return { term = self.parser_functions.parse_inlines(term),
10666                  definitions = defs }
10667       end
10668
10669       local DefinitionListItemLoose
10670         = C(parsers.line) * blank^0
10671         * Ct((parsers.check_minimal_indent * (defstart
10672               * indented_blocks(dlchunk)
10673               / self.parser_functions.parse_blocks_nested))^1)
10674         * Cc(false) / definition_list_item
10675
10676       local DefinitionListItemTight
10677         = C(parsers.line)
10678         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
10679               / self.parser_functions.parse_blocks_nested))^1)
10680         * Cc(true) / definition_list_item
10681
10682       local DefinitionList
10683         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
10684           + Ct(DefinitionListItemTight^1)
10685           * (blank^0
10686             * -DefinitionListItemLoose * Cc(true))
10687           ) / writer.definitionlist
10688
10689       self.insert_pattern("Block after Heading",
10690                           DefinitionList, "DefinitionList")
10691     end
10692   }
10693 end
```

#### 3.1.7.5 Fancy Lists

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

```
10694 M.extensions.fancy_lists = function()
10695   return {
```

```
10696     name = "built-in fancy_lists syntax extension",
10697     extend_writer = function(self)
10698       local options = self.options
10699
```

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.

```
10700       function self.fancylist(items,tight,startnum,numstyle,numdelim)
10701         if not self.is_writing then return "" end
10702         local buffer = {}
10703         local num = startnum
10704         for _,item in ipairs(items) do
10705           if item ~= "" then
10706             buffer[#buffer + 1] = self.fancyitem(item,num)
10707           end
10708           if num ~= nil and item ~= "" then
10709             num = num + 1
10710           end
10711         end
10712         local contents = util.intersperse(buffer,"\n")
10713         if tight and options.tightLists then
10714           return {"\\markdownRendererFancyOlBeginTight{",
10715                   numstyle,"}{",numdelim,"}",contents,
10716                   "\n\\markdownRendererFancyOlEndTight "}
10717         else
10718           return {"\\markdownRendererFancyOlBegin{",
10719                   numstyle,"}{",numdelim,"}",contents,
10720                   "\n\\markdownRendererFancyOlEnd "}
```

```
10721         end
10722       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.

```
10723       function self.fancyitem(s,num)
10724         if num ~= nil then
10725           return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
10726                   "\\markdownRendererFancyOlItemEnd "}
10727         else
10728           return {"\\markdownRendererFancyOlItem ",s,
10729                   "\\markdownRendererFancyOlItemEnd "}
10730         end
10731       end
10732     end, extend_reader = function(self)
10733       local parsers = self.parsers
10734       local options = self.options
10735       local writer = self.writer
10736
10737       local function combine_markers_and_delims(markers, delims)
10738         local markers_table = {}
10739         for _,marker in ipairs(markers) do
10740           local start_marker
10741           local continuation_marker
10742           if type(marker) == "table" then
10743             start_marker = marker[1]
10744             continuation_marker = marker[2]
10745           else
10746             start_marker = marker
10747             continuation_marker = marker
10748           end
10749           for _,delim in ipairs(delims) do
10750             table.insert(markers_table,
10751                          {start_marker, continuation_marker, delim})
10752           end
10753         end
10754         return markers_table
10755       end
10756
10757       local function join_table_with_func(func, markers_table)
10758         local pattern = func(table.unpack(markers_table[1]))
10759         for i = 2, #markers_table do
10760           pattern = pattern + func(table.unpack(markers_table[i]))
10761         end
10762         return pattern
10763       end
```

```
10764
10765       local lowercase_letter_marker = R("az")
10766       local uppercase_letter_marker = R("AZ")
10767
10768       local roman_marker = function(chars)
10769         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
10770         local l, x, v, i
10771           = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
10772         return  m^-3
10773                 * (c*m + c*d + d^-1 * c^-3)
10774                 * (x*c + x*l + l^-1 * x^-3)
10775                 * (i*x + i*v + v^-1 * i^-3)
10776       end
10777
10778       local lowercase_roman_marker
10779         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10780       local uppercase_roman_marker
10781         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10782
10783       local lowercase_opening_roman_marker  = P("i")
10784       local uppercase_opening_roman_marker  = P("I")
10785
10786       local digit_marker = parsers.dig * parsers.dig^-8
10787
10788       local markers = {
10789         {lowercase_opening_roman_marker, lowercase_roman_marker},
10790         {uppercase_opening_roman_marker, uppercase_roman_marker},
10791         lowercase_letter_marker,
10792         uppercase_letter_marker,
10793         lowercase_roman_marker,
10794         uppercase_roman_marker,
10795         digit_marker
10796       }
10797
10798       local delims = {
10799         parsers.period,
10800         parsers.rparent
10801       }
10802
10803       local markers_table = combine_markers_and_delims(markers, delims)
10804
10805       local function enumerator(start_marker, _,
10806                                 delimiter_type, interrupting)
10807         local delimiter_range
10808         local allowed_end
10809         if interrupting then
10810           delimiter_range = P("1")
```

```
10811           allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10812         else
10813           delimiter_range = start_marker
10814           allowed_end = C(parsers.spacechar^1)
10815                       + #(parsers.newline + parsers.eof)
10816         end
10817
10818         return parsers.check_trail
10819                 * Ct(C(delimiter_range) * C(delimiter_type))
10820                 * allowed_end
10821       end
10822
10823       local starter = join_table_with_func(enumerator, markers_table)
10824
10825       local TightListItem = function(starter)
10826         return  parsers.add_indent(starter, "li")
10827                 * parsers.indented_content_tight
10828       end
10829
10830       local LooseListItem = function(starter)
10831         return  parsers.add_indent(starter, "li")
10832                 * parsers.indented_content_loose
10833                 * remove_indent("li")
10834       end
10835
10836       local function roman2number(roman)
10837         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
10838                          ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
10839         local numeral = 0
10840
10841         local i = 1
10842         local len = string.len(roman)
10843         while i < len do
10844           local z1, z2 = romans[ string.sub(roman, i, i) ],
10845                          romans[ string.sub(roman, i+1, i+1) ]
10846           if z1 < z2 then
10847               numeral = numeral + (z2 - z1)
10848               i = i + 2
10849           else
10850               numeral = numeral + z1
10851               i = i + 1
10852           end
10853         end
10854         if i <= len then
10855           numeral = numeral + romans[ string.sub(roman,i,i) ]
10856         end
10857         return numeral
```

```
10858       end
10859
10860       local function sniffstyle(numstr, delimend)
10861         local numdelim
10862         if delimend == ")" then
10863           numdelim = "OneParen"
10864         elseif delimend == "." then
10865           numdelim = "Period"
10866         else
10867           numdelim = "Default"
10868         end
10869
10870         local num
10871         num = numstr:match("^([I])$")
10872         if num then
10873           return roman2number(num), "UpperRoman", numdelim
10874         end
10875         num = numstr:match("^([i])$")
10876         if num then
10877           return roman2number(string.upper(num)), "LowerRoman", numdelim
10878         end
10879         num = numstr:match("^([A-Z])$")
10880         if num then
10881           return string.byte(num) - string.byte("A") + 1,
10882                  "UpperAlpha", numdelim
10883         end
10884         num = numstr:match("^([a-z])$")
10885         if num then
10886           return string.byte(num) - string.byte("a") + 1,
10887                  "LowerAlpha", numdelim
10888         end
10889         num = numstr:match("^([IVXLCDM]+)")
10890         if num then
10891           return roman2number(num), "UpperRoman", numdelim
10892         end
10893         num = numstr:match("^([ivxlcdm]+)")
10894         if num then
10895           return roman2number(string.upper(num)), "LowerRoman", numdelim
10896         end
10897         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10898       end
10899
10900       local function fancylist(items,tight,start)
10901         local startnum, numstyle, numdelim
10902           = sniffstyle(start[2][1], start[2][2])
10903         return writer.fancylist(items,tight,
10904                                 options.startNumber and startnum or 1,
```

```
                                numstyle or "Decimal",
                                numdelim or "Default")
      end

      local FancyListOfType
        = function(start_marker, continuation_marker, delimiter_type)
          local enumerator_start
            = enumerator(start_marker, continuation_marker,
                         delimiter_type)
          local enumerator_cont
            = enumerator(continuation_marker, continuation_marker,
                         delimiter_type)
          return Cg(enumerator_start, "listtype")
               * (Ct( TightListItem(Cb("listtype"))
                   * ((parsers.check_minimal_indent / "")
                   * TightListItem(enumerator_cont))^0)
               * Cc(true)
               * -#((parsers.conditionally_indented_blankline^0 / "")
                   * parsers.check_minimal_indent * enumerator_cont)
               + Ct( LooseListItem(Cb("listtype"))
                   * ((parsers.conditionally_indented_blankline^0 / "")
                     * (parsers.check_minimal_indent / "")
                     * LooseListItem(enumerator_cont))^0)
               * Cc(false)
               ) * Ct(Cb("listtype")) / fancylist
        end

      local FancyList
        = join_table_with_func(FancyListOfType, markers_table)

      local ListStarter = starter

      self.update_rule("OrderedList", FancyList)
      self.update_rule("ListStarter", ListStarter)
    end
  }
end
```

#### 3.1.7.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the

syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```
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),"}{",self.infostring(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
```

```
10981       local writer = self.writer
10982
10983       local function captures_geq_length(_,i,a,b)
10984         return #a >= #b and i
10985       end
10986
10987       local function strip_enclosing_whitespaces(str)
10988         return str:gsub("^%s*(.-)%s*$", "%1")
10989       end
10990
10991       local tilde_infostring = Cs(Cs((V("HtmlEntity")
10992                                     + parsers.anyescaped
10993                                     - parsers.newline)^0)
10994                                / strip_enclosing_whitespaces)
10995
10996       local backtick_infostring
10997         = Cs( Cs((V("HtmlEntity")
10998             + ( -#(parsers.backslash * parsers.backtick)
10999               * parsers.anyescaped)
11000               - parsers.newline
11001               - parsers.backtick)^0)
11002         / strip_enclosing_whitespaces)
11003
11004       local fenceindent
11005
11006       local function has_trail(indent_table)
11007         return indent_table ~= nil and
11008           indent_table.trail ~= nil and
11009           next(indent_table.trail) ~= nil
11010       end
11011
11012       local function has_indents(indent_table)
11013         return indent_table ~= nil and
11014           indent_table.indents ~= nil and
11015           next(indent_table.indents) ~= nil
11016       end
11017
11018       local function get_last_indent_name(indent_table)
11019         if has_indents(indent_table) then
11020           return indent_table.indents[#indent_table.indents].name
11021         end
11022       end
11023
11024       local count_fenced_start_indent =
11025         function(_, _, indent_table, trail)
11026           local last_indent_name = get_last_indent_name(indent_table)
11027           fenceindent = 0
```

```
11028           if last_indent_name ~= "li" then
11029             fenceindent = #trail
11030           end
11031           return true
11032         end
11033
11034       local fencehead = function(char, infostring)
11035         return Cmt( Cb("indent_info")
11036                   * parsers.check_trail, count_fenced_start_indent)
11037              * Cg(char^3, "fencelength")
11038              * parsers.optionalspace
11039              * infostring
11040              * (parsers.newline + parsers.eof)
11041       end
11042
11043       local fencetail = function(char)
11044         return parsers.check_trail_no_rem
11045              * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11046              * parsers.optionalspace * (parsers.newline + parsers.eof)
11047              + parsers.eof
11048       end
11049
11050       local process_fenced_line =
11051         function(s, i, -- luacheck: ignore s i
11052                  indent_table, line_content, is_blank)
11053           local remainder = ""
11054           if has_trail(indent_table) then
11055             remainder = indent_table.trail.internal_remainder
11056           end
11057
11058           if is_blank
11059              and get_last_indent_name(indent_table) == "li" then
11060             remainder = ""
11061           end
11062
11063           local str = remainder .. line_content
11064           local index = 1
11065           local remaining = fenceindent
11066
11067           while true do
11068             local c = str:sub(index, index)
11069             if c == " " and remaining > 0 then
11070               remaining = remaining - 1
11071               index = index + 1
11072             elseif c == "\t" and remaining > 3 then
11073               remaining = remaining - 4
11074               index = index + 1
```

```
        else
          break
        end
      end

      return true, str:sub(index)
    end

  local fencedline = function(char)
    return Cmt( Cb("indent_info")
              * C(parsers.line - fencetail(char))
              * Cc(false), process_fenced_line)
  end

  local blankfencedline
    = Cmt( Cb("indent_info")
         * C(parsers.blankline)
         * Cc(true), process_fenced_line)

  local TildeFencedCode
    = fencehead(parsers.tilde, tilde_infostring)
    * Cs(( (parsers.check_minimal_blank_indent / "")
         * blankfencedline
         + ( parsers.check_minimal_indent / "")
           * fencedline(parsers.tilde))^0)
    * ( (parsers.check_minimal_indent / "")
      * fencetail(parsers.tilde) + parsers.succeed)

  local BacktickFencedCode
         = fencehead(parsers.backtick, backtick_infostring)
         * Cs(( (parsers.check_minimal_blank_indent / "")
              * blankfencedline
              + (parsers.check_minimal_indent / "")
              * fencedline(parsers.backtick))^0)
         * ( (parsers.check_minimal_indent / "")
           * fencetail(parsers.backtick) + parsers.succeed)

  local infostring_with_attributes
                    = Ct(C((parsers.linechar
                          - ( parsers.optionalspace
                            * parsers.attributes))^0)
                        * parsers.optionalspace
                        * Ct(parsers.attributes))

  local FencedCode
    = ((TildeFencedCode + BacktickFencedCode)
    / function(infostring, code)
```

```
11122           local expanded_code = self.expandtabs(code)
11123
11124           if allow_raw_blocks then
11125             local raw_attr = lpeg.match(parsers.raw_attribute,
11126                                         infostring)
11127             if raw_attr then
11128               return writer.rawBlock(expanded_code, raw_attr)
11129             end
11130           end
11131
11132           local attr = nil
11133           if allow_attributes then
11134             local match = lpeg.match(infostring_with_attributes,
11135                                      infostring)
11136             if match then
11137               infostring, attr = table.unpack(match)
11138             end
11139           end
11140           return writer.fencedCode(expanded_code, infostring, attr)
11141         end)
11142
11143       self.insert_pattern("Block after Verbatim",
11144                           FencedCode, "FencedCode")
11145
11146       local fencestart
11147       if blank_before_code_fence then
11148         fencestart = parsers.fail
11149       else
11150         fencestart = fencehead(parsers.backtick, backtick_infostring)
11151                    + fencehead(parsers.tilde, tilde_infostring)
11152       end
11153
11154       self.update_rule("EndlineExceptions", function(previous_pattern)
11155         if previous_pattern == nil then
11156           previous_pattern = parsers.EndlineExceptions
11157         end
11158         return previous_pattern + fencestart
11159       end)
11160
11161       self.add_special_character("`")
11162       self.add_special_character("~")
11163     end
11164   }
11165 end
```

#### 3.1.7.7 Fenced Divs

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

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

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

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

```
      function self.div_end()
        return self.pop_attributes("div")
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer
```

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

```
      local fenced_div_infostring
                              = C((parsers.linechar
                                 - ( parsers.spacechar^1
                                   * parsers.colon^1))^1)

      local fenced_div_begin = parsers.nonindentspace
                             * parsers.colon^3
                             * parsers.optionalspace
                             * fenced_div_infostring
                             * ( parsers.spacechar^1
                               * parsers.colon^1)^0
                             * parsers.optionalspace
                             * (parsers.newline + parsers.eof)

      local fenced_div_end = parsers.nonindentspace
                           * parsers.colon^3
```

```
                          * parsers.optionalspace
                          * (parsers.newline + parsers.eof)
```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```
    self.initialize_named_group("fenced_div_level", "0")
    self.initialize_named_group("fenced_div_num_opening_indents")

    local function increment_div_level()
      local push_indent_table =
        function(s, i, indent_table, -- luacheck: ignore s i
                 fenced_div_num_opening_indents, fenced_div_level)
          fenced_div_level = tonumber(fenced_div_level) + 1
          local num_opening_indents = 0
          if indent_table.indents ~= nil then
            num_opening_indents = #indent_table.indents
          end
          fenced_div_num_opening_indents[fenced_div_level]
            = num_opening_indents
          return true, fenced_div_num_opening_indents
        end

      local increment_level =
        function(s, i, fenced_div_level) -- luacheck: ignore s i
          fenced_div_level = tonumber(fenced_div_level) + 1
          return true, tostring(fenced_div_level)
        end

      return Cg( Cmt( Cb("indent_info")
                    * Cb("fenced_div_num_opening_indents")
                    * Cb("fenced_div_level"), push_indent_table)
               , "fenced_div_num_opening_indents")
           * Cg( Cmt( Cb("fenced_div_level"), increment_level)
               , "fenced_div_level")
    end

    local function decrement_div_level()
      local pop_indent_table =
        function(s, i, -- luacheck: ignore s i
                 fenced_div_indent_table, fenced_div_level)
          fenced_div_level = tonumber(fenced_div_level)
          fenced_div_indent_table[fenced_div_level] = nil
          return true, tostring(fenced_div_level - 1)
```

```
11241           end
11242
11243         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11244                       * Cb("fenced_div_level"), pop_indent_table)
11245                  , "fenced_div_level")
11246       end
11247
11248
11249       local non_fenced_div_block
11250         = parsers.check_minimal_indent * V("Block")
11251         - parsers.check_minimal_indent_and_trail * fenced_div_end
11252
11253       local non_fenced_div_paragraph
11254         = parsers.check_minimal_indent * V("Paragraph")
11255         - parsers.check_minimal_indent_and_trail * fenced_div_end
11256
11257       local blank = parsers.minimally_indented_blank
11258
11259       local block_separated = parsers.block_sep_group(blank)
11260                             * non_fenced_div_block
11261
11262       local loop_body_pair
11263         = parsers.create_loop_body_pair(block_separated,
11264                                         non_fenced_div_paragraph,
11265                                         parsers.block_sep_group(blank),
11266                                         parsers.par_sep_group(blank))
11267
11268       local content_loop  = ( non_fenced_div_block
11269                             * loop_body_pair.block^0
11270                             + non_fenced_div_paragraph
11271                             * block_separated
11272                             * loop_body_pair.block^0
11273                             + non_fenced_div_paragraph
11274                             * loop_body_pair.par^0)
11275                           * blank^0
11276
11277       local FencedDiv = fenced_div_begin
11278                       / function (infostring)
11279                           local attr
11280                             = lpeg.match(Ct(parsers.attributes),
11281                                          infostring)
11282                           if attr == nil then
11283                             attr = {"." .. infostring}
11284                           end
11285                           return attr
11286                         end
11287                       / writer.div_begin
```

```
                    * increment_div_level()
                    * parsers.skipblanklines
                    * Ct(content_loop)
                    * parsers.minimally_indented_blank^0
                    * parsers.check_minimal_indent_and_trail
                    * fenced_div_end
                    * decrement_div_level()
                    * (Cc("") / writer.div_end)

    self.insert_pattern("Block after Verbatim",
                        FencedDiv, "FencedDiv")

    self.add_special_character(":")

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```
    local function is_inside_div()
      local check_div_level =
        function(s, i, fenced_div_level) -- luacheck: ignore s i
          fenced_div_level = tonumber(fenced_div_level)
          return fenced_div_level > 0
        end

      return Cmt(Cb("fenced_div_level"), check_div_level)
    end

    local function check_indent()
      local compare_indent =
        function(s, i, indent_table, -- luacheck: ignore s i
                 fenced_div_num_opening_indents, fenced_div_level)
          fenced_div_level = tonumber(fenced_div_level)
          local num_current_indents
            = ( indent_table.current_line_indents ~= nil and
                #indent_table.current_line_indents) or 0
          local num_opening_indents
            = fenced_div_num_opening_indents[fenced_div_level]
          return num_current_indents == num_opening_indents
        end

      return Cmt( Cb("indent_info")
                * Cb("fenced_div_num_opening_indents")
                * Cb("fenced_div_level"), compare_indent)
    end

    local fencestart = is_inside_div()
```

```
                        * fenced_div_end
                        * check_indent()

      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
```

#### 3.1.7.8 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 function strip_atx_end(s)
        return s:gsub("%s+#*%s*$","")
      end

      local AtxHeading = Cg(parsers.heading_start, "level")
                       * parsers.optionalspace
                       * (C(((parsers.linechar
                              - (parsers.attributes
                                * parsers.optionalspace
                                * parsers.newline))
                             * (parsers.linechar
                               - parsers.lbrace)^0)^1)
                             / strip_atx_end
                             / parsers.parse_heading_text)
                       * Cg(Ct(parsers.newline
                              + (parsers.attributes
                                * parsers.optionalspace
                                * parsers.newline)), "attributes")
                       * Cb("level")
                       * Cb("attributes")
```

```
11373                       / writer.heading
11374
11375       local function strip_trailing_spaces(s)
11376         return s:gsub("%s*$","")
11377       end
11378
11379       local heading_line  = (parsers.linechar
11380                             - (parsers.attributes
11381                               * parsers.optionalspace
11382                               * parsers.newline))^1
11383                           - parsers.thematic_break_lines
11384
11385       local heading_text
11386         = heading_line
11387         * ( (V("Endline") / "\n")
11388           * (heading_line - parsers.heading_level))^0
11389         * parsers.newline^-1
11390
11391       local SetextHeading
11392         = parsers.freeze_trail * parsers.check_trail_no_rem
11393         * #(heading_text
11394            * (parsers.attributes
11395              * parsers.optionalspace
11396              * parsers.newline)^-1
11397            * parsers.check_minimal_indent
11398            * parsers.check_trail
11399            * parsers.heading_level)
11400         * Cs(heading_text) / strip_trailing_spaces
11401         / parsers.parse_heading_text
11402         * Cg(Ct((parsers.attributes
11403               * parsers.optionalspace
11404               * parsers.newline)^-1), "attributes")
11405         * parsers.check_minimal_indent_and_trail * parsers.heading_level
11406         * Cb("attributes")
11407         * parsers.newline
11408         * parsers.unfreeze_trail
11409         / writer.heading
11410
11411       local Heading = AtxHeading + SetextHeading
11412       self.update_rule("Heading", Heading)
11413     end
11414   }
11415 end
```

#### 3.1.7.9 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
```

#### 3.1.7.10 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
        buffer[#buffer + 1] = lines[#lines]

        return {"\\markdownRendererLineBlockBegin\n"
                  ,buffer,
                  "\n\\markdownRendererLineBlockEnd "}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

```

```
      local LineBlock
        = Ct((Cs(( (parsers.pipe * parsers.space) / ""
                 * ((parsers.space)/entities.char_entity("nbsp"))^0
                 * parsers.linechar^0 * (parsers.newline/""))
                 * (-parsers.pipe
                   * (parsers.space^1/" ")
                   * parsers.linechar^1
                   * (parsers.newline/"")
                   )^0
                 * (parsers.blankline/"")^0)
              / self.parser_functions.parse_inlines)^1)
        / writer.lineblock

      self.insert_pattern("Block after Blockquote",
                          LineBlock, "LineBlock")
    end
  }
end
```

#### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

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

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

```
      function self.mark(s)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererMark{", s, "}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local doubleequals = P("==")

      local Mark
        = parsers.between(V("Inline"), doubleequals, doubleequals)
        / function (inlines) return writer.mark(inlines) end

      self.add_special_character("=")
      self.insert_pattern("Inline before LinkAndEmph",
                          Mark, "Mark")
    end
  }
```

```
end
```

#### 3.1.7.12 Link Attributes

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

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

```

The following patterns define link reference definitions with attributes.

```
      local define_reference_parser
        = (parsers.check_trail / "")
        * parsers.link_label
        * parsers.colon
        * parsers.spnlc * parsers.url
        * ( parsers.spnlc_sep * parsers.title
          * (parsers.spnlc * Ct(parsers.attributes))
          * parsers.only_blank
          + parsers.spnlc_sep * parsers.title * parsers.only_blank
          + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
          * parsers.only_blank
          + Cc("") * parsers.only_blank)

      local ReferenceWithAttributes = define_reference_parser
                                    / self.register_link

      self.update_rule("Reference", ReferenceWithAttributes)

```

The following patterns define direct and indirect links with attributes.

```

      local LinkWithAttributesAndEmph
        = Ct(parsers.link_and_emph_table * Cg(Cc(true),
             "match_link_attributes"))
        / self.defer_link_and_emphasis_processing

      self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)

```

The following patterns define autolinks with attributes.

```
      local AutoLinkUrlWithAttributes
                      = parsers.auto_link_url
```

```
                      * Ct(parsers.attributes)
                      / self.auto_link_url

      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
```

#### 3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```
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.

```
11569       function self.note(s)
11570         if self.flatten_inlines then return "" end
11571         return {"\\markdownRendererNote{",s,"}"}
11572       end
11573     end, extend_reader = function(self)
11574       local parsers = self.parsers
11575       local writer = self.writer
11576 
11577       local rawnotes = parsers.rawnotes
11578 
11579       if inline_notes then
11580         local InlineNote
11581           = parsers.circumflex
11582           * ( parsers.link_label
11583             / self.parser_functions.parse_inlines_no_inline_note)
11584           / writer.note
11585 
11586         self.insert_pattern("Inline after LinkAndEmph",
11587                             InlineNote, "InlineNote")
11588       end
11589       if notes then
11590         local function strip_first_char(s)
11591           return s:sub(2)
11592         end
11593 
11594         local RawNoteRef
11595                       = #(parsers.lbracket * parsers.circumflex)
11596                       * parsers.link_label / strip_first_char
11597 
11598         -- like indirect_link
11599         local function lookup_note(ref)
11600           return writer.defer_call(function()
11601             local found = rawnotes[self.normalize_tag(ref)]
11602             if found then
11603               return writer.note(
11604                 self.parser_functions.parse_blocks_nested(found))
11605             else
11606               return {"[",
11607                 self.parser_functions.parse_inlines("^" .. ref), "]"}
11608             end
11609           end)
11610         end
11611 
11612         local function register_note(ref,rawnote)
11613           local normalized_tag = self.normalize_tag(ref)
```

```
11614           if rawnotes[normalized_tag] == nil then
11615             rawnotes[normalized_tag] = rawnote
11616           end
11617           return ""
11618         end
11619
11620         local NoteRef = RawNoteRef / lookup_note
11621
11622         local optionally_indented_line
11623           = parsers.check_optional_indent_and_any_trail * parsers.line
11624
11625         local blank
11626           = parsers.check_optional_blank_indent_and_any_trail
11627           * parsers.optionalspace * parsers.newline
11628
11629         local chunk
11630           = Cs(parsers.line
11631           * (optionally_indented_line - blank)^0)
11632
11633         local indented_blocks = function(bl)
11634           return Cs( bl
11635                 * ( blank^1 * (parsers.check_optional_indent / "")
11636                   * parsers.check_code_trail
11637                   * -parsers.blankline * bl)^0)
11638         end
11639
11640         local NoteBlock
11641                     = parsers.check_trail_no_rem
11642                     * RawNoteRef * parsers.colon
11643                     * parsers.spnlc * indented_blocks(chunk)
11644                     / register_note
11645
11646         local Reference = NoteBlock + parsers.Reference
11647
11648         self.update_rule("Reference", Reference)
11649         self.insert_pattern("Inline before LinkAndEmph",
11650                             NoteRef, "NoteRef")
11651       end
11652
11653       self.add_special_character("^")
11654     end
11655   }
11656 end
```

#### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syn-

tax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```
11657 M.extensions.pipe_tables = function(table_captions, table_attributes)
11658
11659   local function make_pipe_table_rectangular(rows)
11660     local num_columns = #rows[2]
11661     local rectangular_rows = {}
11662     for i = 1, #rows do
11663       local row = rows[i]
11664       local rectangular_row = {}
11665       for j = 1, num_columns do
11666         rectangular_row[j] = row[j] or ""
11667       end
11668       table.insert(rectangular_rows, rectangular_row)
11669     end
11670     return rectangular_rows
11671   end
11672
11673   local function pipe_table_row(allow_empty_first_column
11674                                 , nonempty_column
11675                                 , column_separator
11676                                 , column)
11677     local row_beginning
11678     if allow_empty_first_column then
11679       row_beginning = -- empty first column
11680                       #(parsers.spacechar^4
11681                         * column_separator)
11682                     * parsers.optionalspace
11683                     * column
11684                     * parsers.optionalspace
11685                     -- non-empty first column
11686                     + parsers.nonindentspace
11687                     * nonempty_column^-1
11688                     * parsers.optionalspace
11689     else
11690       row_beginning = parsers.nonindentspace
11691                     * nonempty_column^-1
11692                     * parsers.optionalspace
11693     end
11694
11695     return Ct(row_beginning
11696             * (-- single column with no leading pipes
11697                #(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, attributes)
      if not self.is_writing then return "" end
      local buffer = {}
      if attributes ~= nil then
        table.insert(buffer,
                     "\\markdownRendererTableAttributeContextBegin\n")
        table.insert(buffer, self.attributes(attributes))
      end
      table.insert(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
```

```
11741           table.insert(buffer, "}")
11742         end
11743         if attributes ~= nil then
11744           table.insert(buffer,
11745                        "\\markdownRendererTableAttributeContextEnd{}")
11746         end
11747         return buffer
11748       end
11749     end, extend_reader = function(self)
11750       local parsers = self.parsers
11751       local writer = self.writer
11752
11753       local table_hline_separator = parsers.pipe + parsers.plus
11754
11755       local table_hline_column = (parsers.dash
11756                                   - #(parsers.dash
11757                                       * (parsers.spacechar
11758                                         + table_hline_separator
11759                                         + parsers.newline)))^1
11760                                 * (parsers.colon * Cc("r")
11761                                   + parsers.dash * Cc("d"))
11762                                 + parsers.colon
11763                                 * (parsers.dash
11764                                   - #(parsers.dash
11765                                       * (parsers.spacechar
11766                                         + table_hline_separator
11767                                         + parsers.newline)))^1
11768                                 * (parsers.colon * Cc("c")
11769                                   + parsers.dash * Cc("l"))
11770
11771       local table_hline = pipe_table_row(false
11772                                          , table_hline_column
11773                                          , table_hline_separator
11774                                          , table_hline_column)
11775
11776       local table_caption_beginning
11777         = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
11778           * parsers.optionalspace * parsers.newline)^0
11779         * parsers.check_minimal_indent_and_trail
11780         * (P("Table")^-1 * parsers.colon)
11781         * parsers.optionalspace
11782
11783       local function strip_trailing_spaces(s)
11784         return s:gsub("%s*$","")
11785       end
11786
11787       local table_row
```

```
      = pipe_table_row(true
                       , (C((parsers.linechar - parsers.pipe)^1)
                         / strip_trailing_spaces
                         / self.parser_functions.parse_inlines)
                       , parsers.pipe
                       , (C((parsers.linechar - parsers.pipe)^0)
                         / strip_trailing_spaces
                         / self.parser_functions.parse_inlines))

    local table_caption
    if table_captions then
      table_caption = #table_caption_beginning
                    * table_caption_beginning
      if table_attributes then
        table_caption = table_caption
                      * (C(((( parsers.linechar
                             - (parsers.attributes
                               * parsers.optionalspace
                               * parsers.newline
                               * -#( parsers.optionalspace
                                   * parsers.linechar)))
                           + ( parsers.newline
                             * #( parsers.optionalspace
                                * parsers.linechar)
                             * C(parsers.optionalspace)
                             / writer.space))
                          * (parsers.linechar
                            - parsers.lbrace)^0)^1)
                         / self.parser_functions.parse_inlines)
                      * (parsers.newline
                        + ( Ct(parsers.attributes)
                          * parsers.optionalspace
                          * parsers.newline))
      else
        table_caption = table_caption
                      * C(( parsers.linechar
                          + ( parsers.newline
                            * #( parsers.optionalspace
                               * parsers.linechar)
                            * C(parsers.optionalspace)
                            / writer.space))^1)
                      / self.parser_functions.parse_inlines
                      * parsers.newline
      end
    else
      table_caption = parsers.fail
    end
```

```
11835
11836       local PipeTable
11837         = Ct( table_row * parsers.newline
11838             * (parsers.check_minimal_indent_and_trail / {})
11839           * table_hline * parsers.newline
11840           * ( (parsers.check_minimal_indent / {})
11841             * table_row * parsers.newline)^0)
11842         / make_pipe_table_rectangular
11843         * table_caption^-1
11844         / writer.table
11845
11846       self.insert_pattern("Block after Blockquote",
11847                           PipeTable, "PipeTable")
11848     end
11849   }
11850 end
```

#### 3.1.7.15 Raw Attributes

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

```
11851 M.extensions.raw_inline = function()
11852   return {
11853     name = "built-in raw_inline syntax extension",
11854     extend_writer = function(self)
11855       local options = self.options
11856
```

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

```
11857       function self.rawInline(s, attr)
11858         if not self.is_writing then return "" end
11859         if self.flatten_inlines then return s end
11860         local name = util.cache_verbatim(options.cacheDir, s)
11861         return {"\\markdownRendererInputRawInline{",
11862                 name,"}{", self.string(attr),"}"}
11863       end
11864     end, extend_reader = function(self)
11865       local writer = self.writer
11866
11867       local RawInline = parsers.inticks
11868                       * parsers.raw_attribute
11869                       / writer.rawInline
11870
11871       self.insert_pattern("Inline before Code",
11872                           RawInline, "RawInline")
11873     end
11874   }
```

```
11875 end
```

#### 3.1.7.16 Strike-Through

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

```
11876 M.extensions.strike_through = function()
11877   return {
11878     name = "built-in strike_through syntax extension",
11879     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.

```
11880       function self.strike_through(s)
11881         if self.flatten_inlines then return s end
11882         return {"\\markdownRendererStrikeThrough{",s,"}"}
11883       end
11884     end, extend_reader = function(self)
11885       local parsers = self.parsers
11886       local writer = self.writer
11887
11888       local StrikeThrough = (
11889         parsers.between(parsers.Inline, parsers.doubletildes,
11890                         parsers.doubletildes)
11891       ) / writer.strike_through
11892
11893       self.insert_pattern("Inline after LinkAndEmph",
11894                           StrikeThrough, "StrikeThrough")
11895
11896       self.add_special_character("~")
11897     end
11898   }
11899 end
```

#### 3.1.7.17 Subscripts

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

```
11900 M.extensions.subscripts = function()
11901   return {
11902     name = "built-in subscripts syntax extension",
11903     extend_writer = function(self)
```

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

```
11904       function self.subscript(s)
11905         if self.flatten_inlines then return s end
11906         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 LinkAndEmph",
                          Subscript, "Subscript")

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

#### 3.1.7.18 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)
        if self.flatten_inlines then return s end
        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 LinkAndEmph",
                          Superscript, "Superscript")

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

#### 3.1.7.19 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 self.flatten_inlines then return s 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 self.flatten_inlines then return s 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 * Cs(p * (p - ender)^0) * ender)
      end

      local function strip_preceding_whitespaces(str)
        return str:gsub("^%s*(.-)$", "%1")
      end

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

      local allowed_before_closing_no_space
        = B( parsers.backslash * parsers.any
           + parsers.any * (parsers.nonspacechar - parsers.backslash))

```

The following patterns implement the Pandoc dollar math syntax extension.

```
      local dollar_math_content
        = (parsers.newline * (parsers.check_optional_indent / ""))
        + parsers.backslash^-1
```

```
      * parsers.linechar)
      - parsers.blankline^2
      - parsers.dollar

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

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

    local inline_math_dollars = between(Cs( 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(Cs( 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.newline * (parsers.check_optional_indent / ""))
      + parsers.linechar)
      - 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

    local inline_math_closing_double  = allowed_before_closing
                                      * parsers.spacechar^0
                                      * parsers.backslash
                                      * parsers.backslash
                                      * parsers.rparent

    local inline_math_double  = between(Cs( backslash_math_content),
                                        inline_math_opening_double,
                                        inline_math_closing_double)
```

```
                                / strip_preceding_whitespaces

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

    local display_math_closing_double = allowed_before_closing
                                      * parsers.spacechar^0
                                      * parsers.backslash
                                      * parsers.backslash
                                      * parsers.rbracket

    local display_math_double = between(Cs( backslash_math_content),
                                        display_math_opening_double,
                                        display_math_closing_double)
                                / strip_preceding_whitespaces
```

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

```
    local inline_math_opening_single  = parsers.backslash
                                      * parsers.lparent

    local inline_math_closing_single  = allowed_before_closing
                                      * parsers.spacechar^0
                                      * parsers.backslash
                                      * parsers.rparent

    local inline_math_single  = between(Cs( backslash_math_content),
                                        inline_math_opening_single,
                                        inline_math_closing_single)
                                / strip_preceding_whitespaces

    local display_math_opening_single = parsers.backslash
                                      * parsers.lbracket

    local display_math_closing_single = allowed_before_closing
                                      * parsers.spacechar^0
                                      * parsers.backslash
                                      * parsers.rbracket

    local display_math_single = between(Cs( backslash_math_content),
                                        display_math_opening_single,
                                        display_math_closing_single)
                                / strip_preceding_whitespaces

    local display_math = parsers.fail

    local inline_math = parsers.fail

```

```
12072       if tex_math_dollars then
12073         display_math = display_math + display_math_dollars
12074         inline_math = inline_math + inline_math_dollars
12075       end
12076
12077       if tex_math_double_backslash then
12078         display_math = display_math + display_math_double
12079         inline_math = inline_math + inline_math_double
12080       end
12081
12082       if tex_math_single_backslash then
12083         display_math = display_math + display_math_single
12084         inline_math = inline_math + inline_math_single
12085       end
12086
12087       local TexMath = display_math / writer.display_math
12088                     + inline_math / writer.inline_math
12089
12090       self.insert_pattern("Inline after LinkAndEmph",
12091                           TexMath, "TexMath")
12092
12093       if tex_math_dollars then
12094         self.add_special_character("$")
12095       end
12096
12097       if tex_math_single_backslash or tex_math_double_backslash then
12098         self.add_special_character("\\")
12099         self.add_special_character("[")
12100         self.add_special_character("]")
12101         self.add_special_character(")")
12102         self.add_special_character("(")
12103       end
12104     end
12105   }
12106 end
```

#### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. When both `expect_jekyll_data` and `ensure_jekyll_data` parameters are `true`, then a a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata.

```
12107 M.extensions.jekyll_data = function(expect_jekyll_data,
12108                                     ensure_jekyll_data)
12109   return {
```

```
12110     name = "built-in jekyll_data syntax extension",
12111     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` for the typographic output format used by the `\markdownRendererJekyllDataTypographicString` macro.

```
12112       function self.jekyllData(d, t, p)
12113         if not self.is_writing then return "" end
12114
12115         local buf = {}
12116
12117         local keys = {}
12118         for k, _ in pairs(d) do
12119           table.insert(keys, k)
12120         end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
12121         table.sort(keys, function(first, second)
12122           if type(first) ~= type(second) then
12123             return type(first) < type(second)
12124           else
12125             return first < second
12126           end
12127         end)
12128
12129         if not p then
12130           table.insert(buf, "\\markdownRendererJekyllDataBegin")
12131         end
12132
12133         local is_sequence = false
12134         if #d > 0 and #d == #keys then
12135           for i=1, #d do
12136             if d[i] == nil then
12137               goto not_a_sequence
12138             end
12139           end
12140           is_sequence = true
12141         end
12142         ::not_a_sequence::
12143
12144         if is_sequence then
12145           table.insert(buf,
12146             "\\markdownRendererJekyllDataSequenceBegin{")
```

```
12147           table.insert(buf, self.identifier(p or "null"))
12148           table.insert(buf, "}{")
12149           table.insert(buf, #keys)
12150           table.insert(buf, "}")
12151         else
12152           table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12153           table.insert(buf, self.identifier(p or "null"))
12154           table.insert(buf, "}{")
12155           table.insert(buf, #keys)
12156           table.insert(buf, "}")
12157         end
12158
12159         for _, k in ipairs(keys) do
12160           local v = d[k]
12161           local typ = type(v)
12162           k = tostring(k or "null")
12163           if typ == "table" and next(v) ~= nil then
12164             table.insert(
12165               buf,
12166               self.jekyllData(v, t, k)
12167             )
12168           else
12169             k = self.identifier(k)
12170             v = tostring(v)
12171             if typ == "boolean" then
12172               table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12173               table.insert(buf, k)
12174               table.insert(buf, "}{")
12175               table.insert(buf, v)
12176               table.insert(buf, "}")
12177             elseif typ == "number" then
12178               table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12179               table.insert(buf, k)
12180               table.insert(buf, "}{")
12181               table.insert(buf, v)
12182               table.insert(buf, "}")
12183             elseif typ == "string" then
12184               table.insert(buf,
12185                 "\\markdownRendererJekyllDataProgrammaticString{")
12186               table.insert(buf, k)
12187               table.insert(buf, "}{")
12188               table.insert(buf, self.identifier(v))
12189               table.insert(buf, "}")
12190               table.insert(buf,
12191                 "\\markdownRendererJekyllDataTypographicString{")
12192               table.insert(buf, k)
12193               table.insert(buf, "}{")
```

```
12194             table.insert(buf, t(v))
12195             table.insert(buf, "}")
12196           elseif typ == "table" then
12197             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12198             table.insert(buf, k)
12199             table.insert(buf, "}")
12200           else
12201             local error = self.error(format(
12202               "Unexpected type %s for value of "
12203               .. "YAML key %s.", typ, k))
12204             table.insert(buf, error)
12205           end
12206         end
12207       end
12208
12209       if is_sequence then
12210         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12211       else
12212         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12213       end
12214
12215       if not p then
12216         table.insert(buf, "\\markdownRendererJekyllDataEnd")
12217       end
12218
12219       return buf
12220     end
12221   end, extend_reader = function(self)
12222     local parsers = self.parsers
12223     local writer = self.writer
12224
12225     local JekyllData
12226       = Cmt( C((parsers.line - P("---") - P("..."))^0)
12227            , function(s, i, text) -- luacheck: ignore s i
12228                local data
12229                local ran_ok, _ = pcall(function()
12230                  -- TODO: Use `require("tinyyaml")` in TeX Live 2023
12231                  local tinyyaml = require("markdown-tinyyaml")
12232                  data = tinyyaml.parse(text, {timestamps=false})
12233                end)
12234                if ran_ok and data ~= nil then
12235                  return true, writer.jekyllData(data, function(s)
12236                    return self.parser_functions.parse_blocks_nested(s)
12237                  end, nil)
12238                else
12239                  return false
12240                end
```

```
12241                 end
12242               )
12243
12244       local UnexpectedJekyllData
12245         = P("---")
12246         * parsers.blankline / 0
12247         -- if followed by blank, it's thematic break
12248         * #(-parsers.blankline)
12249         * JekyllData
12250         * (P("---") + P("..."))
12251
12252       local ExpectedJekyllData
12253         = ( P("---")
12254           * parsers.blankline / 0
12255           -- if followed by blank, it's thematic break
12256           * #(-parsers.blankline)
12257           )^-1
12258         * JekyllData
12259         * (P("---") + P("..."))^-1
12260
12261       if ensure_jekyll_data then
12262         ExpectedJekyllData = ExpectedJekyllData
12263                            * parsers.eof
12264       else
12265         ExpectedJekyllData = ( ExpectedJekyllData
12266                              * (V("Blank")^0 / writer.interblocksep)
12267                              )^-1
12268       end
12269
12270       self.insert_pattern("Block before Blockquote",
12271                           UnexpectedJekyllData, "UnexpectedJekyllData")
12272       if expect_jekyll_data then
12273         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12274       end
12275     end
12276   }
12277 end
```

### 3.1.8 Conversion from Markdown to Plain TEX

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its own function `new` unless option `eagerCache` or `finalizeCache` has been enabled and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```
12278 function M.new(options)
```

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

```
12279   options = options or {}
12280   setmetatable(options, { __index = function (_, key)
12281     return defaultOptions[key] end })
```

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```
12282   local parser_convert = nil
12283   return function(input)
12284     local function convert(input)
12285       if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
12286         local parser = require("markdown-parser")
12287         if metadata.version ~= parser.metadata.version then
12288           warn("markdown.lua " .. metadata.version .. " used with " ..
12289                "markdown-parser.lua " .. parser.metadata.version .. ".")
12290         end
12291         parser_convert = parser.new(options)
12292       end
12293       return parser_convert(input)
12294     end
```

If we cache markdown documents, produce the cache file and transform its filename to plain TEX output.

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).

```
12295     local output
12296     if options.eagerCache or options.finalizeCache then
12297       local salt = util.salt(options)
12298       local name = util.cache(options.cacheDir, input, salt, convert,
12299                               ".md.tex")
12300       output = [[\input{]] .. name .. [[}\relax]]
```

Otherwise, return the result of the conversion directly.

```
12301     else
12302       output = convert(input)
12303     end
```

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

```
12304     if options.finalizeCache then
12305       local file, mode
12306       if options.frozenCacheCounter > 0 then
12307         mode = "a"
```

```
12308       else
12309         mode = "w"
12310       end
12311       file = assert(io.open(options.frozenCacheFileName, mode),
12312         [[Could not open file "]] .. options.frozenCacheFileName
12313         .. [[" for writing]])
12314       assert(file:write(
12315         [[\expandafter\global\expandafter\def\csname ]]
12316         .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12317         .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
12318       assert(file:close())
12319     end
12320     return output
12321   end
12322 end
```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain TEX output. See Section 2.1.1.

```
12323 function M.new(options)
```

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

```
12324   options = options or {}
12325   setmetatable(options, { __index = function (_, key)
12326     return defaultOptions[key] end })
```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
12327   if options.singletonCache and singletonCache.convert then
12328     for k, v in pairs(defaultOptions) do
12329       if type(v) == "table" then
12330         for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12331           if singletonCache.options[k][i] ~= options[k][i] then
12332             goto miss
12333           end
12334         end
```

The `cacheDir` option is disregarded.

```
12335       elseif k ~= "cacheDir"
12336          and singletonCache.options[k] ~= options[k] then
12337         goto miss
12338       end
12339     end
12340     return singletonCache.convert
12341   end
12342   ::miss::
```

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

```
12343   local extensions = {}
```

```
12344
12345   if options.bracketedSpans then
12346     local bracketed_spans_extension = M.extensions.bracketed_spans()
12347     table.insert(extensions, bracketed_spans_extension)
12348   end
12349
12350   if options.contentBlocks then
12351     local content_blocks_extension = M.extensions.content_blocks(
12352       options.contentBlocksLanguageMap)
12353     table.insert(extensions, content_blocks_extension)
12354   end
12355
12356   if options.definitionLists then
12357     local definition_lists_extension = M.extensions.definition_lists(
12358       options.tightLists)
12359     table.insert(extensions, definition_lists_extension)
12360   end
12361
12362   if options.fencedCode then
12363     local fenced_code_extension = M.extensions.fenced_code(
12364       options.blankBeforeCodeFence,
12365       options.fencedCodeAttributes,
12366       options.rawAttribute)
12367     table.insert(extensions, fenced_code_extension)
12368   end
12369
12370   if options.fencedDivs then
12371     local fenced_div_extension = M.extensions.fenced_divs(
12372       options.blankBeforeDivFence)
12373     table.insert(extensions, fenced_div_extension)
12374   end
12375
12376   if options.headerAttributes then
12377     local header_attributes_extension = M.extensions.header_attributes()
12378     table.insert(extensions, header_attributes_extension)
12379   end
12380
12381   if options.inlineCodeAttributes then
12382     local inline_code_attributes_extension =
12383       M.extensions.inline_code_attributes()
12384     table.insert(extensions, inline_code_attributes_extension)
12385   end
12386
12387   if options.jekyllData then
12388     local jekyll_data_extension = M.extensions.jekyll_data(
12389       options.expectJekyllData, options.ensureJekyllData)
12390     table.insert(extensions, jekyll_data_extension)
```

```
12391   end
12392
12393   if options.linkAttributes then
12394     local link_attributes_extension =
12395       M.extensions.link_attributes()
12396     table.insert(extensions, link_attributes_extension)
12397   end
12398
12399   if options.lineBlocks then
12400     local line_block_extension = M.extensions.line_blocks()
12401     table.insert(extensions, line_block_extension)
12402   end
12403
12404   if options.mark then
12405     local mark_extension = M.extensions.mark()
12406     table.insert(extensions, mark_extension)
12407   end
12408
12409   if options.pipeTables then
12410     local pipe_tables_extension = M.extensions.pipe_tables(
12411       options.tableCaptions, options.tableAttributes)
12412     table.insert(extensions, pipe_tables_extension)
12413   end
12414
12415   if options.rawAttribute then
12416     local raw_inline_extension = M.extensions.raw_inline()
12417     table.insert(extensions, raw_inline_extension)
12418   end
12419
12420   if options.strikeThrough then
12421     local strike_through_extension = M.extensions.strike_through()
12422     table.insert(extensions, strike_through_extension)
12423   end
12424
12425   if options.subscripts then
12426     local subscript_extension = M.extensions.subscripts()
12427     table.insert(extensions, subscript_extension)
12428   end
12429
12430   if options.superscripts then
12431     local superscript_extension = M.extensions.superscripts()
12432     table.insert(extensions, superscript_extension)
12433   end
12434
12435   if options.texMathDollars or
12436      options.texMathSingleBackslash or
12437      options.texMathDoubleBackslash then
```

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

  if options.notes or options.inlineNotes then
    local notes_extension = M.extensions.notes(
      options.notes, 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

  if options.fancyLists then
    local fancy_lists_extension = M.extensions.fancy_lists()
    table.insert(extensions, fancy_lists_extension)
  end
```

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

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

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

```
      local pathname = assert(kpse.find_file(filename),
        [[Could not locate user-defined syntax extension "]]
        .. filename)
      local input_file = assert(io.open(pathname, "r"),
        [[Could not open user-defined syntax extension "]]
        .. pathname .. [[" for reading]])
      local input = assert(input_file:read("*a"))
      assert(input_file:close())
      local user_extension, err = load([[
        local sandbox = {}
        setmetatable(sandbox, {__index = _G})
        _ENV = sandbox
      ]] .. input)()
      assert(user_extension,
        [[Failed to compile user-defined syntax extension "]]
        .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
      assert(user_extension.api_version ~= nil,
```

```
12480         [[User-defined syntax extension "]] .. pathname
12481         .. [[" does not specify mandatory field "api_version"]])
12482       assert(type(user_extension.api_version) == "number",
12483         [[User-defined syntax extension "]] .. pathname
12484         .. [[" specifies field "api_version" of type "]]
12485         .. type(user_extension.api_version)
12486         .. [[" but "number" was expected]])
12487       assert(user_extension.api_version > 0
12488          and user_extension.api_version
12489           <= metadata.user_extension_api_version,
12490         [[User-defined syntax extension "]] .. pathname
12491         .. [[" uses syntax extension API version "]]
12492         .. user_extension.api_version .. [[ but markdown.lua ]]
12493         .. metadata.version .. [[ uses API version ]]
12494         .. metadata.user_extension_api_version
12495         .. [[, which is incompatible]])
12496
12497       assert(user_extension.grammar_version ~= nil,
12498         [[User-defined syntax extension "]] .. pathname
12499         .. [[" does not specify mandatory field "grammar_version"]])
12500       assert(type(user_extension.grammar_version) == "number",
12501         [[User-defined syntax extension "]] .. pathname
12502         .. [[" specifies field "grammar_version" of type "]]
12503         .. type(user_extension.grammar_version)
12504         .. [[" but "number" was expected]])
12505       assert(user_extension.grammar_version == metadata.grammar_version,
12506         [[User-defined syntax extension "]] .. pathname
12507         .. [[" uses grammar version "]]
12508         .. user_extension.grammar_version
12509         .. [[ but markdown.lua ]] .. metadata.version
12510         .. [[ uses grammar version ]] .. metadata.grammar_version
12511         .. [[, which is incompatible]])
12512
12513       assert(user_extension.finalize_grammar ~= nil,
12514         [[User-defined syntax extension "]] .. pathname
12515         .. [[" does not specify mandatory "finalize_grammar" field]])
12516       assert(type(user_extension.finalize_grammar) == "function",
12517         [[User-defined syntax extension "]] .. pathname
12518         .. [[" specifies field "finalize_grammar" of type "]]
12519         .. type(user_extension.finalize_grammar)
12520         .. [[" but "function" was expected]])
```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```
12521       local extension = {
12522         name = [[user-defined "]] .. pathname .. [[" syntax extension]],
12523         extend_reader = user_extension.finalize_grammar,
```

```
        extend_writer = function() end,
      }
      return extension
    end)(user_extension_filename)
    table.insert(extensions, user_extension)
  end
```

Produce 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)
```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```
  collectgarbage("collect")
```

Update the singleton cache.

```
  if options.singletonCache then
    local singletonCacheOptions = {}
    for k, v in pairs(options) do
      singletonCacheOptions[k] = v
    end
    setmetatable(singletonCacheOptions,
      { __index = function (_, key)
        return defaultOptions[key] end })
    singletonCache.options = singletonCacheOptions
    singletonCache.convert = convert
  end
```

Return the conversion function from markdown to plain TEX.

```
  return convert
end

return M
```

### 3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

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
```

```
12558
```

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

```
12559 local lfs = require("lfs")
12560 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12561   assert(lfs.mkdir(options["cacheDir"]))
12562 end
```

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

```
12563 local kpse
12564 (function()
12565   local should_initialize = package.loaded.kpse == nil
12566                         or tex.initialize ~= nil
12567   kpse = require("kpse")
12568   if should_initialize then
12569     kpse.set_program_name("luatex")
12570   end
12571 end)()
12572 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.

```
12573 if metadata.version ~= md.metadata.version then
12574   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12575        "markdown.lua " .. md.metadata.version .. ".")
12576 end
12577 local convert = md.new(options)
12578 local output = convert(input)
12579
12580 if output_filename then
12581   local output_file = assert(io.open(output_filename, "w"),
12582     [[Could not open file "]] .. output_filename .. [[" for writing]])
12583   assert(output_file:write(output))
12584   assert(output_file:close())
12585 else
12586   assert(io.write(output))
12587 end
```

Remove the `options.cacheDir` directory if it is empty.

```
12588 if options.cacheDir then
12589   lfs.rmdir(options.cacheDir)
12590 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

```
\ExplSyntaxOn
\cs_if_free:NT
  \markdownInfo
  {
    \cs_new:Npn
      \markdownInfo #1
      {
        \msg_info:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownWarning
  {
    \cs_new:Npn
      \markdownWarning #1
      {
        \msg_warning:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownError
  {
    \cs_new:Npn
      \markdownError #1 #2
      {
        \msg_error:nnee
          { markdown }
          { generic-message-with-help-text }
          { #1 }
          { #2 }
      }
  }
\msg_new:nnn
  { markdown }
  { generic-message }
  { #1 }
```

```
\msg_new:nnnn
  { markdown }
  { generic-message-with-help-text }
  { #1 }
  { #2 }
\cs_generate_variant:Nn
  \msg_info:nnn
  { nne }
\cs_generate_variant:Nn
  \msg_warning:nnn
  { nne }
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnee }
\ExplSyntaxOff
```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain TEX themes provided with the Markdown package.

```
\ExplSyntaxOn
\prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
\prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
\cs_new:Nn
  \@@_plain_tex_load_theme:nnn
  {
    \prop_get:NnNTF
      \g_@@_plain_tex_loaded_themes_linenos_prop
      { #1 }
      \l_tmpa_tl
      {
        \prop_get:NnN
          \g_@@_plain_tex_loaded_themes_versions_prop
          { #1 }
          \l_tmpb_tl
        \str_if_eq:nVTF
          { #2 }
          \l_tmpb_tl
          {
            \msg_warning:nnnVn
              { markdown }
              { repeatedly-loaded-plain-tex-theme }
              { #1 }
              \l_tmpa_tl
              { #2 }
          }
```

```
          {
            \msg_error:nnnnVV
              { markdown }
              { different-versions-of-plain-tex-theme }
              { #1 }
              { #2 }
              \l_tmpb_tl
              \l_tmpa_tl
          }
      }
      {
        \msg_info:nnnn
          { markdown }
          { loading-plain-tex-theme }
          { #1 }
          { #2 }
        \prop_gput:Nnx
          \g_@@_plain_tex_loaded_themes_linenos_prop
          { #1 }
          { \tex_the:D \tex_inputlineno:D }
        \prop_gput:Nnn
          \g_@@_plain_tex_loaded_themes_versions_prop
          { #1 }
          { #2 }
        \file_input:n
          { markdown theme #3 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-plain-tex-theme }
  { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-plain-tex-theme }
  {
    Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\msg_new:nnn
  { markdown }
  { different-versions-of-plain-tex-theme }
  {
    Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
    but~version~#3~has~already~been~loaded~on~line~#4
  }
\cs_generate_variant:Nn
```

```
  \prop_gput:Nnn
  { Nnx }
\cs_gset_eq:NN
  \@@_load_theme:nnn
  \@@_plain_tex_load_theme:nnn
\cs_generate_variant:Nn
  \@@_load_theme:nnn
  { VeV }
\cs_generate_variant:Nn
  \msg_error:nnnnnn
  { nnnnVV }
\cs_generate_variant:Nn
  \msg_warning:nnnnn
  { nnnVn }
```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as LaTeX and ConTeXt.

```
\cs_new:Npn
  \markdownLoadPlainTeXTheme
  {
```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_reverse:N
      \l_tmpa_tl
    \tl_set:Ne
      \l_tmpb_tl
      {
        \tl_tail:V
          \l_tmpa_tl
      }
    \tl_reverse:N
      \l_tmpb_tl
```

Next, we munge the theme name.

```
    \str_set:NV
      \l_tmpa_str
      \l_tmpb_tl
    \str_replace_all:Nnn
      \l_tmpa_str
      { / }
      { _ }
```

Finally, we load the plain TeX theme.

```
    \@@_plain_tex_load_theme:VeV
      \l_tmpb_tl
      { \markdownThemeVersion }
      \l_tmpa_str
  }
\cs_generate_variant:Nn
  \tl_set:Nn
  { Ne }
\cs_generate_variant:Nn
  \@@_plain_tex_load_theme:nnn
  { VeV }
\ExplSyntaxOff
```

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

```
\markdownSetup {
  rendererPrototypes = {
    tilde = {~},
  },
}
```

The witiko/markdown/defaults plain TEX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererParagraphSeparatorPrototype{%
  \markdownRendererInterblockSeparator}%
\def\markdownRendererHardLineBreakPrototype{\hfil\break}%
\def\markdownRendererSoftLineBreakPrototype{ }%
\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}{#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#3{%
```

```
  \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
  { \codepoint_str_generate:n { fffd } }
\ExplSyntaxOff
\def\markdownRendererSectionBeginPrototype{}%
\def\markdownRendererSectionEndPrototype{}%
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererWarningPrototype
  #1#2#3#4
  {
    \tl_set:Nn
      \l_tmpa_tl
      { #2 }
    \tl_if_empty:nF
      { #4 }
      {
        \tl_put_right:Nn
          \l_tmpa_tl
          { \iow_newline: #4 }
      }
    \exp_args:NV
      \markdownWarning
      \l_tmpa_tl
  }
\ExplSyntaxOff
\def\markdownRendererErrorPrototype#1#2#3#4{%
  \markdownError{#2}{#4}}%
```

#### 3.2.3.1 Raw Attributes

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

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_plain_tex_default_input_raw_inline:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} \unskip }
      }
```

```
  }
\cs_new:Nn
  \@@_plain_tex_default_input_raw_block:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \@@_plain_tex_default_input_raw_inline:nn
      { #1 }
      { #2 }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \@@_plain_tex_default_input_raw_block:nn
      { #1 }
      { #2 }
  }
\ExplSyntaxOff
```

#### 3.2.3.2 YAML Metadata Renderer Prototypes

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

`\c_@@_jekyll_data_sequence_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 }
}
```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```
\def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
\def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\ExplSyntaxOff
```

If plain TEX is the top layer, we load the `witiko/markdown/defaults` plain TEX theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
\ExplSyntaxOn
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \ExplSyntaxOff
    \@@_if_option:nF
      { noDefaults }
      {
        \@@_if_option:nTF
          { experimental }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults@experimental }
          }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults }
          }
      }
    \ExplSyntaxOn
```

```
  }
\ExplSyntaxOff
```

### 3.2.4 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
            {
              \@@_lua_escape:xN
                { ##1 }
                \l_tmpb_tl
              \tl_gput_right:Nn
                \g_@@_formatted_lua_options_tl
                { " }
              \tl_gput_right:NV
                \g_@@_formatted_lua_options_tl
                \l_tmpb_tl
              \tl_gput_right:Nn
                \g_@@_formatted_lua_options_tl
                { " ,~ }
            }
          \tl_gput_right:Nx
            \g_@@_formatted_lua_options_tl
            { \c_right_brace_str ,~ }
        }
    }
    {
      \@@_get_option_value:nN
        { #1 }
        \l_tmpa_tl
      \@@_lua_escape:xN
        { \l_tmpa_tl }
        \l_tmpb_tl
      \tl_gput_right:Nn
        \g_@@_formatted_lua_options_tl
        { #1~=~ " }
      \tl_gput_right:NV
        \g_@@_formatted_lua_options_tl
        \l_tmpb_tl
      \tl_gput_right:Nn
```

```
          \g_@@_formatted_lua_options_tl
          { " ,~ }
      }
  }
\cs_generate_variant:Nn
  \clist_map_inline:nn
  { Vn }
\let\markdownPrepareLuaOptions=\@@_format_lua_options:
\def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
\sys_if_engine_luatex:TF
  {
    \cs_new:Nn
      \@@_lua_escape:nN
      {
        \tl_set:Nx
          #2
          {
            \lua_escape:n
              { #1 }
          }
      }
  }
  {
    \regex_const:Nn
      \c_@@_lua_escape_regex
      { [\\"'] }
    \cs_new:Nn
      \@@_lua_escape:nN
      {
        \tl_set:Nn
          #2
          { #1 }
        \regex_replace_all:NnN
          \c_@@_lua_escape_regex
          { \u { c_backslash_str } \0 }
          #2
      }
  }
\cs_generate_variant:Nn
  \@@_lua_escape:nN
  { xN }
```

After the \markdownPrepareInputFilename macro has been fully expanded, the \markdownInputFilename macro will expands to a Lua string that contains the input filename passed as the first argument.

```
\tl_new:N
  \markdownInputFilename
```

```
\cs_new:Npn
  \markdownPrepareInputFilename
  #1
  {
    \@@_lua_escape:xN
      { #1 }
      \markdownInputFilename
    \tl_gset:Nx
      \markdownInputFilename
      { " \markdownInputFilename " }
  }
```

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.

```
\cs_new:Npn
  \markdownPrepare
  {
```

First, ensure that the `cacheDir` directory exists.

```
    local~lfs = require("lfs")
    local~options = \markdownLuaOptions
    if~not~lfs.isdir(options.cacheDir) then~
      assert(lfs.mkdir(options.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(options)
  }
```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain TEX. It opens the input file, converts it, and prints the conversion result.

```
\cs_new:Npn
  \markdownConvert
  {
    local~filename = \markdownInputFilename
    local~file = assert(io.open(filename, "r"),
      [[Could~not~open~file~"]] .. filename .. [["~for~reading]])
    local~input = assert(file:read("*a"))
    assert(file:close())
    print(convert(input))
  }
\ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TEX.

```
\def\markdownCleanup{%
```

Remove the `options.cacheDir` directory if it is empty.

```
  if options.cacheDir then
    lfs.rmdir(options.cacheDir)
  end
}%
```

### 3.2.5 Buffering Block-Level Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```
\csname newread\endcsname\markdownInputFileStream
\csname newwrite\endcsname\markdownOutputFileStream
```

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

```
\begingroup
  \catcode`\^^I=12%
  \gdef\markdownReadAndConvertTab{^^I}%
\endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the LaTeX $2_\varepsilon$ `\filecontents` macro to plain TeX.

```
\begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `stripPercentSigns` is enabled.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |catcode`@=14%
  |catcode`|%=12@
  |gdef|markdownReadAndConvert#1#2{@
    |begingroup@
```

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

```
    |markdownIfOption{frozenCache}{}{@
      |immediate|openout|markdownOutputFileStream@
        |markdownOptionInputTempFileName|relax@
      |markdownInfo{@
        Buffering block-level markdown input into the temporary @
        input file "|markdownOptionInputTempFileName" and scanning @
```

```
13288         for the closing token sequence "#1"}@
13289     }@
```

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.

```
13290     |def|do##1{|catcode`##1=12}|dospecials@
13291     |catcode`| =12@
13292     |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.

```
13293     |def|markdownReadAndConvertStripPercentSign##1{@
13294       |markdownIfOption{stripPercentSigns}{@
13295         |if##1%@
13296           |expandafter|expandafter|expandafter@
13297             |markdownReadAndConvertProcessLine@
13298         |else@
13299           |expandafter|expandafter|expandafter@
13300             |markdownReadAndConvertProcessLine@
13301             |expandafter|expandafter|expandafter##1@
13302         |fi@
13303       }{@
13304         |expandafter@
13305           |markdownReadAndConvertProcessLine@
13306           |expandafter##1@
13307       }@
13308     }@
```

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.

```
13309     |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.

```
13310       |ifx|relax##3|relax@
13311         |markdownIfOption{frozenCache}{}{@
13312           |immediate|write|markdownOutputFileStream{##1}@
13313         }@
13314       |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
```

Use the lt3luabridge library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownLuaExecute
  #1
  {
    \int_compare:nNnT
      { \g_luabridge_method_int }
      =
      { \c_luabridge_method_shell_int }
      {
```

```
        \sys_if_shell_unrestricted:F
          {
            \sys_if_shell:TF
              {
                \msg_error:nn
                  { markdown }
                  { restricted-shell-access }
              }
              {
                \msg_error:nn
                  { markdown }
                  { disabled-shell-access }
              }
          }
      }
    \str_gset:NV
      \g_luabridge_output_dirname_str
      \markdownOptionOutputDir
    \luabridge_now:e
      { #1 }
  }
\cs_generate_variant:Nn
  \msg_new:nnnn
  { nnnV }
\tl_set:Nn
  \l_tmpa_tl
  {
    You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
    --enable-write18~flag,~or~write~shell_escape=t~in~the~
    texmf.cnf~file.
  }
\msg_new:nnnV
  { markdown }
  { restricted-shell-access }
  { Shell~escape~is~restricted }
  \l_tmpa_tl
\msg_new:nnnV
  { markdown }
  { disabled-shell-access }
  { Shell~escape~is~disabled }
  \l_tmpa_tl
\ExplSyntaxOff
```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```
\ExplSyntaxOn
```

```
\tl_new:N
  \g_@@_after_markinline_tl
\tl_gset:Nn
  \g_@@_after_markinline_tl
  { \unskip }
\cs_new:Npn
  \markinline
  {
```

Locally change the category of the special plain TEX characters to *other* in order to prevent unwanted interpretation of the input markdown text as TEX code.

```
    \group_begin:
    \cctab_select:N
      \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
    \@@_if_option:nF
      { frozenCache }
      {
        \immediate
          \openout
          \markdownOutputFileStream
          \markdownOptionInputTempFileName
          \relax
        \msg_info:nne
          { markdown }
          { buffering-markinline }
          { \markdownOptionInputTempFileName }
      }
```

Peek ahead and extract the inline markdown text.

```
    \peek_regex_replace_once:nnF
      { { (.*?) } }
      {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
        \c { @@_if_option:nF }
          \cB { frozenCache \cE }
          \cB {
            \c { immediate }
              \c { write }
              \c { markdownOutputFileStream }
              \cB { \1 \cE }
            \c { immediate }
              \c { closeout }
              \c { markdownOutputFileStream }
          \cE }
```

Reset the category codes and \input the result of the conversion.

```
        \c { group_end: }
        \c { group_begin: }
        \c { @@_setup:n }
          \cB { contentLevel = inline \cE }
        \c { markdownInput }
          \cB {
            \c { markdownOptionOutputDir } /
            \c { markdownOptionInputTempFileName }
          \cE }
        \c { group_end: }
        \c { tl_use:N }
          \c { g_@@_after_markinline_tl }
      }
      {
        \msg_error:nn
          { markdown }
          { markinline-peek-failure }
        \group_end:
        \tl_use:N
          \g_@@_after_markinline_tl
      }
  }
\msg_new:nnn
  { markdown }
  { buffering-markinline }
  { Buffering~inline~markdown~input~into~
    the~temporary~input~file~"#1". }
\msg_new:nnnn
  { markdown }
  { markinline-peek-failure }
  { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
  { The~macro~should~be~followed~by~inline~
    markdown~text~in~curly~braces }
\ExplSyntaxOff
```

### 3.2.7 Typesetting Markdown

The \markdownInput macro uses an implementation of the \markdownLuaExecute macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TEX.

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownInput
  #1
  {
```

```
    \@@_if_option:nTF
      { frozenCache }
      {
        \markdownInputRaw
          { #1 }
      }
      {
```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On LaTeX, this also includes the directories specified in `\input@path`.

```
        \tl_set:Nx
          \l_tmpa_tl
          { #1 }
        \file_get_full_name:VNTF
          \l_tmpa_tl
          \l_tmpb_tl
          {
            \exp_args:NV
              \markdownInputRaw
              \l_tmpb_tl
          }
          {
            \msg_error:nnV
              { markdown }
              { markdown-file-does-not-exist }
              \l_tmpa_tl
          }
      }
  }
\msg_new:nnn
  { markdown }
  { markdown-file-does-not-exist }
  {
    Markdown~file~#1~does~not~exist
  }
\ExplSyntaxOff
\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|markdownInputRaw#1{%
```

Change the category code of the percent sign (%) to other, so that a user of the hybrid Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
    |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
      |markdownPrepareInputFilename{&1}%
      |markdownLuaExecute{%
        |markdownPrepare
        |markdownConvert
        |markdownCleanup}%
```

If we are finalizing the frozen cache, increment frozenCacheCounter.

```
      |markdownIfOption{finalizeCache}{%
        |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 implementation makes use of the fact that, apart from some subtle differences, LaTeX implements the majority of the plain TEX format [15, 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]%
```

### 3.3.1 Typesetting Markdown

The \markinlinePlainTeX macro is used to store the original plain TEX implementation of the \markinline macro. The \markinline macro is then redefined to accept an optional argument with options recognized by the LaTeX interface (see Section 2.3.3).

```
\ExplSyntaxOn
\cs_gset_eq:NN
  \markinlinePlainTeX
  \markinline
\cs_gset:Npn
  \markinline
  {
    \peek_regex_replace_once:nn
      { ( \[ (.*?) \] ) ? }
      {
```

Apply the options locally.

```
        \c { group_begin: }
        \c { @@_setup:n }
          \cB { \2 \cE }
        \c { tl_put_right:Nn }
          \c { g_@@_after_markinline_tl }
          \cB { \c { group_end: } \cE }
```

```
        \c { markinlinePlainTeX }
      }
  }
\ExplSyntaxOff
```

The `\markdownInputPlainTeX` macro is used to store the original plain TEX implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the LATEX interface (see Section 2.3.3).

```
\let\markdownInputPlainTeX\markdownInput
\renewcommand\markdownInput[2][]{%
  \begingroup
    \markdownSetup{#1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
\renewcommand\yamlInput[2][]{%
  \begingroup
    \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
```

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

```
\ExplSyntaxOn
\renewenvironment
  { markdown }
  {
```

In our implementation of the `markdown` LATEX environment, we want to distinguish between the following two cases:

```
\begin{markdown} [smartEllipses]
% This is an optional argument ^
% ...
\end{markdown}
```

```
\begin{markdown}
  [smartEllipses]
% ^ This is link
\end{markdown}
```

Therefore, we cannot use the built-in LATEX support for environments with optional arguments or packages such as xparse. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` LATEX environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return

character will be produced by TEX via the \endlinechar plain TEX macro and it needs to have the correct category code, so that \markdownReadAndConvert processes it correctly.

```
    \group_begin:
    \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (#, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
    \char_set_catcode_letter:n { 35 }
```

After we have matched the opening [ that begins the optional argument, we accept carriage returns as well.

```
    \peek_regex_replace_once:nnF
      { \ *\[\r*([^]]*)\][^\r]* }
      {
```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce \par tokens. Furthermore, this will cause hash signs followed by a number to be recognized as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.

```
        \c { group_end: }
        \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the \markdownSetup macro.

```
        \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the \markdownReadAndConvert macro process the rest of the LATEX environment.

We also make provision for using the \markdown command as a part of a different LATEX environment as follows:

```
\newenvironment{foo}%
               {code before \markdown[some, options]}%
               {\markdownEnd code after}
```

```
        \c { exp_args:NV }
          \c { markdownReadAndConvert@ }
          \c { @currenvir }
      }
      {
        \group_end:
        \exp_args:NV
          \markdownReadAndConvert@
          \@currenvir
```

```
      }
  }
  { \markdownEnd }
\renewenvironment
  { markdown* }
  [ 1 ]
  {
    \@@_if_option:nTF
      { experimental }
      {
        \msg_error:nnn
          { markdown }
          { latex-markdown-star-deprecated }
          { #1 }
      }
      {
        \msg_warning:nnn
          { markdown }
          { latex-markdown-star-deprecated }
          { #1 }
      }
    \@@_setup:n
      { #1 }
    \markdownReadAndConvert@
      { markdown* }
  }
  { \markdownEnd }
\renewenvironment
  { yaml }
  {
    \group_begin:
    \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
    \markdown
  }
  { \yamlEnd }
\msg_new:nnn
  { markdown }
  { latex-markdown-star-deprecated }
  {
    The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
    be~removed~in~the~next~major~version~of~the~Markdown~package.
  }
\cs_generate_variant:Nn
  \@@_setup:n
  { V }
\ExplSyntaxOff
\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@#1<%
    |markdownReadAndConvert<\end{#1}>%
                           <|end<#1>>>%
|endgroup
```

### 3.3.2 Themes

This section overrides the plain TEX implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in LATEX themes provided with the Markdown package.

```
\ExplSyntaxOn
\prop_new:N \g_@@_latex_loaded_themes_linenos_prop
\prop_new:N \g_@@_latex_loaded_themes_versions_prop
\cs_gset:Nn
  \@@_load_theme:nnn
  {
```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme`⟨*munged theme name*⟩`.sty` exists and whether we are still in the preamble.

```
    \ifmarkdownLaTeXLoaded
      \ifx\@onlypreamble\@notprerr
```

If both conditions are true does, end with an error, since we cannot load LATEX themes after the preamble. Otherwise, try loading a plain TEX theme instead.

```
        \file_if_exist:nTF
          { markdown theme #3.sty }
          {
            \msg_error:nnn
              { markdown }
              { latex-theme-after-preamble }
              { #1 }
          }
          {
            \@@_plain_tex_load_theme:nnn
              { #1 }
              { #2 }
              { #3 }
          }
      \else
```

If the Markdown package has already been loaded but we are still in the preamble, load a LaTeX theme if it exists or load a plain TeX theme otherwise.

```
\file_if_exist:nTF
  { markdown theme #3.sty }
  {
    \prop_get:NnNTF
      \g_@@_latex_loaded_themes_linenos_prop
      { #1 }
      \l_tmpa_tl
      {
        \prop_get:NnN
          \g_@@_latex_loaded_themes_versions_prop
          { #1 }
          \l_tmpb_tl
        \str_if_eq:nVTF
          { #2 }
          \l_tmpb_tl
          {
            \msg_warning:nnnVn
              { markdown }
              { repeatedly-loaded-latex-theme }
              { #1 }
              \l_tmpa_tl
              { #2 }
          }
          {
            \msg_error:nnnnVV
              { markdown }
              { different-versions-of-latex-theme }
              { #1 }
              { #2 }
              \l_tmpb_tl
              \l_tmpa_tl
          }
      }
      {
        \msg_info:nnnn
          { markdown }
          { loading-latex-theme }
          { #1 }
          { #2 }
        \prop_gput:Nnx
          \g_@@_latex_loaded_themes_linenos_prop
          { #1 }
          { \tex_the:D \tex_inputlineno:D }
        \prop_gput:Nnn
          \g_@@_latex_loaded_themes_versions_prop
```

```
                  { #1 }
                  { #2 }
                \RequirePackage
                  { markdown theme #3 }
              }
          }
          {
            \@@_plain_tex_load_theme:nnn
              { #1 }
              { #2 }
              { #3 }
          }
      \fi
    \else
```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```
      \msg_info:nnnn
        { markdown }
        { theme-loading-postponed }
        { #1 }
        { #2 }
      \AtEndOfPackage
        {
          \@@_set_theme:n
            { #1 @ #2 }
        }
    \fi
  }
\msg_new:nnn
  { markdown }
  { theme-loading-postponed }
  {
    Postponing~loading~version~#2~of~Markdown~theme~#1~until~
    Markdown~package~has~finished~loading
  }
\msg_new:nnn
  { markdown }
  { loading-latex-theme }
  { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-latex-theme }
  {
    Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
```

```
\msg_new:nnn
  { markdown }
  { different-versions-of-latex-theme }
  {
    Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
    but~version~#3~has~already~been~loaded~on~line~#4
  }
\cs_generate_variant:Nn
  \msg_new:nnnn
  { nnVV }
\tl_set:Nn
  \l_tmpa_tl
  { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
\tl_put_right:NV
  \l_tmpa_tl
  \c_backslash_str
\tl_put_right:Nn
  \l_tmpa_tl
  { begin{document} }
\tl_set:Nn
  \l_tmpb_tl
  { Load~Markdown~theme~#1~before~ }
\tl_put_right:NV
  \l_tmpb_tl
  \c_backslash_str
\tl_put_right:Nn
  \l_tmpb_tl
  { begin{document} }
\msg_new:nnVV
  { markdown }
  { latex-theme-after-preamble }
  \l_tmpa_tl
  \l_tmpb_tl
\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[3]{%
  \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}{#3}%
    }%
  }%
  \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/markdown/defaults LaTeX theme provides default definitions for token renderer prototypes. First, the LaTeX theme loads the plain TeX theme with the default definitions for plain TeX:

```
\markdownLoadPlainTeXTheme
```

Next, the LaTeX theme overrides some of the plain TeX definitions. See Section 3.3.4 for the actual definitions.

### 3.3.3 Options

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

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

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option plain has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
\markdownIfOption{plain}{\iffalse}{\iftrue}
```

#### 3.3.4.1 Lists

If either the tightLists or the fancyLists Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package paralist or the package enumitem have already been loaded, use them. Otherwise, if the option experimental or any test phase has been enabled, use the package enumitem. Otherwise, use the package paralist.

```
\ExplSyntaxOn
\bool_new:N
  \g_@@_tight_or_fancy_lists_bool
\bool_gset_false:N
  \g_@@_tight_or_fancy_lists_bool
\@@_if_option:nTF
  { tightLists }
  {
    \bool_gset_true:N
      \g_@@_tight_or_fancy_lists_bool
  }
  {
    \@@_if_option:nT
      { fancyLists }
      {
        \bool_gset_true:N
          \g_@@_tight_or_fancy_lists_bool
```

```
      }
  }
\bool_new:N
  \g_@@_beamer_paralist_or_enumitem_bool
\bool_gset_true:N
  \g_@@_beamer_paralist_or_enumitem_bool
\@ifclassloaded
  { beamer }
  { }
  {
    \@ifpackageloaded
      { paralist }
      { }
      {
        \@ifpackageloaded
        { enumitem }
        { }
        {
          \bool_gset_false:N
            \g_@@_beamer_paralist_or_enumitem_bool
        }
      }
  }
\bool_if:nT
  {
    \g_@@_tight_or_fancy_lists_bool &&
    ! \g_@@_beamer_paralist_or_enumitem_bool
  }
  {
    \bool_if:nTF
      {
        \bool_lazy_or_p:nn
          {
            \str_if_eq_p:en
              { \markdownThemeVersion }
              { experimental }
          }
          {
            \bool_lazy_and_p:nn
              {
                \prop_if_exist_p:N
                  \g__pdfmanagement_documentproperties_prop
              }
              {
                \bool_lazy_any_p:n
                  {
                    {
```

```
                    \prop_if_in_p:Nn
                      \g__pdfmanagement_documentproperties_prop
                      { document / testphase / phase-I }
                  }
                  {
                    \prop_if_in_p:Nn
                      \g__pdfmanagement_documentproperties_prop
                      { document / testphase / phase-II }
                  }
                  {
                    \prop_if_in_p:Nn
                      \g__pdfmanagement_documentproperties_prop
                      { document / testphase / phase-III }
                  }
                  {
                    \prop_if_in_p:Nn
                      \g__pdfmanagement_documentproperties_prop
                      { document / testphase / phase-IV }
                  }
                  {
                    \prop_if_in_p:Nn
                      \g__pdfmanagement_documentproperties_prop
                      { document / testphase / phase-V }
                  }
                  {
                    \prop_if_in_p:Nn
                      \g__pdfmanagement_documentproperties_prop
                      { document / testphase / phase-VI }
                  }
              }
          }
      }
      {
        \RequirePackage
          { enumitem }
      }
      {
        \RequirePackage
          { paralist }
      }
  }
\ExplSyntaxOff
```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
\ExplSyntaxOn
```

```
\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_generate_variant:Nn
  \@@_latex_fancy_list_item_label:nnn
  { VVn }
\tl_new:N
  \l_@@_latex_fancy_list_item_label_number_style_tl
\tl_new:N
  \l_@@_latex_fancy_list_item_label_delimiter_style_tl
\@ifpackageloaded{enumitem}{
  \markdownSetup{rendererPrototypes={
```

First, let's define the tight list item renderer prototypes.

```
    ulBeginTight = {
      \begin
        { itemize }
        [ noitemsep ]
```

```
  },
  ulEndTight = {
    \end
      { itemize }
  },
  olBeginTight = {
    \begin
      { enumerate }
      [ noitemsep ]
  },
  olEndTight = {
    \end
      { enumerate }
  },
  dlBeginTight = {
    \begin
      { description }
      [ noitemsep ]
  },
  dlEndTight = {
    \end
      { description }
  },
```

Second, let's define the fancy list item renderer prototypes.

```
  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 }
    \begin
      { enumerate }
  },
  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 }
    \begin
      { enumerate }
      [ noitemsep ]
  },
```

```
    fancyOlEnd(|Tight) = {
      \end { enumerate }
      \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 }
        ]
    },
  }}
```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
}{\@ifpackageloaded{paralist}{
  \markdownSetup{rendererPrototypes={
```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```
    ulBeginTight = {%
      \group_begin:
      \pltopsep=\topsep
      \plpartopsep=\partopsep
      \begin{compactitem}
    },
    ulEndTight = {
      \end{compactitem}
      \group_end:
    },
    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 }
      \begin{enumerate}
    },
    fancyOlEnd = {
      \end{enumerate}
      \group_end:
    },
```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```
    olBeginTight = {%
      \group_begin:
      \plpartopsep=\partopsep
      \pltopsep=\topsep
      \begin{compactenum}
    },
    olEndTight = {
      \end{compactenum}
      \group_end:
    },
    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 }
      \plpartopsep=\partopsep
      \pltopsep=\topsep
      \begin{compactenum}
    },
    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 }
        ]
    },
```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```
    dlBeginTight = {
      \group_begin:
      \plpartopsep=\partopsep
      \pltopsep=\topsep
      \begin{compactdesc}
    },
    dlEndTight = {
```

```
      \end{compactdesc}
      \group_end:
    }
  }}
}{
```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```
  \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,[34] we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```
  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} }
    },
  },
}
```

[34] 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.

```
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  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 HTML identifiers appear after a heading, we make them produce `\label` macros.

```
\ExplSyntaxOn
\seq_new:N
  \l_@@_header_identifiers_seq
\markdownSetup
  {
    rendererPrototypes = {
      headerAttributeContextBegin = {
        \markdownSetup
          {
            rendererPrototypes = {
              attributeIdentifier = {
                \seq_put_right:Nn
                  \l_@@_header_identifiers_seq
                  { ##1 }
              },
            },
          }
      },
      headerAttributeContextEnd = {
        \seq_map_inline:Nn
          \l_@@_header_identifiers_seq
          { \label { ##1 } }
        \seq_clear:N
          \l_@@_header_identifiers_seq
      },
    },
  }
```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```
\bool_new:N
  \l_@@_header_unnumbered_bool
```

```
\markdownSetup
  {
    rendererPrototypes = {
      headerAttributeContextBegin += {
        \markdownSetup
          {
            rendererPrototypes = {
              attributeClassName = {
                \bool_if:nT
                  {
                    \str_if_eq_p:nn
                      { ##1 }
                      { unnumbered } &&
                    ! \l_@@_header_unnumbered_bool
                  }
                  {
                    \group_begin:
                    \bool_set_true:N
                      \l_@@_header_unnumbered_bool
                    \c@secnumdepth = 0
                    \markdownSetup
                      {
                        rendererPrototypes = {
                          sectionBegin = {
                            \group_begin:
                          },
                          sectionEnd = {
                            \group_end:
                          },
                        },
                      }
                  }
              },
            },
          }
      },
    },
  }
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  superscript = {\textsuperscript{#1}},
  subscript = {\textsubscript{#1}},
  blockQuoteBegin = {\begin{quotation}},
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  note = {\footnote{#1}}}}
```

#### 3.3.4.2 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\RequirePackage{ltxcmds}
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputFencedCodePrototype#1#2#3
  {
    \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`\%=14\relax
            \catcode`\#=6\relax
            \exp_args:NV
              \inputminted
              \l_tmpa_tl
              { #1 }
            \catcode`\%=12\relax
            \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}},
    headingFive = {\subparagraph{#1}}}}
}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}},
    headingSix = {\subparagraph{#1}}}}
}%
```

#### 3.3.4.3 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
}
```

#### 3.3.4.4 HTML elements

If the `html` option is enabled and we are using TEX4ht[35], 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
      },
    },
  }
}
```

#### 3.3.4.5 Citations

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

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}%  No prenotes/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}%  No prenotes/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}%  No prenotes/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}%
    }}}}
```

#### 3.3.4.6 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}}}
```

#### 3.3.4.7 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}
```

#### 3.3.4.8 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
    },
  }}
}{}

```

#### 3.3.4.9 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.

```
\markdownSetup{
  rendererPrototypes = {
    jekyllDataEnd = {
      \AddToHook{begindocument/end}{\maketitle}
    },
  },
}
```

#### 3.3.4.10 Marked Text

If the `mark` option is enabled, we will load either the soulutf8 package or the lua-ul package and use it to implement marked text.

```
\@@_if_option:nT
  { mark }
  {
    \sys_if_engine_luatex:TF
      {
        \RequirePackage
          { luacolor }
        \RequirePackage
          { lua-ul }
        \markdownSetup
          {
            rendererPrototypes = {
              mark = {
                \highLight
                  { #1 }
              },
            }
          }
      }
```

```
    {
      \RequirePackage
        { xcolor }
      % TODO: Use just package soul after TeX Live 2023.
      \IfFormatAtLeastTF
        { 2023-02-18 }
        {
          \RequirePackage
            { soul }
        }
        {
          \RequirePackage
            { soulutf8 }
        }
      \markdownSetup
        {
          rendererPrototypes = {
            mark = {
              \hl
                { #1 }
            },
          }
        }
    }
}
```

#### 3.3.4.11 Strike-Through

If the `strikeThrough` option is enabled, we will load either the soulutf8 package or the lua-ul package and use it to implement strike-throughs.

```
\@@_if_option:nT
  { strikeThrough }
  {
    \sys_if_engine_luatex:TF
      {
        \RequirePackage
          { lua-ul }
        \markdownSetup
          {
            rendererPrototypes = {
              strikeThrough = {
                \strikeThrough
                  { #1 }
              },
            }
          }
      }
```

```
    {
      % TODO: Use just package soul after TeX Live 2023.
      \IfFormatAtLeastTF
        { 2023-02-18 }
        {
          \RequirePackage
            { soul }
        }
        {
          \RequirePackage
            { soulutf8 }
        }
      \markdownSetup
        {
          rendererPrototypes = {
            strikeThrough = {
              \st
                { #1 }
            },
          }
        }
    }
}
```

#### 3.3.4.12 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form ⟨*key*⟩=⟨*value*⟩ set the corresponding keys of the graphicx package to the corresponding values and we will register any identifiers, so that they can be used as LaTeX labels for referencing figures.

```
\ExplSyntaxOn
\seq_new:N
  \l_@@_image_identifiers_seq
\markdownSetup {
  rendererPrototypes = {
    image = {
      \begin { figure }
        \begin { center }
          \includegraphics
            [ alt = { #1 } ]
            { #3 }
          \tl_if_empty:nF
            { #4 }
            { \caption { #4 } }
```

```
          \seq_map_inline:Nn
            \l_@@_image_identifiers_seq
            { \label { ##1 } }
        \end { center }
      \end { figure }
    },
  }
}
\@@_if_option:nT
  { linkAttributes }
  {
    \RequirePackage { graphicx }
    \markdownSetup {
      rendererPrototypes = {
        imageAttributeContextBegin = {
          \group_begin:
          \markdownSetup {
            rendererPrototypes = {
              attributeIdentifier = {
                \seq_put_right:Nn
                  \l_@@_image_identifiers_seq
                  { ##1 }
              },
              attributeKeyValue = {
                \setkeys
                  { Gin }
                  { { ##1 } = { ##2 } }
              },
            },
          }
        },
        imageAttributeContextEnd = {
          \group_end:
        },
      },
    }
  }
\ExplSyntaxOff
```

#### 3.3.4.13 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:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_block:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block: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` and `\inputyaml` macros are 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}%
\long\def\inputyaml{%
  \dosingleempty
  \doinputyaml}%
\long\def\doinputyaml[#1]#2{%
```

```
  \doinputmarkdown
    [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%
```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` 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. [16, sec. 31]. According to Eijkhout [17, 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.

```
\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}%
  |gdef|startyaml{%
    |begingroup
    |ctxlua{document.markdown_buffering = true}%
    |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
    |markdownReadAndConvert{\stopyaml}%
                           {|stopyaml}}%
  |gdef|stopyaml{%
    |ctxlua{document.markdown_buffering = false}%
    |yamlEnd}%
|endgroup
```

### 3.4.2 Themes

This section overrides the plain TEX implementation of the theme-loading mechanism

from Section 3.2.2. Furthermore, this section also implements the built-in ConTEXt themes provided with the Markdown package.

```
\ExplSyntaxOn
\prop_new:N \g_@@_context_loaded_themes_linenos_prop
\prop_new:N \g_@@_context_loaded_themes_versions_prop
\cs_gset:Nn
  \@@_load_theme:nnn
  {
```

Determine whether a file named `t-markdowntheme`⟨*munged theme name*⟩`.tex` exists. If it does, load it. Otherwise, try loading a plain TEX theme instead.

```
    \file_if_exist:nTF
      { t - markdown theme #3.tex }
      {
        \prop_get:NnNTF
          \g_@@_context_loaded_themes_linenos_prop
          { #1 }
          \l_tmpa_tl
          {
            \prop_get:NnN
              \g_@@_context_loaded_themes_versions_prop
              { #1 }
              \l_tmpb_tl
            \str_if_eq:nVTF
              { #2 }
              \l_tmpb_tl
              {
                \msg_warning:nnnVn
                  { markdown }
                  { repeatedly-loaded-context-theme }
                  { #1 }
                  \l_tmpa_tl
                  { #2 }
              }
              {
                \msg_error:nnnnVV
                  { markdown }
                  { different-versions-of-context-theme }
                  { #1 }
                  { #2 }
                  \l_tmpb_tl
                  \l_tmpa_tl
              }
          }
          {
            \msg_info:nnn
              { markdown }
```

```
              { loading-context-theme }
              { #1 }
              { #2 }
            \prop_gput:Nnx
              \g_@@_context_loaded_themes_linenos_prop
              { #1 }
              { \tex_the:D \tex_inputlineno:D }
            \prop_gput:Nnn
              \g_@@_context_loaded_themes_versions_prop
              { #1 }
              { #2 }
            \usemodule
              [ t ]
              [ markdown theme #3 ]
          }
      }
      {
        \@@_plain_tex_load_theme:nnn
          { #1 }
          { #2 }
          { #3 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-context-theme }
  { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-context-theme }
  {
    Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\msg_new:nnn
  { markdown }
  { different-versions-of-context-theme }
  {
    Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
    but~version~#3~has~already~been~loaded~on~line~#4
  }
\ExplSyntaxOff
```

The witiko/markdown/defaults ConTEXt theme provides default definitions for token renderer prototypes. First, the ConTEXt theme loads the plain TEX theme with the default definitions for plain TEX:

```
\markdownLoadPlainTeXTheme
```

Next, the ConTEXt theme overrides some of the plain TEX definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
\markdownIfOption{plain}{\iffalse}{\iftrue}
\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}}%
```

#### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputFencedCodePrototype#1#2#3
  {
    \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}%
```

#### 3.4.3.2 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}
```

#### 3.4.3.3 Raw Attributes

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

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
```

```
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline:nn
              { #1 }
              { context }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { context }
          {
            \@@_plain_tex_default_input_raw_block:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset_eq:NN
  \markdownRendererInputRawBlockPrototype
  \markdownRendererInputRawInlinePrototype
\fi % Closes `\markdownIfOption{plain}{\iffalse}{\iftrue}`
\ExplSyntaxOff
\stopmodule
\protect
```

At the end of the ConTEXt module, we load the `witiko/markdown/defaults` ConTEXt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
\ExplSyntaxOn
```

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_context_tl
  {
    \ExplSyntaxOff
    \@@_if_option:nF
      { noDefaults }
      {
        \@@_if_option:nTF
          { experimental }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults@experimental }
          }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults }
          }
      }
    \ExplSyntaxOn
  }
\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] Frank Mittelbach, Ulrike Fischer, and LATEX Project. *The documentmetadata-support code*. June 1, 2024. URL: https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf (visited on 10/21/2024).

[3] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[4] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[5] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: https://pandoc.org/ (visited on 10/05/2022).

[6] 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.

[7] Donald Ervin Knuth. *The TEXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[8] 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).

[9] 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).

[10] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: https://github.com/Witiko/markdown/discussions/514 (visited on 10/21/2024).

[11] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: https://tex.stackexchange.com/q/716362/70941 (visited on 04/28/2024).

[12] Frank Mittelbach. *LATEX's hook management*. June 26, 2024. URL: https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf (visited on 10/02/2024).

[13] 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).

[14] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[15] 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).

[16] Donald Ervin Knuth. *TEX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.

[17] Victor Eijkhout. *TEX by Topic. A TEXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index