# A Markdown Interpreter for TEX

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

**Version 3.7.1-0-g8e726800**
**2024-09-30**

## 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 XƎTEX), 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,

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

a TEX engine that extends ε-TEX, and all the plain TEX prerequisites (see Section 1.1.2):

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or LATEX themes (see Section 2.3.3) 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
```

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

```
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 2.3.3).

```
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 2.3.3.

```
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 2.3.3.

```
49 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 3.3.4.8, and also in the default renderer prototype for identifier attributes.

```
50 soft etoolbox
```

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

```
51 soft soul
```

**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.

```
52 soft ltxcmds
```

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

```
53 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.

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

### 1.3 Acknowledgements

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

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

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

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

The TEX implementation of the package draws inspiration from several sources including the source code of LATEX $2_\varepsilon$, 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.

### 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.

```
54 local M = {metadata = metadata}
```

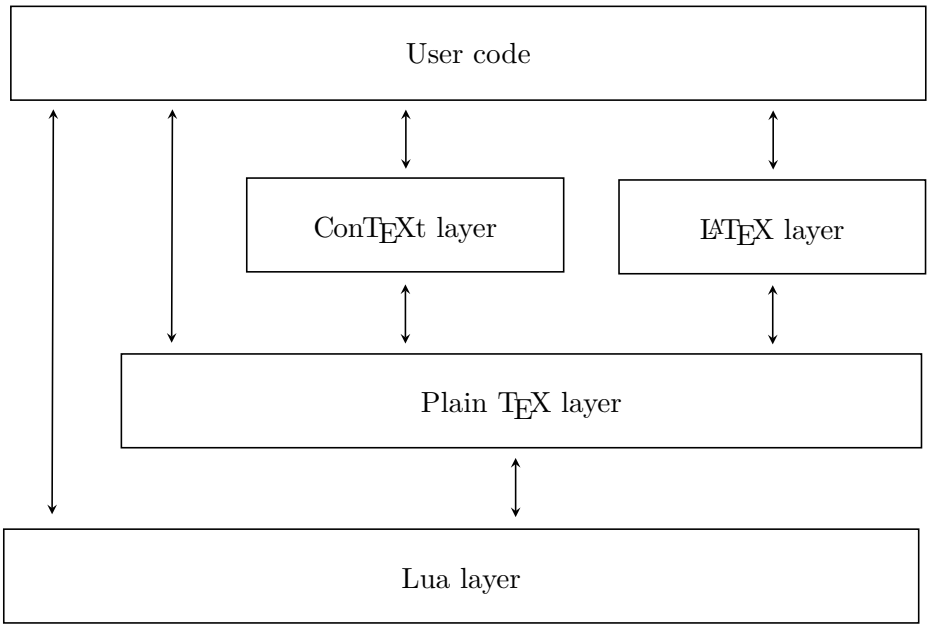


**Figure 1: A block diagram of the Markdown package**

</lua,lua-loader,lua-unicode-data> <*lua>

### 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
```

`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,
}
```

`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
```

[6] See https://github.com/witiko/markdown/pull/226#issuecomment-1599641634.
[7] See https://unicode.org/faq/normalization.html.

`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

`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.

```
\@@_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
```

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

`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
```

`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
```

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

`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
```

`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
```

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

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

true

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

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

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

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

`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:

```
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);
}
~~~~~~

  ``` 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
```

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

`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
```

`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
```

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

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

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

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

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

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

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

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

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

`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.

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

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
```

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}
```

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

`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
```

`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.
---
```

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

`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
```

`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
```

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

`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.

[^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
```

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

`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 \\
```

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

```
        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
```

`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
```

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

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

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

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

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

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

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

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

`stripIndent`=true, false default: `false`

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

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

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

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

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

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

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

`false` Disable the Pandoc subscript syntax extension.

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

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

`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}
```
```

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

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

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

`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.

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

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

`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.

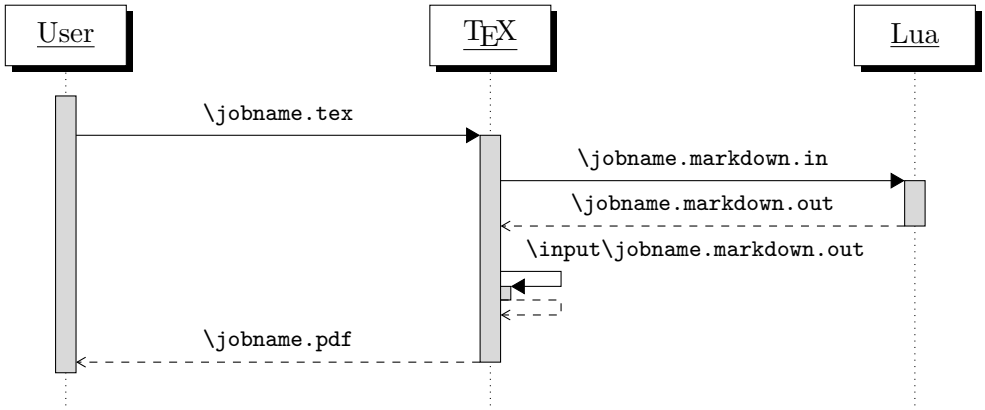


**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface**

```
729
730 local HELP_STRING = [[
731 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
732 where OPTIONS are documented in the Lua interface section of the
733 technical Markdown package documentation.
734
735 When OUTPUT_FILE is unspecified, the result of the conversion will be
736 written to the standard output. When INPUT_FILE is also unspecified, the
737 result of the conversion will be read from the standard input.
738
739 Report bugs to: witiko@mail.muni.cz
740 Markdown package home page: <https://github.com/witiko/markdown>]]
741
```

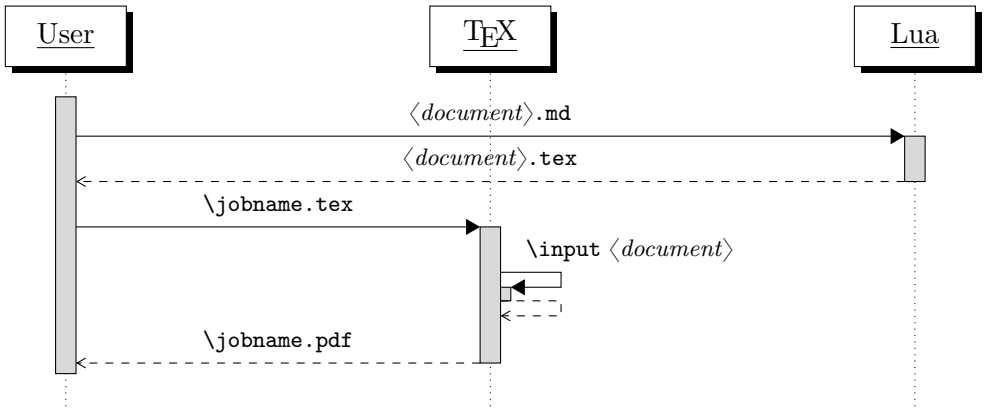


**Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface**

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

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

License: ]] .. metadata.license

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

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

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

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

local function snake_case(option_name)
  local cased_option_name = option_name:gsub("%l%u", function(match)
```

```
767     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
768   end)
769   return cased_option_name
770 end
771
772 local cases = {camel_case, snake_case}
773 local various_case_options = {}
774 for option_name, _ in pairs(defaultOptions) do
775   for _, case in ipairs(cases) do
776     various_case_options[case(option_name)] = option_name
777   end
778 end
779
780 local process_options = true
781 local options = {}
782 local input_filename
783 local output_filename
784 for i = 1, #arg do
785   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.

```
786     if arg[i] == "--" then
787       process_options = false
788       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.

```
789     elseif arg[i]:match("=") then
790       local key, value = arg[i]:match("(.-)=(.*)")
791       if defaultOptions[key] == nil and
792          various_case_options[key] ~= nil then
793         key = various_case_options[key]
794       end
```

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

```
795       local default_type = type(defaultOptions[key])
796       if default_type == "boolean" then
797         options[key] = (value == "true")
798       elseif default_type == "number" then
799         options[key] = tonumber(value)
800       elseif default_type == "table" then
801         options[key] = {}
802         for item in value:gmatch("[^ ,]+") do
```

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

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

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

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

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

The first argument that matches none of the above 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.

```
  if input_filename == nil then
    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.

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

```
end
```

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

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

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

```
\input hello
```

## 2.2 Plain TEX Interface

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

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

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown

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

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

```
\let\markdownBegin\relax
\let\markdownEnd\relax
```

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

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

it will not be recognized as the end of the markdown string. As a corrolary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

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

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

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

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

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

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

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.

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
```

This macro 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
```

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 `\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 [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
\seq_gput_right:Nn
  \g_@@_plain_tex_options_seq
  { stripPercentSigns }
\prop_gput:Nnn
  \g_@@_plain_tex_option_types_prop
  { stripPercentSigns }
  { boolean }
\prop_gput:Nnx
  \g_@@_default_plain_tex_options_prop
  { stripPercentSigns }
  { false }
```

#### 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 [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [5] also show that snake_case is faster to read than camelCase.

```
              \@@_with_various_cases:nn
                { ####1 }
                {
                  \@@_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 }
  { 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*⟩ 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 is *qualified* and contains no underscores. A theme name is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer LATEX package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes 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`.

```
\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 `\g_@@_themes_seq` stack of theme names. For convenience, the name of the current theme is also available in the `\g_@@_current_theme_tl` macro.

```
\seq_new:N
  \g_@@_themes_seq
\tl_new:N
  \g_@@_current_theme_tl
\tl_gset:Nn
  \g_@@_current_theme_tl
  { }
\seq_gput_right:NV
  \g_@@_themes_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 munge the theme name.

```
    \str_set:Nn
      \l_tmpa_str
      { #1 }
    \str_replace_all:Nnn
      \l_tmpa_str
      { / }
      { _ }
```

Finally, we load the theme.

```
    \tl_gset:Nn
      \g_@@_current_theme_tl
      { #1 / }
    \seq_gput_right:NV
      \g_@@_themes_seq
      \g_@@_current_theme_tl
    \@@_load_theme:nV
      { #1 }
      \l_tmpa_str
    \seq_gpop_right:NN
      \g_@@_themes_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_themes_seq
      \l_tmpa_tl
    \tl_gset:NV
      \g_@@_current_theme_tl
      \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 }
\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.

```
\prg_new_conditional:Nnn
  \@@_if_snippet_exists:n
  { TF, T, F }
  {
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_put_right:Nn
      \l_tmpa_tl
      { #1 }
    \prop_get:NVNTF
      \g_@@_snippets_prop
      \l_tmpa_tl
      \l_tmpb_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 }
        {
          \prop_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,

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

you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```
\def\markdownRendererContentBlockCode{%
  \markdownRendererContentBlockCodePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockCode }
  { 5 }
\ExplSyntaxOff
```

#### 2.2.5.10 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
```

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 `\markdownRendererFencedDi` 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 `\markdownRendererHeaderAttril` 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
      {
        \markdownWarning
          {
            The~jekyllDataString~renderer~has~been~deprecated,~
            to~be~removed~in~Markdown~4.0.0
          }
        \markdownRendererJekyllDataString
      }
      {
        \cs_if_exist:NTF
          \markdownRendererJekyllDataStringPrototype
          {
            \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` [9]. 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.
}
```

```
2999          {
3000            \@@_glob_seq:VnN
3001              \l_keys_key_str
3002              { g_@@_renderers_seq }
3003              \l_@@_renderer_glob_results_seq
3004            \seq_if_empty:NTF
3005              \l_@@_renderer_glob_results_seq
3006              {
3007                \msg_error:nnV
3008                  { markdown }
3009                  { undefined-renderer }
3010                  \l_keys_key_str
3011              }
3012              {
3013                \tl_set:Nn
3014                  \l_@@_renderer_glob_definition_tl
3015                  { \exp_not:n { #1 } }
3016                \seq_map_inline:Nn
3017                  \l_@@_renderer_glob_results_seq
3018                  {
3019                    \tl_set:Nn
3020                      \l_tmpa_tl
3021                      { { ##1 } = }
3022                    \tl_put_right:Nx
3023                      \l_tmpa_tl
3024                      { { \l_@@_renderer_glob_definition_tl } }
3025                    \keys_set:nV
3026                      { markdown/options/renderers }
3027                      \l_tmpa_tl
```

```
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer }
  {
    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 += {...},
        },
      }
    },
  },
}
```

```
      \regex_match:NVTF
        \c_@@_appending_key_regex
        \l_keys_key_str
        {
          \bool_gset_true:N
            \g_@@_appending_renderer_prototype_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/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` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain TEX special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
\let\markdownReadAndConvert\relax
\begingroup
```

Locally swap the category code of the backslash symbol (\) with the pipe symbol (|). This is required in order that all the special symbols in the first argument of the markdownReadAndConvert macro have the category code *other*.

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

The macro is exposed in the interface, so that 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
```

```
3379 \input markdown/markdown
```

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

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

where ⟨*options*⟩ are the LaTeX interface options (see Section 2.3.2). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) 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 2.3.3) 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` and `markdown*` LaTeX environments, and redefines the `\markinline` and `\markdownInput` commands.

#### 2.3.1.1 The `markdown` and `markdown*` LaTeX environments

The `markdown` and `markdown*` LaTeX environments are used to typeset markdown document fragments. Both LaTeX environments accept LaTeX interface options (see Section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

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

```
3380 \newenvironment{markdown}\relax\relax
3381 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\markdownEnd` macro to produce special effects before and after the `markdown` LaTeX environment (and likewise for the starred version).

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown}[smartEllipses]
```

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

```
_Hello_ **world** ...
\end{markdown}
% ...
\end{document}
```

```
_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 text and must be followed by text that has not yet been ingested by TeX's input processor. Furthermore, using the `\markdownEnd` macro 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 is 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}
```

#### 2.3.1.2 The `\markinline` and `\markdownInput` macros

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 2.3.2) as its optional argument. These options will only influnce this markdown content.

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

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

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

### 2.3.2 Options

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

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.

#### 2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [10, 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}}
```

#### 2.3.2.2 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,
}
```

### 2.3.3 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.

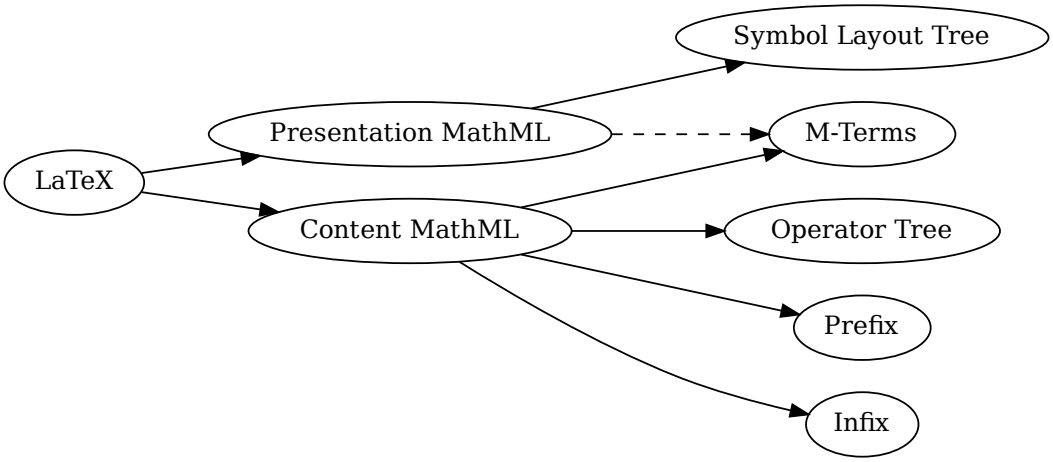


**Figure 4: Various formats of mathemathical formulae**

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.

```
3396 \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

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

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

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

Chapter 1

# Introduction

## 1.1 Section

### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

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

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

```
\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).

```
  \markdownIfOption{noDefaults}{}{
    \markdownSetup{theme=witiko/markdown/defaults}
  }
}
\ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%
```

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

The interface exposes the `\startmarkdown` and `\stopmarkdown` macro pair for the typesetting of a markdown document fragment, and defines the `\inputmarkdown` macro.

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

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

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

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

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

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

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

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

### 2.4.2 Options

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

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

The ConTEXt options may be specified when using the `\inputmarkdown` macro (see Section 2.4), 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 }
  }
\ExplSyntaxOff
```

#### 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.

```
\ExplSyntaxOn
\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 de-

fine 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).

```
3458 local upper, format, length =
3459   string.upper, string.format, string.len
3460 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3461   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3462   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.

```
3463 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.

```
3464 function util.err(msg, exit_code)
3465   io.stderr:write("markdown.lua: " .. msg .. "\n")
3466   os.exit(exit_code or 1)
3467 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"),
      [[Could not open file "]] .. name .. [[" for writing]])
    local result = string
    if transform ~= nil then
      result = transform(result)
    end
    assert(file:write(result))
    assert(file:close())
  end
  return name
end
```

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

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

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

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

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

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

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

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

```
            corr = corr - 1 + sp
            return string.rep(" ", sp)
          end))
end
```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

```
function util.walk(t, f)
  local typ = type(t)
  if typ == "string" then
    f(t)
  elseif typ == "table" then
    local i = 1
    local n
    n = t[i]
    while n do
      util.walk(n, f)
      i = i + 1
      n = t[i]
    end
  elseif typ == "function" then
    local ok, val = pcall(t)
    if ok then
      util.walk(val,f)
    end
  else
    f(tostring(t))
  end
end
```

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

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

The `util.rope_to_string` method converts a rope `rope` to a string and returns it. For the definition of a rope, see the definition of the `util.walk` method.

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

The `util.rope_last` method retrieves the last item in a rope. For the definition of a rope, see the definition of the `util.walk` method.

```
function util.rope_last(rope)
  if #rope == 0 then
    return nil
  else
    local l = rope[#rope]
    if type(l) == "table" then
      return util.rope_last(l)
    else
      return l
    end
  end
end
```

Given an array `ary` and a string `x`, the `util.intersperse` method returns an array `new`, such that `ary[i] == new[2*(i-1)+1]` and `new[2*i] == x` for all $1 \leqslant$ `i` $\leqslant$ `#ary`.

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

Given an array `ary` and a function `f`, the `util.map` method returns an array `new`, such that `new[i] == f(ary[i])` for all $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.map(ary, f)
  local new = {}
  for i,v in ipairs(ary) do
    new[i] = f(v)
  end
  return new
end
```

Given a table `char_escapes` mapping escapable characters to escaped strings and optionally a table `string_escapes` mapping escapable strings to escaped strings, the `util.escaper` method returns an escaper function that escapes all occurrences of escapable strings and characters (in this order).

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

```
3578 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3579   local char_escapes_list = ""
3580   for i,_ in pairs(char_escapes) do
3581     char_escapes_list = char_escapes_list .. i
3582   end
```

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

```
3583   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.

```
3584   if string_escapes then
3585     for k,v in pairs(string_escapes) do
3586       escapable = P(k) / v + escapable
3587     end
3588   end
```

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

```
3589   local escape_string = Cs((escapable + any)^0)
```

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

```
3590   return function(s)
3591     return lpeg.match(escape_string, s)
3592   end
3593 end
```

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

```
3594 function util.pathname(dir, file)
3595   if #dir == 0 then
3596     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,
  ["apos"] = 39,
  ["lpar"] = 40,
  ["rpar"] = 41,
  ["ast"] = 42,
  ["midast"] = 42,
  ["plus"] = 43,
  ["comma"] = 44,
  ["period"] = 46,
  ["sol"] = 47,
  ["colon"] = 58,
  ["semi"] = 59,
  ["LT"] = 60,
  ["lt"] = 60,
  ["nvlt"] = {60, 8402},
  ["bne"] = {61, 8421},
  ["equals"] = 61,
  ["GT"] = 62,
  ["gt"] = 62,
  ["nvgt"] = {62, 8402},
  ["quest"] = 63,
  ["commat"] = 64,
  ["lbrack"] = 91,
  ["lsqb"] = 91,
  ["bsol"] = 92,
  ["rbrack"] = 93,
  ["rsqb"] = 93,
  ["Hat"] = 94,
  ["UnderBar"] = 95,
  ["lowbar"] = 95,
  ["DiacriticalGrave"] = 96,
  ["grave"] = 96,
  ["fjlig"] = {102, 106},
  ["lbrace"] = 123,
  ["lcub"] = 123,
  ["VerticalLine"] = 124,
  ["verbar"] = 124,
  ["vert"] = 124,
  ["rbrace"] = 125,
  ["rcub"] = 125,
  ["NonBreakingSpace"] = 160,
  ["nbsp"] = 160,
  ["iexcl"] = 161,
```

```
3676   ["cent"] = 162,
3677   ["pound"] = 163,
3678   ["curren"] = 164,
3679   ["yen"] = 165,
3680   ["brvbar"] = 166,
3681   ["sect"] = 167,
3682   ["Dot"] = 168,
3683   ["DoubleDot"] = 168,
3684   ["die"] = 168,
3685   ["uml"] = 168,
3686   ["COPY"] = 169,
3687   ["copy"] = 169,
3688   ["ordf"] = 170,
3689   ["laquo"] = 171,
3690   ["not"] = 172,
3691   ["shy"] = 173,
3692   ["REG"] = 174,
3693   ["circledR"] = 174,
3694   ["reg"] = 174,
3695   ["macr"] = 175,
3696   ["strns"] = 175,
3697   ["deg"] = 176,
3698   ["PlusMinus"] = 177,
3699   ["plusmn"] = 177,
3700   ["pm"] = 177,
3701   ["sup2"] = 178,
3702   ["sup3"] = 179,
3703   ["DiacriticalAcute"] = 180,
3704   ["acute"] = 180,
3705   ["micro"] = 181,
3706   ["para"] = 182,
3707   ["CenterDot"] = 183,
3708   ["centerdot"] = 183,
3709   ["middot"] = 183,
3710   ["Cedilla"] = 184,
3711   ["cedil"] = 184,
3712   ["sup1"] = 185,
3713   ["ordm"] = 186,
3714   ["raquo"] = 187,
3715   ["frac14"] = 188,
3716   ["frac12"] = 189,
3717   ["half"] = 189,
3718   ["frac34"] = 190,
3719   ["iquest"] = 191,
3720   ["Agrave"] = 192,
3721   ["Aacute"] = 193,
3722   ["Acirc"] = 194,
```

```
3723   ["Atilde"] = 195,
3724   ["Auml"] = 196,
3725   ["Aring"] = 197,
3726   ["angst"] = 197,
3727   ["AElig"] = 198,
3728   ["Ccedil"] = 199,
3729   ["Egrave"] = 200,
3730   ["Eacute"] = 201,
3731   ["Ecirc"] = 202,
3732   ["Euml"] = 203,
3733   ["Igrave"] = 204,
3734   ["Iacute"] = 205,
3735   ["Icirc"] = 206,
3736   ["Iuml"] = 207,
3737   ["ETH"] = 208,
3738   ["Ntilde"] = 209,
3739   ["Ograve"] = 210,
3740   ["Oacute"] = 211,
3741   ["Ocirc"] = 212,
3742   ["Otilde"] = 213,
3743   ["Ouml"] = 214,
3744   ["times"] = 215,
3745   ["Oslash"] = 216,
3746   ["Ugrave"] = 217,
3747   ["Uacute"] = 218,
3748   ["Ucirc"] = 219,
3749   ["Uuml"] = 220,
3750   ["Yacute"] = 221,
3751   ["THORN"] = 222,
3752   ["szlig"] = 223,
3753   ["agrave"] = 224,
3754   ["aacute"] = 225,
3755   ["acirc"] = 226,
3756   ["atilde"] = 227,
3757   ["auml"] = 228,
3758   ["aring"] = 229,
3759   ["aelig"] = 230,
3760   ["ccedil"] = 231,
3761   ["egrave"] = 232,
3762   ["eacute"] = 233,
3763   ["ecirc"] = 234,
3764   ["euml"] = 235,
3765   ["igrave"] = 236,
3766   ["iacute"] = 237,
3767   ["icirc"] = 238,
3768   ["iuml"] = 239,
3769   ["eth"] = 240,
```

```
3770    ["ntilde"] = 241,
3771    ["ograve"] = 242,
3772    ["oacute"] = 243,
3773    ["ocirc"] = 244,
3774    ["otilde"] = 245,
3775    ["ouml"] = 246,
3776    ["div"] = 247,
3777    ["divide"] = 247,
3778    ["oslash"] = 248,
3779    ["ugrave"] = 249,
3780    ["uacute"] = 250,
3781    ["ucirc"] = 251,
3782    ["uuml"] = 252,
3783    ["yacute"] = 253,
3784    ["thorn"] = 254,
3785    ["yuml"] = 255,
3786    ["Amacr"] = 256,
3787    ["amacr"] = 257,
3788    ["Abreve"] = 258,
3789    ["abreve"] = 259,
3790    ["Aogon"] = 260,
3791    ["aogon"] = 261,
3792    ["Cacute"] = 262,
3793    ["cacute"] = 263,
3794    ["Ccirc"] = 264,
3795    ["ccirc"] = 265,
3796    ["Cdot"] = 266,
3797    ["cdot"] = 267,
3798    ["Ccaron"] = 268,
3799    ["ccaron"] = 269,
3800    ["Dcaron"] = 270,
3801    ["dcaron"] = 271,
3802    ["Dstrok"] = 272,
3803    ["dstrok"] = 273,
3804    ["Emacr"] = 274,
3805    ["emacr"] = 275,
3806    ["Edot"] = 278,
3807    ["edot"] = 279,
3808    ["Eogon"] = 280,
3809    ["eogon"] = 281,
3810    ["Ecaron"] = 282,
3811    ["ecaron"] = 283,
3812    ["Gcirc"] = 284,
3813    ["gcirc"] = 285,
3814    ["Gbreve"] = 286,
3815    ["gbreve"] = 287,
3816    ["Gdot"] = 288,
```

```
3817   ["gdot"] = 289,
3818   ["Gcedil"] = 290,
3819   ["Hcirc"] = 292,
3820   ["hcirc"] = 293,
3821   ["Hstrok"] = 294,
3822   ["hstrok"] = 295,
3823   ["Itilde"] = 296,
3824   ["itilde"] = 297,
3825   ["Imacr"] = 298,
3826   ["imacr"] = 299,
3827   ["Iogon"] = 302,
3828   ["iogon"] = 303,
3829   ["Idot"] = 304,
3830   ["imath"] = 305,
3831   ["inodot"] = 305,
3832   ["IJlig"] = 306,
3833   ["ijlig"] = 307,
3834   ["Jcirc"] = 308,
3835   ["jcirc"] = 309,
3836   ["Kcedil"] = 310,
3837   ["kcedil"] = 311,
3838   ["kgreen"] = 312,
3839   ["Lacute"] = 313,
3840   ["lacute"] = 314,
3841   ["Lcedil"] = 315,
3842   ["lcedil"] = 316,
3843   ["Lcaron"] = 317,
3844   ["lcaron"] = 318,
3845   ["Lmidot"] = 319,
3846   ["lmidot"] = 320,
3847   ["Lstrok"] = 321,
3848   ["lstrok"] = 322,
3849   ["Nacute"] = 323,
3850   ["nacute"] = 324,
3851   ["Ncedil"] = 325,
3852   ["ncedil"] = 326,
3853   ["Ncaron"] = 327,
3854   ["ncaron"] = 328,
3855   ["napos"] = 329,
3856   ["ENG"] = 330,
3857   ["eng"] = 331,
3858   ["Omacr"] = 332,
3859   ["omacr"] = 333,
3860   ["Odblac"] = 336,
3861   ["odblac"] = 337,
3862   ["OElig"] = 338,
3863   ["oelig"] = 339,
```

```
  ["Racute"] = 340,
  ["racute"] = 341,
  ["Rcedil"] = 342,
  ["rcedil"] = 343,
  ["Rcaron"] = 344,
  ["rcaron"] = 345,
  ["Sacute"] = 346,
  ["sacute"] = 347,
  ["Scirc"] = 348,
  ["scirc"] = 349,
  ["Scedil"] = 350,
  ["scedil"] = 351,
  ["Scaron"] = 352,
  ["scaron"] = 353,
  ["Tcedil"] = 354,
  ["tcedil"] = 355,
  ["Tcaron"] = 356,
  ["tcaron"] = 357,
  ["Tstrok"] = 358,
  ["tstrok"] = 359,
  ["Utilde"] = 360,
  ["utilde"] = 361,
  ["Umacr"] = 362,
  ["umacr"] = 363,
  ["Ubreve"] = 364,
  ["ubreve"] = 365,
  ["Uring"] = 366,
  ["uring"] = 367,
  ["Udblac"] = 368,
  ["udblac"] = 369,
  ["Uogon"] = 370,
  ["uogon"] = 371,
  ["Wcirc"] = 372,
  ["wcirc"] = 373,
  ["Ycirc"] = 374,
  ["ycirc"] = 375,
  ["Yuml"] = 376,
  ["Zacute"] = 377,
  ["zacute"] = 378,
  ["Zdot"] = 379,
  ["zdot"] = 380,
  ["Zcaron"] = 381,
  ["zcaron"] = 382,
  ["fnof"] = 402,
  ["imped"] = 437,
  ["gacute"] = 501,
  ["jmath"] = 567,
```

```
3911   ["circ"] = 710,
3912   ["Hacek"] = 711,
3913   ["caron"] = 711,
3914   ["Breve"] = 728,
3915   ["breve"] = 728,
3916   ["DiacriticalDot"] = 729,
3917   ["dot"] = 729,
3918   ["ring"] = 730,
3919   ["ogon"] = 731,
3920   ["DiacriticalTilde"] = 732,
3921   ["tilde"] = 732,
3922   ["DiacriticalDoubleAcute"] = 733,
3923   ["dblac"] = 733,
3924   ["DownBreve"] = 785,
3925   ["Alpha"] = 913,
3926   ["Beta"] = 914,
3927   ["Gamma"] = 915,
3928   ["Delta"] = 916,
3929   ["Epsilon"] = 917,
3930   ["Zeta"] = 918,
3931   ["Eta"] = 919,
3932   ["Theta"] = 920,
3933   ["Iota"] = 921,
3934   ["Kappa"] = 922,
3935   ["Lambda"] = 923,
3936   ["Mu"] = 924,
3937   ["Nu"] = 925,
3938   ["Xi"] = 926,
3939   ["Omicron"] = 927,
3940   ["Pi"] = 928,
3941   ["Rho"] = 929,
3942   ["Sigma"] = 931,
3943   ["Tau"] = 932,
3944   ["Upsilon"] = 933,
3945   ["Phi"] = 934,
3946   ["Chi"] = 935,
3947   ["Psi"] = 936,
3948   ["Omega"] = 937,
3949   ["ohm"] = 937,
3950   ["alpha"] = 945,
3951   ["beta"] = 946,
3952   ["gamma"] = 947,
3953   ["delta"] = 948,
3954   ["epsi"] = 949,
3955   ["epsilon"] = 949,
3956   ["zeta"] = 950,
3957   ["eta"] = 951,
```

```
  ["theta"] = 952,
  ["iota"] = 953,
  ["kappa"] = 954,
  ["lambda"] = 955,
  ["mu"] = 956,
  ["nu"] = 957,
  ["xi"] = 958,
  ["omicron"] = 959,
  ["pi"] = 960,
  ["rho"] = 961,
  ["sigmaf"] = 962,
  ["sigmav"] = 962,
  ["varsigma"] = 962,
  ["sigma"] = 963,
  ["tau"] = 964,
  ["upsi"] = 965,
  ["upsilon"] = 965,
  ["phi"] = 966,
  ["chi"] = 967,
  ["psi"] = 968,
  ["omega"] = 969,
  ["thetasym"] = 977,
  ["thetav"] = 977,
  ["vartheta"] = 977,
  ["Upsi"] = 978,
  ["upsih"] = 978,
  ["phiv"] = 981,
  ["straightphi"] = 981,
  ["varphi"] = 981,
  ["piv"] = 982,
  ["varpi"] = 982,
  ["Gammad"] = 988,
  ["digamma"] = 989,
  ["gammad"] = 989,
  ["kappav"] = 1008,
  ["varkappa"] = 1008,
  ["rhov"] = 1009,
  ["varrho"] = 1009,
  ["epsiv"] = 1013,
  ["straightepsilon"] = 1013,
  ["varepsilon"] = 1013,
  ["backepsilon"] = 1014,
  ["bepsi"] = 1014,
  ["IOcy"] = 1025,
  ["DJcy"] = 1026,
  ["GJcy"] = 1027,
  ["Jukcy"] = 1028,
```

```
4005    ["DScy"] = 1029,
4006    ["Iukcy"] = 1030,
4007    ["YIcy"] = 1031,
4008    ["Jsercy"] = 1032,
4009    ["LJcy"] = 1033,
4010    ["NJcy"] = 1034,
4011    ["TSHcy"] = 1035,
4012    ["KJcy"] = 1036,
4013    ["Ubrcy"] = 1038,
4014    ["DZcy"] = 1039,
4015    ["Acy"] = 1040,
4016    ["Bcy"] = 1041,
4017    ["Vcy"] = 1042,
4018    ["Gcy"] = 1043,
4019    ["Dcy"] = 1044,
4020    ["IEcy"] = 1045,
4021    ["ZHcy"] = 1046,
4022    ["Zcy"] = 1047,
4023    ["Icy"] = 1048,
4024    ["Jcy"] = 1049,
4025    ["Kcy"] = 1050,
4026    ["Lcy"] = 1051,
4027    ["Mcy"] = 1052,
4028    ["Ncy"] = 1053,
4029    ["Ocy"] = 1054,
4030    ["Pcy"] = 1055,
4031    ["Rcy"] = 1056,
4032    ["Scy"] = 1057,
4033    ["Tcy"] = 1058,
4034    ["Ucy"] = 1059,
4035    ["Fcy"] = 1060,
4036    ["KHcy"] = 1061,
4037    ["TScy"] = 1062,
4038    ["CHcy"] = 1063,
4039    ["SHcy"] = 1064,
4040    ["SHCHcy"] = 1065,
4041    ["HARDcy"] = 1066,
4042    ["Ycy"] = 1067,
4043    ["SOFTcy"] = 1068,
4044    ["Ecy"] = 1069,
4045    ["YUcy"] = 1070,
4046    ["YAcy"] = 1071,
4047    ["acy"] = 1072,
4048    ["bcy"] = 1073,
4049    ["vcy"] = 1074,
4050    ["gcy"] = 1075,
4051    ["dcy"] = 1076,
```

```
4052   ["iecy"] = 1077,
4053   ["zhcy"] = 1078,
4054   ["zcy"] = 1079,
4055   ["icy"] = 1080,
4056   ["jcy"] = 1081,
4057   ["kcy"] = 1082,
4058   ["lcy"] = 1083,
4059   ["mcy"] = 1084,
4060   ["ncy"] = 1085,
4061   ["ocy"] = 1086,
4062   ["pcy"] = 1087,
4063   ["rcy"] = 1088,
4064   ["scy"] = 1089,
4065   ["tcy"] = 1090,
4066   ["ucy"] = 1091,
4067   ["fcy"] = 1092,
4068   ["khcy"] = 1093,
4069   ["tscy"] = 1094,
4070   ["chcy"] = 1095,
4071   ["shcy"] = 1096,
4072   ["shchcy"] = 1097,
4073   ["hardcy"] = 1098,
4074   ["ycy"] = 1099,
4075   ["softcy"] = 1100,
4076   ["ecy"] = 1101,
4077   ["yucy"] = 1102,
4078   ["yacy"] = 1103,
4079   ["iocy"] = 1105,
4080   ["djcy"] = 1106,
4081   ["gjcy"] = 1107,
4082   ["jukcy"] = 1108,
4083   ["dscy"] = 1109,
4084   ["iukcy"] = 1110,
4085   ["yicy"] = 1111,
4086   ["jsercy"] = 1112,
4087   ["ljcy"] = 1113,
4088   ["njcy"] = 1114,
4089   ["tshcy"] = 1115,
4090   ["kjcy"] = 1116,
4091   ["ubrcy"] = 1118,
4092   ["dzcy"] = 1119,
4093   ["ensp"] = 8194,
4094   ["emsp"] = 8195,
4095   ["emsp13"] = 8196,
4096   ["emsp14"] = 8197,
4097   ["numsp"] = 8199,
4098   ["puncsp"] = 8200,
```

```
4099  ["ThinSpace"] = 8201,
4100  ["thinsp"] = 8201,
4101  ["VeryThinSpace"] = 8202,
4102  ["hairsp"] = 8202,
4103  ["NegativeMediumSpace"] = 8203,
4104  ["NegativeThickSpace"] = 8203,
4105  ["NegativeThinSpace"] = 8203,
4106  ["NegativeVeryThinSpace"] = 8203,
4107  ["ZeroWidthSpace"] = 8203,
4108  ["zwnj"] = 8204,
4109  ["zwj"] = 8205,
4110  ["lrm"] = 8206,
4111  ["rlm"] = 8207,
4112  ["dash"] = 8208,
4113  ["hyphen"] = 8208,
4114  ["ndash"] = 8211,
4115  ["mdash"] = 8212,
4116  ["horbar"] = 8213,
4117  ["Verbar"] = 8214,
4118  ["Vert"] = 8214,
4119  ["OpenCurlyQuote"] = 8216,
4120  ["lsquo"] = 8216,
4121  ["CloseCurlyQuote"] = 8217,
4122  ["rsquo"] = 8217,
4123  ["rsquor"] = 8217,
4124  ["lsquor"] = 8218,
4125  ["sbquo"] = 8218,
4126  ["OpenCurlyDoubleQuote"] = 8220,
4127  ["ldquo"] = 8220,
4128  ["CloseCurlyDoubleQuote"] = 8221,
4129  ["rdquo"] = 8221,
4130  ["rdquor"] = 8221,
4131  ["bdquo"] = 8222,
4132  ["ldquor"] = 8222,
4133  ["dagger"] = 8224,
4134  ["Dagger"] = 8225,
4135  ["ddagger"] = 8225,
4136  ["bull"] = 8226,
4137  ["bullet"] = 8226,
4138  ["nldr"] = 8229,
4139  ["hellip"] = 8230,
4140  ["mldr"] = 8230,
4141  ["permil"] = 8240,
4142  ["pertenk"] = 8241,
4143  ["prime"] = 8242,
4144  ["Prime"] = 8243,
4145  ["tprime"] = 8244,
```

```
  ["backprime"] = 8245,
  ["bprime"] = 8245,
  ["lsaquo"] = 8249,
  ["rsaquo"] = 8250,
  ["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,
```

```
4193  ["Lscr"] = 8466,
4194  ["lagran"] = 8466,
4195  ["ell"] = 8467,
4196  ["Nopf"] = 8469,
4197  ["naturals"] = 8469,
4198  ["numero"] = 8470,
4199  ["copysr"] = 8471,
4200  ["weierp"] = 8472,
4201  ["wp"] = 8472,
4202  ["Popf"] = 8473,
4203  ["primes"] = 8473,
4204  ["Qopf"] = 8474,
4205  ["rationals"] = 8474,
4206  ["Rscr"] = 8475,
4207  ["realine"] = 8475,
4208  ["Re"] = 8476,
4209  ["Rfr"] = 8476,
4210  ["real"] = 8476,
4211  ["realpart"] = 8476,
4212  ["Ropf"] = 8477,
4213  ["reals"] = 8477,
4214  ["rx"] = 8478,
4215  ["TRADE"] = 8482,
4216  ["trade"] = 8482,
4217  ["Zopf"] = 8484,
4218  ["integers"] = 8484,
4219  ["mho"] = 8487,
4220  ["Zfr"] = 8488,
4221  ["zeetrf"] = 8488,
4222  ["iiota"] = 8489,
4223  ["Bernoullis"] = 8492,
4224  ["Bscr"] = 8492,
4225  ["bernou"] = 8492,
4226  ["Cayleys"] = 8493,
4227  ["Cfr"] = 8493,
4228  ["escr"] = 8495,
4229  ["Escr"] = 8496,
4230  ["expectation"] = 8496,
4231  ["Fouriertrf"] = 8497,
4232  ["Fscr"] = 8497,
4233  ["Mellintrf"] = 8499,
4234  ["Mscr"] = 8499,
4235  ["phmmat"] = 8499,
4236  ["order"] = 8500,
4237  ["orderof"] = 8500,
4238  ["oscr"] = 8500,
4239  ["alefsym"] = 8501,
```

```
4240  ["aleph"] = 8501,
4241  ["beth"] = 8502,
4242  ["gimel"] = 8503,
4243  ["daleth"] = 8504,
4244  ["CapitalDifferentialD"] = 8517,
4245  ["DD"] = 8517,
4246  ["DifferentialD"] = 8518,
4247  ["dd"] = 8518,
4248  ["ExponentialE"] = 8519,
4249  ["ee"] = 8519,
4250  ["exponentiale"] = 8519,
4251  ["ImaginaryI"] = 8520,
4252  ["ii"] = 8520,
4253  ["frac13"] = 8531,
4254  ["frac23"] = 8532,
4255  ["frac15"] = 8533,
4256  ["frac25"] = 8534,
4257  ["frac35"] = 8535,
4258  ["frac45"] = 8536,
4259  ["frac16"] = 8537,
4260  ["frac56"] = 8538,
4261  ["frac18"] = 8539,
4262  ["frac38"] = 8540,
4263  ["frac58"] = 8541,
4264  ["frac78"] = 8542,
4265  ["LeftArrow"] = 8592,
4266  ["ShortLeftArrow"] = 8592,
4267  ["larr"] = 8592,
4268  ["leftarrow"] = 8592,
4269  ["slarr"] = 8592,
4270  ["ShortUpArrow"] = 8593,
4271  ["UpArrow"] = 8593,
4272  ["uarr"] = 8593,
4273  ["uparrow"] = 8593,
4274  ["RightArrow"] = 8594,
4275  ["ShortRightArrow"] = 8594,
4276  ["rarr"] = 8594,
4277  ["rightarrow"] = 8594,
4278  ["srarr"] = 8594,
4279  ["DownArrow"] = 8595,
4280  ["ShortDownArrow"] = 8595,
4281  ["darr"] = 8595,
4282  ["downarrow"] = 8595,
4283  ["LeftRightArrow"] = 8596,
4284  ["harr"] = 8596,
4285  ["leftrightarrow"] = 8596,
4286  ["UpDownArrow"] = 8597,
```

```
  ["updownarrow"] = 8597,
  ["varr"] = 8597,
  ["UpperLeftArrow"] = 8598,
  ["nwarr"] = 8598,
  ["nwarrow"] = 8598,
  ["UpperRightArrow"] = 8599,
  ["nearr"] = 8599,
  ["nearrow"] = 8599,
  ["LowerRightArrow"] = 8600,
  ["searr"] = 8600,
  ["searrow"] = 8600,
  ["LowerLeftArrow"] = 8601,
  ["swarr"] = 8601,
  ["swarrow"] = 8601,
  ["nlarr"] = 8602,
  ["nleftarrow"] = 8602,
  ["nrarr"] = 8603,
  ["nrightarrow"] = 8603,
  ["nrarrw"] = {8605, 824},
  ["rarrw"] = 8605,
  ["rightsquigarrow"] = 8605,
  ["Larr"] = 8606,
  ["twoheadleftarrow"] = 8606,
  ["Uarr"] = 8607,
  ["Rarr"] = 8608,
  ["twoheadrightarrow"] = 8608,
  ["Darr"] = 8609,
  ["larrtl"] = 8610,
  ["leftarrowtail"] = 8610,
  ["rarrtl"] = 8611,
  ["rightarrowtail"] = 8611,
  ["LeftTeeArrow"] = 8612,
  ["mapstoleft"] = 8612,
  ["UpTeeArrow"] = 8613,
  ["mapstoup"] = 8613,
  ["RightTeeArrow"] = 8614,
  ["map"] = 8614,
  ["mapsto"] = 8614,
  ["DownTeeArrow"] = 8615,
  ["mapstodown"] = 8615,
  ["hookleftarrow"] = 8617,
  ["larrhk"] = 8617,
  ["hookrightarrow"] = 8618,
  ["rarrhk"] = 8618,
  ["larrlp"] = 8619,
  ["looparrowleft"] = 8619,
  ["looparrowright"] = 8620,
```

```
4334  ["rarrlp"] = 8620,
4335  ["harrw"] = 8621,
4336  ["leftrightsquigarrow"] = 8621,
4337  ["nharr"] = 8622,
4338  ["nleftrightarrow"] = 8622,
4339  ["Lsh"] = 8624,
4340  ["lsh"] = 8624,
4341  ["Rsh"] = 8625,
4342  ["rsh"] = 8625,
4343  ["ldsh"] = 8626,
4344  ["rdsh"] = 8627,
4345  ["crarr"] = 8629,
4346  ["cularr"] = 8630,
4347  ["curvearrowleft"] = 8630,
4348  ["curarr"] = 8631,
4349  ["curvearrowright"] = 8631,
4350  ["circlearrowleft"] = 8634,
4351  ["olarr"] = 8634,
4352  ["circlearrowright"] = 8635,
4353  ["orarr"] = 8635,
4354  ["LeftVector"] = 8636,
4355  ["leftharpoonup"] = 8636,
4356  ["lharu"] = 8636,
4357  ["DownLeftVector"] = 8637,
4358  ["leftharpoondown"] = 8637,
4359  ["lhard"] = 8637,
4360  ["RightUpVector"] = 8638,
4361  ["uharr"] = 8638,
4362  ["upharpoonright"] = 8638,
4363  ["LeftUpVector"] = 8639,
4364  ["uharl"] = 8639,
4365  ["upharpoonleft"] = 8639,
4366  ["RightVector"] = 8640,
4367  ["rharu"] = 8640,
4368  ["rightharpoonup"] = 8640,
4369  ["DownRightVector"] = 8641,
4370  ["rhard"] = 8641,
4371  ["rightharpoondown"] = 8641,
4372  ["RightDownVector"] = 8642,
4373  ["dharr"] = 8642,
4374  ["downharpoonright"] = 8642,
4375  ["LeftDownVector"] = 8643,
4376  ["dharl"] = 8643,
4377  ["downharpoonleft"] = 8643,
4378  ["RightArrowLeftArrow"] = 8644,
4379  ["rightleftarrows"] = 8644,
4380  ["rlarr"] = 8644,
```

```
4381   ["UpArrowDownArrow"] = 8645,
4382   ["udarr"] = 8645,
4383   ["LeftArrowRightArrow"] = 8646,
4384   ["leftrightarrows"] = 8646,
4385   ["lrarr"] = 8646,
4386   ["leftleftarrows"] = 8647,
4387   ["llarr"] = 8647,
4388   ["upuparrows"] = 8648,
4389   ["uuarr"] = 8648,
4390   ["rightrightarrows"] = 8649,
4391   ["rrarr"] = 8649,
4392   ["ddarr"] = 8650,
4393   ["downdownarrows"] = 8650,
4394   ["ReverseEquilibrium"] = 8651,
4395   ["leftrightharpoons"] = 8651,
4396   ["lrhar"] = 8651,
4397   ["Equilibrium"] = 8652,
4398   ["rightleftharpoons"] = 8652,
4399   ["rlhar"] = 8652,
4400   ["nLeftarrow"] = 8653,
4401   ["nlArr"] = 8653,
4402   ["nLeftrightarrow"] = 8654,
4403   ["nhArr"] = 8654,
4404   ["nRightarrow"] = 8655,
4405   ["nrArr"] = 8655,
4406   ["DoubleLeftArrow"] = 8656,
4407   ["Leftarrow"] = 8656,
4408   ["lArr"] = 8656,
4409   ["DoubleUpArrow"] = 8657,
4410   ["Uparrow"] = 8657,
4411   ["uArr"] = 8657,
4412   ["DoubleRightArrow"] = 8658,
4413   ["Implies"] = 8658,
4414   ["Rightarrow"] = 8658,
4415   ["rArr"] = 8658,
4416   ["DoubleDownArrow"] = 8659,
4417   ["Downarrow"] = 8659,
4418   ["dArr"] = 8659,
4419   ["DoubleLeftRightArrow"] = 8660,
4420   ["Leftrightarrow"] = 8660,
4421   ["hArr"] = 8660,
4422   ["iff"] = 8660,
4423   ["DoubleUpDownArrow"] = 8661,
4424   ["Updownarrow"] = 8661,
4425   ["vArr"] = 8661,
4426   ["nwArr"] = 8662,
4427   ["neArr"] = 8663,
```

```
4428    ["seArr"] = 8664,
4429    ["swArr"] = 8665,
4430    ["Lleftarrow"] = 8666,
4431    ["lAarr"] = 8666,
4432    ["Rrightarrow"] = 8667,
4433    ["rAarr"] = 8667,
4434    ["zigrarr"] = 8669,
4435    ["LeftArrowBar"] = 8676,
4436    ["larrb"] = 8676,
4437    ["RightArrowBar"] = 8677,
4438    ["rarrb"] = 8677,
4439    ["DownArrowUpArrow"] = 8693,
4440    ["duarr"] = 8693,
4441    ["loarr"] = 8701,
4442    ["roarr"] = 8702,
4443    ["hoarr"] = 8703,
4444    ["ForAll"] = 8704,
4445    ["forall"] = 8704,
4446    ["comp"] = 8705,
4447    ["complement"] = 8705,
4448    ["PartialD"] = 8706,
4449    ["npart"] = {8706, 824},
4450    ["part"] = 8706,
4451    ["Exists"] = 8707,
4452    ["exist"] = 8707,
4453    ["NotExists"] = 8708,
4454    ["nexist"] = 8708,
4455    ["nexists"] = 8708,
4456    ["empty"] = 8709,
4457    ["emptyset"] = 8709,
4458    ["emptyv"] = 8709,
4459    ["varnothing"] = 8709,
4460    ["Del"] = 8711,
4461    ["nabla"] = 8711,
4462    ["Element"] = 8712,
4463    ["in"] = 8712,
4464    ["isin"] = 8712,
4465    ["isinv"] = 8712,
4466    ["NotElement"] = 8713,
4467    ["notin"] = 8713,
4468    ["notinva"] = 8713,
4469    ["ReverseElement"] = 8715,
4470    ["SuchThat"] = 8715,
4471    ["ni"] = 8715,
4472    ["niv"] = 8715,
4473    ["NotReverseElement"] = 8716,
4474    ["notni"] = 8716,
```

```
4475  ["notniva"] = 8716,
4476  ["Product"] = 8719,
4477  ["prod"] = 8719,
4478  ["Coproduct"] = 8720,
4479  ["coprod"] = 8720,
4480  ["Sum"] = 8721,
4481  ["sum"] = 8721,
4482  ["minus"] = 8722,
4483  ["MinusPlus"] = 8723,
4484  ["mnplus"] = 8723,
4485  ["mp"] = 8723,
4486  ["dotplus"] = 8724,
4487  ["plusdo"] = 8724,
4488  ["Backslash"] = 8726,
4489  ["setminus"] = 8726,
4490  ["setmn"] = 8726,
4491  ["smallsetminus"] = 8726,
4492  ["ssetmn"] = 8726,
4493  ["lowast"] = 8727,
4494  ["SmallCircle"] = 8728,
4495  ["compfn"] = 8728,
4496  ["Sqrt"] = 8730,
4497  ["radic"] = 8730,
4498  ["Proportional"] = 8733,
4499  ["prop"] = 8733,
4500  ["propto"] = 8733,
4501  ["varpropto"] = 8733,
4502  ["vprop"] = 8733,
4503  ["infin"] = 8734,
4504  ["angrt"] = 8735,
4505  ["ang"] = 8736,
4506  ["angle"] = 8736,
4507  ["nang"] = {8736, 8402},
4508  ["angmsd"] = 8737,
4509  ["measuredangle"] = 8737,
4510  ["angsph"] = 8738,
4511  ["VerticalBar"] = 8739,
4512  ["mid"] = 8739,
4513  ["shortmid"] = 8739,
4514  ["smid"] = 8739,
4515  ["NotVerticalBar"] = 8740,
4516  ["nmid"] = 8740,
4517  ["nshortmid"] = 8740,
4518  ["nsmid"] = 8740,
4519  ["DoubleVerticalBar"] = 8741,
4520  ["par"] = 8741,
4521  ["parallel"] = 8741,
```

```
4522  ["shortparallel"] = 8741,
4523  ["spar"] = 8741,
4524  ["NotDoubleVerticalBar"] = 8742,
4525  ["npar"] = 8742,
4526  ["nparallel"] = 8742,
4527  ["nshortparallel"] = 8742,
4528  ["nspar"] = 8742,
4529  ["and"] = 8743,
4530  ["wedge"] = 8743,
4531  ["or"] = 8744,
4532  ["vee"] = 8744,
4533  ["cap"] = 8745,
4534  ["caps"] = {8745, 65024},
4535  ["cup"] = 8746,
4536  ["cups"] = {8746, 65024},
4537  ["Integral"] = 8747,
4538  ["int"] = 8747,
4539  ["Int"] = 8748,
4540  ["iiint"] = 8749,
4541  ["tint"] = 8749,
4542  ["ContourIntegral"] = 8750,
4543  ["conint"] = 8750,
4544  ["oint"] = 8750,
4545  ["Conint"] = 8751,
4546  ["DoubleContourIntegral"] = 8751,
4547  ["Cconint"] = 8752,
4548  ["cwint"] = 8753,
4549  ["ClockwiseContourIntegral"] = 8754,
4550  ["cwconint"] = 8754,
4551  ["CounterClockwiseContourIntegral"] = 8755,
4552  ["awconint"] = 8755,
4553  ["Therefore"] = 8756,
4554  ["there4"] = 8756,
4555  ["therefore"] = 8756,
4556  ["Because"] = 8757,
4557  ["becaus"] = 8757,
4558  ["because"] = 8757,
4559  ["ratio"] = 8758,
4560  ["Colon"] = 8759,
4561  ["Proportion"] = 8759,
4562  ["dotminus"] = 8760,
4563  ["minusd"] = 8760,
4564  ["mDDot"] = 8762,
4565  ["homtht"] = 8763,
4566  ["Tilde"] = 8764,
4567  ["nvsim"] = {8764, 8402},
4568  ["sim"] = 8764,
```

```
["thicksim"] = 8764,
["thksim"] = 8764,
["backsim"] = 8765,
["bsim"] = 8765,
["race"] = {8765, 817},
["ac"] = 8766,
["acE"] = {8766, 819},
["mstpos"] = 8766,
["acd"] = 8767,
["VerticalTilde"] = 8768,
["wr"] = 8768,
["wreath"] = 8768,
["NotTilde"] = 8769,
["nsim"] = 8769,
["EqualTilde"] = 8770,
["NotEqualTilde"] = {8770, 824},
["eqsim"] = 8770,
["esim"] = 8770,
["nesim"] = {8770, 824},
["TildeEqual"] = 8771,
["sime"] = 8771,
["simeq"] = 8771,
["NotTildeEqual"] = 8772,
["nsime"] = 8772,
["nsimeq"] = 8772,
["TildeFullEqual"] = 8773,
["cong"] = 8773,
["simne"] = 8774,
["NotTildeFullEqual"] = 8775,
["ncong"] = 8775,
["TildeTilde"] = 8776,
["ap"] = 8776,
["approx"] = 8776,
["asymp"] = 8776,
["thickapprox"] = 8776,
["thkap"] = 8776,
["NotTildeTilde"] = 8777,
["nap"] = 8777,
["napprox"] = 8777,
["ape"] = 8778,
["approxeq"] = 8778,
["apid"] = 8779,
["napid"] = {8779, 824},
["backcong"] = 8780,
["bcong"] = 8780,
["CupCap"] = 8781,
["asympeq"] = 8781,
```

```
4616    ["nvap"] = {8781, 8402},
4617    ["Bumpeq"] = 8782,
4618    ["HumpDownHump"] = 8782,
4619    ["NotHumpDownHump"] = {8782, 824},
4620    ["bump"] = 8782,
4621    ["nbump"] = {8782, 824},
4622    ["HumpEqual"] = 8783,
4623    ["NotHumpEqual"] = {8783, 824},
4624    ["bumpe"] = 8783,
4625    ["bumpeq"] = 8783,
4626    ["nbumpe"] = {8783, 824},
4627    ["DotEqual"] = 8784,
4628    ["doteq"] = 8784,
4629    ["esdot"] = 8784,
4630    ["nedot"] = {8784, 824},
4631    ["doteqdot"] = 8785,
4632    ["eDot"] = 8785,
4633    ["efDot"] = 8786,
4634    ["fallingdotseq"] = 8786,
4635    ["erDot"] = 8787,
4636    ["risingdotseq"] = 8787,
4637    ["Assign"] = 8788,
4638    ["colone"] = 8788,
4639    ["coloneq"] = 8788,
4640    ["ecolon"] = 8789,
4641    ["eqcolon"] = 8789,
4642    ["ecir"] = 8790,
4643    ["eqcirc"] = 8790,
4644    ["circeq"] = 8791,
4645    ["cire"] = 8791,
4646    ["wedgeq"] = 8793,
4647    ["veeeq"] = 8794,
4648    ["triangleq"] = 8796,
4649    ["trie"] = 8796,
4650    ["equest"] = 8799,
4651    ["questeq"] = 8799,
4652    ["NotEqual"] = 8800,
4653    ["ne"] = 8800,
4654    ["Congruent"] = 8801,
4655    ["bnequiv"] = {8801, 8421},
4656    ["equiv"] = 8801,
4657    ["NotCongruent"] = 8802,
4658    ["nequiv"] = 8802,
4659    ["le"] = 8804,
4660    ["leq"] = 8804,
4661    ["nvle"] = {8804, 8402},
4662    ["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,
```

```
4710    ["nge"] = 8817,
4711    ["ngeq"] = 8817,
4712    ["LessTilde"] = 8818,
4713    ["lesssim"] = 8818,
4714    ["lsim"] = 8818,
4715    ["GreaterTilde"] = 8819,
4716    ["gsim"] = 8819,
4717    ["gtrsim"] = 8819,
4718    ["NotLessTilde"] = 8820,
4719    ["nlsim"] = 8820,
4720    ["NotGreaterTilde"] = 8821,
4721    ["ngsim"] = 8821,
4722    ["LessGreater"] = 8822,
4723    ["lessgtr"] = 8822,
4724    ["lg"] = 8822,
4725    ["GreaterLess"] = 8823,
4726    ["gl"] = 8823,
4727    ["gtrless"] = 8823,
4728    ["NotLessGreater"] = 8824,
4729    ["ntlg"] = 8824,
4730    ["NotGreaterLess"] = 8825,
4731    ["ntgl"] = 8825,
4732    ["Precedes"] = 8826,
4733    ["pr"] = 8826,
4734    ["prec"] = 8826,
4735    ["Succeeds"] = 8827,
4736    ["sc"] = 8827,
4737    ["succ"] = 8827,
4738    ["PrecedesSlantEqual"] = 8828,
4739    ["prcue"] = 8828,
4740    ["preccurlyeq"] = 8828,
4741    ["SucceedsSlantEqual"] = 8829,
4742    ["sccue"] = 8829,
4743    ["succcurlyeq"] = 8829,
4744    ["PrecedesTilde"] = 8830,
4745    ["precsim"] = 8830,
4746    ["prsim"] = 8830,
4747    ["NotSucceedsTilde"] = {8831, 824},
4748    ["SucceedsTilde"] = 8831,
4749    ["scsim"] = 8831,
4750    ["succsim"] = 8831,
4751    ["NotPrecedes"] = 8832,
4752    ["npr"] = 8832,
4753    ["nprec"] = 8832,
4754    ["NotSucceeds"] = 8833,
4755    ["nsc"] = 8833,
4756    ["nsucc"] = 8833,
```

```
["NotSubset"] = {8834, 8402},
["nsubset"] = {8834, 8402},
["sub"] = 8834,
["subset"] = 8834,
["vnsub"] = {8834, 8402},
["NotSuperset"] = {8835, 8402},
["Superset"] = 8835,
["nsupset"] = {8835, 8402},
["sup"] = 8835,
["supset"] = 8835,
["vnsup"] = {8835, 8402},
["nsub"] = 8836,
["nsup"] = 8837,
["SubsetEqual"] = 8838,
["sube"] = 8838,
["subseteq"] = 8838,
["SupersetEqual"] = 8839,
["supe"] = 8839,
["supseteq"] = 8839,
["NotSubsetEqual"] = 8840,
["nsube"] = 8840,
["nsubseteq"] = 8840,
["NotSupersetEqual"] = 8841,
["nsupe"] = 8841,
["nsupseteq"] = 8841,
["subne"] = 8842,
["subsetneq"] = 8842,
["varsubsetneq"] = {8842, 65024},
["vsubne"] = {8842, 65024},
["supne"] = 8843,
["supsetneq"] = 8843,
["varsupsetneq"] = {8843, 65024},
["vsupne"] = {8843, 65024},
["cupdot"] = 8845,
["UnionPlus"] = 8846,
["uplus"] = 8846,
["NotSquareSubset"] = {8847, 824},
["SquareSubset"] = 8847,
["sqsub"] = 8847,
["sqsubset"] = 8847,
["NotSquareSuperset"] = {8848, 824},
["SquareSuperset"] = 8848,
["sqsup"] = 8848,
["sqsupset"] = 8848,
["SquareSubsetEqual"] = 8849,
["sqsube"] = 8849,
["sqsubseteq"] = 8849,
```

```
4804    ["SquareSupersetEqual"] = 8850,
4805    ["sqsupe"] = 8850,
4806    ["sqsupseteq"] = 8850,
4807    ["SquareIntersection"] = 8851,
4808    ["sqcap"] = 8851,
4809    ["sqcaps"] = {8851, 65024},
4810    ["SquareUnion"] = 8852,
4811    ["sqcup"] = 8852,
4812    ["sqcups"] = {8852, 65024},
4813    ["CirclePlus"] = 8853,
4814    ["oplus"] = 8853,
4815    ["CircleMinus"] = 8854,
4816    ["ominus"] = 8854,
4817    ["CircleTimes"] = 8855,
4818    ["otimes"] = 8855,
4819    ["osol"] = 8856,
4820    ["CircleDot"] = 8857,
4821    ["odot"] = 8857,
4822    ["circledcirc"] = 8858,
4823    ["ocir"] = 8858,
4824    ["circledast"] = 8859,
4825    ["oast"] = 8859,
4826    ["circleddash"] = 8861,
4827    ["odash"] = 8861,
4828    ["boxplus"] = 8862,
4829    ["plusb"] = 8862,
4830    ["boxminus"] = 8863,
4831    ["minusb"] = 8863,
4832    ["boxtimes"] = 8864,
4833    ["timesb"] = 8864,
4834    ["dotsquare"] = 8865,
4835    ["sdotb"] = 8865,
4836    ["RightTee"] = 8866,
4837    ["vdash"] = 8866,
4838    ["LeftTee"] = 8867,
4839    ["dashv"] = 8867,
4840    ["DownTee"] = 8868,
4841    ["top"] = 8868,
4842    ["UpTee"] = 8869,
4843    ["bot"] = 8869,
4844    ["bottom"] = 8869,
4845    ["perp"] = 8869,
4846    ["models"] = 8871,
4847    ["DoubleRightTee"] = 8872,
4848    ["vDash"] = 8872,
4849    ["Vdash"] = 8873,
4850    ["Vvdash"] = 8874,
```

```
4851  ["VDash"] = 8875,
4852  ["nvdash"] = 8876,
4853  ["nvDash"] = 8877,
4854  ["nVdash"] = 8878,
4855  ["nVDash"] = 8879,
4856  ["prurel"] = 8880,
4857  ["LeftTriangle"] = 8882,
4858  ["vartriangleleft"] = 8882,
4859  ["vltri"] = 8882,
4860  ["RightTriangle"] = 8883,
4861  ["vartriangleright"] = 8883,
4862  ["vrtri"] = 8883,
4863  ["LeftTriangleEqual"] = 8884,
4864  ["ltrie"] = 8884,
4865  ["nvltrie"] = {8884, 8402},
4866  ["trianglelefteq"] = 8884,
4867  ["RightTriangleEqual"] = 8885,
4868  ["nvrtrie"] = {8885, 8402},
4869  ["rtrie"] = 8885,
4870  ["trianglerighteq"] = 8885,
4871  ["origof"] = 8886,
4872  ["imof"] = 8887,
4873  ["multimap"] = 8888,
4874  ["mumap"] = 8888,
4875  ["hercon"] = 8889,
4876  ["intcal"] = 8890,
4877  ["intercal"] = 8890,
4878  ["veebar"] = 8891,
4879  ["barvee"] = 8893,
4880  ["angrtvb"] = 8894,
4881  ["lrtri"] = 8895,
4882  ["Wedge"] = 8896,
4883  ["bigwedge"] = 8896,
4884  ["xwedge"] = 8896,
4885  ["Vee"] = 8897,
4886  ["bigvee"] = 8897,
4887  ["xvee"] = 8897,
4888  ["Intersection"] = 8898,
4889  ["bigcap"] = 8898,
4890  ["xcap"] = 8898,
4891  ["Union"] = 8899,
4892  ["bigcup"] = 8899,
4893  ["xcup"] = 8899,
4894  ["Diamond"] = 8900,
4895  ["diam"] = 8900,
4896  ["diamond"] = 8900,
4897  ["sdot"] = 8901,
```

```
["Star"] = 8902,
["sstarf"] = 8902,
["divideontimes"] = 8903,
["divonx"] = 8903,
["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,
```

```
4945    ["NotPrecedesSlantEqual"] = 8928,
4946    ["nprcue"] = 8928,
4947    ["NotSucceedsSlantEqual"] = 8929,
4948    ["nsccue"] = 8929,
4949    ["NotSquareSubsetEqual"] = 8930,
4950    ["nsqsube"] = 8930,
4951    ["NotSquareSupersetEqual"] = 8931,
4952    ["nsqsupe"] = 8931,
4953    ["lnsim"] = 8934,
4954    ["gnsim"] = 8935,
4955    ["precnsim"] = 8936,
4956    ["prnsim"] = 8936,
4957    ["scnsim"] = 8937,
4958    ["succnsim"] = 8937,
4959    ["NotLeftTriangle"] = 8938,
4960    ["nltri"] = 8938,
4961    ["ntriangleleft"] = 8938,
4962    ["NotRightTriangle"] = 8939,
4963    ["nrtri"] = 8939,
4964    ["ntriangleright"] = 8939,
4965    ["NotLeftTriangleEqual"] = 8940,
4966    ["nltrie"] = 8940,
4967    ["ntrianglelefteq"] = 8940,
4968    ["NotRightTriangleEqual"] = 8941,
4969    ["nrtrie"] = 8941,
4970    ["ntrianglerighteq"] = 8941,
4971    ["vellip"] = 8942,
4972    ["ctdot"] = 8943,
4973    ["utdot"] = 8944,
4974    ["dtdot"] = 8945,
4975    ["disin"] = 8946,
4976    ["isinsv"] = 8947,
4977    ["isins"] = 8948,
4978    ["isindot"] = 8949,
4979    ["notindot"] = {8949, 824},
4980    ["notinvc"] = 8950,
4981    ["notinvb"] = 8951,
4982    ["isinE"] = 8953,
4983    ["notinE"] = {8953, 824},
4984    ["nisd"] = 8954,
4985    ["xnis"] = 8955,
4986    ["nis"] = 8956,
4987    ["notnivc"] = 8957,
4988    ["notnivb"] = 8958,
4989    ["barwed"] = 8965,
4990    ["barwedge"] = 8965,
4991    ["Barwed"] = 8966,
```

```
4992  ["doublebarwedge"] = 8966,
4993  ["LeftCeiling"] = 8968,
4994  ["lceil"] = 8968,
4995  ["RightCeiling"] = 8969,
4996  ["rceil"] = 8969,
4997  ["LeftFloor"] = 8970,
4998  ["lfloor"] = 8970,
4999  ["RightFloor"] = 8971,
5000  ["rfloor"] = 8971,
5001  ["drcrop"] = 8972,
5002  ["dlcrop"] = 8973,
5003  ["urcrop"] = 8974,
5004  ["ulcrop"] = 8975,
5005  ["bnot"] = 8976,
5006  ["profline"] = 8978,
5007  ["profsurf"] = 8979,
5008  ["telrec"] = 8981,
5009  ["target"] = 8982,
5010  ["ulcorn"] = 8988,
5011  ["ulcorner"] = 8988,
5012  ["urcorn"] = 8989,
5013  ["urcorner"] = 8989,
5014  ["dlcorn"] = 8990,
5015  ["llcorner"] = 8990,
5016  ["drcorn"] = 8991,
5017  ["lrcorner"] = 8991,
5018  ["frown"] = 8994,
5019  ["sfrown"] = 8994,
5020  ["smile"] = 8995,
5021  ["ssmile"] = 8995,
5022  ["cylcty"] = 9005,
5023  ["profalar"] = 9006,
5024  ["topbot"] = 9014,
5025  ["ovbar"] = 9021,
5026  ["solbar"] = 9023,
5027  ["angzarr"] = 9084,
5028  ["lmoust"] = 9136,
5029  ["lmoustache"] = 9136,
5030  ["rmoust"] = 9137,
5031  ["rmoustache"] = 9137,
5032  ["OverBracket"] = 9140,
5033  ["tbrk"] = 9140,
5034  ["UnderBracket"] = 9141,
5035  ["bbrk"] = 9141,
5036  ["bbrktbrk"] = 9142,
5037  ["OverParenthesis"] = 9180,
5038  ["UnderParenthesis"] = 9181,
```

```
5039   ["OverBrace"] = 9182,
5040   ["UnderBrace"] = 9183,
5041   ["trpezium"] = 9186,
5042   ["elinters"] = 9191,
5043   ["blank"] = 9251,
5044   ["circledS"] = 9416,
5045   ["oS"] = 9416,
5046   ["HorizontalLine"] = 9472,
5047   ["boxh"] = 9472,
5048   ["boxv"] = 9474,
5049   ["boxdr"] = 9484,
5050   ["boxdl"] = 9488,
5051   ["boxur"] = 9492,
5052   ["boxul"] = 9496,
5053   ["boxvr"] = 9500,
5054   ["boxvl"] = 9508,
5055   ["boxhd"] = 9516,
5056   ["boxhu"] = 9524,
5057   ["boxvh"] = 9532,
5058   ["boxH"] = 9552,
5059   ["boxV"] = 9553,
5060   ["boxdR"] = 9554,
5061   ["boxDr"] = 9555,
5062   ["boxDR"] = 9556,
5063   ["boxdL"] = 9557,
5064   ["boxDl"] = 9558,
5065   ["boxDL"] = 9559,
5066   ["boxuR"] = 9560,
5067   ["boxUr"] = 9561,
5068   ["boxUR"] = 9562,
5069   ["boxuL"] = 9563,
5070   ["boxUl"] = 9564,
5071   ["boxUL"] = 9565,
5072   ["boxvR"] = 9566,
5073   ["boxVr"] = 9567,
5074   ["boxVR"] = 9568,
5075   ["boxvL"] = 9569,
5076   ["boxVl"] = 9570,
5077   ["boxVL"] = 9571,
5078   ["boxHd"] = 9572,
5079   ["boxhD"] = 9573,
5080   ["boxHD"] = 9574,
5081   ["boxHu"] = 9575,
5082   ["boxhU"] = 9576,
5083   ["boxHU"] = 9577,
5084   ["boxvH"] = 9578,
5085   ["boxVh"] = 9579,
```

```
  ["boxVH"] = 9580,
  ["uhblk"] = 9600,
  ["lhblk"] = 9604,
  ["block"] = 9608,
  ["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,
```

```
5133   ["EmptySmallSquare"] = 9723,
5134   ["FilledSmallSquare"] = 9724,
5135   ["bigstar"] = 9733,
5136   ["starf"] = 9733,
5137   ["star"] = 9734,
5138   ["phone"] = 9742,
5139   ["female"] = 9792,
5140   ["male"] = 9794,
5141   ["spades"] = 9824,
5142   ["spadesuit"] = 9824,
5143   ["clubs"] = 9827,
5144   ["clubsuit"] = 9827,
5145   ["hearts"] = 9829,
5146   ["heartsuit"] = 9829,
5147   ["diamondsuit"] = 9830,
5148   ["diams"] = 9830,
5149   ["sung"] = 9834,
5150   ["flat"] = 9837,
5151   ["natur"] = 9838,
5152   ["natural"] = 9838,
5153   ["sharp"] = 9839,
5154   ["check"] = 10003,
5155   ["checkmark"] = 10003,
5156   ["cross"] = 10007,
5157   ["malt"] = 10016,
5158   ["maltese"] = 10016,
5159   ["sext"] = 10038,
5160   ["VerticalSeparator"] = 10072,
5161   ["lbbrk"] = 10098,
5162   ["rbbrk"] = 10099,
5163   ["bsolhsub"] = 10184,
5164   ["suphsol"] = 10185,
5165   ["LeftDoubleBracket"] = 10214,
5166   ["lobrk"] = 10214,
5167   ["RightDoubleBracket"] = 10215,
5168   ["robrk"] = 10215,
5169   ["LeftAngleBracket"] = 10216,
5170   ["lang"] = 10216,
5171   ["langle"] = 10216,
5172   ["RightAngleBracket"] = 10217,
5173   ["rang"] = 10217,
5174   ["rangle"] = 10217,
5175   ["Lang"] = 10218,
5176   ["Rang"] = 10219,
5177   ["loang"] = 10220,
5178   ["roang"] = 10221,
5179   ["LongLeftArrow"] = 10229,
```

```
5180   ["longleftarrow"] = 10229,
5181   ["xlarr"] = 10229,
5182   ["LongRightArrow"] = 10230,
5183   ["longrightarrow"] = 10230,
5184   ["xrarr"] = 10230,
5185   ["LongLeftRightArrow"] = 10231,
5186   ["longleftrightarrow"] = 10231,
5187   ["xharr"] = 10231,
5188   ["DoubleLongLeftArrow"] = 10232,
5189   ["Longleftarrow"] = 10232,
5190   ["xlArr"] = 10232,
5191   ["DoubleLongRightArrow"] = 10233,
5192   ["Longrightarrow"] = 10233,
5193   ["xrArr"] = 10233,
5194   ["DoubleLongLeftRightArrow"] = 10234,
5195   ["Longleftrightarrow"] = 10234,
5196   ["xhArr"] = 10234,
5197   ["longmapsto"] = 10236,
5198   ["xmap"] = 10236,
5199   ["dzigrarr"] = 10239,
5200   ["nvlArr"] = 10498,
5201   ["nvrArr"] = 10499,
5202   ["nvHarr"] = 10500,
5203   ["Map"] = 10501,
5204   ["lbarr"] = 10508,
5205   ["bkarow"] = 10509,
5206   ["rbarr"] = 10509,
5207   ["lBarr"] = 10510,
5208   ["dbkarow"] = 10511,
5209   ["rBarr"] = 10511,
5210   ["RBarr"] = 10512,
5211   ["drbkarow"] = 10512,
5212   ["DDotrahd"] = 10513,
5213   ["UpArrowBar"] = 10514,
5214   ["DownArrowBar"] = 10515,
5215   ["Rarrtl"] = 10518,
5216   ["latail"] = 10521,
5217   ["ratail"] = 10522,
5218   ["lAtail"] = 10523,
5219   ["rAtail"] = 10524,
5220   ["larrfs"] = 10525,
5221   ["rarrfs"] = 10526,
5222   ["larrbfs"] = 10527,
5223   ["rarrbfs"] = 10528,
5224   ["nwarhk"] = 10531,
5225   ["nearhk"] = 10532,
5226   ["hksearow"] = 10533,
```

```
  ["searhk"] = 10533,
  ["hkswarow"] = 10534,
  ["swarhk"] = 10534,
  ["nwnear"] = 10535,
  ["nesear"] = 10536,
  ["toea"] = 10536,
  ["seswar"] = 10537,
  ["tosa"] = 10537,
  ["swnwar"] = 10538,
  ["nrarrc"] = {10547, 824},
  ["rarrc"] = 10547,
  ["cudarrr"] = 10549,
  ["ldca"] = 10550,
  ["rdca"] = 10551,
  ["cudarrl"] = 10552,
  ["larrpl"] = 10553,
  ["curarrm"] = 10556,
  ["cularrp"] = 10557,
  ["rarrpl"] = 10565,
  ["harrcir"] = 10568,
  ["Uarrocir"] = 10569,
  ["lurdshar"] = 10570,
  ["ldrushar"] = 10571,
  ["LeftRightVector"] = 10574,
  ["RightUpDownVector"] = 10575,
  ["DownLeftRightVector"] = 10576,
  ["LeftUpDownVector"] = 10577,
  ["LeftVectorBar"] = 10578,
  ["RightVectorBar"] = 10579,
  ["RightUpVectorBar"] = 10580,
  ["RightDownVectorBar"] = 10581,
  ["DownLeftVectorBar"] = 10582,
  ["DownRightVectorBar"] = 10583,
  ["LeftUpVectorBar"] = 10584,
  ["LeftDownVectorBar"] = 10585,
  ["LeftTeeVector"] = 10586,
  ["RightTeeVector"] = 10587,
  ["RightUpTeeVector"] = 10588,
  ["RightDownTeeVector"] = 10589,
  ["DownLeftTeeVector"] = 10590,
  ["DownRightTeeVector"] = 10591,
  ["LeftUpTeeVector"] = 10592,
  ["LeftDownTeeVector"] = 10593,
  ["lHar"] = 10594,
  ["uHar"] = 10595,
  ["rHar"] = 10596,
  ["dHar"] = 10597,
```

```
  ["luruhar"] = 10598,
  ["ldrdhar"] = 10599,
  ["ruluhar"] = 10600,
  ["rdldhar"] = 10601,
  ["lharul"] = 10602,
  ["llhard"] = 10603,
  ["rharul"] = 10604,
  ["lrhard"] = 10605,
  ["UpEquilibrium"] = 10606,
  ["udhar"] = 10606,
  ["ReverseUpEquilibrium"] = 10607,
  ["duhar"] = 10607,
  ["RoundImplies"] = 10608,
  ["erarr"] = 10609,
  ["simrarr"] = 10610,
  ["larrsim"] = 10611,
  ["rarrsim"] = 10612,
  ["rarrap"] = 10613,
  ["ltlarr"] = 10614,
  ["gtrarr"] = 10616,
  ["subrarr"] = 10617,
  ["suplarr"] = 10619,
  ["lfisht"] = 10620,
  ["rfisht"] = 10621,
  ["ufisht"] = 10622,
  ["dfisht"] = 10623,
  ["lopar"] = 10629,
  ["ropar"] = 10630,
  ["lbrke"] = 10635,
  ["rbrke"] = 10636,
  ["lbrkslu"] = 10637,
  ["rbrksld"] = 10638,
  ["lbrksld"] = 10639,
  ["rbrkslu"] = 10640,
  ["langd"] = 10641,
  ["rangd"] = 10642,
  ["lparlt"] = 10643,
  ["rpargt"] = 10644,
  ["gtlPar"] = 10645,
  ["ltrPar"] = 10646,
  ["vzigzag"] = 10650,
  ["vangrt"] = 10652,
  ["angrtvbd"] = 10653,
  ["ange"] = 10660,
  ["range"] = 10661,
  ["dwangle"] = 10662,
  ["uwangle"] = 10663,
```

```
5321   ["angmsdaa"] = 10664,
5322   ["angmsdab"] = 10665,
5323   ["angmsdac"] = 10666,
5324   ["angmsdad"] = 10667,
5325   ["angmsdae"] = 10668,
5326   ["angmsdaf"] = 10669,
5327   ["angmsdag"] = 10670,
5328   ["angmsdah"] = 10671,
5329   ["bemptyv"] = 10672,
5330   ["demptyv"] = 10673,
5331   ["cemptyv"] = 10674,
5332   ["raemptyv"] = 10675,
5333   ["laemptyv"] = 10676,
5334   ["ohbar"] = 10677,
5335   ["omid"] = 10678,
5336   ["opar"] = 10679,
5337   ["operp"] = 10681,
5338   ["olcross"] = 10683,
5339   ["odsold"] = 10684,
5340   ["olcir"] = 10686,
5341   ["ofcir"] = 10687,
5342   ["olt"] = 10688,
5343   ["ogt"] = 10689,
5344   ["cirscir"] = 10690,
5345   ["cirE"] = 10691,
5346   ["solb"] = 10692,
5347   ["bsolb"] = 10693,
5348   ["boxbox"] = 10697,
5349   ["trisb"] = 10701,
5350   ["rtriltri"] = 10702,
5351   ["LeftTriangleBar"] = 10703,
5352   ["NotLeftTriangleBar"] = {10703, 824},
5353   ["NotRightTriangleBar"] = {10704, 824},
5354   ["RightTriangleBar"] = 10704,
5355   ["iinfin"] = 10716,
5356   ["infintie"] = 10717,
5357   ["nvinfin"] = 10718,
5358   ["eparsl"] = 10723,
5359   ["smeparsl"] = 10724,
5360   ["eqvparsl"] = 10725,
5361   ["blacklozenge"] = 10731,
5362   ["lozf"] = 10731,
5363   ["RuleDelayed"] = 10740,
5364   ["dsol"] = 10742,
5365   ["bigodot"] = 10752,
5366   ["xodot"] = 10752,
5367   ["bigoplus"] = 10753,
```

```
5368    ["xoplus"] = 10753,
5369    ["bigotimes"] = 10754,
5370    ["xotime"] = 10754,
5371    ["biguplus"] = 10756,
5372    ["xuplus"] = 10756,
5373    ["bigsqcup"] = 10758,
5374    ["xsqcup"] = 10758,
5375    ["iiiint"] = 10764,
5376    ["qint"] = 10764,
5377    ["fpartint"] = 10765,
5378    ["cirfnint"] = 10768,
5379    ["awint"] = 10769,
5380    ["rppolint"] = 10770,
5381    ["scpolint"] = 10771,
5382    ["npolint"] = 10772,
5383    ["pointint"] = 10773,
5384    ["quatint"] = 10774,
5385    ["intlarhk"] = 10775,
5386    ["pluscir"] = 10786,
5387    ["plusacir"] = 10787,
5388    ["simplus"] = 10788,
5389    ["plusdu"] = 10789,
5390    ["plussim"] = 10790,
5391    ["plustwo"] = 10791,
5392    ["mcomma"] = 10793,
5393    ["minusdu"] = 10794,
5394    ["loplus"] = 10797,
5395    ["roplus"] = 10798,
5396    ["Cross"] = 10799,
5397    ["timesd"] = 10800,
5398    ["timesbar"] = 10801,
5399    ["smashp"] = 10803,
5400    ["lotimes"] = 10804,
5401    ["rotimes"] = 10805,
5402    ["otimesas"] = 10806,
5403    ["Otimes"] = 10807,
5404    ["odiv"] = 10808,
5405    ["triplus"] = 10809,
5406    ["triminus"] = 10810,
5407    ["tritime"] = 10811,
5408    ["intprod"] = 10812,
5409    ["iprod"] = 10812,
5410    ["amalg"] = 10815,
5411    ["capdot"] = 10816,
5412    ["ncup"] = 10818,
5413    ["ncap"] = 10819,
5414    ["capand"] = 10820,
```

```
5415  ["cupor"] = 10821,
5416  ["cupcap"] = 10822,
5417  ["capcup"] = 10823,
5418  ["cupbrcap"] = 10824,
5419  ["capbrcup"] = 10825,
5420  ["cupcup"] = 10826,
5421  ["capcap"] = 10827,
5422  ["ccups"] = 10828,
5423  ["ccaps"] = 10829,
5424  ["ccupssm"] = 10832,
5425  ["And"] = 10835,
5426  ["Or"] = 10836,
5427  ["andand"] = 10837,
5428  ["oror"] = 10838,
5429  ["orslope"] = 10839,
5430  ["andslope"] = 10840,
5431  ["andv"] = 10842,
5432  ["orv"] = 10843,
5433  ["andd"] = 10844,
5434  ["ord"] = 10845,
5435  ["wedbar"] = 10847,
5436  ["sdote"] = 10854,
5437  ["simdot"] = 10858,
5438  ["congdot"] = 10861,
5439  ["ncongdot"] = {10861, 824},
5440  ["easter"] = 10862,
5441  ["apacir"] = 10863,
5442  ["apE"] = 10864,
5443  ["napE"] = {10864, 824},
5444  ["eplus"] = 10865,
5445  ["pluse"] = 10866,
5446  ["Esim"] = 10867,
5447  ["Colone"] = 10868,
5448  ["Equal"] = 10869,
5449  ["ddotseq"] = 10871,
5450  ["eDDot"] = 10871,
5451  ["equivDD"] = 10872,
5452  ["ltcir"] = 10873,
5453  ["gtcir"] = 10874,
5454  ["ltquest"] = 10875,
5455  ["gtquest"] = 10876,
5456  ["LessSlantEqual"] = 10877,
5457  ["NotLessSlantEqual"] = {10877, 824},
5458  ["leqslant"] = 10877,
5459  ["les"] = 10877,
5460  ["nleqslant"] = {10877, 824},
5461  ["nles"] = {10877, 824},
```

```
5462   ["GreaterSlantEqual"] = 10878,
5463   ["NotGreaterSlantEqual"] = {10878, 824},
5464   ["geqslant"] = 10878,
5465   ["ges"] = 10878,
5466   ["ngeqslant"] = {10878, 824},
5467   ["nges"] = {10878, 824},
5468   ["lesdot"] = 10879,
5469   ["gesdot"] = 10880,
5470   ["lesdoto"] = 10881,
5471   ["gesdoto"] = 10882,
5472   ["lesdotor"] = 10883,
5473   ["gesdotol"] = 10884,
5474   ["lap"] = 10885,
5475   ["lessapprox"] = 10885,
5476   ["gap"] = 10886,
5477   ["gtrapprox"] = 10886,
5478   ["lne"] = 10887,
5479   ["lneq"] = 10887,
5480   ["gne"] = 10888,
5481   ["gneq"] = 10888,
5482   ["lnap"] = 10889,
5483   ["lnapprox"] = 10889,
5484   ["gnap"] = 10890,
5485   ["gnapprox"] = 10890,
5486   ["lEg"] = 10891,
5487   ["lesseqqgtr"] = 10891,
5488   ["gEl"] = 10892,
5489   ["gtreqqless"] = 10892,
5490   ["lsime"] = 10893,
5491   ["gsime"] = 10894,
5492   ["lsimg"] = 10895,
5493   ["gsiml"] = 10896,
5494   ["lgE"] = 10897,
5495   ["glE"] = 10898,
5496   ["lesges"] = 10899,
5497   ["gesles"] = 10900,
5498   ["els"] = 10901,
5499   ["eqslantless"] = 10901,
5500   ["egs"] = 10902,
5501   ["eqslantgtr"] = 10902,
5502   ["elsdot"] = 10903,
5503   ["egsdot"] = 10904,
5504   ["el"] = 10905,
5505   ["eg"] = 10906,
5506   ["siml"] = 10909,
5507   ["simg"] = 10910,
5508   ["simlE"] = 10911,
```

```
  ["simgE"] = 10912,
  ["LessLess"] = 10913,
  ["NotNestedLessLess"] = {10913, 824},
  ["GreaterGreater"] = 10914,
  ["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,
["subedot"] = 10947,
["supedot"] = 10948,
["nsubE"] = {10949, 824},
["nsubseteqq"] = {10949, 824},
["subE"] = 10949,
["subseteqq"] = 10949,
["nsupE"] = {10950, 824},
["nsupseteqq"] = {10950, 824},
["supE"] = 10950,
["supseteqq"] = 10950,
["subsim"] = 10951,
["supsim"] = 10952,
["subnE"] = 10955,
["subsetneqq"] = 10955,
["varsubsetneqq"] = {10955, 65024},
["vsubnE"] = {10955, 65024},
["supnE"] = 10956,
["supsetneqq"] = 10956,
["varsupsetneqq"] = {10956, 65024},
["vsupnE"] = {10956, 65024},
["csub"] = 10959,
["csup"] = 10960,
["csube"] = 10961,
["csupe"] = 10962,
["subsup"] = 10963,
["supsub"] = 10964,
["subsub"] = 10965,
["supsup"] = 10966,
["suphsub"] = 10967,
["supdsub"] = 10968,
["forkv"] = 10969,
["topfork"] = 10970,
["mlcp"] = 10971,
["Dashv"] = 10980,
["DoubleLeftTee"] = 10980,
["Vdashl"] = 10982,
["Barv"] = 10983,
["vBar"] = 10984,
["vBarv"] = 10985,
["Vbar"] = 10987,
["Not"] = 10988,
["bNot"] = 10989,
```

```
  ["rnmid"] = 10990,
  ["cirmid"] = 10991,
  ["midcir"] = 10992,
  ["topcir"] = 10993,
  ["nhpar"] = 10994,
  ["parsim"] = 10995,
  ["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)
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.

```
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.

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

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

```
  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.

```
  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.

```
  local slice_specifiers = {}
  for specifier in options.slice:gmatch("[^%s]+") do
    table.insert(slice_specifiers, specifier)
  end

```

```
5795   if #slice_specifiers == 2 then
5796     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5797     local slice_begin_type = self.slice_begin:sub(1, 1)
5798     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5799       self.slice_begin = "^" .. self.slice_begin
5800     end
5801     local slice_end_type = self.slice_end:sub(1, 1)
5802     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5803       self.slice_end = "$" .. self.slice_end
5804     end
5805   elseif #slice_specifiers == 1 then
5806     self.slice_begin = "^" .. slice_specifiers[1]
5807     self.slice_end = "$" .. slice_specifiers[1]
5808   end
5809
5810   self.slice_begin_type = self.slice_begin:sub(1, 1)
5811   self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5812   self.slice_end_type = self.slice_end:sub(1, 1)
5813   self.slice_end_identifier = self.slice_end:sub(2) or ""
5814
5815   if self.slice_begin == "^" and self.slice_end ~= "^" then
5816     self.is_writing = true
5817   else
5818     self.is_writing = false
5819   end
```

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

```
5820   self.space = " "
```

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

```
5821   self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5822   function self.plain(s)
5823     return s
5824   end
```

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

```
5825   function self.paragraph(s)
5826     if not self.is_writing then return "" end
5827     return s
5828   end
```

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

```
5829   self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
5830   function self.interblocksep()
5831     if not self.is_writing then return "" end
```

```
5832      return self.interblocksep_text
5833    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.

```
5834    self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
5835    function self.paragraphsep()
5836      if not self.is_writing then return "" end
5837      return self.paragraphsep_text
5838    end
```

Define writer->undosep as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
5839    self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
5840    function self.undosep()
5841      if not self.is_writing then return "" end
5842      return self.undosep_text
5843    end
```

Define writer->soft_line_break as the output format of a soft line break.

```
5844    self.soft_line_break = function()
5845      if self.flatten_inlines then return "\n" end
5846      return "\\markdownRendererSoftLineBreak\n{}"
5847    end
```

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

```
5848    self.hard_line_break = function()
5849      if self.flatten_inlines then return "\n" end
5850      return "\\markdownRendererHardLineBreak\n{}"
5851    end
```

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

```
5852    self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
5853    function self.thematic_break()
5854      if not self.is_writing then return "" end
5855      return "\\markdownRendererThematicBreak{}"
5856    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.

```
5857    self.escaped_uri_chars = {
5858      ["{"] = "\\markdownRendererLeftBrace{}",
5859      ["}"] = "\\markdownRendererRightBrace{}",
5860      ["\\"] = "\\markdownRendererBackslash{}",
```

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

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

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

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

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

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minima` tables to create the `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.

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

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```
  function self.warning(t, m)
    return {"\\markdownRendererWarning{", self.escape(t), "}{",
            escape_minimal(t), "}{", self.escape(m or ""), "}{",
            escape_minimal(m or ""), "}"}
  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.

```
  function self.error(t, m)
    return {"\\markdownRendererError{", self.escape(t), "}{",
            escape_minimal(t), "}{", self.escape(m or ""), "}{",
            escape_minimal(m or ""), "}"}
  end
```

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

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

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

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

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

```
function self.image(lab, src, tit, attributes)
  if self.flatten_inlines then return lab end
  local buf = {}
```

```
5963      if attributes ~= nil then
5964        table.insert(buf,
5965                     "\\markdownRendererImageAttributeContextBegin\n")
5966        table.insert(buf, self.attributes(attributes))
5967      end
5968      table.insert(buf, {"\\markdownRendererImage{",lab,"}",
5969                         "{",self.string(src),"}",
5970                         "{",self.uri(src),"}",
5971                         "{",self.string(tit or ""),"}"})
5972      if attributes ~= nil then
5973        table.insert(buf,
5974                     "\\markdownRendererImageAttributeContextEnd{}")
5975      end
5976      return buf
5977    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.

```
5978    function self.bulletlist(items,tight)
5979      if not self.is_writing then return "" end
5980      local buffer = {}
5981      for _,item in ipairs(items) do
5982        if item ~= "" then
5983          buffer[#buffer + 1] = self.bulletitem(item)
5984        end
5985      end
5986      local contents = util.intersperse(buffer,"\n")
5987      if tight and options.tightLists then
5988        return {"\\markdownRendererUlBeginTight\n",contents,
5989          "\n\\markdownRendererUlEndTight "}
5990      else
5991        return {"\\markdownRendererUlBegin\n",contents,
5992          "\n\\markdownRendererUlEnd "}
5993      end
5994    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.

```
5995    function self.bulletitem(s)
5996      return {"\\markdownRendererUlItem ",s,
5997              "\\markdownRendererUlItemEnd "}
5998    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
      buffer[#buffer + 1] = self.ordereditem(item,num)
    end
    if num ~= nil and item ~= "" then
      num = num + 1
    end
  end
  local contents = util.intersperse(buffer,"\n")
  if tight and options.tightLists then
    return {"\\markdownRendererOlBeginTight\n",contents,
            "\n\\markdownRendererOlEndTight "}
  else
    return {"\\markdownRendererOlBegin\n",contents,
            "\n\\markdownRendererOlEnd "}
  end
end
```

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

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

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

```
function self.inline_html_comment(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
end
```

Define `writer->inline_html_tag` as a function that will transform the contents of an opening, closing, or empty inline HTML tag to the output format, where `contents` are the contents of the HTML tag.

```
function self.inline_html_tag(contents)
  if self.flatten_inlines then return contents end
```

```
6035      return {"\\markdownRendererInlineHtmlTag{",
6036              self.string(contents),"}"}
6037    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.

```
6038    function self.block_html_element(s)
6039      if not self.is_writing then return "" end
6040      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6041      return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
6042    end
```

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

```
6043    function self.emphasis(s)
6044      if self.flatten_inlines then return s end
6045      return {"\\markdownRendererEmphasis{",s,"}"}
6046    end
```

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

```
6047    function self.tickbox(f)
6048      if f == 1.0 then
6049        return "☒ "
6050      elseif f == 0.0 then
6051        return "☐ "
6052      else
6053        return "☐ "
6054      end
6055    end
```

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

```
6056    function self.strong(s)
6057      if self.flatten_inlines then return s end
6058      return {"\\markdownRendererStrongEmphasis{",s,"}"}
6059    end
```

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

```
6060    function self.blockquote(s)
6061      if not self.is_writing then return "" end
6062      return {"\\markdownRendererBlockQuoteBegin\n",s,
6063        "\\markdownRendererBlockQuoteEnd "}
6064    end
```

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

```
6065  function self.verbatim(s)
6066    if not self.is_writing then return "" end
6067    s = s:gsub("\n$", "")
6068    local name = util.cache_verbatim(options.cacheDir, s)
6069    return {"\\markdownRendererInputVerbatim{",name,"}"}
6070  end
```

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

```
6071  function self.document(d)
6072    local buf = {"\\markdownRendererDocumentBegin\n"}
6073
6074    -- warn against the `hybrid` option
6075    if options.hybrid then
6076      local text = "The `hybrid` option has been soft-deprecated."
6077      local more = "Consider using one of the following better options "
6078                .. "for mixing TeX and markdown: `contentBlocks`, "
6079                .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6080                .. "`texMathSingleBackslash`, and "
6081                .. "`texMathDoubleBackslash`. "
6082                .. "For more information, see the user manual at "
6083                .. "<https://witiko.github.io/markdown/>."
6084      table.insert(buf, self.warning(text, more))
6085    end
6086
6087    -- insert the text of the document
6088    table.insert(buf, d)
6089
6090    -- pop all attributes
6091    table.insert(buf, self.pop_attributes())
6092
6093    table.insert(buf, "\\markdownRendererDocumentEnd")
6094
6095    return buf
6096  end
```

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

```
6097  local seen_identifiers = {}
6098  local key_value_regex = "([^= ]+)%s*=%s*(.*)"
6099  local function normalize_attributes(attributes, auto_identifiers)
6100    -- normalize attributes
6101    local normalized_attributes = {}
6102    local has_explicit_identifiers = false
6103    local key, value
6104    for _, attribute in ipairs(attributes or {}) do
6105      if attribute:sub(1, 1) == "#" then
6106        table.insert(normalized_attributes, attribute)
```

```
      has_explicit_identifiers = true
      seen_identifiers[attribute:sub(2)] = true
    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
```

```
6154        end
6155      end
6156    end
6157
6158    -- sort and deduplicate normalized attributes
6159    table.sort(normalized_attributes)
6160    local seen_normalized_attributes = {}
6161    local deduplicated_normalized_attributes = {}
6162    for _, attribute in ipairs(normalized_attributes) do
6163      if seen_normalized_attributes[attribute] == nil then
6164        seen_normalized_attributes[attribute] = true
6165        table.insert(deduplicated_normalized_attributes, attribute)
6166      end
6167    end
6168
6169    return deduplicated_normalized_attributes
6170  end
6171
6172  function self.attributes(attributes, should_normalize_attributes)
6173    local normalized_attributes
6174    if should_normalize_attributes == false then
6175      normalized_attributes = attributes
6176    else
6177      normalized_attributes = normalize_attributes(attributes)
6178    end
6179
6180    local buf = {}
6181    local key, value
6182    for _, attribute in ipairs(normalized_attributes) do
6183      if attribute:sub(1, 1) == "#" then
6184        table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
6185                           attribute:sub(2), "}"})
6186      elseif attribute:sub(1, 1) == "." then
6187        table.insert(buf, {"\\markdownRendererAttributeClassName{",
6188                           attribute:sub(2), "}"})
6189      else
6190        key, value = attribute:match(key_value_regex)
6191        table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
6192                           key, "}{", value, "}"})
6193      end
6194    end
6195
6196    return buf
6197  end
```

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

```
6198  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.

```
6199  self.attribute_type_levels = {}
6200  setmetatable(self.attribute_type_levels,
6201               { __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.

```
6202  local function apply_attributes()
6203    local buf = {}
6204    for i = 1, #self.active_attributes do
6205      local start_output = self.active_attributes[i][3]
6206      if start_output ~= nil then
6207        table.insert(buf, start_output)
6208      end
6209    end
6210    return buf
6211  end
6212
6213  local function tear_down_attributes()
6214    local buf = {}
6215    for i = #self.active_attributes, 1, -1 do
6216      local end_output = self.active_attributes[i][4]
6217      if end_output ~= nil then
6218        table.insert(buf, end_output)
6219      end
6220    end
6221    return buf
6222  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.

```
6223  function self.push_attributes(attribute_type, attributes,
6224                                start_output, end_output)
6225    local attribute_type_level
6226      = self.attribute_type_levels[attribute_type]
6227    self.attribute_type_levels[attribute_type]
6228      = attribute_type_level + 1
6229
6230    -- index attributes in a hash table for easy lookup
6231    attributes = attributes or {}
```

```
6232    for i = 1, #attributes do
6233      attributes[attributes[i]] = true
6234    end
6235
6236    local buf = {}
6237    -- handle slicing
6238    if attributes["#" .. self.slice_end_identifier] ~= nil and
6239       self.slice_end_type == "^" then
6240      if self.is_writing then
6241        table.insert(buf, self.undosep())
6242        table.insert(buf, tear_down_attributes())
6243      end
6244      self.is_writing = false
6245    end
6246    if attributes["#" .. self.slice_begin_identifier] ~= nil and
6247       self.slice_begin_type == "^" then
6248      table.insert(buf, apply_attributes())
6249      self.is_writing = true
6250    end
6251    if self.is_writing and start_output ~= nil then
6252      table.insert(buf, start_output)
6253    end
6254    table.insert(self.active_attributes,
6255                 {attribute_type, attributes,
6256                  start_output, end_output})
6257    return buf
6258  end
6259
```

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).

```
6260  function self.pop_attributes(attribute_type)
6261    local buf = {}
6262    -- pop attributes until we find attributes of correct type
6263    -- or until no attributes remain
6264    local current_attribute_type = false
6265    while current_attribute_type ~= attribute_type and
6266          #self.active_attributes > 0 do
6267      local attributes, _, end_output
6268      current_attribute_type, attributes, _, end_output = table.unpack(
6269        self.active_attributes[#self.active_attributes])
6270      local attribute_type_level
6271        = self.attribute_type_levels[current_attribute_type]
```

```
      self.attribute_type_levels[current_attribute_type]
        = attribute_type_level - 1
      if self.is_writing and end_output ~= nil then
        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
```

```
6362        local char = utf8.char(code)
6363
6364        -- Remove everything up to the first non-space.
6365        if not letter_found then
6366          local is_letter = unicode.utf8.match(char, "%S")
6367          if is_letter then
6368            letter_found = true
6369          else
6370            goto continue
6371          end
6372        end
6373
6374        -- Remove all non-alphanumeric characters, except underscores
6375        -- and hyphens.
6376        if not unicode.utf8.match(char, "[%w_%-%s]") then
6377          prev_space = false
6378          goto continue
6379        end
6380
6381        -- Replace all spaces and newlines with hyphens.
6382        if unicode.utf8.match(char, "[%s\n]") then
6383          char = "-"
6384          if prev_space then
6385            goto continue
6386          else
6387            prev_space = true
6388          end
6389        else
6390          -- Convert all alphabetic characters to lowercase.
6391          char = unicode.utf8.lower(char)
6392          prev_space = false
6393        end
6394
6395        table.insert(buffer, char)
6396
6397        ::continue::
6398      end
6399
6400      if prev_space then
6401        table.remove(buffer)
6402      end
6403
6404      local identifier = #buffer == 0 and "section"
6405                      or table.concat(buffer, "")
6406      return identifier
6407    end
```

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

```
6408  self.secbegin_text = "\\markdownRendererSectionBegin\n"
6409  self.secend_text = "\n\\markdownRendererSectionEnd "
6410  function self.heading(s, level, attributes)
6411    local buf = {}
6412    local flat_text, inlines = table.unpack(s)
6413
6414    -- push empty attributes for implied sections
6415    while self.attribute_type_levels["heading"] < level - 1 do
6416      table.insert(buf,
6417                   self.push_attributes("heading",
6418                                        nil,
6419                                        self.secbegin_text,
6420                                        self.secend_text))
6421    end
6422
6423    -- pop attributes for sections that have ended
6424    while self.attribute_type_levels["heading"] >= level do
6425      table.insert(buf, self.pop_attributes("heading"))
6426    end
6427
6428    -- construct attributes for the new section
6429    local auto_identifiers = {}
6430    if self.options.autoIdentifiers then
6431      table.insert(auto_identifiers, create_auto_identifier(flat_text))
6432    end
6433    if self.options.gfmAutoIdentifiers then
6434      table.insert(auto_identifiers,
6435                   create_gfm_auto_identifier(flat_text))
6436    end
6437    local normalized_attributes = normalize_attributes(attributes,
6438                                                       auto_identifiers)
6439
6440    -- push attributes for the new section
6441    local start_output = {}
6442    local end_output = {}
6443    table.insert(start_output, self.secbegin_text)
6444    table.insert(end_output, self.secend_text)
6445
6446    table.insert(buf, self.push_attributes("heading",
6447                                           normalized_attributes,
6448                                           start_output,
6449                                           end_output))
6450    assert(self.attribute_type_levels["heading"] == level)
6451
6452    -- render the heading and its attributes
```

```
6453    if self.is_writing and #normalized_attributes > 0 then
6454      table.insert(buf,
6455                   "\\markdownRendererHeaderAttributeContextBegin\n")
6456      table.insert(buf, self.attributes(normalized_attributes, false))
6457    end
6458
6459    local cmd
6460    level = level + options.shiftHeadings
6461    if level <= 1 then
6462      cmd = "\\markdownRendererHeadingOne"
6463    elseif level == 2 then
6464      cmd = "\\markdownRendererHeadingTwo"
6465    elseif level == 3 then
6466      cmd = "\\markdownRendererHeadingThree"
6467    elseif level == 4 then
6468      cmd = "\\markdownRendererHeadingFour"
6469    elseif level == 5 then
6470      cmd = "\\markdownRendererHeadingFive"
6471    elseif level >= 6 then
6472      cmd = "\\markdownRendererHeadingSix"
6473    else
6474      cmd = ""
6475    end
6476    if self.is_writing then
6477      table.insert(buf, {cmd, "{", inlines, "}"})
6478    end
6479
6480    if self.is_writing and #normalized_attributes > 0 then
6481      table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{}")
6482    end
6483
6484    return buf
6485  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.

```
6486  function self.get_state()
6487    return {
6488      is_writing=self.is_writing,
6489      flatten_inlines=self.flatten_inlines,
6490      active_attributes={table.unpack(self.active_attributes)},
6491    }
6492  end
```

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

```
6493  function self.set_state(s)
6494    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
```

```
6772        count = count + tab_value
6773        tab_value = 4
6774      elseif character == " " then
6775        count = count + 1
6776        tab_value = 4 - (1 - tab_value) % 4
6777      end
6778
6779      if (left_strip_length ~= 0) then
6780        local possible_to_strip = math.min(count, left_strip_length)
6781        count = count - possible_to_strip
6782        left_strip_length = left_strip_length - possible_to_strip
6783        left_total_stripped = left_total_stripped + possible_to_strip
6784      else
6785        full_remainder =  full_remainder .. character
6786      end
6787
6788      if (minimum_found) then
6789        minimum_remainder = minimum_remainder .. character
6790      elseif (count >= minimum) then
6791        minimum_found = true
6792        minimum_remainder = minimum_remainder
6793                            .. string.rep(" ", count - minimum)
6794      end
6795
6796      if (code_started) then
6797        code_start = code_start .. character
6798      elseif (count >= minimum + 4) then
6799        code_started = true
6800        code_start = code_start
6801                    .. string.rep(" ", count - (minimum + 4))
6802      end
6803    end
6804  end
6805
6806  local remainder
6807  if (code_started) then
6808    remainder = code_start
6809  else
6810    remainder = string.rep(" ", count - minimum)
6811  end
6812
6813  local is_minimum = count >= minimum
6814  return {
6815    is_code = code_started,
6816    remainder = remainder,
6817    left_total_stripped = left_total_stripped,
6818    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)
```

```
6943
6944  if current_line_indents == nil then
6945    current_line_indents = {}
6946  end
6947
6948  for index = 1,#indent_table.indents do
6949    local value = indent_table.indents[index]
6950    local pattern = decode_pattern(value.name)
6951
6952    -- match decoded pattern
6953    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6954    if new_indent_info == nil then
6955      local blankline_end = lpeg.match(
6956        Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
6957      if is_optional or not indent_table.ignore_blockquote_blank
6958         or not blankline_end then
6959        return is_optional, new_index, current_trail,
6960               current_line_indents
6961      end
6962
6963      return traverse_indent(s, tonumber(blankline_end.pos),
6964                             indent_table, is_optional, is_blank,
6965                             current_line_indents)
6966    end
6967
6968    local raw_last_trail = new_indent_info[1]
6969    local delimiter = new_indent_info[2]
6970    local raw_new_trail = new_indent_info[3]
6971    local next_index = new_indent_info[4]
6972
6973    local space_only = delimiter == ""
6974
6975    -- check previous trail
6976    if not space_only and next(current_trail) == nil then
6977      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6978      current_trail = {is_code=sp.is_code, remainder=sp.remainder,
6979                       total_length=sp.total_length,
6980                       full_remainder=sp.full_remainder}
6981    end
6982
6983    if next(current_trail) ~= nil then
6984      if not space_only and current_trail.is_code then
6985        return is_optional, new_index, current_trail,
6986               current_line_indents
6987      end
6988      if current_trail.internal_remainder ~= nil then
6989        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
```

```
7079      return result, remainder
7080    end
7081
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
7082  local check_trail_length =
7083    function(s, i, indent_table, -- luacheck: ignore s i
7084             spacing, min, max)
7085      local trail
7086
7087      if has_trail(indent_table) then
7088        trail = indent_table.trail
7089      else
7090        trail = process_starter_spacing(0, spacing, 0, 0)
7091      end
7092
7093      local total_length = trail.total_length
7094      if total_length == nil then
7095        return false
7096      end
7097
7098      return min <= total_length and total_length <= max
7099    end
7100
```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```
7101  local function check_continuation_indentation(s, i, indent_table,
7102                                                 is_optional, is_blank)
7103    if not has_indents(indent_table) then
7104      return true
7105    end
7106
7107    local passes, new_index, current_trail, current_line_indents =
7108      traverse_indent(s, i, indent_table, is_optional, is_blank)
7109
7110    if passes then
7111      indent_table.current_line_indents = current_line_indents
7112      indent_table = add_trail(indent_table, current_trail)
7113      return new_index, indent_table
7114    end
7115    return false
7116  end
7117
```

Get name of the last indent from the `indent_table`.

```
7118  local function get_last_indent_name(indent_table)
7119    if has_indents(indent_table) then
```

```
7120      return indent_table.indents[#indent_table.indents].name
7121    end
7122  end
7123
```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
7124  local function remove_remainder_if_blank(indent_table, remainder)
7125    if get_last_indent_name(indent_table) == "li" then
7126      return ""
7127    end
7128    return remainder
7129  end
7130
```

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.

```
7131  local check_trail_type =
7132    function(s, i, -- luacheck: ignore s i
7133             trail, spacing, trail_type)
7134      if trail == nil then
7135        trail = process_starter_spacing(0, spacing, 0, 0)
7136      end
7137
7138      if trail_type == "non-code" then
7139        return check_trail(false, trail.is_code)
7140      end
7141      if trail_type == "code" then
7142        return check_trail(true, trail.is_code)
7143      end
7144      if trail_type == "full-code" then
7145        if (trail.is_code) then
7146          return i, trail.remainder
7147        end
7148        return i, ""
7149      end
7150      if trail_type == "full-any" then
7151        return i, trail.internal_remainder
7152      end
7153    end
7154
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
7155  local trail_freezing =
7156    function(s, i, -- luacheck: ignore s i
7157             indent_table, is_freezing)
7158      if is_freezing then
7159        if indent_table.is_trail_frozen then
```

```
        indent_table.trail = indent_table.frozen_trail
      else
        indent_table.frozen_trail = indent_table.trail
        indent_table.is_trail_frozen = true
      end
    else
      indent_table.frozen_trail = nil
      indent_table.is_trail_frozen = false
    end
    return true, indent_table
  end

```

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`.

```
local check_continuation_indentation_and_trail =
  function (s, i, indent_table, is_optional, is_blank, trail_type,
            reset_rem, omit_remainder)
    if not has_indents(indent_table) then
      local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
                                           * Cp(), s, i)
      local result, remainder = check_trail_type(s, i,
        indent_table.trail, spacing, trail_type)
      if remainder == nil then
        if result then
          return new_index
        end
        return false
      end
      if result then
        return new_index, remainder
      end
      return false
    end

    local passes, new_index, current_trail = traverse_indent(s, i,
      indent_table, is_optional, is_blank)

    if passes then
      local spacing
      if current_trail == nil then
        local newer_spacing, newer_index = lpeg.match(
          C(parsers.spacechar^0) * Cp(), s, i)
        current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
        new_index = newer_index
        spacing = newer_spacing
```

```
      else
        spacing = current_trail.remainder
      end
      local result, remainder = check_trail_type(s, new_index,
        current_trail, spacing, trail_type)
      if remainder == nil or omit_remainder then
        if result then
          return new_index
        end
        return false
      end

      if is_blank and reset_rem then
        remainder = remove_remainder_if_blank(indent_table, remainder)
      end
      if result then
        return new_index, remainder
      end
      return false
    end
    return false
  end

```

The following patterns check whitespace indentation at the start of a block.

```
parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
                         * Cc(false), check_trail_joined)

parsers.check_trail_no_rem = Cmt( Cb("indent_info")
                                * C(parsers.spacechar^0) * Cc(false)
                                * Cc(true), check_trail_joined)

parsers.check_code_trail  = Cmt( Cb("indent_info")
                               * C(parsers.spacechar^0)
                               * Cc(true), check_trail_joined)

parsers.check_trail_length_range  = function(min, max)
  return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
            * Cc(max), check_trail_length)
end

parsers.check_trail_length = function(n)
  return parsers.check_trail_length_range(n, n)
end

```

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)
```

```
7288
7289 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7290   Cmt( Cb("indent_info")
7291      * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7292      , check_continuation_indentation_and_trail)
7293
7294 parsers.check_optional_indent_and_any_trail =
7295   Cmt( Cb("indent_info")
7296      * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7297      , check_continuation_indentation_and_trail)
7298
7299 parsers.check_optional_blank_indent_and_any_trail =
7300   Cmt( Cb("indent_info")
7301      * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7302      , check_continuation_indentation_and_trail)
7303
```

The following patterns specify behaviour around newlines.

```
7304
7305 parsers.spnlc_noexc = parsers.optionalspace
7306                     * ( parsers.newline
7307                       * parsers.check_minimal_indent_and_any_trail)^-1
7308
7309 parsers.spnlc = parsers.optionalspace
7310               * (V("EndlineNoSub"))^-1
7311
7312 parsers.spnlc_sep  = parsers.optionalspace * V("EndlineNoSub")
7313                    + parsers.spacechar^1
7314
7315 parsers.only_blank = parsers.spacechar^0
7316                    * (parsers.newline + parsers.eof)
7317
7318 %   \end{macrocode}
7319 % \begin{figure}
7320 % \hspace*{-0.1\textwidth}
7321 % \begin{minipage}{1.2\textwidth}
7322 % \centering
7323 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
7324 % \node[state, initial by diamond, accepting] (noop) {initial};
7325 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
7326 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
7327 % \node[state] (comment) [below=of noop] {comment};
7328 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs
7329 % \node[state] (blank_line) [below right=of comment] {blank line};
7330 % \path[->]
7331 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^\
7332 %        edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
7333 %        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)
                         * Cg((parsers.backslash  -- even backslash
                              * parsers.backslash)^1, "backslashes")
                         + (parsers.backslash
                           * (#parsers.percent
                             * Cb("backslashes")
                             / function(backslashes)
                               return string.rep("\\", #backslashes / 2)
                             end
                             * C(parsers.percent)
```

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

```
7471
7472 parsers.html_entities
7473   = parsers.hexentity / entities.hex_entity_with_x_char
7474   + parsers.decentity / entities.dec_entity
7475   + parsers.tagentity / entities.char_entity
```

#### 3.1.5.3 Parsers Used for Markdown Lists

```
7476 parsers.bullet = function(bullet_char, interrupting)
7477   local allowed_end
7478   if interrupting then
7479     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7480   else
7481     allowed_end = C(parsers.spacechar^1)
7482                 + #(parsers.newline + parsers.eof)
7483   end
7484   return parsers.check_trail
7485         * Ct(C(bullet_char) * Cc(""))
7486         * allowed_end
7487 end
7488
7489 local function tickbox(interior)
7490   return parsers.optionalspace * parsers.lbracket
7491        * interior * parsers.rbracket * parsers.spacechar^1
7492 end
7493
7494 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7495 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7496 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7497
```

#### 3.1.5.4 Parsers Used for Markdown Code Spans

```
7498 parsers.openticks   = Cg(parsers.backtick^1, "ticks")
7499
7500 local function captures_equal_length(_,i,a,b)
7501   return #a == #b and i
7502 end
7503
7504 parsers.closeticks  = Cmt(C(parsers.backtick^1)
7505                           * Cb("ticks"), captures_equal_length)
7506
7507 parsers.intickschar = (parsers.any - S("\n\r`"))
7508                     + V("NoSoftLineBreakEndline")
7509                     + (parsers.backtick^1 - parsers.closeticks)
7510
7511 local function process_inticks(s)
7512   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
```

```
7698
7699 parsers.html_attribute
7700   = parsers.html_attribute_spacing
7701   * parsers.html_attribute_name
7702   * parsers.html_inline_attribute_value_specification^-1
7703
7704 parsers.html_non_newline_attribute
7705   = parsers.spacechar^1
7706   * parsers.html_attribute_name
7707   * parsers.html_attribute_value_specification^-1
7708
7709 parsers.nested_breaking_blank = parsers.newline
7710                               * parsers.check_minimal_blank_indent
7711                               * parsers.blankline
7712
7713 parsers.html_comment_start = P("<!--")
7714
7715 parsers.html_comment_end = P("-->")
7716
7717 parsers.html_comment
7718   = Cs( parsers.html_comment_start
7719       * parsers.html_until_end(parsers.html_comment_end))
7720
7721 parsers.html_inline_comment = (parsers.html_comment_start / "")
7722                             * -P(">") * -P("->")
7723                             * Cs(( V("NoSoftLineBreakEndline")
7724                                  + parsers.any
7725                                  - parsers.nested_breaking_blank
7726                                  - parsers.html_comment_end)^0)
7727                             * (parsers.html_comment_end / "")
7728
7729 parsers.html_cdatasection_start = P("<![CDATA[")
7730
7731 parsers.html_cdatasection_end = P("]]>")
7732
7733 parsers.html_cdatasection
7734   = Cs( parsers.html_cdatasection_start
7735       * parsers.html_until_end(parsers.html_cdatasection_end))
7736
7737 parsers.html_inline_cdatasection
7738   = parsers.html_cdatasection_start
7739   * Cs(V("NoSoftLineBreakEndline") + parsers.any
7740       - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
7741   * parsers.html_cdatasection_end
7742
7743 parsers.html_declaration_start = P("<!") * parsers.letter
7744
```

```
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

-- closing tags
```

```
parsers.html_close_tag  = parsers.html_tag_closing_start
                        * parsers.block_keyword
                        * parsers.html_tag_end

parsers.html_any_close_tag  = parsers.html_tag_closing_start
                            * parsers.keyword
                            * parsers.html_tag_end

parsers.html_close_special_tag = parsers.html_tag_closing_start
                               * parsers.special_block_keyword
                               * parsers.html_tag_end

-- empty tags
parsers.html_any_empty_inline_tag = parsers.html_tag_start
                                  * parsers.keyword
                                  * parsers.html_attribute^0
                                  * parsers.html_empty_tag_end

parsers.html_any_empty_tag  = parsers.html_tag_start
                            * parsers.keyword
                            * parsers.html_non_newline_attribute^0
                            * parsers.optionalspace
                            * parsers.slash
                            * parsers.more

parsers.html_empty_tag  = parsers.html_tag_start
                        * parsers.block_keyword
                        * parsers.html_attribute^0
                        * parsers.html_empty_tag_end

parsers.html_empty_special_tag  = parsers.html_tag_start
                                * parsers.special_block_keyword
                                * parsers.html_attribute^0
                                * parsers.html_empty_tag_end

parsers.html_incomplete_blocks
  = parsers.html_incomplete_open_tag
  + parsers.html_incomplete_open_special_tag
  + parsers.html_incomplete_close_tag

-- parse special html blocks
parsers.html_blankline_ending_special_block_opening
  = ( parsers.html_close_special_tag
    + parsers.html_empty_special_tag)
  * #( parsers.optionalspace
     * (parsers.newline + parsers.eof))

```

```
parsers.html_blankline_ending_special_block
  = parsers.html_blankline_ending_special_block_opening
  * parsers.html_blankline_end_condition

parsers.html_special_block_opening
  = parsers.html_incomplete_open_special_tag
  - parsers.html_empty_special_tag

parsers.html_closing_special_block
  = parsers.html_special_block_opening
  * parsers.html_until_end(parsers.html_close_special_tag)

parsers.html_special_block
  = parsers.html_blankline_ending_special_block
  + parsers.html_closing_special_block

-- parse html blocks
parsers.html_block_opening  = parsers.html_incomplete_open_tag
                            + parsers.html_incomplete_close_tag

parsers.html_block  = parsers.html_block_opening
                    * parsers.html_blankline_end_condition

-- parse any html blocks
parsers.html_any_block_opening
  = ( parsers.html_any_open_tag
    + parsers.html_any_close_tag
    + parsers.html_any_empty_tag)
  * #(parsers.optionalspace * (parsers.newline + parsers.eof))

parsers.html_any_block  = parsers.html_any_block_opening
                        * parsers.html_blankline_end_condition

parsers.html_inline_comment_full  = parsers.html_comment_start
                                  * -P(">") * -P("->")
                                  * Cs(( V("NoSoftLineBreakEndline")
                                       + parsers.any - P("--")
                                       - parsers.nested_breaking_blank
                                       - parsers.html_comment_end)^0)
                                  * parsers.html_comment_end

parsers.html_inline_tags  = parsers.html_inline_comment_full
                          + parsers.html_any_empty_inline_tag
                          + parsers.html_inline_instruction
                          + 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.

```
8179  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.

```
8180  function self.normalize_tag(tag)
8181    tag = util.rope_to_string(tag)
8182    tag = tag:gsub("[ \n\r\t]+", " ")
8183    tag = tag:gsub("^ ", ""):gsub(" $", "")
8184    tag = uni_algos.case.casefold(tag, true, false)
8185    return tag
8186  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.

```
8187  local function iterlines(s, f)
8188    local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8189    return util.rope_to_string(rope)
8190  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.

```
8191  if options.preserveTabs then
8192    self.expandtabs = function(s) return s end
8193  else
8194    self.expandtabs = function(s)
8195                        if s:find("\t") then
8196                          return iterlines(s, util.expand_tabs_in_line)
8197                        else
8198                          return s
8199                        end
8200                      end
8201  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.

```
8202  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
```

```
8241
8242  self.create_parser("parse_blocks",
8243                     function()
8244                       return parsers.blocks
8245                     end, true)
8246
8247  self.create_parser("parse_blocks_nested",
8248                     function()
8249                       return parsers.blocks_nested
8250                     end, false)
8251
8252  self.create_parser("parse_inlines",
8253                     function()
8254                       return parsers.inlines
8255                     end, false)
8256
8257  self.create_parser("parse_inlines_no_inline_note",
8258                     function()
8259                       return parsers.inlines_no_inline_note
8260                     end, false)
8261
8262  self.create_parser("parse_inlines_no_html",
8263                     function()
8264                       return parsers.inlines_no_html
8265                     end, false)
8266
8267  self.create_parser("parse_inlines_nbsp",
8268                     function()
8269                       return parsers.inlines_nbsp
8270                     end, false)
8271  self.create_parser("parse_inlines_no_link_or_emphasis",
8272                    function()
8273                      return parsers.inlines_no_link_or_emphasis
8274                    end, false)
```

#### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
8275  parsers.minimally_indented_blankline
8276    = parsers.check_minimal_indent * (parsers.blankline / "")
8277
8278  parsers.minimally_indented_block
8279    = parsers.check_minimal_indent * V("Block")
8280
8281  parsers.minimally_indented_block_or_paragraph
8282    = parsers.check_minimal_indent * V("BlockOrParagraph")
8283
```

```
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
```

```
8419
8420    return parsers.check_trail
8421             * Ct(C(delimiter_range) * C(delimiter_type))
8422             * allowed_end
8423  end
8424
8425  parsers.starter = parsers.bullet(parsers.dash)
8426                  + parsers.bullet(parsers.asterisk)
8427                  + parsers.bullet(parsers.plus)
8428                  + parsers.enumerator(parsers.period)
8429                  + parsers.enumerator(parsers.rparent)
8430
```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```
8431  parsers.blockquote_start
8432    = parsers.check_trail
8433    * C(parsers.more)
8434    * C(parsers.spacechar^0)
8435
8436  parsers.blockquote_body
8437    = parsers.add_indent(parsers.blockquote_start, "bq", true)
8438    * parsers.indented_content()
8439    * remove_indent("bq")
8440
8441  if not options.breakableBlockquotes then
8442    parsers.blockquote_body
8443      = parsers.add_indent(parsers.blockquote_start, "bq", false)
8444      * parsers.indented_content()
8445      * remove_indent("bq")
8446  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.

```
8447  local function parse_content_part(content_part)
8448    local rope = util.rope_to_string(content_part)
8449    local parsed
8450      = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8451    parsed.indent_info = nil
8452    return parsed
8453  end
8454
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8455  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.

```
8498  local function fill_strong(t, opening_index, closing_index)
8499    local content
8500      = collect_emphasis_content(t, opening_index + 1,
8501                                 closing_index - 1)
8502    t[opening_index + 1].is_active = true
8503    t[opening_index + 1].rendered = writer.strong(content)
8504  end
8505
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
8506  local function breaks_three_rule(opening_delimiter, closing_delimiter)
8507    return ( opening_delimiter.is_closing
8508          or closing_delimiter.is_opening)
8509       and (( opening_delimiter.original_count
8510            + closing_delimiter.original_count) % 3 == 0)
8511       and ( opening_delimiter.original_count % 3 ~= 0
8512          or closing_delimiter.original_count % 3 ~= 0)
8513  end
8514
```

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`.

```
8515  local find_emphasis_opener = function(t, bottom_index, latest_index,
8516                                        character, closing_delimiter)
8517    for i = latest_index, bottom_index, -1 do
8518      local value = t[i]
8519      if value.is_active and
8520         value.is_opening and
8521         value.type == "delimiter" and
8522         value.element == "emphasis" and
8523         (value.character == character) and
8524         (value.current_count > 0) then
8525        if not breaks_three_rule(value, closing_delimiter) then
8526          return i
8527        end
8528      end
8529    end
8530  end
8531
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```
8532  local function process_emphasis(t, opening_index, closing_index)
8533    for i = opening_index, closing_index do
8534      local value = t[i]
8535      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)
```

```
8626    local c
8627    local char_length
8628    for pos = start_pos, end_pos, 1 do
8629      if (start_pos < 0) then
8630        char_length = -pos
8631      else
8632        char_length = pos + 1
8633      end
8634
8635      if (chartype == "punctuation") then
8636        if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8637          return i
8638        end
8639      else
8640        c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
8641        if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8642          return i
8643        end
8644      end
8645    end
8646  end
8647
8648  local function check_preceding_unicode_punctuation(s, i)
8649    return check_unicode_type(s, i, -4, -1, "punctuation")
8650  end
8651
8652  local function check_preceding_unicode_whitespace(s, i)
8653    return check_unicode_type(s, i, -4, -1, "%s")
8654  end
8655
8656  local function check_following_unicode_punctuation(s, i)
8657    return check_unicode_type(s, i, 0, 3, "punctuation")
8658  end
8659
8660  local function check_following_unicode_whitespace(s, i)
8661    return check_unicode_type(s, i, 0, 3, "%s")
8662  end
8663
8664  parsers.unicode_preceding_punctuation
8665    = B(parsers.escapable)
8666    + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8667
8668  parsers.unicode_preceding_whitespace
8669    = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8670
8671  parsers.unicode_following_punctuation
8672    = #parsers.escapable
```

```
8673    + Cmt(parsers.succeed, check_following_unicode_punctuation)
8674
8675  parsers.unicode_following_whitespace
8676    = Cmt(parsers.succeed, check_following_unicode_whitespace)
8677
8678  parsers.delimiter_run = function(character)
8679    return  (B(parsers.backslash * character) + -B(character))
8680          * character^1
8681          * -#character
8682  end
8683
8684  parsers.left_flanking_delimiter_run = function(character)
8685    return  (B( parsers.any)
8686              * ( parsers.unicode_preceding_punctuation
8687                + parsers.unicode_preceding_whitespace)
8688             + -B(parsers.any))
8689            * parsers.delimiter_run(character)
8690            * parsers.unicode_following_punctuation
8691            + parsers.delimiter_run(character)
8692            * -#( parsers.unicode_following_punctuation
8693                + parsers.unicode_following_whitespace
8694                + parsers.eof)
8695  end
8696
8697  parsers.right_flanking_delimiter_run = function(character)
8698    return  parsers.unicode_preceding_punctuation
8699          * parsers.delimiter_run(character)
8700          * ( parsers.unicode_following_punctuation
8701            + parsers.unicode_following_whitespace
8702            + parsers.eof)
8703          + (B(parsers.any)
8704            * -( parsers.unicode_preceding_punctuation
8705               + parsers.unicode_preceding_whitespace))
8706          * parsers.delimiter_run(character)
8707  end
8708
8709  if options.underscores then
8710    parsers.emph_start
8711      = parsers.left_flanking_delimiter_run(parsers.asterisk)
8712      + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
8713        + ( parsers.unicode_preceding_punctuation
8714          * #parsers.right_flanking_delimiter_run(parsers.underscore)))
8715      * parsers.left_flanking_delimiter_run(parsers.underscore)
8716
8717    parsers.emph_end
8718      = parsers.right_flanking_delimiter_run(parsers.asterisk)
8719      + ( -#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

```

```
8764  -- List of note references defined in the document
8765  parsers.rawnotes = {}
8766
```

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.

```
8767  function self.register_link(_, tag, url, title,
8768                              attributes)
8769    local normalized_tag = self.normalize_tag(tag)
8770      if references[normalized_tag] == nil then
8771        references[normalized_tag] = {
8772          url = url,
8773          title = title,
8774          attributes = attributes
8775        }
8776      end
8777    return ""
8778  end
8779
```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
8780  function self.lookup_reference(tag)
8781    return references[self.normalize_tag(tag)]
8782  end
8783
```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
8784  function self.lookup_note_reference(tag)
8785    return parsers.rawnotes[self.normalize_tag(tag)]
8786  end
8787
8788  parsers.title_s_direct_ref  = parsers.squote
8789                              * Cs((parsers.html_entities
8790                                  + ( parsers.anyescaped
8791                                    - parsers.squote
8792                                    - parsers.blankline^2))^0)
8793                              * parsers.squote
8794
8795  parsers.title_d_direct_ref  = parsers.dquote
8796                              * Cs((parsers.html_entities
8797                                  + ( parsers.anyescaped
8798                                    - parsers.dquote
8799                                    - parsers.blankline^2))^0)
8800                              * parsers.dquote
8801
8802  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

```

```
8897  if options.inlineNotes then
8898    opening_elements = opening_elements + inline_note_element
8899  end
8900
8901  opening_elements = opening_elements + image_element
8902
8903  if options.notes then
8904    opening_elements = opening_elements + note_element
8905  end
8906
8907  opening_elements = opening_elements + link_element
8908
8909  parsers.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
8910                                  * Cg(Cc(true), "is_opening")
8911                                  * Cg(Cc(false), "is_closing")
8912                                  * opening_elements)
8913
8914  parsers.link_image_closing  = Ct( Cg(Cc("delimiter"), "type")
8915                                  * Cg(Cc("link"), "element")
8916                                  * Cg(Cc(false), "is_opening")
8917                                  * Cg(Cc(true), "is_closing")
8918                                  * ( Cg(Cc(true), "is_direct")
8919                                    * Cg( parsers.rbracket
8920                                        * #parsers.inline_direct_ref,
8921                                         "content")
8922                                    + Cg(Cc(false), "is_direct")
8923                                    * Cg(parsers.rbracket, "content")))
8924
8925  parsers.link_image_open_or_close  = parsers.link_image_opening
8926                                    + parsers.link_image_closing
8927
8928  if options.html then
8929    parsers.link_emph_precedence  = parsers.inticks
8930                                  + parsers.autolink
8931                                  + parsers.html_inline_tags
8932  else
8933    parsers.link_emph_precedence  = parsers.inticks
8934                                  + parsers.autolink
8935  end
8936
8937  parsers.link_and_emph_endline = parsers.newline
8938                                * ((parsers.check_minimal_indent
8939                                  * -V("EndlineExceptions")
8940                                  + parsers.check_optional_indent
8941                                  * -V("EndlineExceptions")
8942                                  * -parsers.starter) / "")
8943                                * parsers.spacechar^0 / "\n"
```

```
8944
8945  parsers.link_and_emph_content
8946    = Ct( Cg(Cc("content"), "type")
8947        * Cg(Cs(( parsers.link_emph_precedence
8948                + parsers.backslash * parsers.linechar
8949                + parsers.link_and_emph_endline
8950                + (parsers.linechar
8951                  - parsers.blankline^2
8952                  - parsers.link_image_open_or_close
8953                  - parsers.emph_open_or_close))^0), "content"))
8954
8955  parsers.link_and_emph_table
8956    = (parsers.link_image_opening + parsers.emph_open)
8957    * parsers.link_and_emph_content
8958    * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
8959      * parsers.link_and_emph_content)^1
8960
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8961  local function collect_link_content(t, opening_index, closing_index)
8962    local content = {}
8963    for i = opening_index, closing_index do
8964      content[#content + 1] = t[i].content
8965    end
8966    return util.rope_to_string(content)
8967  end
8968
```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
8969  local function find_link_opener(t, bottom_index, latest_index)
8970    for i = latest_index, bottom_index, -1 do
8971      local value = t[i]
8972      if value.type == "delimiter" and
8973         value.is_opening and
8974         ( value.element == "link"
8975        or value.element == "image"
8976        or value.element == "note")
8977         and not value.removed then
8978        if value.is_active then
8979          return i
8980        end
8981        value.removed = true
8982        return nil
8983      end
8984    end
8985  end
```

```
8986
```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
8987  local function find_next_link_closing_index(t, latest_index)
8988    for i = latest_index, #t do
8989      local value = t[i]
8990      if value.is_closing and
8991         value.element == "link" and
8992         not value.removed then
8993        return i
8994      end
8995    end
8996  end
8997
```

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.

```
8998  local function disable_previous_link_openers(t, opening_index)
8999    if t[opening_index].element == "image" then
9000      return
9001    end
9002
9003    for i = opening_index, 1, -1 do
9004      local value = t[i]
9005      if value.is_active and
9006         value.type == "delimiter" and
9007         value.is_opening and
9008         value.element == "link" then
9009        value.is_active = false
9010      end
9011    end
9012  end
9013
```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
9014  local function disable_range(t, opening_index, closing_index)
9015    for i = opening_index, closing_index do
9016      local value = t[i]
9017      if value.is_active then
9018        value.is_active = false
9019        if value.type == "delimiter" then
9020          value.removed = true
9021        end
9022      end
9023    end
9024  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 = {}
```

```
9061        if (t[opening_index].element == "link") then
9062          rendered = writer.link(mapped, reference.url,
9063                                 reference.title, reference.attributes)
9064        end
9065
9066        if (t[opening_index].element == "image") then
9067          rendered = writer.image(mapped, reference.url, reference.title,
9068                                  reference.attributes)
9069        end
9070
9071        if (t[opening_index].element == "note") then
9072          if (t[opening_index].link_type == "note_inline") then
9073            rendered = writer.note(mapped)
9074          end
9075          if (t[opening_index].link_type == "raw_note") then
9076            rendered = writer.note(reference)
9077          end
9078        end
9079
9080        t[opening_index].rendered = rendered
9081        delete_parsed_content_in_range(t, opening_index + 1,
9082                                       closing_index)
9083        empty_content_in_range(t, opening_index, closing_index)
9084        disable_previous_link_openers(t, opening_index)
9085        disable_range(t, opening_index, closing_index)
9086      end
9087
```

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.

```
9088    local resolve_inline_following_content =
9089      function(t, closing_index, match_reference, match_link_attributes)
9090        local content = ""
9091        for i = closing_index + 1, #t do
9092          content = content .. t[i].content
9093        end
9094
9095        local matching_content = parsers.succeed
9096
9097        if match_reference then
9098          matching_content = matching_content
9099                           * parsers.inline_direct_ref_inside
9100        end
9101
9102        if match_link_attributes then
9103          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.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")
                  * -parsers.starter) / function(_) return end
                * parsers.spacechar^0

parsers.Endline = parsers.endline
```

```
                / writer.soft_line_break

parsers.EndlineNoSub = parsers.endline

parsers.NoSoftLineBreakEndline
                      = parsers.newline
                      * (parsers.check_minimal_indent
                        * -V("NoSoftLineBreakEndlineExceptions")
                        + parsers.check_optional_indent
                        * -V("NoSoftLineBreakEndlineExceptions")
                        * -parsers.starter)
                      * parsers.spacechar^0
                      / writer.space

parsers.EndlineBreak = parsers.backslash * parsers.Endline
                                         / writer.hard_line_break

parsers.OptionalIndent
                    = parsers.spacechar^1 / writer.space

parsers.Space       = parsers.spacechar^2 * parsers.Endline
                                         / writer.hard_line_break
                    + parsers.spacechar^1
                    * parsers.Endline^-1
                    * parsers.eof / self.expandtabs
                    + parsers.spacechar^1 * parsers.Endline
                                         / writer.soft_line_break
                    + parsers.spacechar^1
                    * -parsers.newline / self.expandtabs

parsers.NoSoftLineBreakSpace
                    = parsers.spacechar^2 * parsers.Endline
                                         / writer.hard_line_break
                    + parsers.spacechar^1
                    * parsers.Endline^-1
                    * parsers.eof / self.expandtabs
                    + parsers.spacechar^1 * parsers.Endline
                                         / writer.soft_line_break
                    + parsers.spacechar^1
                    * -parsers.newline / self.expandtabs

parsers.NonbreakingEndline
                   = 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

  parsers.InlineHtml = Cs(parsers.html_inline_comment)
                     / writer.inline_html_comment
                     + Cs(parsers.html_any_empty_inline_tag
```

```
                          + parsers.html_inline_instruction
                          + parsers.html_inline_cdatasection
                          + parsers.html_inline_declaration
                          + parsers.html_any_open_inline_tag
                          + parsers.html_any_close_tag)
                        / writer.inline_html_tag

parsers.HtmlEntity = parsers.html_entities / writer.string
```

#### 3.1.6.9 Block Elements (local)

```
parsers.DisplayHtml = Cs(parsers.check_trail
                           * ( parsers.html_comment
                             + parsers.html_special_block
                             + parsers.html_block
                             + parsers.html_any_block
                             + parsers.html_instruction
                             + parsers.html_cdatasection
                             + parsers.html_declaration))
                         / writer.block_html_element

parsers.indented_non_blank_line = parsers.indentedline
                                  - parsers.blankline

parsers.Verbatim
  = Cs( parsers.check_code_trail
      * (parsers.line - parsers.blankline)
      * (( parsers.check_minimal_blank_indent_and_full_code_trail
         * parsers.blankline)^0
        * ( (parsers.check_minimal_indent / "")
          * parsers.check_code_trail
          * (parsers.line - parsers.blankline))^1)^0)
  / self.expandtabs / writer.verbatim

parsers.Blockquote   = parsers.blockquote_body
                     / writer.blockquote

parsers.ThematicBreak = parsers.thematic_break_lines
                      / writer.thematic_break

parsers.Reference    = parsers.define_reference_parser
                     / self.register_link

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),
    parsers.par_sep_group(parsers.list_blank))

parsers.tight_list_content_loop
  = V("Block")
```

```
    * parsers.tight_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.ref_or_block_non_separated
    * parsers.tight_list_loop_body_pair.block^0
    +  (V("Paragraph") + V("Plain"))
    * parsers.tight_list_loop_body_pair.par^0

  parsers.loose_list_content_loop
    = V("Block")
    * parsers.loose_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.ref_or_block_list_separated
    * parsers.loose_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.loose_list_loop_body_pair.par^0

  parsers.list_item_tightness_condition
    = -#( parsers.list_blank^0
        * parsers.minimally_indented_ref_or_block_or_par)
    * remove_indent("li")
    + remove_indent("li")
    * parsers.fail

  parsers.indented_content_tight
    = Ct( (parsers.blankline / "")
        * #parsers.list_blank
        * remove_indent("li")
        + ( (V("Reference") + (parsers.blankline / ""))
          * parsers.check_minimal_indent
          * parsers.tight_list_content_loop
          + (V("Reference") + (parsers.blankline / ""))
          + (parsers.tickbox^-1 / writer.escape)
          * parsers.tight_list_content_loop
          )
        * parsers.list_item_tightness_condition)

  parsers.indented_content_loose
    = Ct( (parsers.blankline / "")
        * #parsers.list_blank
        + ( (V("Reference") + (parsers.blankline / ""))
          * parsers.check_minimal_indent
          * parsers.loose_list_content_loop
          + (V("Reference") + (parsers.blankline / ""))
          + (parsers.tickbox^-1 / writer.escape)
          * parsers.loose_list_content_loop))

  parsers.TightListItem = function(starter)
```

```
  return  -parsers.ThematicBreak
          * parsers.add_indent(starter, "li")
          * parsers.indented_content_tight
end

parsers.LooseListItem = function(starter)
  return  -parsers.ThematicBreak
          * parsers.add_indent(starter, "li")
          * parsers.indented_content_loose
          * remove_indent("li")
end

parsers.BulletListOfType = function(bullet_type)
  local bullet = parsers.bullet(bullet_type)
  return  ( Ct( parsers.TightListItem(bullet)
              * ( (parsers.check_minimal_indent / "")
                * parsers.TightListItem(bullet)
                )^0
            )
            * Cc(true)
            * -#( (parsers.list_blank^0 / "")
                * parsers.check_minimal_indent
                * (bullet - parsers.ThematicBreak)
            )
            + Ct( parsers.LooseListItem(bullet)
                * ( (parsers.list_blank^0 / "")
                  * (parsers.check_minimal_indent / "")
                  * parsers.LooseListItem(bullet)
                  )^0
            )
            * Cc(false)
          ) / writer.bulletlist
end

parsers.BulletList = parsers.BulletListOfType(parsers.dash)
                   + parsers.BulletListOfType(parsers.asterisk)
                   + parsers.BulletListOfType(parsers.plus)

local function ordered_list(items,tight,starter)
  local startnum = starter[2][1]
  if options.startNumber then
    startnum = tonumber(startnum) or 1  -- fallback for '#'
    if startnum ~= nil then
      startnum = math.floor(startnum)
    end
  else
    startnum = nil
```

```
9625    end
9626    return writer.orderedlist(items,tight,startnum)
9627  end
9628
9629  parsers.OrderedListOfType = function(delimiter_type)
9630    local enumerator = parsers.enumerator(delimiter_type)
9631    return  Cg(enumerator, "listtype")
9632          * (Ct( parsers.TightListItem(Cb("listtype"))
9633              * ( (parsers.check_minimal_indent / "")
9634                * parsers.TightListItem(enumerator))^0)
9635          * Cc(true)
9636          * -#((parsers.list_blank^0 / "")
9637              * parsers.check_minimal_indent * enumerator)
9638          + Ct( parsers.LooseListItem(Cb("listtype"))
9639              * ((parsers.list_blank^0 / "")
9640                * (parsers.check_minimal_indent / "")
9641                * parsers.LooseListItem(enumerator))^0)
9642          * Cc(false)
9643          ) * Ct(Cb("listtype")) / ordered_list
9644  end
9645
9646  parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
9647                      + parsers.OrderedListOfType(parsers.rparent)
```

#### 3.1.6.11 Blank (local)

```
9648  parsers.Blank        = parsers.blankline / ""
9649                       + V("Reference")
```

#### 3.1.6.12 Headings (local)

```
9650  function parsers.parse_heading_text(s)
9651    local inlines = self.parser_functions.parse_inlines(s)
9652    local flatten_inlines = self.writer.flatten_inlines
9653    self.writer.flatten_inlines = true
9654    local flat_text = self.parser_functions.parse_inlines(s)
9655    flat_text = util.rope_to_string(flat_text)
9656    self.writer.flatten_inlines = flatten_inlines
9657    return {flat_text, inlines}
9658  end
9659
9660  -- parse atx header
9661  parsers.AtxHeading = parsers.check_trail_no_rem
9662                     * Cg(parsers.heading_start, "level")
9663                     * ((C( parsers.optionalspace
9664                          * parsers.hash^0
9665                          * parsers.optionalspace
9666                          * 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
```

```
9702          local_walkable_syntax[lhs] = util.table_copy(rule)
9703        end
9704        return local_walkable_syntax
9705      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.

```
9706      local current_extension_name = nil
9707      self.insert_pattern = function(selector, pattern, pattern_name)
9708        assert(pattern_name == nil or type(pattern_name) == "string")
9709        local _, _, lhs, pos, rhs
9710          = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9711        assert(lhs ~= nil,
9712          [[Expected selector in form ]]
9713          .. [["LHS (before|after|instead of) RHS", not "]]
9714          .. selector .. [["]])
9715        assert(walkable_syntax[lhs] ~= nil,
9716          [[Rule ]] .. lhs
9717          .. [[ -> ... does not exist in markdown grammar]])
9718        assert(pos == "before" or pos == "after" or pos == "instead of",
9719          [[Expected positional specifier "before", "after", ]]
9720          .. [[or "instead of", not "]]
9721          .. pos .. [["]])
9722        local rule = walkable_syntax[lhs]
9723        local index = nil
9724        for current_index, current_rhs in ipairs(rule) do
9725          if type(current_rhs) == "string" and current_rhs == rhs then
9726            index = current_index
9727            if pos == "after" then
9728              index = index + 1
9729            end
9730            break
9731          end
9732        end
9733        assert(index ~= nil,
9734          [[Rule ]] .. lhs .. [[ -> ]] .. rhs
9735            .. [[ does not exist in markdown grammar]])
9736        local accountable_pattern
9737        if current_extension_name then
9738          accountable_pattern
9739            = {pattern, current_extension_name, pattern_name}
9740        else
9741          assert(type(pattern) == "string",
9742            [[reader->insert_pattern() was called outside ]]
9743            .. [[an extension with ]]
9744            .. [[a PEG pattern instead of a rule name]])
```

```
9745          accountable_pattern = pattern
9746        end
9747        if pos == "instead of" then
9748          rule[index] = accountable_pattern
9749        else
9750          table.insert(rule, index, accountable_pattern)
9751        end
9752      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.

```
9753      local syntax =
9754        { "Blocks",
9755
9756          Blocks = V("InitializeState")
9757                 * V("ExpectedJekyllData")
9758                 * 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.

```
9759                 * ( V("Block")
9760                   * ( V("Blank")^0 * parsers.eof
9761                     + ( V("Blank")^2 / writer.paragraphsep
9762                       + V("Blank")^0 / writer.interblocksep
9763                       )
9764                     )
9765                   + ( V("Paragraph") + V("Plain") )
9766                   * ( V("Blank")^0 * parsers.eof
9767                     + ( V("Blank")^2 / writer.paragraphsep
9768                       + V("Blank")^0 / writer.interblocksep
9769                       )
9770                     )
9771                   * V("Block")
9772                   * ( V("Blank")^0 * parsers.eof
9773                     + ( V("Blank")^2 / writer.paragraphsep
9774                       + V("Blank")^0 / writer.interblocksep
9775                       )
9776                     )
9777                   + ( V("Paragraph") + V("Plain") )
9778                   * ( V("Blank")^0 * parsers.eof
9779                     + V("Blank")^0 / writer.paragraphsep
9780                     )
9781                 )^0,
9782
9783          ExpectedJekyllData = parsers.succeed,
9784
9785          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,

        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)
```

```
9827        assert(current_extension_name ~= nil)
9828        assert(syntax[rule_name] ~= nil,
9829          [[Rule ]] .. rule_name
9830          .. [[ -> ... does not exist in markdown grammar]])
9831        local previous_pattern
9832        local extension_name
9833        if walkable_syntax[rule_name] then
9834          local previous_accountable_pattern
9835            = walkable_syntax[rule_name][1]
9836          previous_pattern = previous_accountable_pattern[1]
9837          extension_name
9838            = previous_accountable_pattern[2]
9839            .. ", " .. current_extension_name
9840        else
9841          previous_pattern = nil
9842          extension_name = current_extension_name
9843        end
9844        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
```

```
9845        if type(get_pattern) == "function" then
9846          pattern = get_pattern(previous_pattern)
9847        else
9848          assert(previous_pattern == nil,
9849                 [[Rule ]] .. rule_name ..
9850                 [[ has already been updated by ]] .. extension_name)
9851          pattern = get_pattern
9852        end
9853        local accountable_pattern = { pattern, extension_name, rule_name }
9854        walkable_syntax[rule_name] = { accountable_pattern }
9855      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.

```
9856      local special_characters = {}
9857      self.add_special_character = function(c)
```

```
9858        table.insert(special_characters, c)
9859        syntax.SpecialChar = S(table.concat(special_characters, ""))
9860      end
9861
9862      self.add_special_character("*")
9863      self.add_special_character("[")
9864      self.add_special_character("]")
9865      self.add_special_character("<")
9866      self.add_special_character("!")
9867      self.add_special_character("\\")
```

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

```
9868      self.initialize_named_group = function(name, value)
9869        local pattern = Ct("")
9870        if value ~= nil then
9871          pattern = pattern / value
9872        end
9873        syntax.InitializeState = syntax.InitializeState
9874                               * Cg(pattern, name)
9875      end
```

Add a named group for indentation.

```
9876      self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
9877      for _, extension in ipairs(extensions) do
9878        current_extension_name = extension.name
9879        extension.extend_writer(writer)
9880        extension.extend_reader(self)
9881      end
9882      current_extension_name = nil
```

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

```
9883      if options.debugExtensions then
9884        local sorted_lhs = {}
9885        for lhs, _ in pairs(walkable_syntax) do
9886          table.insert(sorted_lhs, lhs)
9887        end
9888        table.sort(sorted_lhs)
9889
9890        local output_lines = {"{"}
9891        for lhs_index, lhs in ipairs(sorted_lhs) do
9892          local encoded_lhs = util.encode_json_string(lhs)
9893          table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
9894          local rule = walkable_syntax[lhs]
9895          for rhs_index, rhs in ipairs(rule) do
```

```
          local human_readable_rhs
          if type(rhs) == "string" then
            human_readable_rhs = rhs
          else
            local pattern_name
            if rhs[3] then
              pattern_name = rhs[3]
            else
              pattern_name = "Anonymous Pattern"
            end
            local extension_name = rhs[2]
            human_readable_rhs = pattern_name .. [[ (]]
                                 .. extension_name .. [[)]]
          end
          local encoded_rhs
            = util.encode_json_string(human_readable_rhs)
          local output_line = [[        ]] .. encoded_rhs
          if rhs_index < #rule then
            output_line = output_line .. ","
          end
          table.insert(output_lines, output_line)
        end
        local output_line = "    ]"
        if lhs_index < #sorted_lhs then
          output_line = output_line .. ","
        end
        table.insert(output_lines, output_line)
      end
      table.insert(output_lines, "}")

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

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

Although the interface of the `reader->insert_pattern` method does not doc-

ument 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.

```
        if type(rhs) == "string" then
          pattern = V(rhs)
        else
          pattern = rhs[1]
          if type(pattern) == "string" then
            pattern = V(pattern)
          end
        end
        syntax[lhs] = syntax[lhs] + pattern
      end
    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.

```
    if options.underscores then
      self.add_special_character("_")
    end

    if not options.codeSpans then
      syntax.Code = parsers.fail
    else
      self.add_special_character("`")
    end

    if not options.html then
      syntax.DisplayHtml = parsers.fail
      syntax.InlineHtml = parsers.fail
      syntax.HtmlEntity  = parsers.fail
    else
      self.add_special_character("&")
    end

    if options.preserveTabs then
      options.stripIndent = false
    end

    if not options.smartEllipses then
      syntax.Smart = parsers.fail
    else
      self.add_special_character(".")
    end

    if not options.relativeReferences then
```

```
9978        syntax.AutoLinkRelativeReference = parsers.fail
9979      end
9980
9981      if options.contentLevel == "inline" then
9982        syntax[1] = "Inlines"
9983        syntax.Inlines = V("InitializeState")
9984                         * parsers.Inline^0
9985                         * ( parsers.spacing^0
9986                           * parsers.eof / "")
9987        syntax.Space = parsers.Space + parsers.blankline / writer.space
9988      end
9989
9990      local blocks_nested_t = util.table_copy(syntax)
9991      blocks_nested_t.ExpectedJekyllData = parsers.succeed
9992      parsers.blocks_nested = Ct(blocks_nested_t)
9993
9994      parsers.blocks = Ct(syntax)
9995
9996      local inlines_t = util.table_copy(syntax)
9997      inlines_t[1] = "Inlines"
9998      inlines_t.Inlines = V("InitializeState")
9999                         * parsers.Inline^0
10000                        * ( parsers.spacing^0
10001                          * parsers.eof / "")
10002     parsers.inlines = Ct(inlines_t)
10003
10004     local inlines_no_inline_note_t = util.table_copy(inlines_t)
10005     inlines_no_inline_note_t.InlineNote = parsers.fail
10006     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10007
10008     local inlines_no_html_t = util.table_copy(inlines_t)
10009     inlines_no_html_t.DisplayHtml = parsers.fail
10010     inlines_no_html_t.InlineHtml = parsers.fail
10011     inlines_no_html_t.HtmlEntity = parsers.fail
10012     parsers.inlines_no_html = Ct(inlines_no_html_t)
10013
10014     local inlines_nbsp_t = util.table_copy(inlines_t)
10015     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10016     inlines_nbsp_t.Space = parsers.NonbreakingSpace
10017     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10018
10019     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10020     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10021     inlines_no_link_or_emphasis_t.EndlineExceptions
10022       = parsers.EndlineExceptions - parsers.eof
10023     parsers.inlines_no_link_or_emphasis
10024       = Ct(inlines_no_link_or_emphasis_t)
```

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

```
10025     return function(input)
```

Unicode-normalize the input.

```
10026       if options.unicodeNormalization then
10027         local form = options.unicodeNormalizationForm
10028         if form == "nfc" then
10029           input = uni_algos.normalize.NFC(input)
10030         elseif form == "nfd" then
10031           input = uni_algos.normalize.NFD(input)
10032         elseif form == "nfkc" then
10033           input = uni_algos.normalize.NFKC(input)
10034         elseif form == "nfkd" then
10035           input = uni_algos.normalize.NFKD(input)
10036         else
10037           return writer.error(
10038             format("Unknown normalization form %s.", form))
10039         end
10040       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.

```
10041       input = input:gsub("\r\n?", "\n")
10042       if input:sub(-1) ~= "\n" then
10043         input = input .. "\n"
10044       end
```

Clear the table of references.

```
10045       references = {}
10046       local document = self.parser_functions.parse_blocks(input)
10047       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`.

```
10048       local undosep_start, undosep_end
10049       local potential_secend_start, secend_start
10050       local potential_sep_start, sep_start
10051       while true do
10052         -- find a `writer->undosep`
10053         undosep_start, undosep_end
10054           = output:find(writer.undosep_text, 1, true)
10055         if undosep_start == nil then break end
10056         -- skip any preceding section ends
10057         secend_start = undosep_start
10058         while true do
10059           potential_secend_start = secend_start - #writer.secend_text
```

```
10060           if potential_secend_start < 1
10061              or output:sub(potential_secend_start,
10062                            secend_start - 1) ~= writer.secend_text
10063              then
10064             break
10065           end
10066           secend_start = potential_secend_start
10067         end
10068         -- find an immediately preceding
10069         -- block element / paragraph separator
10070         sep_start = secend_start
10071         potential_sep_start = sep_start - #writer.interblocksep_text
10072         if potential_sep_start >= 1
10073            and output:sub(potential_sep_start,
10074                           sep_start - 1) == writer.interblocksep_text
10075            then
10076           sep_start = potential_sep_start
10077         else
10078           potential_sep_start = sep_start - #writer.paragraphsep_text
10079           if potential_sep_start >= 1
10080              and output:sub(potential_sep_start,
10081                             sep_start - 1) == writer.paragraphsep_text
10082              then
10083             sep_start = potential_sep_start
10084           end
10085         end
10086         -- remove `writer->undosep` and immediately preceding
10087         -- block element / paragraph separator
10088         output = output:sub(1, sep_start - 1)
10089               .. output:sub(secend_start, undosep_start - 1)
10090               .. output:sub(undosep_end + 1)
10091       end
10092       return output
10093     end
10094   end
10095   return self
10096 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.

```
10097 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
          base = curr
        else
          setmetatable(prev, { __index = curr })
        end
        prev = curr
      end
      ::continue::
    end
```

```
    return base or {}
  end)()

  return {
    name = "built-in content_blocks syntax extension",
    extend_writer = function(self)
```

Define `writer->contentblock` as a function that will transform an input iA Writer content block to the output format, where `src` corresponds to the URI prefix, `suf` to the URI extension, `type` to the type of the content block (`localfile` or `onlineimage`), and `tit` to the title of the content block.

```
      function self.contentblock(src,suf,type,tit)
        if not self.is_writing then return "" end
        src = src.."."..suf
        suf = suf:lower()
        if type == "onlineimage" then
          return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
                  "{",self.string(src),"}",
                  "{",self.uri(src),"}",
                  "{",self.string(tit or ""),"}"}
        elseif languages_json[suf] then
          return {"\\markdownRendererContentBlockCode{",suf,"}",
                  "{",self.string(languages_json[suf]),"}",
                  "{",self.string(src),"}",
                  "{",self.uri(src),"}",
                  "{",self.string(tit or ""),"}"}
        else
          return {"\\markdownRendererContentBlock{",suf,"}",
                  "{",self.string(src),"}",
                  "{",self.uri(src),"}",
                  "{",self.string(tit or ""),"}"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

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

      -- case insensitive online image suffix:
      local onlineimagesuffix
                    = (function(...)
                        local parser = nil
                        for _, suffix in ipairs({...}) do
                          local pattern=nil
```

```
                        for i=1,#suffix do
                          local char=suffix:sub(i,i)
                          char = S(char:lower()..char:upper())
                          if pattern == nil then
                            pattern = char
                          else
                            pattern = pattern * char
                          end
                        end
                        if parser == nil then
                          parser = pattern
                        else
                          parser = parser + pattern
                        end
                      end
                      return parser
                    end)("png", "jpg", "jpeg", "gif", "tif", "tiff")

    -- online image url for iA Writer content blocks with
    -- mandatory suffix, allowing nested brackets:
    local onlineimageurl
                  = (parsers.less
                    * Cs((parsers.anyescaped
                         - parsers.more
                         - parsers.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)
```

```
10466         if not self.is_writing then return "" end
10467         local buffer = {}
10468         for _,item in ipairs(items) do
10469           buffer[#buffer + 1] = dlitem(item.term, item.definitions)
10470         end
10471         if tight and tight_lists then
10472           return {"\\markdownRendererDlBeginTight\n", buffer,
10473             "\n\\markdownRendererDlEndTight"}
10474         else
10475           return {"\\markdownRendererDlBegin\n", buffer,
10476             "\n\\markdownRendererDlEnd"}
10477         end
10478       end
10479     end, extend_reader = function(self)
10480       local parsers = self.parsers
10481       local writer = self.writer
10482
10483       local defstartchar = S("~:")
10484
10485       local defstart
10486         = parsers.check_trail_length(0) * defstartchar
10487         * #parsers.spacing
10488         * (parsers.tab + parsers.space^-3)
10489         + parsers.check_trail_length(1)
10490         * defstartchar * #parsers.spacing
10491         * (parsers.tab + parsers.space^-2)
10492         + parsers.check_trail_length(2)
10493         * defstartchar * #parsers.spacing
10494         * (parsers.tab + parsers.space^-1)
10495         + parsers.check_trail_length(3)
10496         * defstartchar * #parsers.spacing
10497
10498       local indented_line
10499         = (parsers.check_minimal_indent / "")
10500         * parsers.check_code_trail * parsers.line
10501
10502       local blank
10503         = parsers.check_minimal_blank_indent_and_any_trail
10504         * parsers.optionalspace * parsers.newline
10505
10506       local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10507
10508       local indented_blocks = function(bl)
10509         return Cs( bl
10510               * (blank^1 * (parsers.check_minimal_indent / "")
10511                 * parsers.check_code_trail * -parsers.blankline * bl)^0
10512               * (blank^1 + parsers.eof))
```

```
10513       end
10514
10515       local function definition_list_item(term, defs, _)
10516         return { term = self.parser_functions.parse_inlines(term),
10517                  definitions = defs }
10518       end
10519
10520       local DefinitionListItemLoose
10521         = C(parsers.line) * blank^0
10522         * Ct((parsers.check_minimal_indent * (defstart
10523             * indented_blocks(dlchunk)
10524             / self.parser_functions.parse_blocks_nested))^1)
10525         * Cc(false) / definition_list_item
10526
10527       local DefinitionListItemTight
10528         = C(parsers.line)
10529         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
10530             / self.parser_functions.parse_blocks_nested))^1)
10531         * Cc(true) / definition_list_item
10532
10533       local DefinitionList
10534         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
10535           + Ct(DefinitionListItemTight^1)
10536           * (blank^0
10537             * -DefinitionListItemLoose * Cc(true))
10538           ) / writer.definitionlist
10539
10540       self.insert_pattern("Block after Heading",
10541                           DefinitionList, "DefinitionList")
10542     end
10543   }
10544 end
```

#### 3.1.7.5 Fancy Lists

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

```
10545 M.extensions.fancy_lists = function()
10546   return {
10547     name = "built-in fancy_lists syntax extension",
10548     extend_writer = function(self)
10549       local options = self.options
10550
```

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.

```
10551       function self.fancylist(items,tight,startnum,numstyle,numdelim)
10552         if not self.is_writing then return "" end
10553         local buffer = {}
10554         local num = startnum
10555         for _,item in ipairs(items) do
10556           if item ~= "" then
10557             buffer[#buffer + 1] = self.fancyitem(item,num)
10558           end
10559           if num ~= nil and item ~= "" then
10560             num = num + 1
10561           end
10562         end
10563         local contents = util.intersperse(buffer,"\n")
10564         if tight and options.tightLists then
10565           return {"\\markdownRendererFancyOlBeginTight{",
10566                   numstyle,"}{",numdelim,"}",contents,
10567                   "\n\\markdownRendererFancyOlEndTight "}
10568         else
10569           return {"\\markdownRendererFancyOlBegin{",
10570                   numstyle,"}{",numdelim,"}",contents,
10571                   "\n\\markdownRendererFancyOlEnd "}
10572         end
10573       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.

```
10574       function self.fancyitem(s,num)
10575         if num ~= nil then
```

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

      local function combine_markers_and_delims(markers, delims)
        local markers_table = {}
        for _,marker in ipairs(markers) do
          local start_marker
          local continuation_marker
          if type(marker) == "table" then
            start_marker = marker[1]
            continuation_marker = marker[2]
          else
            start_marker = marker
            continuation_marker = marker
          end
          for _,delim in ipairs(delims) do
            table.insert(markers_table,
                         {start_marker, continuation_marker, delim})
          end
        end
        return markers_table
      end

      local function join_table_with_func(func, markers_table)
        local pattern = func(table.unpack(markers_table[1]))
        for i = 2, #markers_table do
          pattern = pattern + func(table.unpack(markers_table[i]))
        end
        return pattern
      end

      local lowercase_letter_marker = R("az")
      local uppercase_letter_marker = R("AZ")

      local roman_marker = function(chars)
        local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
        local l, x, v, i
          = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
```

```
10623         return  m^-3
10624                 * (c*m + c*d + d^-1 * c^-3)
10625                 * (x*c + x*l + l^-1 * x^-3)
10626                 * (i*x + i*v + v^-1 * i^-3)
10627       end
10628
10629       local lowercase_roman_marker
10630         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10631       local uppercase_roman_marker
10632         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10633
10634       local lowercase_opening_roman_marker  = P("i")
10635       local uppercase_opening_roman_marker  = P("I")
10636
10637       local digit_marker = parsers.dig * parsers.dig^-8
10638
10639       local markers = {
10640         {lowercase_opening_roman_marker, lowercase_roman_marker},
10641         {uppercase_opening_roman_marker, uppercase_roman_marker},
10642         lowercase_letter_marker,
10643         uppercase_letter_marker,
10644         lowercase_roman_marker,
10645         uppercase_roman_marker,
10646         digit_marker
10647       }
10648
10649       local delims = {
10650         parsers.period,
10651         parsers.rparent
10652       }
10653
10654       local markers_table = combine_markers_and_delims(markers, delims)
10655
10656       local function enumerator(start_marker, _,
10657                                 delimiter_type, interrupting)
10658         local delimiter_range
10659         local allowed_end
10660         if interrupting then
10661           delimiter_range = P("1")
10662           allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10663         else
10664           delimiter_range = start_marker
10665           allowed_end = C(parsers.spacechar^1)
10666                       + #(parsers.newline + parsers.eof)
10667         end
10668
10669         return parsers.check_trail
```

```
10670             * Ct(C(delimiter_range) * C(delimiter_type))
10671             * allowed_end
10672     end
10673
10674     local starter = join_table_with_func(enumerator, markers_table)
10675
10676     local TightListItem = function(starter)
10677       return  parsers.add_indent(starter, "li")
10678             * parsers.indented_content_tight
10679     end
10680
10681     local LooseListItem = function(starter)
10682       return  parsers.add_indent(starter, "li")
10683             * parsers.indented_content_loose
10684             * remove_indent("li")
10685     end
10686
10687     local function roman2number(roman)
10688       local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
10689                        ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
10690       local numeral = 0
10691
10692       local i = 1
10693       local len = string.len(roman)
10694       while i < len do
10695         local z1, z2 = romans[ string.sub(roman, i, i) ],
10696                        romans[ string.sub(roman, i+1, i+1) ]
10697         if z1 < z2 then
10698             numeral = numeral + (z2 - z1)
10699             i = i + 2
10700         else
10701             numeral = numeral + z1
10702             i = i + 1
10703         end
10704       end
10705       if i <= len then
10706         numeral = numeral + romans[ string.sub(roman,i,i) ]
10707       end
10708       return numeral
10709     end
10710
10711     local function sniffstyle(numstr, delimend)
10712       local numdelim
10713       if delimend == ")" then
10714         numdelim = "OneParen"
10715       elseif delimend == "." then
10716         numdelim = "Period"
```

```
10717         else
10718           numdelim = "Default"
10719         end
10720
10721         local num
10722         num = numstr:match("^([I])$")
10723         if num then
10724           return roman2number(num), "UpperRoman", numdelim
10725         end
10726         num = numstr:match("^([i])$")
10727         if num then
10728           return roman2number(string.upper(num)), "LowerRoman", numdelim
10729         end
10730         num = numstr:match("^([A-Z])$")
10731         if num then
10732           return string.byte(num) - string.byte("A") + 1,
10733                  "UpperAlpha", numdelim
10734         end
10735         num = numstr:match("^([a-z])$")
10736         if num then
10737           return string.byte(num) - string.byte("a") + 1,
10738                  "LowerAlpha", numdelim
10739         end
10740         num = numstr:match("^([IVXLCDM]+)")
10741         if num then
10742           return roman2number(num), "UpperRoman", numdelim
10743         end
10744         num = numstr:match("^([ivxlcdm]+)")
10745         if num then
10746           return roman2number(string.upper(num)), "LowerRoman", numdelim
10747         end
10748         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10749       end
10750
10751       local function fancylist(items,tight,start)
10752         local startnum, numstyle, numdelim
10753           = sniffstyle(start[2][1], start[2][2])
10754         return writer.fancylist(items,tight,
10755                                 options.startNumber and startnum or 1,
10756                                 numstyle or "Decimal",
10757                                 numdelim or "Default")
10758       end
10759
10760       local FancyListOfType
10761         = function(start_marker, continuation_marker, delimiter_type)
10762           local enumerator_start
10763             = enumerator(start_marker, continuation_marker,
```

```
10764                         delimiter_type)
10765           local enumerator_cont
10766             = enumerator(continuation_marker, continuation_marker,
10767                         delimiter_type)
10768           return Cg(enumerator_start, "listtype")
10769                * (Ct( TightListItem(Cb("listtype"))
10770                    * ((parsers.check_minimal_indent / "")
10771                    * TightListItem(enumerator_cont))^0)
10772                * Cc(true)
10773                * -#((parsers.conditionally_indented_blankline^0 / "")
10774                    * parsers.check_minimal_indent * enumerator_cont)
10775                + Ct( LooseListItem(Cb("listtype"))
10776                    * ((parsers.conditionally_indented_blankline^0 / "")
10777                      * (parsers.check_minimal_indent / "")
10778                      * LooseListItem(enumerator_cont))^0)
10779                * Cc(false)
10780                ) * Ct(Cb("listtype")) / fancylist
10781         end
10782
10783       local FancyList
10784         = join_table_with_func(FancyListOfType, markers_table)
10785
10786       local Endline   = parsers.newline
10787                       * (parsers.check_minimal_indent
10788                         * -parsers.EndlineExceptions
10789                         + parsers.check_optional_indent
10790                         * -parsers.EndlineExceptions
10791                         * -starter)
10792                       * parsers.spacechar^0
10793                       / writer.soft_line_break
10794
10795       self.update_rule("OrderedList", FancyList)
10796       self.update_rule("Endline", Endline)
10797     end
10798   }
10799 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
```

```
10839       local writer = self.writer
10840
10841       local function captures_geq_length(_,i,a,b)
10842         return #a >= #b and i
10843       end
10844
10845       local function strip_enclosing_whitespaces(str)
10846         return str:gsub("^%s*(.-)%s*$", "%1")
10847       end
10848
10849       local tilde_infostring = Cs(Cs((V("HtmlEntity")
10850                                     + parsers.anyescaped
10851                                     - parsers.newline)^0)
10852                                / strip_enclosing_whitespaces)
10853
10854       local backtick_infostring
10855         = Cs( Cs((V("HtmlEntity")
10856             + ( -#(parsers.backslash * parsers.backtick)
10857               * parsers.anyescaped)
10858               - parsers.newline
10859               - parsers.backtick)^0)
10860         / strip_enclosing_whitespaces)
10861
10862       local fenceindent
10863
10864       local function has_trail(indent_table)
10865         return indent_table ~= nil and
10866           indent_table.trail ~= nil and
10867           next(indent_table.trail) ~= nil
10868       end
10869
10870       local function has_indents(indent_table)
10871         return indent_table ~= nil and
10872           indent_table.indents ~= nil and
10873           next(indent_table.indents) ~= nil
10874       end
10875
10876       local function get_last_indent_name(indent_table)
10877         if has_indents(indent_table) then
10878           return indent_table.indents[#indent_table.indents].name
10879         end
10880       end
10881
10882       local count_fenced_start_indent =
10883         function(_, _, indent_table, trail)
10884           local last_indent_name = get_last_indent_name(indent_table)
10885           fenceindent = 0
```

```
      if last_indent_name ~= "li" then
        fenceindent = #trail
      end
      return true
    end

  local fencehead = function(char, infostring)
    return Cmt( Cb("indent_info")
              * parsers.check_trail, count_fenced_start_indent)
          * Cg(char^3, "fencelength")
          * parsers.optionalspace
          * infostring
          * (parsers.newline + parsers.eof)
  end

  local fencetail = function(char)
    return parsers.check_trail_no_rem
          * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
          * parsers.optionalspace * (parsers.newline + parsers.eof)
          + parsers.eof
  end

  local process_fenced_line =
    function(s, i, -- luacheck: ignore s i
             indent_table, line_content, is_blank)
      local remainder = ""
      if has_trail(indent_table) then
        remainder = indent_table.trail.internal_remainder
      end

      if is_blank
         and get_last_indent_name(indent_table) == "li" then
        remainder = ""
      end

      local str = remainder .. line_content
      local index = 1
      local remaining = fenceindent

      while true do
        local c = str:sub(index, index)
        if c == " " and remaining > 0 then
          remaining = remaining - 1
          index = index + 1
        elseif c == "\t" and remaining > 3 then
          remaining = remaining - 4
          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)
```

```
10980           local expanded_code = self.expandtabs(code)
10981
10982           if allow_raw_blocks then
10983             local raw_attr = lpeg.match(parsers.raw_attribute,
10984                                         infostring)
10985             if raw_attr then
10986               return writer.rawBlock(expanded_code, raw_attr)
10987             end
10988           end
10989
10990           local attr = nil
10991           if allow_attributes then
10992             local match = lpeg.match(infostring_with_attributes,
10993                                      infostring)
10994             if match then
10995               infostring, attr = table.unpack(match)
10996             end
10997           end
10998           return writer.fencedCode(expanded_code, infostring, attr)
10999         end)
11000
11001       self.insert_pattern("Block after Verbatim",
11002                           FencedCode, "FencedCode")
11003
11004       local fencestart
11005       if blank_before_code_fence then
11006         fencestart = parsers.fail
11007       else
11008         fencestart = fencehead(parsers.backtick, backtick_infostring)
11009                    + fencehead(parsers.tilde, tilde_infostring)
11010       end
11011
11012       self.update_rule("EndlineExceptions", function(previous_pattern)
11013         if previous_pattern == nil then
11014           previous_pattern = parsers.EndlineExceptions
11015         end
11016         return previous_pattern + fencestart
11017       end)
11018
11019       self.add_special_character("`")
11020       self.add_special_character("~")
11021     end
11022   }
11023 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)
```

```
      end

    return Cg( Cmt( Cb("fenced_div_num_opening_indents")
                  * Cb("fenced_div_level"), pop_indent_table)
             , "fenced_div_level")
  end


  local non_fenced_div_block
    = parsers.check_minimal_indent * V("Block")
    - parsers.check_minimal_indent_and_trail * fenced_div_end

  local non_fenced_div_paragraph
    = parsers.check_minimal_indent * V("Paragraph")
    - parsers.check_minimal_indent_and_trail * fenced_div_end

  local blank = parsers.minimally_indented_blank

  local block_separated = parsers.block_sep_group(blank)
                        * non_fenced_div_block

  local loop_body_pair
    = parsers.create_loop_body_pair(block_separated,
                                    non_fenced_div_paragraph,
                                    parsers.block_sep_group(blank),
                                    parsers.par_sep_group(blank))

  local content_loop  = ( non_fenced_div_block
                        * loop_body_pair.block^0
                        + non_fenced_div_paragraph
                        * block_separated
                        * loop_body_pair.block^0
                        + non_fenced_div_paragraph
                        * loop_body_pair.par^0)
                      * blank^0

  local FencedDiv = fenced_div_begin
                  / function (infostring)
                      local attr
                        = lpeg.match(Ct(parsers.attributes),
                                     infostring)
                      if attr == nil then
                        attr = {"." .. infostring}
                      end
                      return attr
                    end
                  / 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")
```

```
                      / writer.heading

      local function strip_trailing_spaces(s)
        return s:gsub("%s*$","")
      end

      local heading_line  = (parsers.linechar
                            - (parsers.attributes
                              * parsers.optionalspace
                              * parsers.newline))^1
                          - parsers.thematic_break_lines

      local heading_text
        = heading_line
        * ( (V("Endline") / "\n")
          * (heading_line - parsers.heading_level))^0
        * parsers.newline^-1

      local SetextHeading
        = parsers.freeze_trail * parsers.check_trail_no_rem
        * #(heading_text
           * (parsers.attributes
             * parsers.optionalspace
             * parsers.newline)^-1
           * parsers.check_minimal_indent
           * parsers.check_trail
           * parsers.heading_level)
        * Cs(heading_text) / strip_trailing_spaces
        / parsers.parse_heading_text
        * Cg(Ct((parsers.attributes
              * parsers.optionalspace
              * parsers.newline)^-1), "attributes")
        * parsers.check_minimal_indent_and_trail * parsers.heading_level
        * Cb("attributes")
        * parsers.newline
        * parsers.unfreeze_trail
        / writer.heading

      local Heading = AtxHeading + SetextHeading
      self.update_rule("Heading", Heading)
    end
  }
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.

```
11427       function self.note(s)
11428         if self.flatten_inlines then return "" end
11429         return {"\\markdownRendererNote{",s,"}"}
11430       end
11431     end, extend_reader = function(self)
11432       local parsers = self.parsers
11433       local writer = self.writer
11434
11435       local rawnotes = parsers.rawnotes
11436
11437       if inline_notes then
11438         local InlineNote
11439           = parsers.circumflex
11440           * ( parsers.link_label
11441             / self.parser_functions.parse_inlines_no_inline_note)
11442           / writer.note
11443
11444         self.insert_pattern("Inline after LinkAndEmph",
11445                             InlineNote, "InlineNote")
11446       end
11447       if notes then
11448         local function strip_first_char(s)
11449           return s:sub(2)
11450         end
11451
11452         local RawNoteRef
11453                       = #(parsers.lbracket * parsers.circumflex)
11454                       * parsers.link_label / strip_first_char
11455
11456         -- like indirect_link
11457         local function lookup_note(ref)
11458           return writer.defer_call(function()
11459             local found = rawnotes[self.normalize_tag(ref)]
11460             if found then
11461               return writer.note(
11462                 self.parser_functions.parse_blocks_nested(found))
11463             else
11464               return {"[",
11465                 self.parser_functions.parse_inlines("^" .. ref), "]"}
11466             end
11467           end)
11468         end
11469
11470         local function register_note(ref,rawnote)
11471           local normalized_tag = self.normalize_tag(ref)
```

```
11472           if rawnotes[normalized_tag] == nil then
11473             rawnotes[normalized_tag] = rawnote
11474           end
11475           return ""
11476         end
11477
11478         local NoteRef = RawNoteRef / lookup_note
11479
11480         local optionally_indented_line
11481           = parsers.check_optional_indent_and_any_trail * parsers.line
11482
11483         local blank
11484           = parsers.check_optional_blank_indent_and_any_trail
11485           * parsers.optionalspace * parsers.newline
11486
11487         local chunk
11488           = Cs(parsers.line
11489           * (optionally_indented_line - blank)^0)
11490
11491         local indented_blocks = function(bl)
11492           return Cs( bl
11493                 * ( blank^1 * (parsers.check_optional_indent / "")
11494                   * parsers.check_code_trail
11495                   * -parsers.blankline * bl)^0)
11496         end
11497
11498         local NoteBlock
11499                     = parsers.check_trail_no_rem
11500                     * RawNoteRef * parsers.colon
11501                     * parsers.spnlc * indented_blocks(chunk)
11502                     / register_note
11503
11504         local Reference = NoteBlock + parsers.Reference
11505
11506         self.update_rule("Reference", Reference)
11507         self.insert_pattern("Inline before LinkAndEmph",
11508                             NoteRef, "NoteRef")
11509       end
11510
11511       self.add_special_character("^")
11512     end
11513   }
11514 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.

```
11515 M.extensions.pipe_tables = function(table_captions, table_attributes)
11516
11517   local function make_pipe_table_rectangular(rows)
11518     local num_columns = #rows[2]
11519     local rectangular_rows = {}
11520     for i = 1, #rows do
11521       local row = rows[i]
11522       local rectangular_row = {}
11523       for j = 1, num_columns do
11524         rectangular_row[j] = row[j] or ""
11525       end
11526       table.insert(rectangular_rows, rectangular_row)
11527     end
11528     return rectangular_rows
11529   end
11530
11531   local function pipe_table_row(allow_empty_first_column
11532                                 , nonempty_column
11533                                 , column_separator
11534                                 , column)
11535     local row_beginning
11536     if allow_empty_first_column then
11537       row_beginning = -- empty first column
11538                       #(parsers.spacechar^4
11539                         * column_separator)
11540                     * parsers.optionalspace
11541                     * column
11542                     * parsers.optionalspace
11543                     -- non-empty first column
11544                     + parsers.nonindentspace
11545                     * nonempty_column^-1
11546                     * parsers.optionalspace
11547     else
11548       row_beginning = parsers.nonindentspace
11549                     * nonempty_column^-1
11550                     * parsers.optionalspace
11551     end
11552
11553     return Ct(row_beginning
11554             * (-- single column with no leading pipes
11555                #(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
```

```
11599           table.insert(buffer, "}")
11600         end
11601         if attributes ~= nil then
11602           table.insert(buffer,
11603                        "\\markdownRendererTableAttributeContextEnd{}")
11604         end
11605         return buffer
11606       end
11607     end, extend_reader = function(self)
11608       local parsers = self.parsers
11609       local writer = self.writer
11610
11611       local table_hline_separator = parsers.pipe + parsers.plus
11612
11613       local table_hline_column = (parsers.dash
11614                                   - #(parsers.dash
11615                                       * (parsers.spacechar
11616                                         + table_hline_separator
11617                                         + parsers.newline)))^1
11618                                 * (parsers.colon * Cc("r")
11619                                   + parsers.dash * Cc("d"))
11620                                 + parsers.colon
11621                                 * (parsers.dash
11622                                   - #(parsers.dash
11623                                       * (parsers.spacechar
11624                                         + table_hline_separator
11625                                         + parsers.newline)))^1
11626                                 * (parsers.colon * Cc("c")
11627                                   + parsers.dash * Cc("l"))
11628
11629       local table_hline = pipe_table_row(false
11630                                          , table_hline_column
11631                                          , table_hline_separator
11632                                          , table_hline_column)
11633
11634       local table_caption_beginning
11635         = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
11636           * parsers.optionalspace * parsers.newline)^0
11637         * parsers.check_minimal_indent_and_trail
11638         * (P("Table")^-1 * parsers.colon)
11639         * parsers.optionalspace
11640
11641       local function strip_trailing_spaces(s)
11642         return s:gsub("%s*$","")
11643       end
11644
11645       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
```

```
11693
11694       local PipeTable
11695         = Ct( table_row * parsers.newline
11696             * (parsers.check_minimal_indent_and_trail / {})
11697           * table_hline * parsers.newline
11698           * ( (parsers.check_minimal_indent / {})
11699             * table_row * parsers.newline)^0)
11700         / make_pipe_table_rectangular
11701         * table_caption^-1
11702         / writer.table
11703
11704       self.insert_pattern("Block after Blockquote",
11705                           PipeTable, "PipeTable")
11706     end
11707   }
11708 end
```

#### 3.1.7.15 Raw Attributes

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

```
11709 M.extensions.raw_inline = function()
11710   return {
11711     name = "built-in raw_inline syntax extension",
11712     extend_writer = function(self)
11713       local options = self.options
11714
```

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

```
11715       function self.rawInline(s, attr)
11716         if not self.is_writing then return "" end
11717         if self.flatten_inlines then return s end
11718         local name = util.cache_verbatim(options.cacheDir, s)
11719         return {"\\markdownRendererInputRawInline{",
11720                 name,"}{", self.string(attr),"}"}
11721       end
11722     end, extend_reader = function(self)
11723       local writer = self.writer
11724
11725       local RawInline = parsers.inticks
11726                       * parsers.raw_attribute
11727                       / writer.rawInline
11728
11729       self.insert_pattern("Inline before Code",
11730                           RawInline, "RawInline")
11731     end
11732   }
```

```
11733 end
```

#### 3.1.7.16 Strike-Through

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

```
11734 M.extensions.strike_through = function()
11735   return {
11736     name = "built-in strike_through syntax extension",
11737     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.

```
11738       function self.strike_through(s)
11739         if self.flatten_inlines then return s end
11740         return {"\\markdownRendererStrikeThrough{",s,"}"}
11741       end
11742     end, extend_reader = function(self)
11743       local parsers = self.parsers
11744       local writer = self.writer
11745
11746       local StrikeThrough = (
11747         parsers.between(parsers.Inline, parsers.doubletildes,
11748                         parsers.doubletildes)
11749       ) / writer.strike_through
11750
11751       self.insert_pattern("Inline after LinkAndEmph",
11752                           StrikeThrough, "StrikeThrough")
11753
11754       self.add_special_character("~")
11755     end
11756   }
11757 end
```

#### 3.1.7.17 Subscripts

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

```
11758 M.extensions.subscripts = function()
11759   return {
11760     name = "built-in subscripts syntax extension",
11761     extend_writer = function(self)
```

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

```
11762       function self.subscript(s)
11763         if self.flatten_inlines then return s end
11764         return {"\\markdownRendererSubscript{",s,"}"}
```

```
11765       end
11766     end, extend_reader = function(self)
11767       local parsers = self.parsers
11768       local writer = self.writer
11769
11770       local Subscript = (
11771         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
11772       ) / writer.subscript
11773
11774       self.insert_pattern("Inline after LinkAndEmph",
11775                           Subscript, "Subscript")
11776
11777       self.add_special_character("~")
11778     end
11779   }
11780 end
```

#### 3.1.7.18 Superscripts

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

```
11781 M.extensions.superscripts = function()
11782   return {
11783     name = "built-in superscripts syntax extension",
11784     extend_writer = function(self)
```

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

```
11785       function self.superscript(s)
11786         if self.flatten_inlines then return s end
11787         return {"\\markdownRendererSuperscript{",s,"}"}
11788       end
11789     end, extend_reader = function(self)
11790       local parsers = self.parsers
11791       local writer = self.writer
11792
11793       local Superscript = (
11794         parsers.between(parsers.Str, parsers.circumflex,
11795                         parsers.circumflex)
11796       ) / writer.superscript
11797
11798       self.insert_pattern("Inline after LinkAndEmph",
11799                           Superscript, "Superscript")
11800
11801       self.add_special_character("^")
11802     end
11803   }
11804 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

```

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

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

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

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

      self.insert_pattern("Inline after LinkAndEmph",
                          TexMath, "TexMath")

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

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

#### 3.1.7.20 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.

```
M.extensions.jekyll_data = function(expect_jekyll_data,
                                    ensure_jekyll_data)
  return {
```

```
11968     name = "built-in jekyll_data syntax extension",
11969     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.

```
11970       function self.jekyllData(d, t, p)
11971         if not self.is_writing then return "" end
11972
11973         local buf = {}
11974
11975         local keys = {}
11976         for k, _ in pairs(d) do
11977           table.insert(keys, k)
11978         end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
11979         table.sort(keys, function(first, second)
11980           if type(first) ~= type(second) then
11981             return type(first) < type(second)
11982           else
11983             return first < second
11984           end
11985         end)
11986
11987         if not p then
11988           table.insert(buf, "\\markdownRendererJekyllDataBegin")
11989         end
11990
11991         local is_sequence = false
11992         if #d > 0 and #d == #keys then
11993           for i=1, #d do
11994             if d[i] == nil then
11995               goto not_a_sequence
11996             end
11997           end
11998           is_sequence = true
11999         end
12000         ::not_a_sequence::
12001
12002         if is_sequence then
12003           table.insert(buf,
12004             "\\markdownRendererJekyllDataSequenceBegin{")
```

```
12005           table.insert(buf, self.identifier(p or "null"))
12006           table.insert(buf, "}{")
12007           table.insert(buf, #keys)
12008           table.insert(buf, "}")
12009         else
12010           table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12011           table.insert(buf, self.identifier(p or "null"))
12012           table.insert(buf, "}{")
12013           table.insert(buf, #keys)
12014           table.insert(buf, "}")
12015         end
12016
12017         for _, k in ipairs(keys) do
12018           local v = d[k]
12019           local typ = type(v)
12020           k = tostring(k or "null")
12021           if typ == "table" and next(v) ~= nil then
12022             table.insert(
12023               buf,
12024               self.jekyllData(v, t, k)
12025             )
12026           else
12027             k = self.identifier(k)
12028             v = tostring(v)
12029             if typ == "boolean" then
12030               table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12031               table.insert(buf, k)
12032               table.insert(buf, "}{")
12033               table.insert(buf, v)
12034               table.insert(buf, "}")
12035             elseif typ == "number" then
12036               table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12037               table.insert(buf, k)
12038               table.insert(buf, "}{")
12039               table.insert(buf, v)
12040               table.insert(buf, "}")
12041             elseif typ == "string" then
12042               table.insert(buf,
12043                 "\\markdownRendererJekyllDataProgrammaticString{")
12044               table.insert(buf, k)
12045               table.insert(buf, "}{")
12046               table.insert(buf, self.identifier(v))
12047               table.insert(buf, "}")
12048               table.insert(buf,
12049                 "\\markdownRendererJekyllDataTypographicString{")
12050               table.insert(buf, k)
12051               table.insert(buf, "}{")
```

```
12052             table.insert(buf, t(v))
12053             table.insert(buf, "}")
12054           elseif typ == "table" then
12055             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12056             table.insert(buf, k)
12057             table.insert(buf, "}")
12058           else
12059             local error = self.error(format(
12060               "Unexpected type %s for value of "
12061               .. "YAML key %s.", typ, k))
12062             table.insert(buf, error)
12063           end
12064         end
12065       end
12066
12067       if is_sequence then
12068         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12069       else
12070         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12071       end
12072
12073       if not p then
12074         table.insert(buf, "\\markdownRendererJekyllDataEnd")
12075       end
12076
12077       return buf
12078     end
12079   end, extend_reader = function(self)
12080     local parsers = self.parsers
12081     local writer = self.writer
12082
12083     local JekyllData
12084       = Cmt( C((parsers.line - P("---") - P("..."))^0)
12085            , function(s, i, text) -- luacheck: ignore s i
12086                local data
12087                local ran_ok, _ = pcall(function()
12088                  -- TODO: Use `require("tinyyaml")` in TeX Live 2023
12089                  local tinyyaml = require("markdown-tinyyaml")
12090                  data = tinyyaml.parse(text, {timestamps=false})
12091                end)
12092                if ran_ok and data ~= nil then
12093                  return true, writer.jekyllData(data, function(s)
12094                    return self.parser_functions.parse_blocks_nested(s)
12095                  end, nil)
12096                else
12097                  return false
12098                end
```

```
12099               end
12100             )
12101
12102       local UnexpectedJekyllData
12103         = P("---")
12104         * parsers.blankline / 0
12105         -- if followed by blank, it's thematic break
12106         * #(-parsers.blankline)
12107         * JekyllData
12108         * (P("---") + P("..."))
12109
12110       local ExpectedJekyllData
12111         = ( P("---")
12112           * parsers.blankline / 0
12113           -- if followed by blank, it's thematic break
12114           * #(-parsers.blankline)
12115           )^-1
12116         * JekyllData
12117         * (P("---") + P("..."))^-1
12118
12119       if ensure_jekyll_data then
12120         ExpectedJekyllData = ExpectedJekyllData
12121                            * parsers.eof
12122       else
12123         ExpectedJekyllData = ( ExpectedJekyllData
12124                              * (V("Blank")^0 / writer.interblocksep)
12125                              )^-1
12126       end
12127
12128       self.insert_pattern("Block before Blockquote",
12129                           UnexpectedJekyllData, "UnexpectedJekyllData")
12130       if expect_jekyll_data then
12131         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12132       end
12133     end
12134   }
12135 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`.

```
12136 function M.new(options)
```

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

```
12137   options = options or {}
12138   setmetatable(options, { __index = function (_, key)
12139     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.

```
12140   local parser_convert = nil
12141   return function(input)
12142     local function convert(input)
12143       if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
12144         local parser = require("markdown-parser")
12145         if metadata.version ~= parser.metadata.version then
12146           warn("markdown.lua " .. metadata.version .. " used with " ..
12147                "markdown-parser.lua " .. parser.metadata.version .. ".")
12148         end
12149         parser_convert = parser.new(options)
12150       end
12151       return parser_convert(input)
12152     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).

```
12153     local output
12154     if options.eagerCache or options.finalizeCache then
12155       local salt = util.salt(options)
12156       local name = util.cache(options.cacheDir, input, salt, convert,
12157                               ".md.tex")
12158       output = [[\input{]] .. name .. [[}\relax]]
```

Otherwise, return the result of the conversion directly.

```
12159     else
12160       output = convert(input)
12161     end
```

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

```
12162     if options.finalizeCache then
12163       local file, mode
12164       if options.frozenCacheCounter > 0 then
12165         mode = "a"
```

```
      else
        mode = "w"
      end
      file = assert(io.open(options.frozenCacheFileName, mode),
        [[Could not open file "]] .. options.frozenCacheFileName
        .. [[" for writing]])
      assert(file:write(
        [[\expandafter\global\expandafter\def\csname ]]
        .. [[markdownFrozenCache]] .. options.frozenCacheCounter
        .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
      assert(file:close())
    end
    return output
  end
end
```

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.

```
function M.new(options)
```

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

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

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
  if options.singletonCache and singletonCache.convert then
    for k, v in pairs(defaultOptions) do
      if type(v) == "table" then
        for i = 1, math.max(#singletonCache.options[k], #options[k]) do
          if singletonCache.options[k][i] ~= options[k][i] then
            goto miss
          end
        end
```

The `cacheDir` option is disregarded.

```
      elseif k ~= "cacheDir"
         and singletonCache.options[k] ~= options[k] then
        goto miss
      end
    end
    return singletonCache.convert
  end
  ::miss::
```

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

```
  local extensions = {}
```

```
12202
12203   if options.bracketedSpans then
12204     local bracketed_spans_extension = M.extensions.bracketed_spans()
12205     table.insert(extensions, bracketed_spans_extension)
12206   end
12207
12208   if options.contentBlocks then
12209     local content_blocks_extension = M.extensions.content_blocks(
12210       options.contentBlocksLanguageMap)
12211     table.insert(extensions, content_blocks_extension)
12212   end
12213
12214   if options.definitionLists then
12215     local definition_lists_extension = M.extensions.definition_lists(
12216       options.tightLists)
12217     table.insert(extensions, definition_lists_extension)
12218   end
12219
12220   if options.fencedCode then
12221     local fenced_code_extension = M.extensions.fenced_code(
12222       options.blankBeforeCodeFence,
12223       options.fencedCodeAttributes,
12224       options.rawAttribute)
12225     table.insert(extensions, fenced_code_extension)
12226   end
12227
12228   if options.fencedDivs then
12229     local fenced_div_extension = M.extensions.fenced_divs(
12230       options.blankBeforeDivFence)
12231     table.insert(extensions, fenced_div_extension)
12232   end
12233
12234   if options.headerAttributes then
12235     local header_attributes_extension = M.extensions.header_attributes()
12236     table.insert(extensions, header_attributes_extension)
12237   end
12238
12239   if options.inlineCodeAttributes then
12240     local inline_code_attributes_extension =
12241       M.extensions.inline_code_attributes()
12242     table.insert(extensions, inline_code_attributes_extension)
12243   end
12244
12245   if options.jekyllData then
12246     local jekyll_data_extension = M.extensions.jekyll_data(
12247       options.expectJekyllData, options.ensureJekyllData)
12248     table.insert(extensions, jekyll_data_extension)
```

```
12249   end
12250
12251   if options.linkAttributes then
12252     local link_attributes_extension =
12253       M.extensions.link_attributes()
12254     table.insert(extensions, link_attributes_extension)
12255   end
12256
12257   if options.lineBlocks then
12258     local line_block_extension = M.extensions.line_blocks()
12259     table.insert(extensions, line_block_extension)
12260   end
12261
12262   if options.mark then
12263     local mark_extension = M.extensions.mark()
12264     table.insert(extensions, mark_extension)
12265   end
12266
12267   if options.pipeTables then
12268     local pipe_tables_extension = M.extensions.pipe_tables(
12269       options.tableCaptions, options.tableAttributes)
12270     table.insert(extensions, pipe_tables_extension)
12271   end
12272
12273   if options.rawAttribute then
12274     local raw_inline_extension = M.extensions.raw_inline()
12275     table.insert(extensions, raw_inline_extension)
12276   end
12277
12278   if options.strikeThrough then
12279     local strike_through_extension = M.extensions.strike_through()
12280     table.insert(extensions, strike_through_extension)
12281   end
12282
12283   if options.subscripts then
12284     local subscript_extension = M.extensions.subscripts()
12285     table.insert(extensions, subscript_extension)
12286   end
12287
12288   if options.superscripts then
12289     local superscript_extension = M.extensions.superscripts()
12290     table.insert(extensions, superscript_extension)
12291   end
12292
12293   if options.texMathDollars or
12294      options.texMathSingleBackslash or
12295      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,
```

```
12338         [[User-defined syntax extension "]] .. pathname
12339         .. [[" does not specify mandatory field "api_version"]])
12340       assert(type(user_extension.api_version) == "number",
12341         [[User-defined syntax extension "]] .. pathname
12342         .. [[" specifies field "api_version" of type "]]
12343         .. type(user_extension.api_version)
12344         .. [[" but "number" was expected]])
12345       assert(user_extension.api_version > 0
12346          and user_extension.api_version
12347           <= metadata.user_extension_api_version,
12348         [[User-defined syntax extension "]] .. pathname
12349         .. [[" uses syntax extension API version "]]
12350         .. user_extension.api_version .. [[ but markdown.lua ]]
12351         .. metadata.version .. [[ uses API version ]]
12352         .. metadata.user_extension_api_version
12353         .. [[, which is incompatible]])
12354
12355       assert(user_extension.grammar_version ~= nil,
12356         [[User-defined syntax extension "]] .. pathname
12357         .. [[" does not specify mandatory field "grammar_version"]])
12358       assert(type(user_extension.grammar_version) == "number",
12359         [[User-defined syntax extension "]] .. pathname
12360         .. [[" specifies field "grammar_version" of type "]]
12361         .. type(user_extension.grammar_version)
12362         .. [[" but "number" was expected]])
12363       assert(user_extension.grammar_version == metadata.grammar_version,
12364         [[User-defined syntax extension "]] .. pathname
12365         .. [[" uses grammar version "]]
12366         .. user_extension.grammar_version
12367         .. [[ but markdown.lua ]] .. metadata.version
12368         .. [[ uses grammar version ]] .. metadata.grammar_version
12369         .. [[, which is incompatible]])
12370
12371       assert(user_extension.finalize_grammar ~= nil,
12372         [[User-defined syntax extension "]] .. pathname
12373         .. [[" does not specify mandatory "finalize_grammar" field]])
12374       assert(type(user_extension.finalize_grammar) == "function",
12375         [[User-defined syntax extension "]] .. pathname
12376         .. [[" specifies field "finalize_grammar" of type "]]
12377         .. type(user_extension.finalize_grammar)
12378         .. [[" 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.)

```
12379       local extension = {
12380         name = [[user-defined "]] .. pathname .. [[" syntax extension]],
12381         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
```

```
12416
```

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

```
12417 local lfs = require("lfs")
12418 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12419   assert(lfs.mkdir(options["cacheDir"]))
12420 end
```

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

```
12421 local kpse
12422 (function()
12423   local should_initialize = package.loaded.kpse == nil
12424                         or tex.initialize ~= nil
12425   kpse = require("kpse")
12426   if should_initialize then
12427     kpse.set_program_name("luatex")
12428   end
12429 end)()
12430 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.

```
12431 if metadata.version ~= md.metadata.version then
12432   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12433        "markdown.lua " .. md.metadata.version .. ".")
12434 end
12435 local convert = md.new(options)
12436 local output = convert(input)
12437
12438 if output_filename then
12439   local output_file = assert(io.open(output_filename, "w"),
12440     [[Could not open file "]] .. output_filename .. [[" for writing]])
12441   assert(output_file:write(output))
12442   assert(output_file:close())
12443 else
12444   assert(io.write(output))
12445 end
```

Remove the `options.cacheDir` directory if it is empty.

```
12446 if options.cacheDir then
12447   lfs.rmdir(options.cacheDir)
12448 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
\cs_new:Nn
  \@@_plain_tex_load_theme:nn
  {
    \prop_get:NnNTF
      \g_@@_plain_tex_loaded_themes_linenos_prop
      { #1 }
      \l_tmpa_tl
      {
        \msg_warning:nnnV
          { markdown }
          { repeatedly-loaded-plain-tex-theme }
          { #1 }
          \l_tmpa_tl
      }
      {
        \msg_info:nnn
          { markdown }
          { loading-plain-tex-theme }
          { #1 }
        \prop_gput:Nnx
          \g_@@_plain_tex_loaded_themes_linenos_prop
          { #1 }
          { \tex_the:D \tex_inputlineno:D }
        \file_input:n
```

```
          { markdown theme #2 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-plain-tex-theme }
  { Loading~plain~TeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-plain-tex-theme }
  {
    Plain~TeX~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\cs_generate_variant:Nn
  \prop_gput:Nnn
  { Nnx }
\cs_gset_eq:NN
  \@@_load_theme:nn
  \@@_plain_tex_load_theme:nn
\cs_generate_variant:Nn
  \@@_load_theme:nn
  { nV }
```

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:VV
      \l_tmpb_tl
      \l_tmpa_str
  }
\cs_generate_variant:Nn
  \tl_set:Nn
  { Ne }
\cs_generate_variant:Nn
  \@@_plain_tex_load_theme:nn
  { VV }
\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 }
      {
        \@@_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 @
        for the closing token sequence "#1"}@
    }@
```

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

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

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

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

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

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

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

```
      |ifx|relax##3|relax@
        |markdownIfOption{frozenCache}{}{@
          |immediate|write|markdownOutputFileStream{##1}@
        }@
      |else@
```

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

```
      |def^^M{@
        |markdownInfo{The ending token sequence was found}@
        |markdownIfOption{frozenCache}{}{@
          |immediate|closeout|markdownOutputFileStream@
        }@
        |endgroup@
        |markdownInput{@
          |markdownOptionOutputDir@
          /|markdownOptionInputTempFileName@
        }@
        #2}@
    |fi@
```

Repeat with the next line.

```
    ^^M}@
```

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

```
  |catcode`|^^I=13@
  |def^^I{|markdownReadAndConvertTab}@
```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```
  |catcode`|^^M=13@
  |def^^M##1^^M{@
    |def^^M####1^^M{@
      |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
    ^^M}@
  ^^M}@
```

Reset the character categories back to the former state.

```
|endgroup
```

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 [12, Section 9]. As a consequence, we can directly reuse the existing plain TEX implementation.

```
\def\markdownVersionSpace{ }%
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
  \markdownVersion\markdownVersionSpace markdown renderer]%
```

### 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 2.3.2).

```
\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 `\markdownInput` macro. The `\markdownInput` macro is then redefined to accept an optional argument with options recognized by the LATEX interface (see Section 2.3.2).

```
\let\markdownInputPlainTeX\markdownInput
\renewcommand\markdownInput[2][]{%
  \begingroup
    \markdownSetup{#1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
```

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

```
\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 ]
  {
    \msg_warning:nnn
      { markdown }
      { latex-markdown-star-deprecated }
      { #1 }
    \@@_setup:n
      { #1 }
    \markdownReadAndConvert@
      { markdown* }
  }
  { \markdownEnd }
\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
\cs_gset:Nn
```

```
\@@_load_theme:nn
{
```

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 #2.sty }
        {
          \msg_error:nnn
            { markdown }
            { latex-theme-after-preamble }
            { #1 }
        }
        {
          \@@_plain_tex_load_theme:nn
            { #1 }
            { #2 }
        }
    \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 #2.sty }
        {
          \msg_info:nnn
            { markdown }
            { loading-latex-theme }
            { #1 }
          \RequirePackage
            { markdown theme #2 }
        }
        {
          \@@_plain_tex_load_theme:nn
            { #1 }
            { #2 }
        }
    \fi
  \else
```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```
      \msg_info:nnn
        { markdown }
        { theme-loading-postponed }
        { #1 }
      \AtEndOfPackage
        {
          \@@_set_theme:n
            { #1 }
        }
    \fi
  }
\msg_new:nnn
  { markdown }
  { theme-loading-postponed }
  {
    Postponing~loading~Markdown~theme~#1~until~
    Markdown~package~has~finished~loading
  }
\msg_new:nnn
  { markdown }
  { loading-latex-theme }
  { Loading~LaTeX~Markdown~theme~#1 }
\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"};
```

```
13574             printf '%s' "$OUTPUT" > "\filename";
13575           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:

```
13576         else
13577           printf '%s' '#3' > "\filename";
13578         fi}^^A
```

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

```
13579     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
13580     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
13581       {#1}{#2}{\filename}{#4}^^A
13582   \endgroup
13583   \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
13584 \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:

```
13585 \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.

```
13586 \DeclareOption*{%
13587   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
13588 \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.

```
13589 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

If either the tightLists or the fancyLists Lua option is enabled and the current document class is not beamer, then load the paralist package.

```
13590 \@ifclassloaded{beamer}{}{%
13591   \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
13592   \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
13593 }
```

If we loaded the paralist package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```
\ExplSyntaxOn
\@ifpackageloaded{paralist}{
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_number_style_tl
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_delimiter_style_tl
  \cs_new:Nn
    \@@_latex_fancy_list_item_label_number:nn
    {
      \str_case:nn
        { #1 }
        {
          { Decimal } { #2 }
          { LowerRoman } { \int_to_roman:n { #2 } }
          { UpperRoman } { \int_to_Roman:n { #2 } }
          { LowerAlpha } { \int_to_alph:n { #2 } }
          { UpperAlpha } { \int_to_Alph:n { #2 } }
        }
    }
  \cs_new:Nn
    \@@_latex_fancy_list_item_label_delimiter:n
    {
      \str_case:nn
        { #1 }
        {
          { Default } { . }
          { OneParen } { ) }
          { Period } { . }
        }
    }
  \cs_new:Nn
    \@@_latex_fancy_list_item_label:nnn
    {
      \@@_latex_fancy_list_item_label_number:nn
        { #1 }
        { #3 }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \cs_new:Nn
    \@@_latex_paralist_style:nn
    {
      \str_case:nn
        { #1 }
```

```
        {
          { Decimal } { 1 }
          { LowerRoman } { i }
          { UpperRoman } { I }
          { LowerAlpha } { a }
          { UpperAlpha } { A }
        }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \markdownSetup{rendererPrototypes={
```

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 }
      \@@_if_option:nTF
        { startNumber }
        {
          \tl_set:Nn
            \l_tmpa_tl
            { \begin{enumerate} }
        }
        {
          \tl_set:Nn
            \l_tmpa_tl
            { \begin{enumerate}[ }
          \tl_put_right:Nx
            \l_tmpa_tl
            { \@@_latex_paralist_style:nn { #1 } { #2 } }
          \tl_put_right:Nn
```

```
            \l_tmpa_tl
            { ] }
        }
      \tl_use:N
        \l_tmpa_tl
    },
    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 }
      \@@_if_option:nTF
        { startNumber }
        {
          \tl_set:Nn
            \l_tmpa_tl
            { \begin{compactenum} }
        }
        {
          \tl_set:Nn
            \l_tmpa_tl
            { \begin{compactenum}[ }
          \tl_put_right:Nx
            \l_tmpa_tl
            { \@@_latex_paralist_style:nn { #1 } { #2 } }
          \tl_put_right:Nn
            \l_tmpa_tl
```

```
            { ] }
        }
      \tl_put_left:Nn
        \l_tmpa_tl
        {
          \plpartopsep=\partopsep
          \pltopsep=\topsep
        }
      \tl_use:N
        \l_tmpa_tl
    },
    fancyOlEndTight = {
      \end{compactenum}
      \group_end:
    },
    fancyOlItemWithNumber = {
      \item
        [
          \@@_latex_fancy_list_item_label:VVn
            \l_@@_latex_fancy_list_item_label_number_style_tl
            \l_@@_latex_fancy_list_item_label_delimiter_style_tl
            { #1 }
        ]
    },
```

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:
    }}}
  \cs_generate_variant:Nn
    \@@_latex_fancy_list_item_label:nnn
    { VVn }
}{
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\markdownRendererUlBegin},
    ulEndTight = {\markdownRendererUlEnd},
    fancyOlBegin = {\markdownRendererOlBegin},
    fancyOlEnd = {\markdownRendererOlEnd},
    olBeginTight = {\markdownRendererOlBegin},
```

```
    olEndTight = {\markdownRendererOlEnd},
    fancyOlBeginTight = {\markdownRendererOlBegin},
    fancyOlEndTight = {\markdownRendererOlEnd},
    dlBeginTight = {\markdownRendererDlBegin},
    dlEndTight = {\markdownRendererDlEnd}}}
}
\ExplSyntaxOff
\RequirePackage{amsmath}
```

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

```
\@ifpackageloaded{unicode-math}{
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\mdlgwhtsquare$},
  }}
}{
  \RequirePackage{amssymb}
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\square$},
  }}
}
\RequirePackage{csvsimple}
\RequirePackage{fancyvrb}
\RequirePackage{graphicx}
\markdownSetup{rendererPrototypes={
  hardLineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  pipe = {\textbar},
```

We can capitalize on the fact that the expansion of renderers is performed by TEX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,[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}%
```

[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.

```
    \else
      \texttt{#1}%
    \fi
  }}}
\ExplSyntaxOn
\markdownSetup{
  rendererPrototypes = {
    contentBlock = {
      \str_case:nnF
        { #1 }
        {
          { csv }
            {
              \begin{table}
                \begin{center}
                  \csvautotabular{#3}
                \end{center}
                \tl_if_empty:nF
                  { #4 }
                  { \caption{#4} }
              \end{table}
            }
          { tex } { \markdownEscape{#3} }
        }
        { \markdownInput{#3} }
    },
  },
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  image = {%
    \begin{figure}%
      \begin{center}%
        \includegraphics[alt={#1}]{#3}%
      \end{center}%
      \ifx\empty#4\empty\else
        \caption{#4}%
      \fi
    \end{figure}},
  ulBegin = {\begin{itemize}},
  ulEnd = {\end{itemize}},
  olBegin = {\begin{enumerate}},
  olItem = {\item{}},
  olItemWithNumber = {\item[#1.]},
  olEnd = {\end{enumerate}},
  dlBegin = {\begin{description}},
  dlItem = {\item[#1]},
```

```
  dlEnd = {\end{description}},
  emphasis = {\emph{#1}},
  tickedBox = {$\boxtimes$},
  halfTickedBox = {$\boxdot$}}}
```

If 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}}}}
```

#### 3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\RequirePackage{ltxcmds}
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputFencedCodePrototype#1#2#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`\#=6\relax
            \exp_args:NV
              \inputminted
              \l_tmpa_tl
              { #1 }
            \catcode`\#=12\relax
          }
          {
            \ltx@ifpackageloaded
              { minted2 }
              {
                \catcode`\#=6\relax
                \exp_args:NV
                  \inputminted
                  \l_tmpa_tl
                  { #1 }
                \catcode`\#=12\relax
              }
              {
```

When the listings package is loaded, use it for syntax highlighting.

```
                \ltx@ifpackageloaded
                  { listings }
                  { \lstinputlisting[language=\l_tmpa_tl]{#1} }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
                  { \markdownRendererInputFencedCode{#1}{}{} }
              }
          }
```

```
      }
  }
\ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
\ExplSyntaxOn
\def\markdownLATEXStrongEmphasis#1{%
  \str_if_in:NnTF
    \f@series
    { b }
    { \textnormal{#1} }
    { \textbf{#1} }
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={strongEmphasis={%
  \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support LaTeX document classes that do not provide chapters.

```
\@ifundefined{chapter}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}},
    headingFive = {\subparagraph{#1}}}}
}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}},
    headingSix = {\subparagraph{#1}}}}
}%
```

#### 3.3.4.2 Tickboxes

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

```
\markdownSetup{
  rendererPrototypes = {
    ulItem = {%
      \futurelet\markdownLaTeXCheckbox\markdownLaTeXUlItem
    },
  },
}
\def\markdownLaTeXUlItem{%
  \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
```

```
    \item[\markdownLaTeXCheckbox]%
    \expandafter\@gobble
  \else
    \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
      \item[\markdownLaTeXCheckbox]%
      \expandafter\expandafter\expandafter\@gobble
    \else
      \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
        \item[\markdownLaTeXCheckbox]%
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter\@gobble
      \else
        \item{}%
      \fi
    \fi
  \fi
}
```

#### 3.3.4.3 HTML elements

If the `html` option is enabled and we are using TEX4ht[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
      },
    },
  }
}
```

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

#### 3.3.4.4 Citations

Here is a basic implementation for citations that uses the LaTeX `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibLaTeX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\RequirePackage{gobble}
\def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#4\relax
    \ifx\relax#5\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
        \relax
        \cite{#1#2#6}%  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}%
    }}}}
```

#### 3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```
\RequirePackage{url}
\RequirePackage{expl3}
\ExplSyntaxOn
\def\markdownRendererLinkPrototype#1#2#3#4{
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_set:Nn \l_tmpb_tl { #2 }
  \bool_set:Nn
    \l_tmpa_bool
    {
      \tl_if_eq_p:NN
        \l_tmpa_tl
        \l_tmpb_tl
    }
  \tl_set:Nn \l_tmpa_tl { #4 }
  \bool_set:Nn
    \l_tmpb_bool
    {
      \tl_if_empty_p:N
        \l_tmpa_tl
    }
```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```
  \bool_if:nTF
    {
      \l_tmpa_bool && \l_tmpb_bool
    }
    {
      \markdownLaTeXRendererAutolink { #2 } { #3 }
    }{
      \markdownLaTeXRendererDirectOrIndirectLink
        { #1 } { #2 } { #3 } { #4 }
    }
}
\def\markdownLaTeXRendererAutolink#1#2{%
```

If the URL begins with a hash sign, then we assume that it is a relative reference. Otherwise, we assume that it is an absolute URL.

```
  \tl_set:Nn
    \l_tmpa_tl
    { #2 }
  \tl_trim_spaces:N
    \l_tmpa_tl
  \tl_set:Nx
    \l_tmpb_tl
    {
      \tl_range:Nnn
        \l_tmpa_tl
        { 1 }
        { 1 }
    }
  \str_if_eq:NNTF
    \l_tmpb_tl
    \c_hash_str
    {
      \tl_set:Nx
        \l_tmpb_tl
        {
          \tl_range:Nnn
            \l_tmpa_tl
            { 2 }
            { -1 }
        }
      \exp_args:NV
        \ref
        \l_tmpb_tl
    }{
      \url { #2 }
    }
}
\ExplSyntaxOff
\def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
  #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

#### 3.3.4.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```
\newcount\markdownLaTeXRowCounter
\newcount\markdownLaTeXRowTotal
\newcount\markdownLaTeXColumnCounter
\newcount\markdownLaTeXColumnTotal
\newtoks\markdownLaTeXTable
\newtoks\markdownLaTeXTableAlignment
\newtoks\markdownLaTeXTableEnd
```

```
\AtBeginDocument{%
  \@ifpackageloaded{booktabs}{%
    \def\markdownLaTeXTopRule{\toprule}%
    \def\markdownLaTeXMidRule{\midrule}%
    \def\markdownLaTeXBottomRule{\bottomrule}%
  }{%
    \def\markdownLaTeXTopRule{\hline}%
    \def\markdownLaTeXMidRule{\hline}%
    \def\markdownLaTeXBottomRule{\hline}%
  }%
}
\markdownSetup{rendererPrototypes={
  table = {%
    \markdownLaTeXTable={}%
    \markdownLaTeXTableAlignment={}%
    \markdownLaTeXTableEnd={%
      \markdownLaTeXBottomRule
      \end{tabular}}%
    \ifx\empty#1\empty\else
      \addto@hook\markdownLaTeXTable{%
        \begin{table}
        \centering}%
      \addto@hook\markdownLaTeXTableEnd{%
        \caption{#1}
        \end{table}}%
    \fi
    \addto@hook\markdownLaTeXTable{\begin{tabular}}%
    \markdownLaTeXRowCounter=0%
    \markdownLaTeXRowTotal=#2%
    \markdownLaTeXColumnTotal=#3%
    \markdownLaTeXRenderTableRow
  }
}}
\def\markdownLaTeXRenderTableRow#1{%
  \markdownLaTeXColumnCounter=0%
  \ifnum\markdownLaTeXRowCounter=0\relax
    \markdownLaTeXReadAlignments#1%
    \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
      \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
        \the\markdownLaTeXTableAlignment}}%
    \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
  \else
    \markdownLaTeXRenderTableCell#1%
  \fi
  \ifnum\markdownLaTeXRowCounter=1\relax
    \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
  \fi
```

```
  \advance\markdownLaTeXRowCounter by 1\relax
  \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
    \the\markdownLaTeXTable
    \the\markdownLaTeXTableEnd
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableRow}
\def\markdownLaTeXReadAlignments#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \if#1d%
    \addto@hook\markdownLaTeXTableAlignment{l}%
  \else
    \addto@hook\markdownLaTeXTableAlignment{#1}%
  \fi
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
    \expandafter\@gobble
  \fi\markdownLaTeXReadAlignments}
\def\markdownLaTeXRenderTableCell#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
    \addto@hook\markdownLaTeXTable{#1&}%
  \else
    \addto@hook\markdownLaTeXTable{#1\\}%
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableCell}
```

#### 3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

\markdownIfOption{lineBlocks}{%
  \RequirePackage{verse}
  \markdownSetup{rendererPrototypes={
    lineBlockBegin = {%
      \begingroup
        \def\markdownRendererHardLineBreak{\\}%
        \begin{verse}%
    },
    lineBlockEnd = {%
        \end{verse}%
      \endgroup
    },
  }}
}{}

```

#### 3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  {
    author  .code:n = { \author{#1} },
    date    .code:n = { \date{#1}   },
    title   .code:n = { \title{#1}  },
  }
```

To complement the default setup of our key–values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```
\markdownSetup{
  rendererPrototypes = {
    jekyllDataEnd = {
      \AddToHook{begindocument/end}{\maketitle}
    },
  },
}
\ExplSyntaxOff
```

#### 3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the soulutf8 package and use it to implement strike-throughs.

```
\markdownIfOption{strikeThrough}{%
  \RequirePackage{soulutf8}%
  \markdownSetup{
    rendererPrototypes = {
      strikeThrough = {%
        \st{#1}%
      },
    }
  }
}{}
```

#### 3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the soulutf8 package and use it to implement marked text.

```
\markdownIfOption{mark}{%
  \RequirePackage{soulutf8}%
  \markdownSetup{
```

```
    rendererPrototypes = {
      mark = {%
        \hl{#1}%
      },
    }
  }
}{}
```

#### 3.3.4.11 Image Attributes

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

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

#### 3.3.4.12 Raw Attributes

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

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

```
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline: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` macro is defined to accept an optional argument with options recognized by the ConTEXt interface (see Section 2.4.2).

```
\long\def\inputmarkdown{%
  \dosingleempty
  \doinputmarkdown}%
\long\def\doinputmarkdown[#1]#2{%
  \begingroup
    \iffirstargument
      \setupmarkdown[#1]%
    \fi
    \markdownInput{#2}%
  \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's TEX, trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because "these spaces are hard to see in an editor". At the moment, there is no option to suppress this behavior in (Lua)TEX, but ConTEXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTEXt MkIV and therefore to insert hard line breaks into markdown text.

```
\startluacode
  document.markdown_buffering = false
  local function preserve_trailing_spaces(line)
    if document.markdown_buffering then
      line = line:gsub("[ \t][ \t]$", "\t\t")
    end
    return line
  end
  resolvers.installinputlinehandler(preserve_trailing_spaces)
\stopluacode
\begingroup
  \catcode`\|=0%
  \catcode`\\=12%
  |gdef|startmarkdown{%
    |ctxlua{document.markdown_buffering = true}%
    |markdownReadAndConvert{\stopmarkdown}%
                           {|stopmarkdown}}%
  |gdef|stopmarkdown{%
    |ctxlua{document.markdown_buffering = false}%
    |markdownEnd}%
|endgroup
```

### 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
\cs_gset:Nn
  \@@_load_theme:nn
  {
```

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 #2.tex }
      {
        \msg_info:nnn
          { markdown }
          { loading-context-theme }
```

```
          { #1 }
        \usemodule
          [ t ]
          [ markdown theme #2 ]
      }
      {
        \@@_plain_tex_load_theme:nn
          { #1 }
          { #2 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-context-theme }
  { Loading~ConTeXt~Markdown~theme~#1 }
\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).

```
\markdownIfOption{noDefaults}{}{
  \setupmarkdown[theme=witiko/markdown/defaults]
}
\stopmodule
\protect
```

## References


[1] LuaTEX development team. *LuaTEX reference manual*. Version 1.10 (stable). July 23, 2021. URL: https://www.pragma-ade.com/general/manuals/luatex.pdf (visited on 09/30/2022).

[2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: https://pandoc.org/ (visited on 10/05/2022).

[5] Bonita Sharif and Jonathan I. Maletic. "An Eye Tracking Study on camelCase and under_score Identifier Styles." In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: 10.1109/ICPC.2010.41.

[6] Donald Ervin Knuth. *The TEXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[7] Frank Mittelbach. *The `doc` and `shortvrb` Packages*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/doc.pdf (visited on 02/19/2018).

[8] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf (visited on 02/11/2021).

[9] Vít 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).

[10] Geoffrey M. Poore. *The `minted` Package. Highlighted source code in LaTeX*. July 19, 2017. URL: https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf (visited on 09/01/2020).

[11] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[12] Johannes Braams et al. *The LaTeX 2ε Sources*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/source2e.pdf (visited on 01/08/2018).

[13] Donald Ervin Knuth. *TeX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.

[14] 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