A Markdown Interpreter for TEX

Vít Starý Novotný
witiko@mail.muni.cz

Version 3.4.1-0-g457226ae
2024-02-16

Contents

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

2 Interfaces **6**
2.1 Lua Interface 7
2.2 Plain TEX Interface 49
2.3 LATEX Interface 124
2.4 ConTEXt Interface 131

3 Implementation **134**
3.1 Lua Implementation . . . 134
3.2 Plain TEX Implementation 327
3.3 LATEX Implementation . . 346
3.4 ConTEXt Implementation 376

References **384**

Index **385**

List of Figures

1 A block diagram of the Markdown package 7
2 A sequence diagram of typesetting a document using the TEX interface . . 45
3 A sequence diagram of typesetting a document using the Lua CLI 46
4 Various formats of mathemathical formulae 129
5 The banner of the Markdown package 130

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

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the 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",
    copyright = {"2009-2016 John MacFarlane, Hans Hagen",
                 "2016-2023 Vít Starý Novotný"},
    license   = "LPPL 1.3c"
}

if not modules then modules = { } end
modules['markdown'] = metadata
```

1.1 Requirements

This section gives an overview of all resources required by the package.

1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaTEX engine (though not necessarily in the LuaMetaTEX engine).

LPeg $\geqslant$ 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg $\geqslant$ 0.10 is included in LuaTEX $\geqslant$ 0.72.0 (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");
```

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

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

```
(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.

```
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.

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.

```
⟨/tex⟩
⟨*context⟩
\unprotect
⟨/context⟩
⟨*context, tex⟩
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic
\fi
⟨/context, tex⟩
⟨*tex⟩
```

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

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

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

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

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

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

1.1.3 LaTeX Requirements

The LATEX part of the package requires that the LATEX 2$_\varepsilon$ format is loaded,

```
34 \NeedsTeXFormat{LaTeX2e}%
```

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

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.5) 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.

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

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

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` LaTeX theme (see Section 2.3.3).

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

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

gobble A package that provides the `\@gobblethree` TeX command that is used in the default renderer prototype for citations. The package is included in TeXLive $\geqslant$ 2016.

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

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` LaTeX theme, see Section 2.3.3.

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

grffile A package that extends the name processing of the graphics package to support a larger range of file names in 2006 $\leqslant$ TeX Live $\leqslant$ 2019. Since TeX Live $\geqslant$ 2020, the functionality of the package has been integrated in the LaTeX 2_ε kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` LaTeX themes, see Section 2.3.3.

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.5.8, and also in the default renderer prototype for identifier attributes.

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

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

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

```
35 \RequirePackage{expl3}
```

1.1.4 ConTEXt Prerequisites

The ConTEXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain TEX prerequisites (see Section 1.1.2), and the following ConTEXt modules:

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the CSV filename extension (see Section 2.2.6).

1.2 Feedback

Please use the Markdown project page on GitHub[4] to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the TEX-LATEX Stack Exchange.[5] community question answering web site under the `markdown` tag.

1.3 Acknowledgements

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

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

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

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

The TEX implementation of the package draws inspiration from several sources including the source code of LATEX 2_ε, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from TEX, the filecontents package by Scott Pakin and others.

2 Interfaces

This part of the documentation describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package.

Since neither TEX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of

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

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.

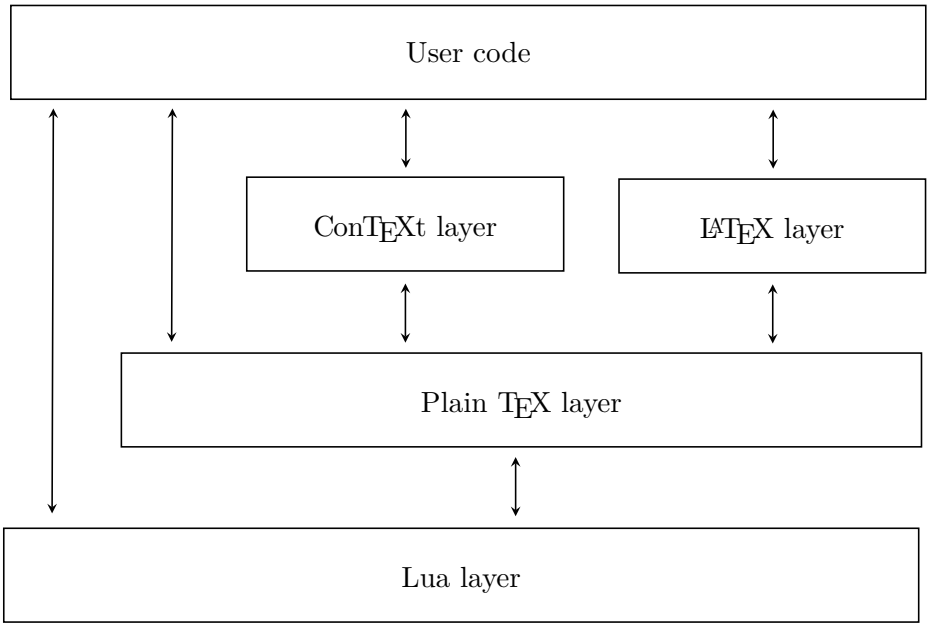

Figure 1: A block diagram of the Markdown package

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain TEX. This interface is used by the plain TEX implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

The Lua interface is implemented by the `markdown` Lua module.

```
36 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain TEX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain TEX according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The

`options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

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

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

2.1.2 User-Defined Syntax Extensions

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

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

```
37 local walkable_syntax = {
38   Block = {
39     "Blockquote",
40     "Verbatim",
41     "ThematicBreak",
42     "BulletList",
43     "OrderedList",
44     "DisplayHtml",
45     "Heading",
46   },
47   BlockOrParagraph = {
48     "Block",
49     "Paragraph",
50     "Plain",
51   },
52   Inline = {
53     "Str",
54     "Space",
55     "Endline",
56     "EndlineBreak",
57     "LinkAndEmph",
58     "Code",
59     "AutoLinkUrl",
60     "AutoLinkEmail",
61     "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: `false`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also

produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

This behavior will always be used if the `finalizeCache` option is enabled.

`false` Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

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

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

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

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

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

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

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

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

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

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 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied.

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

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

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

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

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

2.1.6 Parser Options

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

`true` Enable the Pandoc auto identifiers syntax extension[7]:

```
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.

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

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

`blankBeforeCodeFence`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following fenced code block.

`false` Do not require a blank line between a paragraph and the following fenced code block.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`true` Enable the Pandoc bracketed span syntax extension[8]:

```
[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
```

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

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

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

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

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

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

`true` Enable the Pandoc citation syntax extension[9]:

```
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
```

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

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

`true` Enable the code span syntax:

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

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

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

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

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

true

: Enable the iA Writer content blocks syntax extension [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 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

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

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

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

`true` Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with *inline markup*
```

```
:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

`false` Disable the pandoc definition list syntax extension.

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

`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[10]:

```
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
```

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

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

`true` Enable the Pandoc fenced code attribute syntax extension[11]:

```
~~~~ {#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[12]:

```
::::: {#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
```

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

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

# 123 Sesame Street
```

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

[13] 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 interpretted by TEX. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

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

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

`true` Enable the Pandoc inline code span attribute extension[14]:

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

[14]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[15]:

```
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[16] 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.
---
```

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

```
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[18]:

```
| 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
```

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

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

`true` Enable the Pandoc mark syntax extension[19]:

```
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[20]:

```
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
```

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

```
`$H_2 O$`{=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 \\
```

[21] 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[22] 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
```

[22] 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[23]:

```
This ~~is deleted text.~~
```

`false` Disable the Pandoc strike-through syntax extension.

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

[23] 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[24]:

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

`false` Disable the Pandoc subscript syntax extension.

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

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

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

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

```
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
```

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

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

true

: Enable the Pandoc table caption syntax extension[26] 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[27]:

```
- [ ] 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
```

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

```
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
```

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

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

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

```
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[30]:

```
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:

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

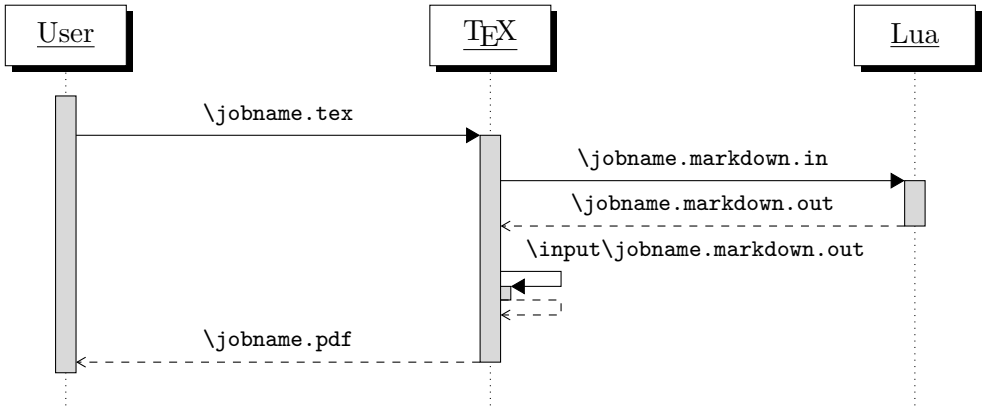

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

```
665
666 local HELP_STRING = [[
667 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
668 where OPTIONS are documented in the Lua interface section of the
669 technical Markdown package documentation.
670
671 When OUTPUT_FILE is unspecified, the result of the conversion will be
672 written to the standard output. When INPUT_FILE is also unspecified, the
673 result of the conversion will be read from the standard input.
674
675 Report bugs to: witiko@mail.muni.cz
676 Markdown package home page: <https://github.com/witiko/markdown>]]
677
```

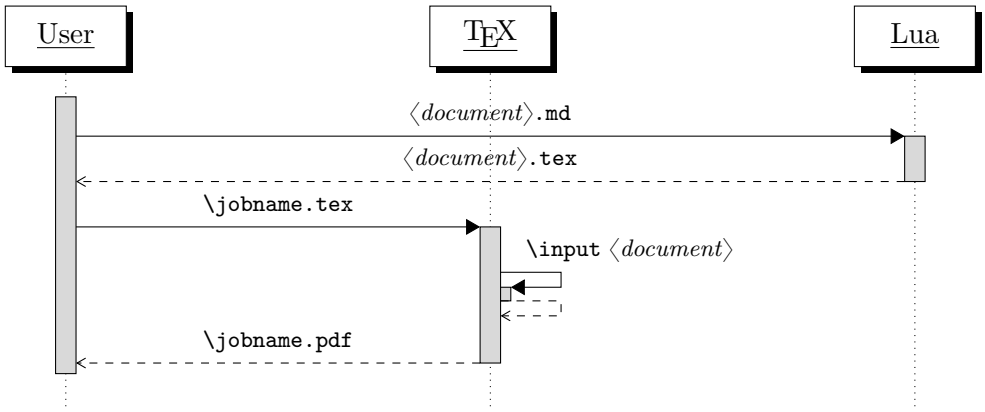

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

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

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

License: ]] .. metadata.license

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

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

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

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

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

```
  end)
  return cased_option_name
end

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

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

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
\input hello
```

2.2 Plain TEX Interface

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

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

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

```
\input markdown
```

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

2.2.1 Typesetting Markdown

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

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

```
773 \let\markdownBegin\relax
774 \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 \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.

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

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

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

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

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

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

2.2.2 Options

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

To 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.

```
\cs_generate_variant:Nn
  \@@_add_plain_tex_option:nnn
  { nnV }
```

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.

```
\ExplSyntaxOff
\input lt3luabridge.tex
\ExplSyntaxOn
\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:
    )
  }
  {
    \luabridge_tl_set:Nn
      \l_tmpa_tl
      { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
  }
  {
    \tl_set:Nn
      \l_tmpa_tl
      { . }
  }
\@@_add_plain_tex_option:nnV
  { outputDir }
  { path }
  \l_tmpa_tl
```

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 Section 3.2.5) 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
  {
```

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
      }
  }
\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
      }
    \prop_gput:NVn
      \g_@@_snippets_prop
      \l_tmpa_tl
      { #2 }
  }
\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
          \@@_setup:V
            \l_tmpb_tl
        }
        {
          \msg_error:nnV
            { markdown }
            { undefined-snippet }
            \l_tmpa_tl
        }
    }
  }
\msg_new:nnn
  { markdown }
  { undefined-snippet }
  { Can't~invoke~undefined~snippet~#1 }
\cs_generate_variant:Nn
  \@@_setup:n
  { V }
\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 contaning the code block contents.

```
\def\markdownRendererInputVerbatim{%
  \markdownRendererInputVerbatimPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputVerbatim }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputVerbatim }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives three arguments that correspond to the filename of a file contaning 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[31] contains a record (k, v), then a non-online-image content block with the filename extension s, `s:lower()` $= k$ is considered to be in a known programming language v. The macro receives five arguments: the local file name extension s cast to the lower case, the language v, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

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

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

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

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

```
  { ellipsis }
  { 0 }
\ExplSyntaxOff
```

2.2.5.12 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` option is enabled.

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

```
\def\markdownRendererFencedCodeAttributeContextBegin{%
  \markdownRendererFencedCodeAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
```

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

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

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

2.2.5.16 Heading Renderers

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.5.17 Inline HTML Comment Renderer

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

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

2.2.5.18 HTML Tag and Element Renderers

The \markdownRendererInlineHtmlTag macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the html option is

enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

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

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

2.2.5.19 Image Renderer

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

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

2.2.5.20 Image Attribute Context Renderers

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

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

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

2.2.5.21 Interblock Separator Renderers

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

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

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

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

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

2.2.5.22 Line Block Renderers

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

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

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

2.2.5.23 Line Break Renderers

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

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

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

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

2.2.5.24 Link Renderer

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

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

2.2.5.25 Link Attribute Context Renderers

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

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

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

2.2.5.26 Marked Text Renderer

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

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

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

2.2.5.27 Markdown Document Renderers

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

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

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

2.2.5.28 Non-Breaking Space Renderer

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

```
\def\markdownRendererNbsp{%
  \markdownRendererNbspPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { nbsp }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { nbsp }
  { 0 }
\ExplSyntaxOff
```

2.2.5.29 Note Renderer

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

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

2.2.5.30 Ordered List Renderers

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

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

The `\markdownRendererOlBeginTight` macro represents the beginning of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

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

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

The \markdownRendererFancyOlBegin macro represents the beginning of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the fancyLists option is enabled. The macro receives two arguments: the style of the list item labels (Decimal, LowerRoman, UpperRoman, LowerAlpha, and UpperAlpha), and the style of delimiters between list item labels and texts (Default, OneParen, and Period).

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

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

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

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

```
\def\markdownRendererOlItem{%
```

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

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

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

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

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

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

```
\def\markdownRendererFancyOlItem{%
```

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

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

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

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

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

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

```
\def\markdownRendererOlEnd{%
  \markdownRendererOlEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olEnd }
  { 0 }
\ExplSyntaxOff
```

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

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

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

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

The `\markdownRendererFancyOlEndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight).

This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```
\def\markdownRendererFancyOlEndTight{%
  \markdownRendererFancyOlEndTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.5.31 Raw Content Renderers

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

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

The `\markdownRendererInputRawBlock` macro represents a raw block. The macro receives two arguments: the filename of a file contaning the raw block and the raw attribute that designates the format of the raw block. This macro will only be produced, when the `rawAttribute` and `fencedCode` options are enabled.

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

```
\ExplSyntaxOff
```

2.2.5.32 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 YAML Metadata Renderers

The \markdownRendererJekyllDataBegin macro represents the beginning of a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives no arguments.

```
\def\markdownRendererJekyllDataBegin{%
  \markdownRendererJekyllDataBeginPrototype}%
```

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

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

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

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

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

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

```
\def\markdownRendererJekyllDataMappingEnd{%
```

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

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

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

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

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

The `\markdownRendererJekyllDataBoolean` macro represents a boolean scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent

structure, and the scalar value, both cast to a string following YAML serialization rules.

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

The `\markdownRendererJekyllDataNumber` macro represents a numeric scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```
\def\markdownRendererJekyllDataNumber{%
  \markdownRendererJekyllDataNumberPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataNumber }
  { 2 }
\ExplSyntaxOff
```

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

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

The `\markdownRendererJekyllDataEmpty` macro represents an empty scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives one argument: the scalar key in the parent structure, cast to a string following YAML serialization rules.

See also Section 2.2.6.1 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
```

2.2.5.44 Generating Plain TEX Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain TEX macros for token renderers. Futhermore, 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_function:NN
      \g_@@_renderers_seq
      \@@_define_renderer:n
  }
\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
\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
          \cs_generate_from_arg_count:NNnV
            #2
            \cs_set:Npn
            { #3 }
            \l_@@_renderer_definition_tl
        },
      }
  }
\cs_generate_variant:Nn
  \@@_define_renderer:nNn
  { ncV }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { NNnV }
\keys_define:nn
  { markdown/options }
  {
    renderers .code:n = {
      \keys_set:nn
```

```
        { markdown/options/renderers }
        { #1 }
    },
  }
\ExplSyntaxOff
```

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.
  }
}
```

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 parameter `#0`:

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

```
\ExplSyntaxOn
\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 }
\seq_new:N
  \l_@@_renderer_glob_results_seq
\keys_define:nn
  { markdown/options/renderers }
  {
    unknown .code:n = {
      \@@_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 }
            \l_keys_key_str
        }
        {
          \tl_set:Nn
            \l_@@_renderer_definition_tl
            { #1 }
          \seq_map_inline:Nn
            \l_@@_renderer_glob_results_seq
            {
              \@@_renderer_tl_to_csname:nN
                { ##1 }
                \l_tmpa_tl
              \prop_get:NnN
                \g_@@_renderer_arities_prop
                { ##1 }
                \l_tmpb_tl
```

```
              \int_set:Nn
                \l_tmpa_int
                \l_tmpb_tl
              \tl_set:NV
                \l_tmpb_tl
                \l_@@_renderer_definition_tl
              \regex_replace_all:nnN
                { \cP\#0 }
                { ##1 }
                \l_tmpb_tl
              \cs_generate_from_arg_count:cNVV
                { \l_tmpa_tl }
                \cs_set:Npn
                \l_tmpa_int
                \l_tmpb_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 }
```

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. Futhermore, 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_function:NN
      \g_@@_renderers_seq
      \@@_define_renderer_prototype:n
  }
\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
\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
          \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, we provide an empty definition.

```
    \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 }
\ExplSyntaxOff
```

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.
  }
}
```

In addition to exact token renderer 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 parameter `#0`:

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

```
\ExplSyntaxOn
\seq_new:N
  \l_@@_renderer_prototype_glob_results_seq
\keys_define:nn
  { markdown/options/renderer-prototypes }
  {
    unknown .code:n = {
      \@@_glob_seq:VnN
        \l_keys_key_str
        { g_@@_renderers_seq }
        \l_@@_renderer_prototype_glob_results_seq
      \seq_if_empty:NTF
        \l_@@_renderer_prototype_glob_results_seq
        {
          \msg_error:nnV
            { markdown }
            { undefined-renderer-prototype }
            \l_keys_key_str
        }
        {
          \tl_set:Nn
            \l_@@_renderer_prototype_definition_tl
            { #1 }
          \seq_map_inline:Nn
            \l_@@_renderer_prototype_glob_results_seq
            {
              \@@_renderer_prototype_tl_to_csname:nN
                { ##1 }
                \l_tmpa_tl
              \prop_get:NnN
                \g_@@_renderer_arities_prop
                { ##1 }
                \l_tmpb_tl
              \int_set:Nn
                \l_tmpa_int
                \l_tmpb_tl
              \tl_set:NV
                \l_tmpb_tl
                \l_@@_renderer_prototype_definition_tl
```

```
              \regex_replace_all:nnN
                { \cP\#0 }
                { ##1 }
                \l_tmpb_tl
              \cs_generate_from_arg_count:cNVV
                { \l_tmpa_tl }
                \cs_set:Npn
                \l_tmpa_int
                \l_tmpb_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
\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 2.2.5.44) 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_ε parses package options.

2.3.1 Typesetting Markdown

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

The `markdown` and `markdown*` LaTeX environments are used to typeset markdown document fragments. Both LaTeX environments accept LaTeX interface options (see ection 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.

```
3113 \newenvironment{markdown}\relax\relax
3114 \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]
_Hello_ **world** ...
\end{markdown}
% ...
\end{document}
```

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

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

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

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

2.3.2 Options

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

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 [9, 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.

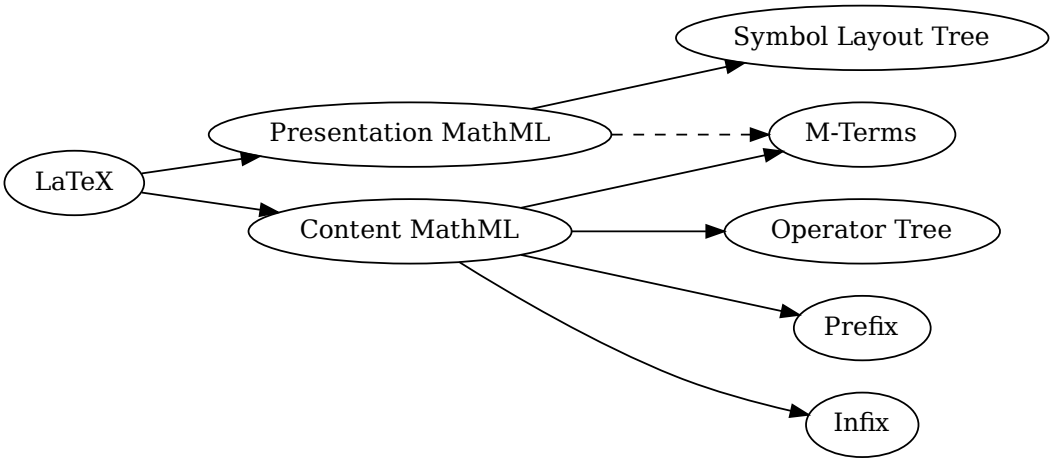

Figure 4: Various formats of mathemathical formulae

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.

```
3129 \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 theme requires the catchfile LATEX package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The

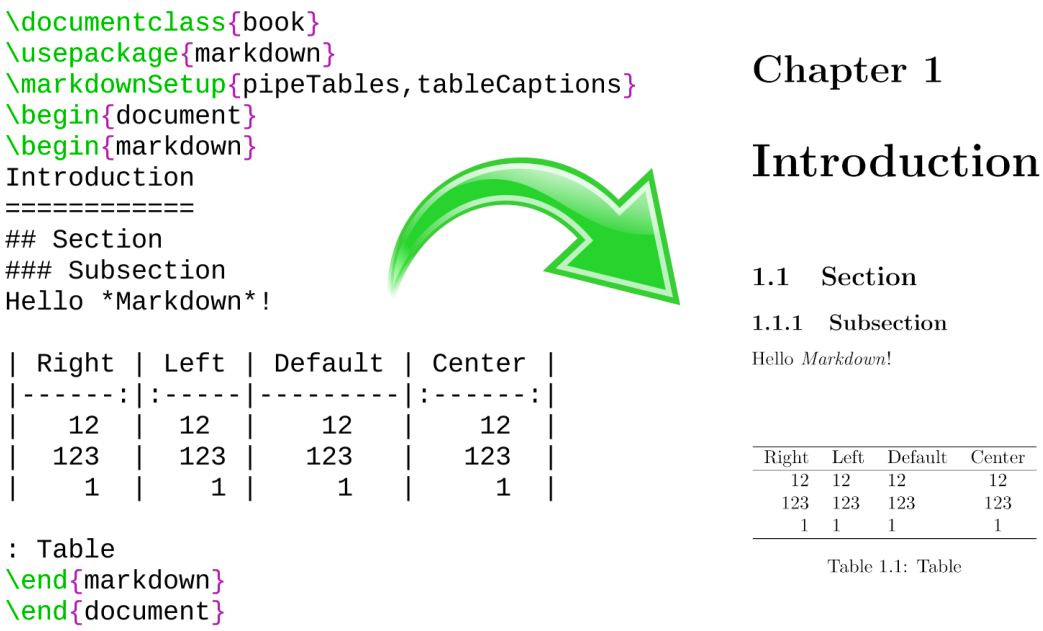

Figure 5: The banner of the Markdown package

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 3.3.4 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}.

```
3159 \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 define plain TEX option, token renderer, and token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

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

2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConTEXt, we expand on the concept of themes by allowing a theme to be a full-blown ConTEXt module. Specifically, the key-values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩ load

a ConTEXt module named `t-markdowntheme`⟨*munged theme name*⟩`.tex` if it exists and a TEX document named `markdowntheme`⟨*munged theme name*⟩`.tex` otherwise.

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

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

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

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

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

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

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

3 Implementation

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

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TEX *token renderers* is performed by the Lua layer. The plain TEX layer provides default definitions for the token renderers. The LATEX and ConTEXt layers correct idiosyncrasies of the respective TEX formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain TEX, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module

and the remaining markdown reader and plain TEX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
local upper, format, length =
  string.upper, string.format, string.len
local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
  lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
  lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the plain TEX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
local util = {}
```

The `util.err` method prints an error message `msg` and exits. If `exit_code` is provided, it specifies the exit code. Otherwise, the exit code will be 1.

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

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

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

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

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

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

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

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

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

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

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

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [10, Chapter 21].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Build a string of escapable characters.

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
local entities = {}

```

```
local character_entities = {
  ["Tab"] = 9,
  ["NewLine"] = 10,
  ["excl"] = 33,
  ["QUOT"] = 34,
  ["quot"] = 34,
  ["num"] = 35,
  ["dollar"] = 36,
  ["percnt"] = 37,
  ["AMP"] = 38,
  ["amp"] = 38,
  ["apos"] = 39,
  ["lpar"] = 40,
  ["rpar"] = 41,
  ["ast"] = 42,
  ["midast"] = 42,
  ["plus"] = 43,
  ["comma"] = 44,
  ["period"] = 46,
  ["sol"] = 47,
  ["colon"] = 58,
  ["semi"] = 59,
  ["LT"] = 60,
  ["lt"] = 60,
  ["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,
```

```
3386   ["vert"] = 124,
3387   ["rbrace"] = 125,
3388   ["rcub"] = 125,
3389   ["NonBreakingSpace"] = 160,
3390   ["nbsp"] = 160,
3391   ["iexcl"] = 161,
3392   ["cent"] = 162,
3393   ["pound"] = 163,
3394   ["curren"] = 164,
3395   ["yen"] = 165,
3396   ["brvbar"] = 166,
3397   ["sect"] = 167,
3398   ["Dot"] = 168,
3399   ["DoubleDot"] = 168,
3400   ["die"] = 168,
3401   ["uml"] = 168,
3402   ["COPY"] = 169,
3403   ["copy"] = 169,
3404   ["ordf"] = 170,
3405   ["laquo"] = 171,
3406   ["not"] = 172,
3407   ["shy"] = 173,
3408   ["REG"] = 174,
3409   ["circledR"] = 174,
3410   ["reg"] = 174,
3411   ["macr"] = 175,
3412   ["strns"] = 175,
3413   ["deg"] = 176,
3414   ["PlusMinus"] = 177,
3415   ["plusmn"] = 177,
3416   ["pm"] = 177,
3417   ["sup2"] = 178,
3418   ["sup3"] = 179,
3419   ["DiacriticalAcute"] = 180,
3420   ["acute"] = 180,
3421   ["micro"] = 181,
3422   ["para"] = 182,
3423   ["CenterDot"] = 183,
3424   ["centerdot"] = 183,
3425   ["middot"] = 183,
3426   ["Cedilla"] = 184,
3427   ["cedil"] = 184,
3428   ["sup1"] = 185,
3429   ["ordm"] = 186,
3430   ["raquo"] = 187,
3431   ["frac14"] = 188,
3432   ["frac12"] = 189,
```

```
3433   ["half"] = 189,
3434   ["frac34"] = 190,
3435   ["iquest"] = 191,
3436   ["Agrave"] = 192,
3437   ["Aacute"] = 193,
3438   ["Acirc"] = 194,
3439   ["Atilde"] = 195,
3440   ["Auml"] = 196,
3441   ["Aring"] = 197,
3442   ["angst"] = 197,
3443   ["AElig"] = 198,
3444   ["Ccedil"] = 199,
3445   ["Egrave"] = 200,
3446   ["Eacute"] = 201,
3447   ["Ecirc"] = 202,
3448   ["Euml"] = 203,
3449   ["Igrave"] = 204,
3450   ["Iacute"] = 205,
3451   ["Icirc"] = 206,
3452   ["Iuml"] = 207,
3453   ["ETH"] = 208,
3454   ["Ntilde"] = 209,
3455   ["Ograve"] = 210,
3456   ["Oacute"] = 211,
3457   ["Ocirc"] = 212,
3458   ["Otilde"] = 213,
3459   ["Ouml"] = 214,
3460   ["times"] = 215,
3461   ["Oslash"] = 216,
3462   ["Ugrave"] = 217,
3463   ["Uacute"] = 218,
3464   ["Ucirc"] = 219,
3465   ["Uuml"] = 220,
3466   ["Yacute"] = 221,
3467   ["THORN"] = 222,
3468   ["szlig"] = 223,
3469   ["agrave"] = 224,
3470   ["aacute"] = 225,
3471   ["acirc"] = 226,
3472   ["atilde"] = 227,
3473   ["auml"] = 228,
3474   ["aring"] = 229,
3475   ["aelig"] = 230,
3476   ["ccedil"] = 231,
3477   ["egrave"] = 232,
3478   ["eacute"] = 233,
3479   ["ecirc"] = 234,
```

```
3480   ["euml"] = 235,
3481   ["igrave"] = 236,
3482   ["iacute"] = 237,
3483   ["icirc"] = 238,
3484   ["iuml"] = 239,
3485   ["eth"] = 240,
3486   ["ntilde"] = 241,
3487   ["ograve"] = 242,
3488   ["oacute"] = 243,
3489   ["ocirc"] = 244,
3490   ["otilde"] = 245,
3491   ["ouml"] = 246,
3492   ["div"] = 247,
3493   ["divide"] = 247,
3494   ["oslash"] = 248,
3495   ["ugrave"] = 249,
3496   ["uacute"] = 250,
3497   ["ucirc"] = 251,
3498   ["uuml"] = 252,
3499   ["yacute"] = 253,
3500   ["thorn"] = 254,
3501   ["yuml"] = 255,
3502   ["Amacr"] = 256,
3503   ["amacr"] = 257,
3504   ["Abreve"] = 258,
3505   ["abreve"] = 259,
3506   ["Aogon"] = 260,
3507   ["aogon"] = 261,
3508   ["Cacute"] = 262,
3509   ["cacute"] = 263,
3510   ["Ccirc"] = 264,
3511   ["ccirc"] = 265,
3512   ["Cdot"] = 266,
3513   ["cdot"] = 267,
3514   ["Ccaron"] = 268,
3515   ["ccaron"] = 269,
3516   ["Dcaron"] = 270,
3517   ["dcaron"] = 271,
3518   ["Dstrok"] = 272,
3519   ["dstrok"] = 273,
3520   ["Emacr"] = 274,
3521   ["emacr"] = 275,
3522   ["Edot"] = 278,
3523   ["edot"] = 279,
3524   ["Eogon"] = 280,
3525   ["eogon"] = 281,
3526   ["Ecaron"] = 282,
```

```
3527   ["ecaron"] = 283,
3528   ["Gcirc"] = 284,
3529   ["gcirc"] = 285,
3530   ["Gbreve"] = 286,
3531   ["gbreve"] = 287,
3532   ["Gdot"] = 288,
3533   ["gdot"] = 289,
3534   ["Gcedil"] = 290,
3535   ["Hcirc"] = 292,
3536   ["hcirc"] = 293,
3537   ["Hstrok"] = 294,
3538   ["hstrok"] = 295,
3539   ["Itilde"] = 296,
3540   ["itilde"] = 297,
3541   ["Imacr"] = 298,
3542   ["imacr"] = 299,
3543   ["Iogon"] = 302,
3544   ["iogon"] = 303,
3545   ["Idot"] = 304,
3546   ["imath"] = 305,
3547   ["inodot"] = 305,
3548   ["IJlig"] = 306,
3549   ["ijlig"] = 307,
3550   ["Jcirc"] = 308,
3551   ["jcirc"] = 309,
3552   ["Kcedil"] = 310,
3553   ["kcedil"] = 311,
3554   ["kgreen"] = 312,
3555   ["Lacute"] = 313,
3556   ["lacute"] = 314,
3557   ["Lcedil"] = 315,
3558   ["lcedil"] = 316,
3559   ["Lcaron"] = 317,
3560   ["lcaron"] = 318,
3561   ["Lmidot"] = 319,
3562   ["lmidot"] = 320,
3563   ["Lstrok"] = 321,
3564   ["lstrok"] = 322,
3565   ["Nacute"] = 323,
3566   ["nacute"] = 324,
3567   ["Ncedil"] = 325,
3568   ["ncedil"] = 326,
3569   ["Ncaron"] = 327,
3570   ["ncaron"] = 328,
3571   ["napos"] = 329,
3572   ["ENG"] = 330,
3573   ["eng"] = 331,
```

```
3574   ["Omacr"] = 332,
3575   ["omacr"] = 333,
3576   ["Odblac"] = 336,
3577   ["odblac"] = 337,
3578   ["OElig"] = 338,
3579   ["oelig"] = 339,
3580   ["Racute"] = 340,
3581   ["racute"] = 341,
3582   ["Rcedil"] = 342,
3583   ["rcedil"] = 343,
3584   ["Rcaron"] = 344,
3585   ["rcaron"] = 345,
3586   ["Sacute"] = 346,
3587   ["sacute"] = 347,
3588   ["Scirc"] = 348,
3589   ["scirc"] = 349,
3590   ["Scedil"] = 350,
3591   ["scedil"] = 351,
3592   ["Scaron"] = 352,
3593   ["scaron"] = 353,
3594   ["Tcedil"] = 354,
3595   ["tcedil"] = 355,
3596   ["Tcaron"] = 356,
3597   ["tcaron"] = 357,
3598   ["Tstrok"] = 358,
3599   ["tstrok"] = 359,
3600   ["Utilde"] = 360,
3601   ["utilde"] = 361,
3602   ["Umacr"] = 362,
3603   ["umacr"] = 363,
3604   ["Ubreve"] = 364,
3605   ["ubreve"] = 365,
3606   ["Uring"] = 366,
3607   ["uring"] = 367,
3608   ["Udblac"] = 368,
3609   ["udblac"] = 369,
3610   ["Uogon"] = 370,
3611   ["uogon"] = 371,
3612   ["Wcirc"] = 372,
3613   ["wcirc"] = 373,
3614   ["Ycirc"] = 374,
3615   ["ycirc"] = 375,
3616   ["Yuml"] = 376,
3617   ["Zacute"] = 377,
3618   ["zacute"] = 378,
3619   ["Zdot"] = 379,
3620   ["zdot"] = 380,
```

```
  ["Zcaron"] = 381,
  ["zcaron"] = 382,
  ["fnof"] = 402,
  ["imped"] = 437,
  ["gacute"] = 501,
  ["jmath"] = 567,
  ["circ"] = 710,
  ["Hacek"] = 711,
  ["caron"] = 711,
  ["Breve"] = 728,
  ["breve"] = 728,
  ["DiacriticalDot"] = 729,
  ["dot"] = 729,
  ["ring"] = 730,
  ["ogon"] = 731,
  ["DiacriticalTilde"] = 732,
  ["tilde"] = 732,
  ["DiacriticalDoubleAcute"] = 733,
  ["dblac"] = 733,
  ["DownBreve"] = 785,
  ["Alpha"] = 913,
  ["Beta"] = 914,
  ["Gamma"] = 915,
  ["Delta"] = 916,
  ["Epsilon"] = 917,
  ["Zeta"] = 918,
  ["Eta"] = 919,
  ["Theta"] = 920,
  ["Iota"] = 921,
  ["Kappa"] = 922,
  ["Lambda"] = 923,
  ["Mu"] = 924,
  ["Nu"] = 925,
  ["Xi"] = 926,
  ["Omicron"] = 927,
  ["Pi"] = 928,
  ["Rho"] = 929,
  ["Sigma"] = 931,
  ["Tau"] = 932,
  ["Upsilon"] = 933,
  ["Phi"] = 934,
  ["Chi"] = 935,
  ["Psi"] = 936,
  ["Omega"] = 937,
  ["ohm"] = 937,
  ["alpha"] = 945,
  ["beta"] = 946,
```

```
3668   ["gamma"] = 947,
3669   ["delta"] = 948,
3670   ["epsi"] = 949,
3671   ["epsilon"] = 949,
3672   ["zeta"] = 950,
3673   ["eta"] = 951,
3674   ["theta"] = 952,
3675   ["iota"] = 953,
3676   ["kappa"] = 954,
3677   ["lambda"] = 955,
3678   ["mu"] = 956,
3679   ["nu"] = 957,
3680   ["xi"] = 958,
3681   ["omicron"] = 959,
3682   ["pi"] = 960,
3683   ["rho"] = 961,
3684   ["sigmaf"] = 962,
3685   ["sigmav"] = 962,
3686   ["varsigma"] = 962,
3687   ["sigma"] = 963,
3688   ["tau"] = 964,
3689   ["upsi"] = 965,
3690   ["upsilon"] = 965,
3691   ["phi"] = 966,
3692   ["chi"] = 967,
3693   ["psi"] = 968,
3694   ["omega"] = 969,
3695   ["thetasym"] = 977,
3696   ["thetav"] = 977,
3697   ["vartheta"] = 977,
3698   ["Upsi"] = 978,
3699   ["upsih"] = 978,
3700   ["phiv"] = 981,
3701   ["straightphi"] = 981,
3702   ["varphi"] = 981,
3703   ["piv"] = 982,
3704   ["varpi"] = 982,
3705   ["Gammad"] = 988,
3706   ["digamma"] = 989,
3707   ["gammad"] = 989,
3708   ["kappav"] = 1008,
3709   ["varkappa"] = 1008,
3710   ["rhov"] = 1009,
3711   ["varrho"] = 1009,
3712   ["epsiv"] = 1013,
3713   ["straightepsilon"] = 1013,
3714   ["varepsilon"] = 1013,
```

```
3715   ["backepsilon"] = 1014,
3716   ["bepsi"] = 1014,
3717   ["IOcy"] = 1025,
3718   ["DJcy"] = 1026,
3719   ["GJcy"] = 1027,
3720   ["Jukcy"] = 1028,
3721   ["DScy"] = 1029,
3722   ["Iukcy"] = 1030,
3723   ["YIcy"] = 1031,
3724   ["Jsercy"] = 1032,
3725   ["LJcy"] = 1033,
3726   ["NJcy"] = 1034,
3727   ["TSHcy"] = 1035,
3728   ["KJcy"] = 1036,
3729   ["Ubrcy"] = 1038,
3730   ["DZcy"] = 1039,
3731   ["Acy"] = 1040,
3732   ["Bcy"] = 1041,
3733   ["Vcy"] = 1042,
3734   ["Gcy"] = 1043,
3735   ["Dcy"] = 1044,
3736   ["IEcy"] = 1045,
3737   ["ZHcy"] = 1046,
3738   ["Zcy"] = 1047,
3739   ["Icy"] = 1048,
3740   ["Jcy"] = 1049,
3741   ["Kcy"] = 1050,
3742   ["Lcy"] = 1051,
3743   ["Mcy"] = 1052,
3744   ["Ncy"] = 1053,
3745   ["Ocy"] = 1054,
3746   ["Pcy"] = 1055,
3747   ["Rcy"] = 1056,
3748   ["Scy"] = 1057,
3749   ["Tcy"] = 1058,
3750   ["Ucy"] = 1059,
3751   ["Fcy"] = 1060,
3752   ["KHcy"] = 1061,
3753   ["TScy"] = 1062,
3754   ["CHcy"] = 1063,
3755   ["SHcy"] = 1064,
3756   ["SHCHcy"] = 1065,
3757   ["HARDcy"] = 1066,
3758   ["Ycy"] = 1067,
3759   ["SOFTcy"] = 1068,
3760   ["Ecy"] = 1069,
3761   ["YUcy"] = 1070,
```

```
3762    ["YAcy"] = 1071,
3763    ["acy"] = 1072,
3764    ["bcy"] = 1073,
3765    ["vcy"] = 1074,
3766    ["gcy"] = 1075,
3767    ["dcy"] = 1076,
3768    ["iecy"] = 1077,
3769    ["zhcy"] = 1078,
3770    ["zcy"] = 1079,
3771    ["icy"] = 1080,
3772    ["jcy"] = 1081,
3773    ["kcy"] = 1082,
3774    ["lcy"] = 1083,
3775    ["mcy"] = 1084,
3776    ["ncy"] = 1085,
3777    ["ocy"] = 1086,
3778    ["pcy"] = 1087,
3779    ["rcy"] = 1088,
3780    ["scy"] = 1089,
3781    ["tcy"] = 1090,
3782    ["ucy"] = 1091,
3783    ["fcy"] = 1092,
3784    ["khcy"] = 1093,
3785    ["tscy"] = 1094,
3786    ["chcy"] = 1095,
3787    ["shcy"] = 1096,
3788    ["shchcy"] = 1097,
3789    ["hardcy"] = 1098,
3790    ["ycy"] = 1099,
3791    ["softcy"] = 1100,
3792    ["ecy"] = 1101,
3793    ["yucy"] = 1102,
3794    ["yacy"] = 1103,
3795    ["iocy"] = 1105,
3796    ["djcy"] = 1106,
3797    ["gjcy"] = 1107,
3798    ["jukcy"] = 1108,
3799    ["dscy"] = 1109,
3800    ["iukcy"] = 1110,
3801    ["yicy"] = 1111,
3802    ["jsercy"] = 1112,
3803    ["ljcy"] = 1113,
3804    ["njcy"] = 1114,
3805    ["tshcy"] = 1115,
3806    ["kjcy"] = 1116,
3807    ["ubrcy"] = 1118,
3808    ["dzcy"] = 1119,
```

```
3809   ["ensp"] = 8194,
3810   ["emsp"] = 8195,
3811   ["emsp13"] = 8196,
3812   ["emsp14"] = 8197,
3813   ["numsp"] = 8199,
3814   ["puncsp"] = 8200,
3815   ["ThinSpace"] = 8201,
3816   ["thinsp"] = 8201,
3817   ["VeryThinSpace"] = 8202,
3818   ["hairsp"] = 8202,
3819   ["NegativeMediumSpace"] = 8203,
3820   ["NegativeThickSpace"] = 8203,
3821   ["NegativeThinSpace"] = 8203,
3822   ["NegativeVeryThinSpace"] = 8203,
3823   ["ZeroWidthSpace"] = 8203,
3824   ["zwnj"] = 8204,
3825   ["zwj"] = 8205,
3826   ["lrm"] = 8206,
3827   ["rlm"] = 8207,
3828   ["dash"] = 8208,
3829   ["hyphen"] = 8208,
3830   ["ndash"] = 8211,
3831   ["mdash"] = 8212,
3832   ["horbar"] = 8213,
3833   ["Verbar"] = 8214,
3834   ["Vert"] = 8214,
3835   ["OpenCurlyQuote"] = 8216,
3836   ["lsquo"] = 8216,
3837   ["CloseCurlyQuote"] = 8217,
3838   ["rsquo"] = 8217,
3839   ["rsquor"] = 8217,
3840   ["lsquor"] = 8218,
3841   ["sbquo"] = 8218,
3842   ["OpenCurlyDoubleQuote"] = 8220,
3843   ["ldquo"] = 8220,
3844   ["CloseCurlyDoubleQuote"] = 8221,
3845   ["rdquo"] = 8221,
3846   ["rdquor"] = 8221,
3847   ["bdquo"] = 8222,
3848   ["ldquor"] = 8222,
3849   ["dagger"] = 8224,
3850   ["Dagger"] = 8225,
3851   ["ddagger"] = 8225,
3852   ["bull"] = 8226,
3853   ["bullet"] = 8226,
3854   ["nldr"] = 8229,
3855   ["hellip"] = 8230,
```

```
3856   ["mldr"] = 8230,
3857   ["permil"] = 8240,
3858   ["pertenk"] = 8241,
3859   ["prime"] = 8242,
3860   ["Prime"] = 8243,
3861   ["tprime"] = 8244,
3862   ["backprime"] = 8245,
3863   ["bprime"] = 8245,
3864   ["lsaquo"] = 8249,
3865   ["rsaquo"] = 8250,
3866   ["OverBar"] = 8254,
3867   ["oline"] = 8254,
3868   ["caret"] = 8257,
3869   ["hybull"] = 8259,
3870   ["frasl"] = 8260,
3871   ["bsemi"] = 8271,
3872   ["qprime"] = 8279,
3873   ["MediumSpace"] = 8287,
3874   ["ThickSpace"] = {8287, 8202},
3875   ["NoBreak"] = 8288,
3876   ["ApplyFunction"] = 8289,
3877   ["af"] = 8289,
3878   ["InvisibleTimes"] = 8290,
3879   ["it"] = 8290,
3880   ["InvisibleComma"] = 8291,
3881   ["ic"] = 8291,
3882   ["euro"] = 8364,
3883   ["TripleDot"] = 8411,
3884   ["tdot"] = 8411,
3885   ["DotDot"] = 8412,
3886   ["Copf"] = 8450,
3887   ["complexes"] = 8450,
3888   ["incare"] = 8453,
3889   ["gscr"] = 8458,
3890   ["HilbertSpace"] = 8459,
3891   ["Hscr"] = 8459,
3892   ["hamilt"] = 8459,
3893   ["Hfr"] = 8460,
3894   ["Poincareplane"] = 8460,
3895   ["Hopf"] = 8461,
3896   ["quaternions"] = 8461,
3897   ["planckh"] = 8462,
3898   ["hbar"] = 8463,
3899   ["hslash"] = 8463,
3900   ["planck"] = 8463,
3901   ["plankv"] = 8463,
3902   ["Iscr"] = 8464,
```

```
["imagline"] = 8464,
["Ifr"] = 8465,
["Im"] = 8465,
["image"] = 8465,
["imagpart"] = 8465,
["Laplacetrf"] = 8466,
["Lscr"] = 8466,
["lagran"] = 8466,
["ell"] = 8467,
["Nopf"] = 8469,
["naturals"] = 8469,
["numero"] = 8470,
["copysr"] = 8471,
["weierp"] = 8472,
["wp"] = 8472,
["Popf"] = 8473,
["primes"] = 8473,
["Qopf"] = 8474,
["rationals"] = 8474,
["Rscr"] = 8475,
["realine"] = 8475,
["Re"] = 8476,
["Rfr"] = 8476,
["real"] = 8476,
["realpart"] = 8476,
["Ropf"] = 8477,
["reals"] = 8477,
["rx"] = 8478,
["TRADE"] = 8482,
["trade"] = 8482,
["Zopf"] = 8484,
["integers"] = 8484,
["mho"] = 8487,
["Zfr"] = 8488,
["zeetrf"] = 8488,
["iiota"] = 8489,
["Bernoullis"] = 8492,
["Bscr"] = 8492,
["bernou"] = 8492,
["Cayleys"] = 8493,
["Cfr"] = 8493,
["escr"] = 8495,
["Escr"] = 8496,
["expectation"] = 8496,
["Fouriertrf"] = 8497,
["Fscr"] = 8497,
["Mellintrf"] = 8499,
```

```
3950   ["Mscr"] = 8499,
3951   ["phmmat"] = 8499,
3952   ["order"] = 8500,
3953   ["orderof"] = 8500,
3954   ["oscr"] = 8500,
3955   ["alefsym"] = 8501,
3956   ["aleph"] = 8501,
3957   ["beth"] = 8502,
3958   ["gimel"] = 8503,
3959   ["daleth"] = 8504,
3960   ["CapitalDifferentialD"] = 8517,
3961   ["DD"] = 8517,
3962   ["DifferentialD"] = 8518,
3963   ["dd"] = 8518,
3964   ["ExponentialE"] = 8519,
3965   ["ee"] = 8519,
3966   ["exponentiale"] = 8519,
3967   ["ImaginaryI"] = 8520,
3968   ["ii"] = 8520,
3969   ["frac13"] = 8531,
3970   ["frac23"] = 8532,
3971   ["frac15"] = 8533,
3972   ["frac25"] = 8534,
3973   ["frac35"] = 8535,
3974   ["frac45"] = 8536,
3975   ["frac16"] = 8537,
3976   ["frac56"] = 8538,
3977   ["frac18"] = 8539,
3978   ["frac38"] = 8540,
3979   ["frac58"] = 8541,
3980   ["frac78"] = 8542,
3981   ["LeftArrow"] = 8592,
3982   ["ShortLeftArrow"] = 8592,
3983   ["larr"] = 8592,
3984   ["leftarrow"] = 8592,
3985   ["slarr"] = 8592,
3986   ["ShortUpArrow"] = 8593,
3987   ["UpArrow"] = 8593,
3988   ["uarr"] = 8593,
3989   ["uparrow"] = 8593,
3990   ["RightArrow"] = 8594,
3991   ["ShortRightArrow"] = 8594,
3992   ["rarr"] = 8594,
3993   ["rightarrow"] = 8594,
3994   ["srarr"] = 8594,
3995   ["DownArrow"] = 8595,
3996   ["ShortDownArrow"] = 8595,
```

```
3997  ["darr"] = 8595,
3998  ["downarrow"] = 8595,
3999  ["LeftRightArrow"] = 8596,
4000  ["harr"] = 8596,
4001  ["leftrightarrow"] = 8596,
4002  ["UpDownArrow"] = 8597,
4003  ["updownarrow"] = 8597,
4004  ["varr"] = 8597,
4005  ["UpperLeftArrow"] = 8598,
4006  ["nwarr"] = 8598,
4007  ["nwarrow"] = 8598,
4008  ["UpperRightArrow"] = 8599,
4009  ["nearr"] = 8599,
4010  ["nearrow"] = 8599,
4011  ["LowerRightArrow"] = 8600,
4012  ["searr"] = 8600,
4013  ["searrow"] = 8600,
4014  ["LowerLeftArrow"] = 8601,
4015  ["swarr"] = 8601,
4016  ["swarrow"] = 8601,
4017  ["nlarr"] = 8602,
4018  ["nleftarrow"] = 8602,
4019  ["nrarr"] = 8603,
4020  ["nrightarrow"] = 8603,
4021  ["nrarrw"] = {8605, 824},
4022  ["rarrw"] = 8605,
4023  ["rightsquigarrow"] = 8605,
4024  ["Larr"] = 8606,
4025  ["twoheadleftarrow"] = 8606,
4026  ["Uarr"] = 8607,
4027  ["Rarr"] = 8608,
4028  ["twoheadrightarrow"] = 8608,
4029  ["Darr"] = 8609,
4030  ["larrtl"] = 8610,
4031  ["leftarrowtail"] = 8610,
4032  ["rarrtl"] = 8611,
4033  ["rightarrowtail"] = 8611,
4034  ["LeftTeeArrow"] = 8612,
4035  ["mapstoleft"] = 8612,
4036  ["UpTeeArrow"] = 8613,
4037  ["mapstoup"] = 8613,
4038  ["RightTeeArrow"] = 8614,
4039  ["map"] = 8614,
4040  ["mapsto"] = 8614,
4041  ["DownTeeArrow"] = 8615,
4042  ["mapstodown"] = 8615,
4043  ["hookleftarrow"] = 8617,
```

```
4044    ["larrhk"] = 8617,
4045    ["hookrightarrow"] = 8618,
4046    ["rarrhk"] = 8618,
4047    ["larrlp"] = 8619,
4048    ["looparrowleft"] = 8619,
4049    ["looparrowright"] = 8620,
4050    ["rarrlp"] = 8620,
4051    ["harrw"] = 8621,
4052    ["leftrightsquigarrow"] = 8621,
4053    ["nharr"] = 8622,
4054    ["nleftrightarrow"] = 8622,
4055    ["Lsh"] = 8624,
4056    ["lsh"] = 8624,
4057    ["Rsh"] = 8625,
4058    ["rsh"] = 8625,
4059    ["ldsh"] = 8626,
4060    ["rdsh"] = 8627,
4061    ["crarr"] = 8629,
4062    ["cularr"] = 8630,
4063    ["curvearrowleft"] = 8630,
4064    ["curarr"] = 8631,
4065    ["curvearrowright"] = 8631,
4066    ["circlearrowleft"] = 8634,
4067    ["olarr"] = 8634,
4068    ["circlearrowright"] = 8635,
4069    ["orarr"] = 8635,
4070    ["LeftVector"] = 8636,
4071    ["leftharpoonup"] = 8636,
4072    ["lharu"] = 8636,
4073    ["DownLeftVector"] = 8637,
4074    ["leftharpoondown"] = 8637,
4075    ["lhard"] = 8637,
4076    ["RightUpVector"] = 8638,
4077    ["uharr"] = 8638,
4078    ["upharpoonright"] = 8638,
4079    ["LeftUpVector"] = 8639,
4080    ["uharl"] = 8639,
4081    ["upharpoonleft"] = 8639,
4082    ["RightVector"] = 8640,
4083    ["rharu"] = 8640,
4084    ["rightharpoonup"] = 8640,
4085    ["DownRightVector"] = 8641,
4086    ["rhard"] = 8641,
4087    ["rightharpoondown"] = 8641,
4088    ["RightDownVector"] = 8642,
4089    ["dharr"] = 8642,
4090    ["downharpoonright"] = 8642,
```

```
["LeftDownVector"] = 8643,
["dharl"] = 8643,
["downharpoonleft"] = 8643,
["RightArrowLeftArrow"] = 8644,
["rightleftarrows"] = 8644,
["rlarr"] = 8644,
["UpArrowDownArrow"] = 8645,
["udarr"] = 8645,
["LeftArrowRightArrow"] = 8646,
["leftrightarrows"] = 8646,
["lrarr"] = 8646,
["leftleftarrows"] = 8647,
["llarr"] = 8647,
["upuparrows"] = 8648,
["uuarr"] = 8648,
["rightrightarrows"] = 8649,
["rrarr"] = 8649,
["ddarr"] = 8650,
["downdownarrows"] = 8650,
["ReverseEquilibrium"] = 8651,
["leftrightharpoons"] = 8651,
["lrhar"] = 8651,
["Equilibrium"] = 8652,
["rightleftharpoons"] = 8652,
["rlhar"] = 8652,
["nLeftarrow"] = 8653,
["nlArr"] = 8653,
["nLeftrightarrow"] = 8654,
["nhArr"] = 8654,
["nRightarrow"] = 8655,
["nrArr"] = 8655,
["DoubleLeftArrow"] = 8656,
["Leftarrow"] = 8656,
["lArr"] = 8656,
["DoubleUpArrow"] = 8657,
["Uparrow"] = 8657,
["uArr"] = 8657,
["DoubleRightArrow"] = 8658,
["Implies"] = 8658,
["Rightarrow"] = 8658,
["rArr"] = 8658,
["DoubleDownArrow"] = 8659,
["Downarrow"] = 8659,
["dArr"] = 8659,
["DoubleLeftRightArrow"] = 8660,
["Leftrightarrow"] = 8660,
["hArr"] = 8660,
```

```
4138  ["iff"] = 8660,
4139  ["DoubleUpDownArrow"] = 8661,
4140  ["Updownarrow"] = 8661,
4141  ["vArr"] = 8661,
4142  ["nwArr"] = 8662,
4143  ["neArr"] = 8663,
4144  ["seArr"] = 8664,
4145  ["swArr"] = 8665,
4146  ["Lleftarrow"] = 8666,
4147  ["lAarr"] = 8666,
4148  ["Rrightarrow"] = 8667,
4149  ["rAarr"] = 8667,
4150  ["zigrarr"] = 8669,
4151  ["LeftArrowBar"] = 8676,
4152  ["larrb"] = 8676,
4153  ["RightArrowBar"] = 8677,
4154  ["rarrb"] = 8677,
4155  ["DownArrowUpArrow"] = 8693,
4156  ["duarr"] = 8693,
4157  ["loarr"] = 8701,
4158  ["roarr"] = 8702,
4159  ["hoarr"] = 8703,
4160  ["ForAll"] = 8704,
4161  ["forall"] = 8704,
4162  ["comp"] = 8705,
4163  ["complement"] = 8705,
4164  ["PartialD"] = 8706,
4165  ["npart"] = {8706, 824},
4166  ["part"] = 8706,
4167  ["Exists"] = 8707,
4168  ["exist"] = 8707,
4169  ["NotExists"] = 8708,
4170  ["nexist"] = 8708,
4171  ["nexists"] = 8708,
4172  ["empty"] = 8709,
4173  ["emptyset"] = 8709,
4174  ["emptyv"] = 8709,
4175  ["varnothing"] = 8709,
4176  ["Del"] = 8711,
4177  ["nabla"] = 8711,
4178  ["Element"] = 8712,
4179  ["in"] = 8712,
4180  ["isin"] = 8712,
4181  ["isinv"] = 8712,
4182  ["NotElement"] = 8713,
4183  ["notin"] = 8713,
4184  ["notinva"] = 8713,
```

```
4185    ["ReverseElement"] = 8715,
4186    ["SuchThat"] = 8715,
4187    ["ni"] = 8715,
4188    ["niv"] = 8715,
4189    ["NotReverseElement"] = 8716,
4190    ["notni"] = 8716,
4191    ["notniva"] = 8716,
4192    ["Product"] = 8719,
4193    ["prod"] = 8719,
4194    ["Coproduct"] = 8720,
4195    ["coprod"] = 8720,
4196    ["Sum"] = 8721,
4197    ["sum"] = 8721,
4198    ["minus"] = 8722,
4199    ["MinusPlus"] = 8723,
4200    ["mnplus"] = 8723,
4201    ["mp"] = 8723,
4202    ["dotplus"] = 8724,
4203    ["plusdo"] = 8724,
4204    ["Backslash"] = 8726,
4205    ["setminus"] = 8726,
4206    ["setmn"] = 8726,
4207    ["smallsetminus"] = 8726,
4208    ["ssetmn"] = 8726,
4209    ["lowast"] = 8727,
4210    ["SmallCircle"] = 8728,
4211    ["compfn"] = 8728,
4212    ["Sqrt"] = 8730,
4213    ["radic"] = 8730,
4214    ["Proportional"] = 8733,
4215    ["prop"] = 8733,
4216    ["propto"] = 8733,
4217    ["varpropto"] = 8733,
4218    ["vprop"] = 8733,
4219    ["infin"] = 8734,
4220    ["angrt"] = 8735,
4221    ["ang"] = 8736,
4222    ["angle"] = 8736,
4223    ["nang"] = {8736, 8402},
4224    ["angmsd"] = 8737,
4225    ["measuredangle"] = 8737,
4226    ["angsph"] = 8738,
4227    ["VerticalBar"] = 8739,
4228    ["mid"] = 8739,
4229    ["shortmid"] = 8739,
4230    ["smid"] = 8739,
4231    ["NotVerticalBar"] = 8740,
```

```
4232  ["nmid"] = 8740,
4233  ["nshortmid"] = 8740,
4234  ["nsmid"] = 8740,
4235  ["DoubleVerticalBar"] = 8741,
4236  ["par"] = 8741,
4237  ["parallel"] = 8741,
4238  ["shortparallel"] = 8741,
4239  ["spar"] = 8741,
4240  ["NotDoubleVerticalBar"] = 8742,
4241  ["npar"] = 8742,
4242  ["nparallel"] = 8742,
4243  ["nshortparallel"] = 8742,
4244  ["nspar"] = 8742,
4245  ["and"] = 8743,
4246  ["wedge"] = 8743,
4247  ["or"] = 8744,
4248  ["vee"] = 8744,
4249  ["cap"] = 8745,
4250  ["caps"] = {8745, 65024},
4251  ["cup"] = 8746,
4252  ["cups"] = {8746, 65024},
4253  ["Integral"] = 8747,
4254  ["int"] = 8747,
4255  ["Int"] = 8748,
4256  ["iiint"] = 8749,
4257  ["tint"] = 8749,
4258  ["ContourIntegral"] = 8750,
4259  ["conint"] = 8750,
4260  ["oint"] = 8750,
4261  ["Conint"] = 8751,
4262  ["DoubleContourIntegral"] = 8751,
4263  ["Cconint"] = 8752,
4264  ["cwint"] = 8753,
4265  ["ClockwiseContourIntegral"] = 8754,
4266  ["cwconint"] = 8754,
4267  ["CounterClockwiseContourIntegral"] = 8755,
4268  ["awconint"] = 8755,
4269  ["Therefore"] = 8756,
4270  ["there4"] = 8756,
4271  ["therefore"] = 8756,
4272  ["Because"] = 8757,
4273  ["becaus"] = 8757,
4274  ["because"] = 8757,
4275  ["ratio"] = 8758,
4276  ["Colon"] = 8759,
4277  ["Proportion"] = 8759,
4278  ["dotminus"] = 8760,
```

```
4279    ["minusd"] = 8760,
4280    ["mDDot"] = 8762,
4281    ["homtht"] = 8763,
4282    ["Tilde"] = 8764,
4283    ["nvsim"] = {8764, 8402},
4284    ["sim"] = 8764,
4285    ["thicksim"] = 8764,
4286    ["thksim"] = 8764,
4287    ["backsim"] = 8765,
4288    ["bsim"] = 8765,
4289    ["race"] = {8765, 817},
4290    ["ac"] = 8766,
4291    ["acE"] = {8766, 819},
4292    ["mstpos"] = 8766,
4293    ["acd"] = 8767,
4294    ["VerticalTilde"] = 8768,
4295    ["wr"] = 8768,
4296    ["wreath"] = 8768,
4297    ["NotTilde"] = 8769,
4298    ["nsim"] = 8769,
4299    ["EqualTilde"] = 8770,
4300    ["NotEqualTilde"] = {8770, 824},
4301    ["eqsim"] = 8770,
4302    ["esim"] = 8770,
4303    ["nesim"] = {8770, 824},
4304    ["TildeEqual"] = 8771,
4305    ["sime"] = 8771,
4306    ["simeq"] = 8771,
4307    ["NotTildeEqual"] = 8772,
4308    ["nsime"] = 8772,
4309    ["nsimeq"] = 8772,
4310    ["TildeFullEqual"] = 8773,
4311    ["cong"] = 8773,
4312    ["simne"] = 8774,
4313    ["NotTildeFullEqual"] = 8775,
4314    ["ncong"] = 8775,
4315    ["TildeTilde"] = 8776,
4316    ["ap"] = 8776,
4317    ["approx"] = 8776,
4318    ["asymp"] = 8776,
4319    ["thickapprox"] = 8776,
4320    ["thkap"] = 8776,
4321    ["NotTildeTilde"] = 8777,
4322    ["nap"] = 8777,
4323    ["napprox"] = 8777,
4324    ["ape"] = 8778,
4325    ["approxeq"] = 8778,
```

```
4326  ["apid"] = 8779,
4327  ["napid"] = {8779, 824},
4328  ["backcong"] = 8780,
4329  ["bcong"] = 8780,
4330  ["CupCap"] = 8781,
4331  ["asympeq"] = 8781,
4332  ["nvap"] = {8781, 8402},
4333  ["Bumpeq"] = 8782,
4334  ["HumpDownHump"] = 8782,
4335  ["NotHumpDownHump"] = {8782, 824},
4336  ["bump"] = 8782,
4337  ["nbump"] = {8782, 824},
4338  ["HumpEqual"] = 8783,
4339  ["NotHumpEqual"] = {8783, 824},
4340  ["bumpe"] = 8783,
4341  ["bumpeq"] = 8783,
4342  ["nbumpe"] = {8783, 824},
4343  ["DotEqual"] = 8784,
4344  ["doteq"] = 8784,
4345  ["esdot"] = 8784,
4346  ["nedot"] = {8784, 824},
4347  ["doteqdot"] = 8785,
4348  ["eDot"] = 8785,
4349  ["efDot"] = 8786,
4350  ["fallingdotseq"] = 8786,
4351  ["erDot"] = 8787,
4352  ["risingdotseq"] = 8787,
4353  ["Assign"] = 8788,
4354  ["colone"] = 8788,
4355  ["coloneq"] = 8788,
4356  ["ecolon"] = 8789,
4357  ["eqcolon"] = 8789,
4358  ["ecir"] = 8790,
4359  ["eqcirc"] = 8790,
4360  ["circeq"] = 8791,
4361  ["cire"] = 8791,
4362  ["wedgeq"] = 8793,
4363  ["veeeq"] = 8794,
4364  ["triangleq"] = 8796,
4365  ["trie"] = 8796,
4366  ["equest"] = 8799,
4367  ["questeq"] = 8799,
4368  ["NotEqual"] = 8800,
4369  ["ne"] = 8800,
4370  ["Congruent"] = 8801,
4371  ["bnequiv"] = {8801, 8421},
4372  ["equiv"] = 8801,
```

```
4373    ["NotCongruent"] = 8802,
4374    ["nequiv"] = 8802,
4375    ["le"] = 8804,
4376    ["leq"] = 8804,
4377    ["nvle"] = {8804, 8402},
4378    ["GreaterEqual"] = 8805,
4379    ["ge"] = 8805,
4380    ["geq"] = 8805,
4381    ["nvge"] = {8805, 8402},
4382    ["LessFullEqual"] = 8806,
4383    ["lE"] = 8806,
4384    ["leqq"] = 8806,
4385    ["nlE"] = {8806, 824},
4386    ["nleqq"] = {8806, 824},
4387    ["GreaterFullEqual"] = 8807,
4388    ["NotGreaterFullEqual"] = {8807, 824},
4389    ["gE"] = 8807,
4390    ["geqq"] = 8807,
4391    ["ngE"] = {8807, 824},
4392    ["ngeqq"] = {8807, 824},
4393    ["lnE"] = 8808,
4394    ["lneqq"] = 8808,
4395    ["lvertneqq"] = {8808, 65024},
4396    ["lvnE"] = {8808, 65024},
4397    ["gnE"] = 8809,
4398    ["gneqq"] = 8809,
4399    ["gvertneqq"] = {8809, 65024},
4400    ["gvnE"] = {8809, 65024},
4401    ["Lt"] = 8810,
4402    ["NestedLessLess"] = 8810,
4403    ["NotLessLess"] = {8810, 824},
4404    ["ll"] = 8810,
4405    ["nLt"] = {8810, 8402},
4406    ["nLtv"] = {8810, 824},
4407    ["Gt"] = 8811,
4408    ["NestedGreaterGreater"] = 8811,
4409    ["NotGreaterGreater"] = {8811, 824},
4410    ["gg"] = 8811,
4411    ["nGt"] = {8811, 8402},
4412    ["nGtv"] = {8811, 824},
4413    ["between"] = 8812,
4414    ["twixt"] = 8812,
4415    ["NotCupCap"] = 8813,
4416    ["NotLess"] = 8814,
4417    ["nless"] = 8814,
4418    ["nlt"] = 8814,
4419    ["NotGreater"] = 8815,
```

```
4420  ["ngt"] = 8815,
4421  ["ngtr"] = 8815,
4422  ["NotLessEqual"] = 8816,
4423  ["nle"] = 8816,
4424  ["nleq"] = 8816,
4425  ["NotGreaterEqual"] = 8817,
4426  ["nge"] = 8817,
4427  ["ngeq"] = 8817,
4428  ["LessTilde"] = 8818,
4429  ["lesssim"] = 8818,
4430  ["lsim"] = 8818,
4431  ["GreaterTilde"] = 8819,
4432  ["gsim"] = 8819,
4433  ["gtrsim"] = 8819,
4434  ["NotLessTilde"] = 8820,
4435  ["nlsim"] = 8820,
4436  ["NotGreaterTilde"] = 8821,
4437  ["ngsim"] = 8821,
4438  ["LessGreater"] = 8822,
4439  ["lessgtr"] = 8822,
4440  ["lg"] = 8822,
4441  ["GreaterLess"] = 8823,
4442  ["gl"] = 8823,
4443  ["gtrless"] = 8823,
4444  ["NotLessGreater"] = 8824,
4445  ["ntlg"] = 8824,
4446  ["NotGreaterLess"] = 8825,
4447  ["ntgl"] = 8825,
4448  ["Precedes"] = 8826,
4449  ["pr"] = 8826,
4450  ["prec"] = 8826,
4451  ["Succeeds"] = 8827,
4452  ["sc"] = 8827,
4453  ["succ"] = 8827,
4454  ["PrecedesSlantEqual"] = 8828,
4455  ["prcue"] = 8828,
4456  ["preccurlyeq"] = 8828,
4457  ["SucceedsSlantEqual"] = 8829,
4458  ["sccue"] = 8829,
4459  ["succcurlyeq"] = 8829,
4460  ["PrecedesTilde"] = 8830,
4461  ["precsim"] = 8830,
4462  ["prsim"] = 8830,
4463  ["NotSucceedsTilde"] = {8831, 824},
4464  ["SucceedsTilde"] = 8831,
4465  ["scsim"] = 8831,
4466  ["succsim"] = 8831,
```

```
["NotPrecedes"] = 8832,
["npr"] = 8832,
["nprec"] = 8832,
["NotSucceeds"] = 8833,
["nsc"] = 8833,
["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},
```

```
4514    ["SquareSuperset"] = 8848,
4515    ["sqsup"] = 8848,
4516    ["sqsupset"] = 8848,
4517    ["SquareSubsetEqual"] = 8849,
4518    ["sqsube"] = 8849,
4519    ["sqsubseteq"] = 8849,
4520    ["SquareSupersetEqual"] = 8850,
4521    ["sqsupe"] = 8850,
4522    ["sqsupseteq"] = 8850,
4523    ["SquareIntersection"] = 8851,
4524    ["sqcap"] = 8851,
4525    ["sqcaps"] = {8851, 65024},
4526    ["SquareUnion"] = 8852,
4527    ["sqcup"] = 8852,
4528    ["sqcups"] = {8852, 65024},
4529    ["CirclePlus"] = 8853,
4530    ["oplus"] = 8853,
4531    ["CircleMinus"] = 8854,
4532    ["ominus"] = 8854,
4533    ["CircleTimes"] = 8855,
4534    ["otimes"] = 8855,
4535    ["osol"] = 8856,
4536    ["CircleDot"] = 8857,
4537    ["odot"] = 8857,
4538    ["circledcirc"] = 8858,
4539    ["ocir"] = 8858,
4540    ["circledast"] = 8859,
4541    ["oast"] = 8859,
4542    ["circleddash"] = 8861,
4543    ["odash"] = 8861,
4544    ["boxplus"] = 8862,
4545    ["plusb"] = 8862,
4546    ["boxminus"] = 8863,
4547    ["minusb"] = 8863,
4548    ["boxtimes"] = 8864,
4549    ["timesb"] = 8864,
4550    ["dotsquare"] = 8865,
4551    ["sdotb"] = 8865,
4552    ["RightTee"] = 8866,
4553    ["vdash"] = 8866,
4554    ["LeftTee"] = 8867,
4555    ["dashv"] = 8867,
4556    ["DownTee"] = 8868,
4557    ["top"] = 8868,
4558    ["UpTee"] = 8869,
4559    ["bot"] = 8869,
4560    ["bottom"] = 8869,
```

```
4561    ["perp"] = 8869,
4562    ["models"] = 8871,
4563    ["DoubleRightTee"] = 8872,
4564    ["vDash"] = 8872,
4565    ["Vdash"] = 8873,
4566    ["Vvdash"] = 8874,
4567    ["VDash"] = 8875,
4568    ["nvdash"] = 8876,
4569    ["nvDash"] = 8877,
4570    ["nVdash"] = 8878,
4571    ["nVDash"] = 8879,
4572    ["prurel"] = 8880,
4573    ["LeftTriangle"] = 8882,
4574    ["vartriangleleft"] = 8882,
4575    ["vltri"] = 8882,
4576    ["RightTriangle"] = 8883,
4577    ["vartriangleright"] = 8883,
4578    ["vrtri"] = 8883,
4579    ["LeftTriangleEqual"] = 8884,
4580    ["ltrie"] = 8884,
4581    ["nvltrie"] = {8884, 8402},
4582    ["trianglelefteq"] = 8884,
4583    ["RightTriangleEqual"] = 8885,
4584    ["nvrtrie"] = {8885, 8402},
4585    ["rtrie"] = 8885,
4586    ["trianglerighteq"] = 8885,
4587    ["origof"] = 8886,
4588    ["imof"] = 8887,
4589    ["multimap"] = 8888,
4590    ["mumap"] = 8888,
4591    ["hercon"] = 8889,
4592    ["intcal"] = 8890,
4593    ["intercal"] = 8890,
4594    ["veebar"] = 8891,
4595    ["barvee"] = 8893,
4596    ["angrtvb"] = 8894,
4597    ["lrtri"] = 8895,
4598    ["Wedge"] = 8896,
4599    ["bigwedge"] = 8896,
4600    ["xwedge"] = 8896,
4601    ["Vee"] = 8897,
4602    ["bigvee"] = 8897,
4603    ["xvee"] = 8897,
4604    ["Intersection"] = 8898,
4605    ["bigcap"] = 8898,
4606    ["xcap"] = 8898,
4607    ["Union"] = 8899,
```

```
4608   ["bigcup"] = 8899,
4609   ["xcup"] = 8899,
4610   ["Diamond"] = 8900,
4611   ["diam"] = 8900,
4612   ["diamond"] = 8900,
4613   ["sdot"] = 8901,
4614   ["Star"] = 8902,
4615   ["sstarf"] = 8902,
4616   ["divideontimes"] = 8903,
4617   ["divonx"] = 8903,
4618   ["bowtie"] = 8904,
4619   ["ltimes"] = 8905,
4620   ["rtimes"] = 8906,
4621   ["leftthreetimes"] = 8907,
4622   ["lthree"] = 8907,
4623   ["rightthreetimes"] = 8908,
4624   ["rthree"] = 8908,
4625   ["backsimeq"] = 8909,
4626   ["bsime"] = 8909,
4627   ["curlyvee"] = 8910,
4628   ["cuvee"] = 8910,
4629   ["curlywedge"] = 8911,
4630   ["cuwed"] = 8911,
4631   ["Sub"] = 8912,
4632   ["Subset"] = 8912,
4633   ["Sup"] = 8913,
4634   ["Supset"] = 8913,
4635   ["Cap"] = 8914,
4636   ["Cup"] = 8915,
4637   ["fork"] = 8916,
4638   ["pitchfork"] = 8916,
4639   ["epar"] = 8917,
4640   ["lessdot"] = 8918,
4641   ["ltdot"] = 8918,
4642   ["gtdot"] = 8919,
4643   ["gtrdot"] = 8919,
4644   ["Ll"] = 8920,
4645   ["nLl"] = {8920, 824},
4646   ["Gg"] = 8921,
4647   ["ggg"] = 8921,
4648   ["nGg"] = {8921, 824},
4649   ["LessEqualGreater"] = 8922,
4650   ["leg"] = 8922,
4651   ["lesg"] = {8922, 65024},
4652   ["lesseqgtr"] = 8922,
4653   ["GreaterEqualLess"] = 8923,
4654   ["gel"] = 8923,
```

```
["gesl"] = {8923, 65024},
["gtreqless"] = 8923,
["cuepr"] = 8926,
["curlyeqprec"] = 8926,
["cuesc"] = 8927,
["curlyeqsucc"] = 8927,
["NotPrecedesSlantEqual"] = 8928,
["nprcue"] = 8928,
["NotSucceedsSlantEqual"] = 8929,
["nsccue"] = 8929,
["NotSquareSubsetEqual"] = 8930,
["nsqsube"] = 8930,
["NotSquareSupersetEqual"] = 8931,
["nsqsupe"] = 8931,
["lnsim"] = 8934,
["gnsim"] = 8935,
["precnsim"] = 8936,
["prnsim"] = 8936,
["scnsim"] = 8937,
["succnsim"] = 8937,
["NotLeftTriangle"] = 8938,
["nltri"] = 8938,
["ntriangleleft"] = 8938,
["NotRightTriangle"] = 8939,
["nrtri"] = 8939,
["ntriangleright"] = 8939,
["NotLeftTriangleEqual"] = 8940,
["nltrie"] = 8940,
["ntrianglelefteq"] = 8940,
["NotRightTriangleEqual"] = 8941,
["nrtrie"] = 8941,
["ntrianglerighteq"] = 8941,
["vellip"] = 8942,
["ctdot"] = 8943,
["utdot"] = 8944,
["dtdot"] = 8945,
["disin"] = 8946,
["isinsv"] = 8947,
["isins"] = 8948,
["isindot"] = 8949,
["notindot"] = {8949, 824},
["notinvc"] = 8950,
["notinvb"] = 8951,
["isinE"] = 8953,
["notinE"] = {8953, 824},
["nisd"] = 8954,
["xnis"] = 8955,
```

```
4702  ["nis"] = 8956,
4703  ["notnivc"] = 8957,
4704  ["notnivb"] = 8958,
4705  ["barwed"] = 8965,
4706  ["barwedge"] = 8965,
4707  ["Barwed"] = 8966,
4708  ["doublebarwedge"] = 8966,
4709  ["LeftCeiling"] = 8968,
4710  ["lceil"] = 8968,
4711  ["RightCeiling"] = 8969,
4712  ["rceil"] = 8969,
4713  ["LeftFloor"] = 8970,
4714  ["lfloor"] = 8970,
4715  ["RightFloor"] = 8971,
4716  ["rfloor"] = 8971,
4717  ["drcrop"] = 8972,
4718  ["dlcrop"] = 8973,
4719  ["urcrop"] = 8974,
4720  ["ulcrop"] = 8975,
4721  ["bnot"] = 8976,
4722  ["profline"] = 8978,
4723  ["profsurf"] = 8979,
4724  ["telrec"] = 8981,
4725  ["target"] = 8982,
4726  ["ulcorn"] = 8988,
4727  ["ulcorner"] = 8988,
4728  ["urcorn"] = 8989,
4729  ["urcorner"] = 8989,
4730  ["dlcorn"] = 8990,
4731  ["llcorner"] = 8990,
4732  ["drcorn"] = 8991,
4733  ["lrcorner"] = 8991,
4734  ["frown"] = 8994,
4735  ["sfrown"] = 8994,
4736  ["smile"] = 8995,
4737  ["ssmile"] = 8995,
4738  ["cylcty"] = 9005,
4739  ["profalar"] = 9006,
4740  ["topbot"] = 9014,
4741  ["ovbar"] = 9021,
4742  ["solbar"] = 9023,
4743  ["angzarr"] = 9084,
4744  ["lmoust"] = 9136,
4745  ["lmoustache"] = 9136,
4746  ["rmoust"] = 9137,
4747  ["rmoustache"] = 9137,
4748  ["OverBracket"] = 9140,
```

```
4749    ["tbrk"] = 9140,
4750    ["UnderBracket"] = 9141,
4751    ["bbrk"] = 9141,
4752    ["bbrktbrk"] = 9142,
4753    ["OverParenthesis"] = 9180,
4754    ["UnderParenthesis"] = 9181,
4755    ["OverBrace"] = 9182,
4756    ["UnderBrace"] = 9183,
4757    ["trpezium"] = 9186,
4758    ["elinters"] = 9191,
4759    ["blank"] = 9251,
4760    ["circledS"] = 9416,
4761    ["oS"] = 9416,
4762    ["HorizontalLine"] = 9472,
4763    ["boxh"] = 9472,
4764    ["boxv"] = 9474,
4765    ["boxdr"] = 9484,
4766    ["boxdl"] = 9488,
4767    ["boxur"] = 9492,
4768    ["boxul"] = 9496,
4769    ["boxvr"] = 9500,
4770    ["boxvl"] = 9508,
4771    ["boxhd"] = 9516,
4772    ["boxhu"] = 9524,
4773    ["boxvh"] = 9532,
4774    ["boxH"] = 9552,
4775    ["boxV"] = 9553,
4776    ["boxdR"] = 9554,
4777    ["boxDr"] = 9555,
4778    ["boxDR"] = 9556,
4779    ["boxdL"] = 9557,
4780    ["boxDl"] = 9558,
4781    ["boxDL"] = 9559,
4782    ["boxuR"] = 9560,
4783    ["boxUr"] = 9561,
4784    ["boxUR"] = 9562,
4785    ["boxuL"] = 9563,
4786    ["boxUl"] = 9564,
4787    ["boxUL"] = 9565,
4788    ["boxvR"] = 9566,
4789    ["boxVr"] = 9567,
4790    ["boxVR"] = 9568,
4791    ["boxvL"] = 9569,
4792    ["boxVl"] = 9570,
4793    ["boxVL"] = 9571,
4794    ["boxHd"] = 9572,
4795    ["boxhD"] = 9573,
```

```
4796   ["boxHD"] = 9574,
4797   ["boxHu"] = 9575,
4798   ["boxhU"] = 9576,
4799   ["boxHU"] = 9577,
4800   ["boxvH"] = 9578,
4801   ["boxVh"] = 9579,
4802   ["boxVH"] = 9580,
4803   ["uhblk"] = 9600,
4804   ["lhblk"] = 9604,
4805   ["block"] = 9608,
4806   ["blk14"] = 9617,
4807   ["blk12"] = 9618,
4808   ["blk34"] = 9619,
4809   ["Square"] = 9633,
4810   ["squ"] = 9633,
4811   ["square"] = 9633,
4812   ["FilledVerySmallSquare"] = 9642,
4813   ["blacksquare"] = 9642,
4814   ["squarf"] = 9642,
4815   ["squf"] = 9642,
4816   ["EmptyVerySmallSquare"] = 9643,
4817   ["rect"] = 9645,
4818   ["marker"] = 9646,
4819   ["fltns"] = 9649,
4820   ["bigtriangleup"] = 9651,
4821   ["xutri"] = 9651,
4822   ["blacktriangle"] = 9652,
4823   ["utrif"] = 9652,
4824   ["triangle"] = 9653,
4825   ["utri"] = 9653,
4826   ["blacktriangleright"] = 9656,
4827   ["rtrif"] = 9656,
4828   ["rtri"] = 9657,
4829   ["triangleright"] = 9657,
4830   ["bigtriangledown"] = 9661,
4831   ["xdtri"] = 9661,
4832   ["blacktriangledown"] = 9662,
4833   ["dtrif"] = 9662,
4834   ["dtri"] = 9663,
4835   ["triangledown"] = 9663,
4836   ["blacktriangleleft"] = 9666,
4837   ["ltrif"] = 9666,
4838   ["ltri"] = 9667,
4839   ["triangleleft"] = 9667,
4840   ["loz"] = 9674,
4841   ["lozenge"] = 9674,
4842   ["cir"] = 9675,
```

```
4843  ["tridot"] = 9708,
4844  ["bigcirc"] = 9711,
4845  ["xcirc"] = 9711,
4846  ["ultri"] = 9720,
4847  ["urtri"] = 9721,
4848  ["lltri"] = 9722,
4849  ["EmptySmallSquare"] = 9723,
4850  ["FilledSmallSquare"] = 9724,
4851  ["bigstar"] = 9733,
4852  ["starf"] = 9733,
4853  ["star"] = 9734,
4854  ["phone"] = 9742,
4855  ["female"] = 9792,
4856  ["male"] = 9794,
4857  ["spades"] = 9824,
4858  ["spadesuit"] = 9824,
4859  ["clubs"] = 9827,
4860  ["clubsuit"] = 9827,
4861  ["hearts"] = 9829,
4862  ["heartsuit"] = 9829,
4863  ["diamondsuit"] = 9830,
4864  ["diams"] = 9830,
4865  ["sung"] = 9834,
4866  ["flat"] = 9837,
4867  ["natur"] = 9838,
4868  ["natural"] = 9838,
4869  ["sharp"] = 9839,
4870  ["check"] = 10003,
4871  ["checkmark"] = 10003,
4872  ["cross"] = 10007,
4873  ["malt"] = 10016,
4874  ["maltese"] = 10016,
4875  ["sext"] = 10038,
4876  ["VerticalSeparator"] = 10072,
4877  ["lbbrk"] = 10098,
4878  ["rbbrk"] = 10099,
4879  ["bsolhsub"] = 10184,
4880  ["suphsol"] = 10185,
4881  ["LeftDoubleBracket"] = 10214,
4882  ["lobrk"] = 10214,
4883  ["RightDoubleBracket"] = 10215,
4884  ["robrk"] = 10215,
4885  ["LeftAngleBracket"] = 10216,
4886  ["lang"] = 10216,
4887  ["langle"] = 10216,
4888  ["RightAngleBracket"] = 10217,
4889  ["rang"] = 10217,
```

```
4890    ["rangle"] = 10217,
4891    ["Lang"] = 10218,
4892    ["Rang"] = 10219,
4893    ["loang"] = 10220,
4894    ["roang"] = 10221,
4895    ["LongLeftArrow"] = 10229,
4896    ["longleftarrow"] = 10229,
4897    ["xlarr"] = 10229,
4898    ["LongRightArrow"] = 10230,
4899    ["longrightarrow"] = 10230,
4900    ["xrarr"] = 10230,
4901    ["LongLeftRightArrow"] = 10231,
4902    ["longleftrightarrow"] = 10231,
4903    ["xharr"] = 10231,
4904    ["DoubleLongLeftArrow"] = 10232,
4905    ["Longleftarrow"] = 10232,
4906    ["xlArr"] = 10232,
4907    ["DoubleLongRightArrow"] = 10233,
4908    ["Longrightarrow"] = 10233,
4909    ["xrArr"] = 10233,
4910    ["DoubleLongLeftRightArrow"] = 10234,
4911    ["Longleftrightarrow"] = 10234,
4912    ["xhArr"] = 10234,
4913    ["longmapsto"] = 10236,
4914    ["xmap"] = 10236,
4915    ["dzigrarr"] = 10239,
4916    ["nvlArr"] = 10498,
4917    ["nvrArr"] = 10499,
4918    ["nvHarr"] = 10500,
4919    ["Map"] = 10501,
4920    ["lbarr"] = 10508,
4921    ["bkarow"] = 10509,
4922    ["rbarr"] = 10509,
4923    ["lBarr"] = 10510,
4924    ["dbkarow"] = 10511,
4925    ["rBarr"] = 10511,
4926    ["RBarr"] = 10512,
4927    ["drbkarow"] = 10512,
4928    ["DDotrahd"] = 10513,
4929    ["UpArrowBar"] = 10514,
4930    ["DownArrowBar"] = 10515,
4931    ["Rarrtl"] = 10518,
4932    ["latail"] = 10521,
4933    ["ratail"] = 10522,
4934    ["lAtail"] = 10523,
4935    ["rAtail"] = 10524,
4936    ["larrfs"] = 10525,
```

```
4937  ["rarrfs"] = 10526,
4938  ["larrbfs"] = 10527,
4939  ["rarrbfs"] = 10528,
4940  ["nwarhk"] = 10531,
4941  ["nearhk"] = 10532,
4942  ["hksearow"] = 10533,
4943  ["searhk"] = 10533,
4944  ["hkswarow"] = 10534,
4945  ["swarhk"] = 10534,
4946  ["nwnear"] = 10535,
4947  ["nesear"] = 10536,
4948  ["toea"] = 10536,
4949  ["seswar"] = 10537,
4950  ["tosa"] = 10537,
4951  ["swnwar"] = 10538,
4952  ["nrarrc"] = {10547, 824},
4953  ["rarrc"] = 10547,
4954  ["cudarrr"] = 10549,
4955  ["ldca"] = 10550,
4956  ["rdca"] = 10551,
4957  ["cudarrl"] = 10552,
4958  ["larrpl"] = 10553,
4959  ["curarrm"] = 10556,
4960  ["cularrp"] = 10557,
4961  ["rarrpl"] = 10565,
4962  ["harrcir"] = 10568,
4963  ["Uarrocir"] = 10569,
4964  ["lurdshar"] = 10570,
4965  ["ldrushar"] = 10571,
4966  ["LeftRightVector"] = 10574,
4967  ["RightUpDownVector"] = 10575,
4968  ["DownLeftRightVector"] = 10576,
4969  ["LeftUpDownVector"] = 10577,
4970  ["LeftVectorBar"] = 10578,
4971  ["RightVectorBar"] = 10579,
4972  ["RightUpVectorBar"] = 10580,
4973  ["RightDownVectorBar"] = 10581,
4974  ["DownLeftVectorBar"] = 10582,
4975  ["DownRightVectorBar"] = 10583,
4976  ["LeftUpVectorBar"] = 10584,
4977  ["LeftDownVectorBar"] = 10585,
4978  ["LeftTeeVector"] = 10586,
4979  ["RightTeeVector"] = 10587,
4980  ["RightUpTeeVector"] = 10588,
4981  ["RightDownTeeVector"] = 10589,
4982  ["DownLeftTeeVector"] = 10590,
4983  ["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,
```

```
5031  ["vangrt"] = 10652,
5032  ["angrtvbd"] = 10653,
5033  ["ange"] = 10660,
5034  ["range"] = 10661,
5035  ["dwangle"] = 10662,
5036  ["uwangle"] = 10663,
5037  ["angmsdaa"] = 10664,
5038  ["angmsdab"] = 10665,
5039  ["angmsdac"] = 10666,
5040  ["angmsdad"] = 10667,
5041  ["angmsdae"] = 10668,
5042  ["angmsdaf"] = 10669,
5043  ["angmsdag"] = 10670,
5044  ["angmsdah"] = 10671,
5045  ["bemptyv"] = 10672,
5046  ["demptyv"] = 10673,
5047  ["cemptyv"] = 10674,
5048  ["raemptyv"] = 10675,
5049  ["laemptyv"] = 10676,
5050  ["ohbar"] = 10677,
5051  ["omid"] = 10678,
5052  ["opar"] = 10679,
5053  ["operp"] = 10681,
5054  ["olcross"] = 10683,
5055  ["odsold"] = 10684,
5056  ["olcir"] = 10686,
5057  ["ofcir"] = 10687,
5058  ["olt"] = 10688,
5059  ["ogt"] = 10689,
5060  ["cirscir"] = 10690,
5061  ["cirE"] = 10691,
5062  ["solb"] = 10692,
5063  ["bsolb"] = 10693,
5064  ["boxbox"] = 10697,
5065  ["trisb"] = 10701,
5066  ["rtriltri"] = 10702,
5067  ["LeftTriangleBar"] = 10703,
5068  ["NotLeftTriangleBar"] = {10703, 824},
5069  ["NotRightTriangleBar"] = {10704, 824},
5070  ["RightTriangleBar"] = 10704,
5071  ["iinfin"] = 10716,
5072  ["infintie"] = 10717,
5073  ["nvinfin"] = 10718,
5074  ["eparsl"] = 10723,
5075  ["smeparsl"] = 10724,
5076  ["eqvparsl"] = 10725,
5077  ["blacklozenge"] = 10731,
```

```
5078   ["lozf"] = 10731,
5079   ["RuleDelayed"] = 10740,
5080   ["dsol"] = 10742,
5081   ["bigodot"] = 10752,
5082   ["xodot"] = 10752,
5083   ["bigoplus"] = 10753,
5084   ["xoplus"] = 10753,
5085   ["bigotimes"] = 10754,
5086   ["xotime"] = 10754,
5087   ["biguplus"] = 10756,
5088   ["xuplus"] = 10756,
5089   ["bigsqcup"] = 10758,
5090   ["xsqcup"] = 10758,
5091   ["iiiint"] = 10764,
5092   ["qint"] = 10764,
5093   ["fpartint"] = 10765,
5094   ["cirfnint"] = 10768,
5095   ["awint"] = 10769,
5096   ["rppolint"] = 10770,
5097   ["scpolint"] = 10771,
5098   ["npolint"] = 10772,
5099   ["pointint"] = 10773,
5100   ["quatint"] = 10774,
5101   ["intlarhk"] = 10775,
5102   ["pluscir"] = 10786,
5103   ["plusacir"] = 10787,
5104   ["simplus"] = 10788,
5105   ["plusdu"] = 10789,
5106   ["plussim"] = 10790,
5107   ["plustwo"] = 10791,
5108   ["mcomma"] = 10793,
5109   ["minusdu"] = 10794,
5110   ["loplus"] = 10797,
5111   ["roplus"] = 10798,
5112   ["Cross"] = 10799,
5113   ["timesd"] = 10800,
5114   ["timesbar"] = 10801,
5115   ["smashp"] = 10803,
5116   ["lotimes"] = 10804,
5117   ["rotimes"] = 10805,
5118   ["otimesas"] = 10806,
5119   ["Otimes"] = 10807,
5120   ["odiv"] = 10808,
5121   ["triplus"] = 10809,
5122   ["triminus"] = 10810,
5123   ["tritime"] = 10811,
5124   ["intprod"] = 10812,
```

```
5125    ["iprod"] = 10812,
5126    ["amalg"] = 10815,
5127    ["capdot"] = 10816,
5128    ["ncup"] = 10818,
5129    ["ncap"] = 10819,
5130    ["capand"] = 10820,
5131    ["cupor"] = 10821,
5132    ["cupcap"] = 10822,
5133    ["capcup"] = 10823,
5134    ["cupbrcap"] = 10824,
5135    ["capbrcup"] = 10825,
5136    ["cupcup"] = 10826,
5137    ["capcap"] = 10827,
5138    ["ccups"] = 10828,
5139    ["ccaps"] = 10829,
5140    ["ccupssm"] = 10832,
5141    ["And"] = 10835,
5142    ["Or"] = 10836,
5143    ["andand"] = 10837,
5144    ["oror"] = 10838,
5145    ["orslope"] = 10839,
5146    ["andslope"] = 10840,
5147    ["andv"] = 10842,
5148    ["orv"] = 10843,
5149    ["andd"] = 10844,
5150    ["ord"] = 10845,
5151    ["wedbar"] = 10847,
5152    ["sdote"] = 10854,
5153    ["simdot"] = 10858,
5154    ["congdot"] = 10861,
5155    ["ncongdot"] = {10861, 824},
5156    ["easter"] = 10862,
5157    ["apacir"] = 10863,
5158    ["apE"] = 10864,
5159    ["napE"] = {10864, 824},
5160    ["eplus"] = 10865,
5161    ["pluse"] = 10866,
5162    ["Esim"] = 10867,
5163    ["Colone"] = 10868,
5164    ["Equal"] = 10869,
5165    ["ddotseq"] = 10871,
5166    ["eDDot"] = 10871,
5167    ["equivDD"] = 10872,
5168    ["ltcir"] = 10873,
5169    ["gtcir"] = 10874,
5170    ["ltquest"] = 10875,
5171    ["gtquest"] = 10876,
```

```
["LessSlantEqual"] = 10877,
["NotLessSlantEqual"] = {10877, 824},
["leqslant"] = 10877,
["les"] = 10877,
["nleqslant"] = {10877, 824},
["nles"] = {10877, 824},
["GreaterSlantEqual"] = 10878,
["NotGreaterSlantEqual"] = {10878, 824},
["geqslant"] = 10878,
["ges"] = 10878,
["ngeqslant"] = {10878, 824},
["nges"] = {10878, 824},
["lesdot"] = 10879,
["gesdot"] = 10880,
["lesdoto"] = 10881,
["gesdoto"] = 10882,
["lesdotor"] = 10883,
["gesdotol"] = 10884,
["lap"] = 10885,
["lessapprox"] = 10885,
["gap"] = 10886,
["gtrapprox"] = 10886,
["lne"] = 10887,
["lneq"] = 10887,
["gne"] = 10888,
["gneq"] = 10888,
["lnap"] = 10889,
["lnapprox"] = 10889,
["gnap"] = 10890,
["gnapprox"] = 10890,
["lEg"] = 10891,
["lesseqqgtr"] = 10891,
["gEl"] = 10892,
["gtreqqless"] = 10892,
["lsime"] = 10893,
["gsime"] = 10894,
["lsimg"] = 10895,
["gsiml"] = 10896,
["lgE"] = 10897,
["glE"] = 10898,
["lesges"] = 10899,
["gesles"] = 10900,
["els"] = 10901,
["eqslantless"] = 10901,
["egs"] = 10902,
["eqslantgtr"] = 10902,
["elsdot"] = 10903,
```

```
["egsdot"] = 10904,
["el"] = 10905,
["eg"] = 10906,
["siml"] = 10909,
["simg"] = 10910,
["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,
```

```
5407   ["nfr"] = 120107,
5408   ["ofr"] = 120108,
5409   ["pfr"] = 120109,
5410   ["qfr"] = 120110,
5411   ["rfr"] = 120111,
5412   ["sfr"] = 120112,
5413   ["tfr"] = 120113,
5414   ["ufr"] = 120114,
5415   ["vfr"] = 120115,
5416   ["wfr"] = 120116,
5417   ["xfr"] = 120117,
5418   ["yfr"] = 120118,
5419   ["zfr"] = 120119,
5420   ["Aopf"] = 120120,
5421   ["Bopf"] = 120121,
5422   ["Dopf"] = 120123,
5423   ["Eopf"] = 120124,
5424   ["Fopf"] = 120125,
5425   ["Gopf"] = 120126,
5426   ["Iopf"] = 120128,
5427   ["Jopf"] = 120129,
5428   ["Kopf"] = 120130,
5429   ["Lopf"] = 120131,
5430   ["Mopf"] = 120132,
5431   ["Oopf"] = 120134,
5432   ["Sopf"] = 120138,
5433   ["Topf"] = 120139,
5434   ["Uopf"] = 120140,
5435   ["Vopf"] = 120141,
5436   ["Wopf"] = 120142,
5437   ["Xopf"] = 120143,
5438   ["Yopf"] = 120144,
5439   ["aopf"] = 120146,
5440   ["bopf"] = 120147,
5441   ["copf"] = 120148,
5442   ["dopf"] = 120149,
5443   ["eopf"] = 120150,
5444   ["fopf"] = 120151,
5445   ["gopf"] = 120152,
5446   ["hopf"] = 120153,
5447   ["iopf"] = 120154,
5448   ["jopf"] = 120155,
5449   ["kopf"] = 120156,
5450   ["lopf"] = 120157,
5451   ["mopf"] = 120158,
5452   ["nopf"] = 120159,
5453   ["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]
```

```
5489   if code_points == nil then
5490     return "&" .. s .. ";"
5491   end
5492   if type(code_points) ~= 'table' then
5493     code_points = {code_points}
5494   end
5495   local char_table = {}
5496     for _, code_point in ipairs(code_points) do
5497         table.insert(char_table, unicode.utf8.char(code_point))
5498     end
5499   return table.concat(char_table)
5500 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.

```
5501 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.

```
5502 function M.writer.new(options)
5503   local self = {}
```

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

```
5504   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.

```
5505   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.

```
5506  local slice_specifiers = {}
5507  for specifier in options.slice:gmatch("[^%s]+") do
5508    table.insert(slice_specifiers, specifier)
5509  end
5510
5511  if #slice_specifiers == 2 then
5512    self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5513    local slice_begin_type = self.slice_begin:sub(1, 1)
5514    if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5515      self.slice_begin = "^" .. self.slice_begin
5516    end
5517    local slice_end_type = self.slice_end:sub(1, 1)
5518    if slice_end_type ~= "^" and slice_end_type ~= "$" then
5519      self.slice_end = "$" .. self.slice_end
5520    end
5521  elseif #slice_specifiers == 1 then
5522    self.slice_begin = "^" .. slice_specifiers[1]
5523    self.slice_end = "$" .. slice_specifiers[1]
5524  end
5525
5526  self.slice_begin_type = self.slice_begin:sub(1, 1)
5527  self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5528  self.slice_end_type = self.slice_end:sub(1, 1)
5529  self.slice_end_identifier = self.slice_end:sub(2) or ""
5530
5531  if self.slice_begin == "^" and self.slice_end ~= "^" then
5532    self.is_writing = true
5533  else
5534    self.is_writing = false
5535  end
```

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

```
5536  self.suffix = ".tex"
```

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

```
5537  self.space = " "
```

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

```
5538  self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5539  function self.plain(s)
5540    return s
5541  end
```

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

```
function self.paragraph(s)
  if not self.is_writing then return "" end
  return s
end
```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

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

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

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

```
function self.paragraphsep()
  if not self.is_writing then return "" end
  return "\\markdownRendererParagraphSeparator\n{}"
end
```

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

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

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

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

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

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

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

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

Define tables `writer->escaped_uri_chars` and `writer->escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

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

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

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

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

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

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

```
local 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)
```

```
5603      end
5604    end
5605    local escape_typographic_text = create_escaper(
5606      self.escaped_chars, self.escaped_strings)
5607    local escape_programmatic_text = create_escaper(
5608      self.escaped_uri_chars, self.escaped_minimal_strings)
5609    local escape_minimal = create_escaper(
5610      {}, 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.

```
5611    self.escape = escape_typographic_text
5612    self.math = escape_minimal
5613    if options.hybrid then
5614      self.identifier = escape_minimal
5615      self.string = escape_minimal
5616      self.uri = escape_minimal
5617      self.infostring = escape_minimal
5618    else
5619      self.identifier = escape_programmatic_text
5620      self.string = escape_typographic_text
5621      self.uri = escape_programmatic_text
5622      self.infostring = escape_programmatic_text
5623    end
```

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

```
5624    function self.code(s, attributes)
5625      if self.flatten_inlines then return s end
5626      local buf = {}
5627      if attributes ~= nil then
5628        table.insert(buf,
5629                     "\\markdownRendererCodeSpanAttributeContextBegin\n")
5630        table.insert(buf, self.attributes(attributes))
5631      end
5632      table.insert(buf,
5633                   {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
5634      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 = {}
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererImageAttributeContextBegin\n")
      table.insert(buf, self.attributes(attributes))
    end
    table.insert(buf, {"\\markdownRendererImage{",lab,"}",
                       "{",self.string(src),"}",
                       "{",self.uri(src),"}",
                       "{",self.string(tit or ""),"}"})
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererImageAttributeContextEnd{}")
    end
```

```
5674      return buf
5675    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.

```
5676    function self.bulletlist(items,tight)
5677      if not self.is_writing then return "" end
5678      local buffer = {}
5679      for _,item in ipairs(items) do
5680        if item ~= "" then
5681          buffer[#buffer + 1] = self.bulletitem(item)
5682        end
5683      end
5684      local contents = util.intersperse(buffer,"\n")
5685      if tight and options.tightLists then
5686        return {"\\markdownRendererUlBeginTight\n",contents,
5687          "\n\\markdownRendererUlEndTight "}
5688      else
5689        return {"\\markdownRendererUlBegin\n",contents,
5690          "\n\\markdownRendererUlEnd "}
5691      end
5692    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.

```
5693    function self.bulletitem(s)
5694      return {"\\markdownRendererUlItem ",s,
5695              "\\markdownRendererUlItemEnd "}
5696    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.

```
5697    function self.orderedlist(items,tight,startnum)
5698      if not self.is_writing then return "" end
5699      local buffer = {}
5700      local num = startnum
5701      for _,item in ipairs(items) do
5702        if item ~= "" then
5703          buffer[#buffer + 1] = self.ordereditem(item,num)
5704        end
5705        if num ~= nil and item ~= "" then
5706          num = num + 1
5707        end
5708      end
5709      local contents = util.intersperse(buffer,"\n")
```

```
5710    if tight and options.tightLists then
5711      return {"\\markdownRendererOlBeginTight\n",contents,
5712              "\n\\markdownRendererOlEndTight "}
5713    else
5714      return {"\\markdownRendererOlBegin\n",contents,
5715              "\n\\markdownRendererOlEnd "}
5716    end
5717  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.

```
5718  function self.ordereditem(s,num)
5719    if num ~= nil then
5720      return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
5721              "\\markdownRendererOlItemEnd "}
5722    else
5723      return {"\\markdownRendererOlItem ",s,
5724              "\\markdownRendererOlItemEnd "}
5725    end
5726  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.

```
5727  function self.inline_html_comment(contents)
5728    if self.flatten_inlines then return contents end
5729    return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
5730  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.

```
5731  function self.inline_html_tag(contents)
5732    if self.flatten_inlines then return contents end
5733    return {"\\markdownRendererInlineHtmlTag{",self.string(contents),"}"}
5734  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.

```
5735  function self.block_html_element(s)
5736    if not self.is_writing then return "" end
5737    local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5738    return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
5739  end
```

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

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

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

```
function self.tickbox(f)
  if f == 1.0 then
    return "☒ "
  elseif f == 0.0 then
    return "▫ "
  else
    return "⊡ "
  end
end
```

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

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

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

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

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

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

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

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

```

```
5771      -- pop all attributes
5772      table.insert(buf, self.pop_attributes())
5773
5774      table.insert(buf, "\\markdownRendererDocumentEnd")
5775
5776      return buf
5777    end
```

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

```
5778    local seen_identifiers = {}
5779    local key_value_regex = "([^= ]+)%s*=%s*(.*)"
5780    local function normalize_attributes(attributes, auto_identifiers)
5781      -- normalize attributes
5782      local normalized_attributes = {}
5783      local has_explicit_identifiers = false
5784      local key, value
5785      for _, attribute in ipairs(attributes or {}) do
5786        if attribute:sub(1, 1) == "#" then
5787          table.insert(normalized_attributes, attribute)
5788          has_explicit_identifiers = true
5789          seen_identifiers[attribute:sub(2)] = true
5790        elseif attribute:sub(1, 1) == "." then
5791          table.insert(normalized_attributes, attribute)
5792        else
5793          key, value = attribute:match(key_value_regex)
5794          if key:lower() == "id" then
5795            table.insert(normalized_attributes, "#" .. value)
5796          elseif key:lower() == "class" then
5797            local classes = {}
5798            for class in value:gmatch("%S+") do
5799              table.insert(classes, class)
5800            end
5801            table.sort(classes)
5802            for _, class in ipairs(classes) do
5803              table.insert(normalized_attributes, "." .. class)
5804            end
5805          else
5806            table.insert(normalized_attributes, attribute)
5807          end
5808        end
5809      end
5810
5811      -- if no explicit identifiers exist, add auto identifiers
5812      if not has_explicit_identifiers and auto_identifiers ~= nil then
5813        local seen_auto_identifiers = {}
5814        for _, auto_identifier in ipairs(auto_identifiers) do
```

```
        if seen_auto_identifiers[auto_identifier] == nil then
          seen_auto_identifiers[auto_identifier] = true
          if seen_identifiers[auto_identifier] == nil then
            seen_identifiers[auto_identifier] = true
            table.insert(normalized_attributes,
                         "#" .. auto_identifier)
          else
            local auto_identifier_number = 1
            while true do
              local numbered_auto_identifier = auto_identifier .. "-"
                                               .. auto_identifier_number
              if seen_identifiers[numbered_auto_identifier] == nil then
                seen_identifiers[numbered_auto_identifier] = true
                table.insert(normalized_attributes,
                             "#" .. numbered_auto_identifier)
                break
              end
              auto_identifier_number = auto_identifier_number + 1
            end
          end
        end
      end
    end

    -- sort and deduplicate normalized attributes
    table.sort(normalized_attributes)
    local seen_normalized_attributes = {}
    local deduplicated_normalized_attributes = {}
    for _, attribute in ipairs(normalized_attributes) do
      if seen_normalized_attributes[attribute] == nil then
        seen_normalized_attributes[attribute] = true
        table.insert(deduplicated_normalized_attributes, attribute)
      end
    end

    return deduplicated_normalized_attributes
  end

  function self.attributes(attributes, should_normalize_attributes)
    local normalized_attributes
    if should_normalize_attributes == false then
      normalized_attributes = attributes
    else
      normalized_attributes = normalize_attributes(attributes)
    end

    local buf = {}
```

```
5862    local key, value
5863    for _, attribute in ipairs(normalized_attributes) do
5864      if attribute:sub(1, 1) == "#" then
5865        table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
5866                           attribute:sub(2), "}"})
5867      elseif attribute:sub(1, 1) == "." then
5868        table.insert(buf, {"\\markdownRendererAttributeClassName{",
5869                           attribute:sub(2), "}"})
5870      else
5871        key, value = attribute:match(key_value_regex)
5872        table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
5873                           key, "}{", value, "}"})
5874      end
5875    end
5876
5877    return buf
5878  end
```

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

```
5879  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`.

```
5880  self.attribute_type_levels = {}
5881  setmetatable(self.attribute_type_levels,
5882               { __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`.

```
5883  local function apply_attributes()
5884    local buf = {}
5885    for i = 1, #self.active_attributes do
5886      local start_output = self.active_attributes[i][3]
5887      if start_output ~= nil then
5888        table.insert(buf, start_output)
5889      end
5890    end
5891    return buf
5892  end
5893
5894  local function tear_down_attributes()
5895    local buf = {}
5896    for i = #self.active_attributes, 1, -1 do
5897      local end_output = self.active_attributes[i][4]
5898      if end_output ~= nil then
5899        table.insert(buf, end_output)
```

```
      end
    end
    return buf
  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.

```
  function self.push_attributes(attribute_type, attributes,
                                start_output, end_output)
    local attribute_type_level = self.attribute_type_levels[attribute_type]
    self.attribute_type_levels[attribute_type] = attribute_type_level + 1

    -- index attributes in a hash table for easy lookup
    attributes = attributes or {}
    for i = 1, #attributes do
      attributes[attributes[i]] = true
    end

    local buf = {}
    -- handle slicing
    if attributes["#" .. self.slice_end_identifier] ~= nil and
       self.slice_end_type == "^" then
      if self.is_writing then
        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
      table.insert(buf, apply_attributes())
      self.is_writing = true
    end
    if self.is_writing and start_output ~= nil then
      table.insert(buf, start_output)
    end
    table.insert(self.active_attributes,
                 {attribute_type, attributes,
                  start_output, end_output})
    return buf
  end

```

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).

```
function self.pop_attributes(attribute_type)
  local buf = {}
  -- pop attributes until we find attributes of correct type
  -- or until no attributes remain
  local current_attribute_type = false
  while current_attribute_type ~= attribute_type and
        #self.active_attributes > 0 do
    local attributes, _, end_output
    current_attribute_type, attributes, _, end_output = table.unpack(
      self.active_attributes[#self.active_attributes])
    local attribute_type_level = 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 == "$" then
      if self.is_writing then
        table.insert(buf, tear_down_attributes())
      end
      self.is_writing = false
    end
    if attributes["#" .. self.slice_begin_identifier] ~= nil and
       self.slice_begin_type == "$" 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

  for _, code in utf8.codes(uni_algos.normalize.NFC(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 perio
    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
```

```
6023
6024    for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
6025      local char = utf8.char(code)
6026
6027      -- Remove everything up to the first non-space.
6028      if not letter_found then
6029        local is_letter = unicode.utf8.match(char, "%S")
6030        if is_letter then
6031          letter_found = true
6032        else
6033          goto continue
6034        end
6035      end
6036
6037      -- Remove all non-alphanumeric characters, except underscores and hyphens.
6038      if not unicode.utf8.match(char, "[%w_%-%s]") then
6039        prev_space = false
6040        goto continue
6041      end
6042
6043      -- Replace all spaces and newlines with hyphens.
6044      if unicode.utf8.match(char, "[%s\n]") then
6045        char = "-"
6046        if prev_space then
6047          goto continue
6048        else
6049          prev_space = true
6050        end
6051      else
6052        -- Convert all alphabetic characters to lowercase.
6053        char = unicode.utf8.lower(char)
6054        prev_space = false
6055      end
6056
6057      table.insert(buffer, char)
6058
6059      ::continue::
6060    end
6061
6062    if prev_space then
6063      table.remove(buffer)
6064    end
6065
6066    local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6067    return identifier
6068  end
```

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

```
6069  function self.heading(s, level, attributes)
6070    local buf = {}
6071    local flat_text, inlines = table.unpack(s)
6072
6073    -- push empty attributes for implied sections
6074    while self.attribute_type_levels["heading"] < level - 1 do
6075      table.insert(buf,
6076                   self.push_attributes("heading",
6077                                        nil,
6078                                        "\\markdownRendererSectionBegin\n",
6079                                        "\n\\markdownRendererSectionEnd "))
6080    end
6081
6082    -- pop attributes for sections that have ended
6083    while self.attribute_type_levels["heading"] >= level do
6084      table.insert(buf, self.pop_attributes("heading"))
6085    end
6086
6087    -- construct attributes for the new section
6088    local auto_identifiers = {}
6089    if self.options.autoIdentifiers then
6090      table.insert(auto_identifiers, create_auto_identifier(flat_text))
6091    end
6092    if self.options.gfmAutoIdentifiers then
6093      table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))
6094    end
6095    local normalized_attributes = normalize_attributes(attributes, auto_identifiers)
6096
6097    -- push attributes for the new section
6098    local start_output = {}
6099    local end_output = {}
6100    table.insert(start_output, "\\markdownRendererSectionBegin\n")
6101    table.insert(end_output, "\n\\markdownRendererSectionEnd ")
6102
6103    table.insert(buf, self.push_attributes("heading",
6104                                           normalized_attributes,
6105                                           start_output,
6106                                           end_output))
6107    assert(self.attribute_type_levels["heading"] == level)
6108
6109    -- render the heading and its attributes
6110    if self.is_writing and #normalized_attributes > 0 then
6111      table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\n")
6112      table.insert(buf, self.attributes(normalized_attributes, false))
6113    end
```

```
6114
6115      local cmd
6116      level = level + options.shiftHeadings
6117      if level <= 1 then
6118        cmd = "\\markdownRendererHeadingOne"
6119      elseif level == 2 then
6120        cmd = "\\markdownRendererHeadingTwo"
6121      elseif level == 3 then
6122        cmd = "\\markdownRendererHeadingThree"
6123      elseif level == 4 then
6124        cmd = "\\markdownRendererHeadingFour"
6125      elseif level == 5 then
6126        cmd = "\\markdownRendererHeadingFive"
6127      elseif level >= 6 then
6128        cmd = "\\markdownRendererHeadingSix"
6129      else
6130        cmd = ""
6131      end
6132      if self.is_writing then
6133        table.insert(buf, {cmd, "{", inlines, "}"})
6134      end
6135
6136      if self.is_writing and #normalized_attributes > 0 then
6137        table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd ")
6138      end
6139
6140      return buf
6141    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.

```
6142    function self.get_state()
6143      return {
6144        is_writing=self.is_writing,
6145        flatten_inlines=self.flatten_inlines,
6146        active_attributes={table.unpack(self.active_attributes)},
6147      }
6148    end
```

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

```
6149    function self.set_state(s)
6150      local previous_state = self.get_state()
6151      for key, value in pairs(s) do
6152        self[key] = value
6153      end
6154      return previous_state
6155    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.internal_punctuation = S(":;,.?")

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

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

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

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

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

3.1.4.2 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
```

```
6273   return indent_table
6274 end
6275
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
6276 local function update_indent_table(indent_table, new_indent, add)
6277   indent_table = remove_trail(indent_table)
6278
6279   if not has_indents(indent_table) then
6280     indent_table.indents = {}
6281   end
6282
6283
6284   if add then
6285     indent_table.indents[#indent_table.indents + 1] = new_indent
6286   else
6287     if indent_table.indents[#indent_table.indents].name == new_indent.name then
6288       indent_table.indents[#indent_table.indents] = nil
6289     end
6290   end
6291
6292   return indent_table
6293 end
6294
```

Remove an indent by its name `name`.

```
6295 local function remove_indent(name)
6296   local function remove_indent_level(s, i, indent_table) -- luacheck: ignore s i
6297     indent_table = update_indent_table(indent_table, {name=name}, false)
6298     return true, indent_table
6299   end
6300
6301   return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6302 end
6303
```

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`.

```
6304 local function process_starter_spacing(indent, spacing, minimum, left_strip_length)
6305   left_strip_length = left_strip_length or 0
6306
6307   local count = 0
6308   local tab_value = 4 - (indent) % 4
6309
6310   local code_started, minimum_found = false, false
```

```
6311  local code_start, minimum_remainder = "", ""
6312
6313  local left_total_stripped = 0
6314  local full_remainder = ""
6315
6316  if spacing ~= nil then
6317    for i = 1, #spacing do
6318      local character = spacing:sub(i, i)
6319
6320      if character == "\t" then
6321        count = count + tab_value
6322        tab_value = 4
6323      elseif character == " " then
6324        count = count + 1
6325        tab_value = 4 - (1 - tab_value) % 4
6326      end
6327
6328      if (left_strip_length ~= 0) then
6329        local possible_to_strip = math.min(count, left_strip_length)
6330        count = count - possible_to_strip
6331        left_strip_length = left_strip_length - possible_to_strip
6332        left_total_stripped = left_total_stripped + possible_to_strip
6333      else
6334        full_remainder =  full_remainder .. character
6335      end
6336
6337      if (minimum_found) then
6338        minimum_remainder = minimum_remainder .. character
6339      elseif (count >= minimum) then
6340        minimum_found = true
6341        minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6342      end
6343
6344      if (code_started) then
6345        code_start = code_start .. character
6346      elseif (count >= minimum + 4) then
6347        code_started = true
6348        code_start = code_start .. string.rep(" ", count - (minimum + 4))
6349      end
6350    end
6351  end
6352
6353  local remainder
6354  if (code_started) then
6355    remainder = code_start
6356  else
6357    remainder = string.rep(" ", count - minimum)
```

```
  end

  local is_minimum = count >= minimum
  return {
    is_code = code_started,
    remainder = remainder,
    left_total_stripped = left_total_stripped,
    is_minimum = is_minimum,
    minimum_remainder = minimum_remainder,
    total_length = count,
    full_remainder = full_remainder
  }
end

```

Count the total width of all indents in the indent table `indent_table`.

```
local function count_indent_tab_level(indent_table)
  local count = 0
  if not has_indents(indent_table) then
    return count
  end

  for i=1, #indent_table.indents do
    count = count + indent_table.indents[i].length
  end
  return count
end

```

Count the total width of a delimiter `delimiter`.

```
local function total_delimiter_length(delimiter)
  local count = 0
  if type(delimiter) == "string" then return #delimiter end
  for _, value in pairs(delimiter) do
    count = count + total_delimiter_length(value)
  end
  return count
end

```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```
local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
  local last_trail = starter[1]
  local delimiter = starter[2]
  local raw_new_trail = starter[3]

  if indent_type == "bq" and not breakable then
```

```
6399      indent_table.ignore_blockquote_blank = true
6400    end
6401
6402    if has_trail(indent_table) then
6403      local trail = indent_table.trail
6404      if trail.is_code then
6405        return false
6406      end
6407      last_trail = trail.remainder
6408    else
6409      local sp = process_starter_spacing(0, last_trail, 0, 0)
6410
6411      if sp.is_code then
6412        return false
6413      end
6414      last_trail = sp.remainder
6415    end
6416
6417    local preceding_indentation = count_indent_tab_level(indent_table) % 4
6418    local last_trail_length = #last_trail
6419    local delimiter_length = total_delimiter_length(delimiter)
6420
6421    local total_indent_level = preceding_indentation + last_trail_length + delimiter_le
6422
6423    local sp = {}
6424    if not is_blank then
6425      sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6426    end
6427
6428    local del_trail_length = sp.left_total_stripped
6429    if is_blank then
6430      del_trail_length = 1
6431    elseif not sp.is_code then
6432      del_trail_length = del_trail_length + #sp.remainder
6433    end
6434
6435    local indent_length = last_trail_length + delimiter_length + del_trail_length
6436    local new_indent_info = {name=indent_type, length=indent_length}
6437
6438    indent_table = update_indent_table(indent_table, new_indent_info, true)
6439    indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,
6440                                            full_remainder=sp.full_remainder})
6441
6442    return true, indent_table
6443  end
6444
```

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)
  local new_index = i

  local preceding_indentation = 0
  local current_trail = {}

  local blank_starter = left_blank_starter(indent_table)

  for index = 1,#indent_table.indents do
```

```
6482    local value = indent_table.indents[index]
6483    local pattern = decode_pattern(value.name)
6484
6485    -- match decoded pattern
6486    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6487    if new_indent_info == nil then
6488      local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, ne
6489      if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end
6490        return is_optional, new_index, current_trail
6491      end
6492
6493      return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optiona
6494    end
6495
6496    local raw_last_trail = new_indent_info[1]
6497    local delimiter = new_indent_info[2]
6498    local raw_new_trail = new_indent_info[3]
6499    local next_index = new_indent_info[4]
6500
6501    local space_only = delimiter == ""
6502
6503    -- check previous trail
6504    if not space_only and next(current_trail) == nil then
6505      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6506      current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.to
6507                       full_remainder=sp.full_remainder}
6508    end
6509
6510    if next(current_trail) ~= nil then
6511      if not space_only and current_trail.is_code then
6512        return is_optional, new_index, current_trail
6513      end
6514      if current_trail.internal_remainder ~= nil then
6515        raw_last_trail = current_trail.internal_remainder
6516      end
6517    end
6518
6519    local raw_last_trail_length = 0
6520    local delimiter_length = 0
6521
6522    if not space_only then
6523      delimiter_length = #delimiter
6524      raw_last_trail_length = #raw_last_trail
6525    end
6526
6527    local total_indent_level = preceding_indentation + raw_last_trail_length + delimi
6528
```

```
    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, minimu

    if space_only and not sp.is_minimum then
      return is_optional or (is_blank and blank_starter <= index), new_index, current
    end

    local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_st

    -- 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=s
                     total_length=sp.total_length, full_remainder=sp.full_remainder}
    new_index = next_index
  end

  return true, new_index, current_trail
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 function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_rema
  local is_code
  local remainder
```

```
6571
6572   if has_trail(indent_table) then
6573     local trail = indent_table.trail
6574     is_code = trail.is_code
6575     if is_code then
6576       remainder = trail.remainder
6577     else
6578       remainder = trail.full_remainder
6579     end
6580   else
6581     local sp = process_starter_spacing(0, spacing, 0, 0)
6582     is_code = sp.is_code
6583     if is_code then
6584       remainder = sp.remainder
6585     else
6586       remainder = sp.full_remainder
6587     end
6588   end
6589
6590   local result = check_trail(expect_code, is_code)
6591   if omit_remainder then
6592     return result
6593   end
6594   return result, remainder
6595 end
6596
```

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

```
6597 local function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:
6598   local trail
6599
6600   if has_trail(indent_table) then
6601     trail = indent_table.trail
6602   else
6603     trail = process_starter_spacing(0, spacing, 0, 0)
6604   end
6605
6606   local total_length = trail.total_length
6607   if total_length == nil then
6608     return false
6609   end
6610
6611   return min <= total_length and total_length <= max
6612 end
6613
```

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

```
local function check_continuation_indentation(s, i, indent_table, is_optional, is_bla
  if not has_indents(indent_table) then
    return true
  end

  local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt

  if passes then
    indent_table = add_trail(indent_table, current_trail)
    return new_index, indent_table
  end
  return false
end

```

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

```
local function get_last_indent_name(indent_table)
  if has_indents(indent_table) then
    return indent_table.indents[#indent_table.indents].name
  end
end

```

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

```
local function remove_remainder_if_blank(indent_table, remainder)
  if get_last_indent_name(indent_table) == "li" then
    return ""
  end
  return remainder
end

```

Take the trail `trail` or create a new one from `spacing` and comapre it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```
local function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore
  if trail == nil then
    trail = process_starter_spacing(0, spacing, 0, 0)
  end

  if trail_type == "non-code" then
    return check_trail(false, trail.is_code)
  end
  if trail_type == "code" then
    return check_trail(true, trail.is_code)
  end
  if trail_type == "full-code" then
    if (trail.is_code) then
```

```
6654        return i, trail.remainder
6655      end
6656      return i, ""
6657    end
6658    if trail_type == "full-any" then
6659      return i, trail.internal_remainder
6660    end
6661  end
6662
```

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

```
6663  local function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
6664    if is_freezing then
6665      if indent_table.is_trail_frozen then
6666        indent_table.trail = indent_table.frozen_trail
6667      else
6668        indent_table.frozen_trail = indent_table.trail
6669        indent_table.is_trail_frozen = true
6670      end
6671    else
6672      indent_table.frozen_trail = nil
6673      indent_table.is_trail_frozen = false
6674    end
6675    return true, indent_table
6676  end
6677
```

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`.

```
6678  local function check_continuation_indentation_and_trail(s, i, indent_table, is_option
6679                                                           reset_rem, omit_remainder)
6680    if not has_indents(indent_table) then
6681      local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6682      local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, tra
6683      if remainder == nil then
6684        if result then
6685          return new_index
6686        end
6687        return false
6688      end
6689      if result then
6690        return new_index, remainder
6691      end
6692      return false
6693    end
6694
6695    local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt
```

```
6696
6697   if passes then
6698     local spacing
6699     if current_trail == nil then
6700       local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s,
6701       current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6702       new_index = newer_index
6703       spacing = newer_spacing
6704     else
6705       spacing = current_trail.remainder
6706     end
6707     local result, remainder = check_trail_type(s, new_index, current_trail, spacing,
6708     if remainder == nil or omit_remainder then
6709       if result then
6710         return new_index
6711       end
6712       return false
6713     end
6714
6715     if is_blank and reset_rem then
6716       remainder = remove_remainder_if_blank(indent_table, remainder)
6717     end
6718     if result then
6719       return new_index, remainder
6720     end
6721     return false
6722   end
6723   return false
6724 end
6725
```

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

```
6726 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(false), che
6727
6728 parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals
6729
6730 parsers.check_code_trail  = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)
6731
6732 parsers.check_trail_length_range  = function(min, max)
6733   return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
6734 end
6735
6736 parsers.check_trail_length = function(n)
6737   return parsers.check_trail_length_range(n, n)
6738 end
6739
```

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_

parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde

```

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_

parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_

parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch

```

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")
                                                 check_continuation_indentation_and_t

parsers.check_minimal_blank_indent_and_full_code_trail = Cmt( Cb("indent_info")
                                                             * Cc(false) * Cc(true) *
    code") * Cc(true),
                                                             check_continuation_indent

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_tr

parsers.check_minimal_blank_indent_and_any_trail = Cmt( Cb("indent_info")
                                                      * Cc(false) * Cc(true) * Cc("fu
    any") * Cc(true) * Cc(false),
                                                       check_continuation_indentation

parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt( Cb("indent_info")
                                                         * Cc(false) * Cc(true) * Cc("
    any") * Cc(true) * Cc(true),
                                                         check_continuation_indentatio

```

```
6774
6775 parsers.check_optional_indent_and_any_trail = Cmt( Cb("indent_info")
6776                                                  * Cc(true) * Cc(false) * Cc("full-
     any") * Cc(true) * Cc(false),
6777                                                  check_continuation_indentation_and_tr
6778
6779 parsers.check_optional_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6780                                                       * Cc(true) * Cc(true) * Cc("ful
     any") * Cc(true) * Cc(false),
6781                                                        check_continuation_indentation
6782
```

The following patterns specify behaviour around newlines.

```
6783
6784 parsers.spnlc_noexc = parsers.optionalspace
6785                     * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
     1
6786
6787 parsers.spnlc = parsers.optionalspace
6788               * (V("EndlineNoSub"))^-1
6789
6790 parsers.spnlc_sep  = parsers.optionalspace * V("EndlineNoSub")
6791                    + parsers.spacechar^1
6792
6793 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6794
6795 %   \end{macrocode}
6796 % \begin{figure}
6797 % \hspace*{-0.1\textwidth}
6798 % \begin{minipage}{1.2\textwidth}
6799 % \centering
6800 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6801 % \node[state, initial by diamond, accepting] (noop) {initial};
6802 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
6803 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
6804 % \node[state] (comment) [below=of noop] {comment};
6805 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs
6806 % \node[state] (blank_line) [below right=of comment] {blank line};
6807 % \path[->]
6808 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^\
6809 %        edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
6810 %        edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
6811 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge$$\drsh
6812 %           edge [bend left=10] node {match $\drsh$} (leading_spaces)
6813 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$
6814 %                  edge [bend right=90] node [right] {match \textbackslash} (odd_back
6815 %                  edge [bend left=10] node {match \%} (comment)
6816 %                  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 tabs and spac
                                  + #(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.rbra
                                    - 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
```

3.1.4.3 Parsers Used for HTML Entities

```
local function repeat_between(pattern, min, max)
  return -pattern^(max + 1) * pattern^min
end

parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
                  * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
parsers.decentity = parsers.ampersand * parsers.hash
                  * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
                  * parsers.semicolon

parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
                      + parsers.decentity / entities.dec_entity
                      + parsers.tagentity / entities.char_entity
```

3.1.4.4 Parsers Used for Markdown Lists

```
parsers.bullet = function(bullet_char, interrupting)
  local allowed_end
  if interrupting then
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
    allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
  end
  return parsers.check_trail
```

```
6952         * Ct(C(bullet_char) * Cc(""))
6953         * allowed_end
6954 end
6955
6956 local function tickbox(interior)
6957   return parsers.optionalspace * parsers.lbracket
6958         * interior * parsers.rbracket * parsers.spacechar^1
6959 end
6960
6961 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
6962 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
6963 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
6964
```

3.1.4.5 Parsers Used for Markdown Code Spans

```
6965 parsers.openticks   = Cg(parsers.backtick^1, "ticks")
6966
6967 local function captures_equal_length(_,i,a,b)
6968   return #a == #b and i
6969 end
6970
6971 parsers.closeticks  = Cmt(C(parsers.backtick^1)
6972                           * Cb("ticks"), captures_equal_length)
6973
6974 parsers.intickschar = (parsers.any - S("\n\r`"))
6975                     + V("NoSoftLineBreakEndline")
6976                     + (parsers.backtick^1 - parsers.closeticks)
6977
6978 local function process_inticks(s)
6979   s = s:gsub("\n", " ")
6980   s = s:gsub("^ (.*) $", "%1")
6981   return s
6982 end
6983
6984 parsers.inticks = parsers.openticks
6985                 * C(parsers.space^0)
6986                 * parsers.closeticks
6987                 + parsers.openticks
6988                 * Cs(Cs(parsers.intickschar^0) / process_inticks)
6989                 * parsers.closeticks
6990
```

3.1.4.6 Parsers Used for HTML

```
6991 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
6992 parsers.keyword_exact = function(s)
6993   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") +
```

```
7041      parsers.keyword_exact("header") +
7042      parsers.keyword_exact("hr") +
7043      parsers.keyword_exact("html") +
7044      parsers.keyword_exact("iframe") +
7045      parsers.keyword_exact("legend") +
7046      parsers.keyword_exact("li") +
7047      parsers.keyword_exact("link") +
7048      parsers.keyword_exact("main") +
7049      parsers.keyword_exact("menu") +
7050      parsers.keyword_exact("menuitem") +
7051      parsers.keyword_exact("nav") +
7052      parsers.keyword_exact("noframes") +
7053      parsers.keyword_exact("ol") +
7054      parsers.keyword_exact("optgroup") +
7055      parsers.keyword_exact("option") +
7056      parsers.keyword_exact("p") +
7057      parsers.keyword_exact("param") +
7058      parsers.keyword_exact("section") +
7059      parsers.keyword_exact("source") +
7060      parsers.keyword_exact("summary") +
7061      parsers.keyword_exact("table") +
7062      parsers.keyword_exact("tbody") +
7063      parsers.keyword_exact("td") +
7064      parsers.keyword_exact("tfoot") +
7065      parsers.keyword_exact("th") +
7066      parsers.keyword_exact("thead") +
7067      parsers.keyword_exact("title") +
7068      parsers.keyword_exact("tr") +
7069      parsers.keyword_exact("track") +
7070      parsers.keyword_exact("ul")
7071
7072  -- end conditions
7073  parsers.html_blankline_end_condition  = parsers.linechar^0
7074                                        * ( parsers.newline
7075                                          * (parsers.check_minimal_blank_indent_and_any
7076                                            * #parsers.blankline
7077                                            + parsers.check_minimal_indent_and_any_trai
7078                                          * parsers.linechar^1)^0
7079                                        * (parsers.newline^-1 / "")
7080
7081  local function remove_trailing_blank_lines(s)
7082    return s:gsub("[\n\r]+%s*$", "")
7083  end
7084
7085  parsers.html_until_end = function(end_marker)
7086    return Cs(Cs((parsers.newline
7087             * (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.undersc
                            + 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.newl
                                      - parsers.dquote - parsers.squote - parsers.bac
                                      - parsers.equal - parsers.less - parsers.more)^

parsers.html_attribute_value_specification  = parsers.optionalspace
                                            * parsers.equal
                                            * parsers.optionalspace
                                            * parsers.html_attribute_value
```

```
7135
7136 parsers.html_spnl = parsers.optionalspace
7137                   * (V("NoSoftLineBreakEndline") * parsers.optionalspace)^-
     1
7138
7139 parsers.html_inline_attribute_value_specification = parsers.html_spnl
7140                                                   * parsers.equal
7141                                                   * parsers.html_spnl
7142                                                   * parsers.html_inline_attribute_val
7143
7144 parsers.html_attribute  = parsers.html_attribute_spacing
7145                         * parsers.html_attribute_name
7146                         * parsers.html_inline_attribute_value_specification^-
     1
7147
7148 parsers.html_non_newline_attribute  = parsers.spacechar^1
7149                                     * parsers.html_attribute_name
7150                                     * parsers.html_attribute_value_specification^-
     1
7151
7152 parsers.nested_breaking_blank = parsers.newline
7153                               * parsers.check_minimal_blank_indent
7154                               * parsers.blankline
7155
7156 parsers.html_comment_start = P("<!--")
7157
7158 parsers.html_comment_end = P("-->")
7159
7160 parsers.html_comment = Cs( parsers.html_comment_start
7161                          * parsers.html_until_end(parsers.html_comment_end))
7162
7163 parsers.html_inline_comment = (parsers.html_comment_start / "")
7164                             * -P(">") * -P("->")
7165                             * Cs((V("NoSoftLineBreakEndline") + parsers.any - P("--
     ")
7166                                 - parsers.nested_breaking_blank - parsers.html_commen
7167                             * (parsers.html_comment_end / "")
7168
7169 parsers.html_cdatasection_start = P("<![CDATA[")
7170
7171 parsers.html_cdatasection_end = P("]]>")
7172
7173 parsers.html_cdatasection = Cs( parsers.html_cdatasection_start
7174                               * parsers.html_until_end(parsers.html_cdatasection_end)
7175
7176 parsers.html_inline_cdatasection  = parsers.html_cdatasection_start
7177                                   * Cs(V("NoSoftLineBreakEndline") + parsers.any
```

```
                                    - parsers.nested_breaking_blank - parsers.html_
                                  * parsers.html_cdatasection_end

parsers.html_declaration_start = P("<!") * parsers.letter

parsers.html_declaration_end = P(">")

parsers.html_declaration  = Cs( parsers.html_declaration_start
                              * parsers.html_until_end(parsers.html_declaration_end))

parsers.html_inline_declaration = parsers.html_declaration_start
                                * Cs(V("NoSoftLineBreakEndline") + parsers.any
                                    - parsers.nested_breaking_blank - parsers.html_de
                                * 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_in
                                * 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
```

```
7272
7273 parsers.html_any_close_tag  = parsers.html_tag_closing_start
7274                             * parsers.keyword
7275                             * parsers.html_tag_end
7276
7277 parsers.html_close_special_tag = parsers.html_tag_closing_start
7278                                * parsers.special_block_keyword
7279                                * parsers.html_tag_end
7280
7281 -- empty tags
7282 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7283                                   * parsers.keyword
7284                                   * parsers.html_attribute^0
7285                                   * parsers.html_empty_tag_end
7286
7287 parsers.html_any_empty_tag  = parsers.html_tag_start
7288                             * parsers.keyword
7289                             * parsers.html_non_newline_attribute^0
7290                             * parsers.optionalspace
7291                             * parsers.slash
7292                             * parsers.more
7293
7294 parsers.html_empty_tag  = parsers.html_tag_start
7295                         * parsers.block_keyword
7296                         * parsers.html_attribute^0
7297                         * parsers.html_empty_tag_end
7298
7299 parsers.html_empty_special_tag  = parsers.html_tag_start
7300                                 * parsers.special_block_keyword
7301                                 * parsers.html_attribute^0
7302                                 * parsers.html_empty_tag_end
7303
7304 parsers.html_incomplete_blocks  = parsers.html_incomplete_open_tag
7305                                 + parsers.html_incomplete_open_special_tag
7306                                 + parsers.html_incomplete_close_tag
7307
7308 -- parse special html blocks
7309 parsers.html_blankline_ending_special_block_opening = (parsers.html_close_special_tag
7310                                                        + parsers.html_empty_special_ta
7311                                                      * #(parsers.optionalspace
7312                                                         * (parsers.newline + parsers.e
7313
7314 parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_b
7315                                             * parsers.html_blankline_end_condition
7316
7317 parsers.html_special_block_opening  = parsers.html_incomplete_open_special_tag
7318                                     - parsers.html_empty_special_tag
```

```
7319
7320 parsers.html_closing_special_block  = parsers.html_special_block_opening
7321                                     * parsers.html_until_end(parsers.html_close_speci
7322
7323 parsers.html_special_block  = parsers.html_blankline_ending_special_block
7324                             + parsers.html_closing_special_block
7325
7326 -- parse html blocks
7327 parsers.html_block_opening  = parsers.html_incomplete_open_tag
7328                             + parsers.html_incomplete_close_tag
7329
7330 parsers.html_block  = parsers.html_block_opening
7331                     * parsers.html_blankline_end_condition
7332
7333 -- parse any html blocks
7334 parsers.html_any_block_opening  = (parsers.html_any_open_tag
7335                                 + parsers.html_any_close_tag
7336                                 + parsers.html_any_empty_tag)
7337                                 * #(parsers.optionalspace * (parsers.newline + parser
7338
7339 parsers.html_any_block  = parsers.html_any_block_opening
7340                         * parsers.html_blankline_end_condition
7341
7342 parsers.html_inline_comment_full  = parsers.html_comment_start
7343                                   * -P(">") * -P("->")
7344                                   * Cs((V("NoSoftLineBreakEndline") + parsers.any - P
    ")
7345                                       - parsers.nested_breaking_blank - parsers.html_
7346                                   * parsers.html_comment_end
7347
7348 parsers.html_inline_tags  = parsers.html_inline_comment_full
7349                           + parsers.html_any_empty_inline_tag
7350                           + parsers.html_inline_instruction
7351                           + parsers.html_inline_cdatasection
7352                           + parsers.html_inline_declaration
7353                           + parsers.html_any_open_inline_tag
7354                           + parsers.html_any_close_tag
7355
```

3.1.4.7 Parsers Used for Markdown Tags and Links

```
7356 parsers.urlchar = parsers.anyescaped
7357                 - parsers.newline
7358                 - parsers.more
7359
7360 parsers.auto_link_scheme_part = parsers.alphanumeric
7361                               + parsers.plus
```

```
7362                               + parsers.period
7363                               + parsers.dash
7364
7365 parsers.auto_link_scheme  = parsers.letter
7366                           * parsers.auto_link_scheme_part
7367                           * parsers.auto_link_scheme_part^-30
7368
7369 parsers.absolute_uri  = parsers.auto_link_scheme * parsers.colon
7370                       * (parsers.any - parsers.spacing - parsers.less - parsers.more)
7371
7372 parsers.printable_characters = S(".!#$%&'*+/=?^_`{|}~-")
7373
7374 parsers.email_address_local_part_char = parsers.alphanumeric
7375                                       + parsers.printable_characters
7376
7377 parsers.email_address_local_part = parsers.email_address_local_part_char^1
7378
7379 parsers.email_address_dns_label = parsers.alphanumeric
7380                                 * (parsers.alphanumeric + parsers.dash)^-
    62
7381                                 * B(parsers.alphanumeric)
7382
7383 parsers.email_address_domain  = parsers.email_address_dns_label
7384                               * (parsers.period * parsers.email_address_dns_label)^0
7385
7386 parsers.email_address = parsers.email_address_local_part
7387                       * parsers.at
7388                       * parsers.email_address_domain
7389
7390 parsers.auto_link_url = parsers.less
7391                       * C(parsers.absolute_uri)
7392                       * parsers.more
7393
7394 parsers.auto_link_email = parsers.less
7395                         * C(parsers.email_address)
7396                         * parsers.more
7397
7398 parsers.auto_link_relative_reference = parsers.less
7399                                      * C(parsers.urlchar^1)
7400                                      * parsers.more
7401
7402 parsers.autolink  = parsers.auto_link_url
7403                   + parsers.auto_link_email
7404
7405 -- content in balanced brackets, parentheses, or quotes:
7406 parsers.bracketed   = P{ parsers.lbracket
7407                        * (( 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.rbracket

parsers.link_label  = parsers.lbracket
                     * -#(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.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 - p
                    * parsers.squote

parsers.title_d     = parsers.dquote
                    * Cs((parsers.html_entities
                         + V("NoSoftLineBreakSpace")
                         + V("NoSoftLineBreakEndline")
                         + (parsers.anyescaped - parsers.newline - parsers.dquote - p
                    * parsers.dquote

parsers.title_p     = parsers.lparent
                    * Cs((parsers.html_entities
                         + V("NoSoftLineBreakSpace")
                         + V("NoSoftLineBreakEndline")
                         + (parsers.anyescaped - parsers.newline - parsers.lparent -
                           - 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("")

```

3.1.4.8 Helpers for Links and Link Reference Definitions

```
-- parse a reference definition:  [foo]: /bar "title"
parsers.define_reference_parser = (parsers.check_trail / "") * parsers.link_label * p
                                  * parsers.spnlc * parsers.url
                                  * ( parsers.spnlc_sep * parsers.title * parsers.only_
                                    + Cc("") * parsers.only_blank)
```

3.1.4.9 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

```

3.1.4.10 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)
```

3.1.4.11 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.optionalsp
                      + parsers.nonindentspace * parsers.dash^1 * parsers.optionalspa

```

```
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))
```

3.1.5 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new TEX reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these ⟨*member*⟩s as `reader->`⟨*member*⟩.

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

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

```
  self.writer = writer
  self.options = options
```

Create a `reader->parsers` hash table that stores PEG patterns that depend on the received `options`. Make `reader->parsers` inherit from the global `parsers` table.

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

Make `reader->parsers` available as a local `parsers` variable that will shadow the global `parsers` table and will make `reader->parsers` easier to type in the rest of the reader code.

```
7564  local parsers = self.parsers
```

3.1.5.1 Top-Level Helper Functions

Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
7565  function self.normalize_tag(tag)
7566    tag = util.rope_to_string(tag)
7567    tag = tag:gsub("[ \n\r\t]+", " ")
7568    tag = tag:gsub("^ ", ""):gsub(" $", "")
7569    tag = uni_algos.case.casefold(tag, true, false)
7570    return tag
7571  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.

```
7572  local function iterlines(s, f)
7573    local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7574    return util.rope_to_string(rope)
7575  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.

```
7576  if options.preserveTabs then
7577    self.expandtabs = function(s) return s end
7578  else
7579    self.expandtabs = function(s)
7580                        if s:find("\t") then
7581                          return iterlines(s, util.expand_tabs_in_line)
7582                        else
7583                          return s
7584                        end
7585                      end
7586  end
```

3.1.5.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
7587  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
              is_shorter = is_shorter or prefix_length < min_prefix_length
              if is_shorter then
                  min_prefix_length, min_prefix = prefix_length, prefix
              end
              return line
          end)
          str = str:gsub('^' .. min_prefix, '')
      end
```

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

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

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

```

```
7626  self.create_parser("parse_blocks_nested",
7627                     function()
7628                       return parsers.blocks_nested
7629                     end, false)
7630
7631  self.create_parser("parse_inlines",
7632                     function()
7633                       return parsers.inlines
7634                     end, false)
7635
7636  self.create_parser("parse_inlines_no_inline_note",
7637                     function()
7638                       return parsers.inlines_no_inline_note
7639                     end, false)
7640
7641  self.create_parser("parse_inlines_no_html",
7642                     function()
7643                       return parsers.inlines_no_html
7644                     end, false)
7645
7646  self.create_parser("parse_inlines_nbsp",
7647                     function()
7648                       return parsers.inlines_nbsp
7649                     end, false)
7650  self.create_parser("parse_inlines_no_link_or_emphasis",
7651                    function()
7652                      return parsers.inlines_no_link_or_emphasis
7653                    end, false)
```

3.1.5.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
7654  parsers.minimally_indented_blankline = parsers.check_minimal_indent * (parsers.blan
7655
7656  parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")
7657
7658  parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("B
7659
7660  parsers.minimally_indented_paragraph = parsers.check_minimal_indent * V("Paragraph"
7661
7662  parsers.minimally_indented_plain = parsers.check_minimal_indent * V("Plain")
7663
7664  parsers.minimally_indented_par_or_plain = parsers.minimally_indented_paragraph
7665                                          + parsers.minimally_indented_plain
7666
7667  parsers.minimally_indented_par_or_plain_no_blank  = parsers.minimally_indented_par_
7668                                                    - parsers.minimally_indented_blan
```

```
7669
7670  parsers.minimally_indented_ref = parsers.check_minimal_indent * V("Reference")
7671
7672  parsers.minimally_indented_blank = parsers.check_minimal_indent * V("Blank")
7673
7674  parsers.conditionally_indented_blankline = parsers.check_minimal_blank_indent * (pa
7675
7676  parsers.minimally_indented_ref_or_block = parsers.minimally_indented_ref
7677                                          + parsers.minimally_indented_block
7678                                          - parsers.minimally_indented_blankline
7679
7680  parsers.minimally_indented_ref_or_block_or_par  = parsers.minimally_indented_ref
7681                                                  + parsers.minimally_indented_block_
7682                                                  - parsers.minimally_indented_blankl
7683
```

The following pattern parses the properly indented content that follows the initial container start.

```
7684
7685  parsers.separator_loop = function(separated_block, paragraph, block_separator, para
7686    return  separated_block
7687          + block_separator
7688            * paragraph
7689            * separated_block
7690          + paragraph_separator
7691          * paragraph
7692  end
7693
7694  parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
7695    return {
7696      block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
7697      par = parsers.separator_loop(separated_block, paragraph, block_separator, parag
7698    }
7699  end
7700
7701  parsers.block_sep_group = function(blank)
7702    return  blank^0 * parsers.eof
7703          + ( blank^2 / writer.paragraphsep
7704            + blank^0 / writer.interblocksep
7705            )
7706  end
7707
7708  parsers.par_sep_group = function(blank)
7709    return  blank^0 * parsers.eof
7710          + blank^0 / writer.paragraphsep
7711  end
7712
```

```
7713  parsers.sep_group_no_output = function(blank)
7714    return  blank^0 * parsers.eof
7715          + blank^0
7716  end
7717
7718  parsers.content_blank = parsers.minimally_indented_blankline
7719
7720  parsers.ref_or_block_separated  = parsers.sep_group_no_output(parsers.content_blank
7721                                  * ( parsers.minimally_indented_ref
7722                                    - parsers.content_blank)
7723                                  + parsers.block_sep_group(parsers.content_blank)
7724                                  * ( parsers.minimally_indented_block
7725                                    - parsers.content_blank)
7726
7727  parsers.loop_body_pair  =
7728    parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7729                                  parsers.minimally_indented_par_or_plain_no_blank,
7730                                  parsers.block_sep_group(parsers.content_blank),
7731                                  parsers.par_sep_group(parsers.content_blank))
7732
7733  parsers.content_loop  = ( V("Block")
7734                          * parsers.loop_body_pair.block^0
7735                          + (V("Paragraph") + V("Plain"))
7736                          * parsers.ref_or_block_separated
7737                          * parsers.loop_body_pair.block^0
7738                          + (V("Paragraph") + V("Plain"))
7739                          * parsers.loop_body_pair.par^0)
7740                        * parsers.content_blank^0
7741
7742  parsers.indented_content = function()
7743    return  Ct( (V("Reference") + (parsers.blankline / ""))
7744              * parsers.content_blank^0
7745              * parsers.check_minimal_indent
7746              * parsers.content_loop
7747              + (V("Reference") + (parsers.blankline / ""))
7748              * parsers.content_blank^0
7749              + parsers.content_loop)
7750  end
7751
7752  parsers.add_indent = function(pattern, name, breakable)
7753    return  Cg(Cmt( Cb("indent_info")
7754                  * Ct(pattern)
7755                  * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
7756                  * Cc(name)
7757                  * Cc(breakable),
7758              process_starter_indent), "indent_info")
7759  end
```

```

```

3.1.5.4 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

  return parsers.check_trail
           * Ct(C(delimiter_range) * C(delimiter_type))
           * allowed_end
end

parsers.starter = parsers.bullet(parsers.dash)
                + parsers.bullet(parsers.asterisk)
                + parsers.bullet(parsers.plus)
                + parsers.enumerator(parsers.period)
                + parsers.enumerator(parsers.rparent)

```

3.1.5.5 Parsers Used for Blockquotes (local)

```
parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.spaced

parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", true)
                        * parsers.indented_content()
                        * remove_indent("bq")

if not options.breakableBlockquotes then
  parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", fals
                          * parsers.indented_content()
                          * remove_indent("bq")
end
```

3.1.5.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```
local function parse_content_part(content_part)
  local rope = util.rope_to_string(content_part)
  local parsed = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
  parsed.indent_info = nil
  return parsed
end

```

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

```
local function collect_emphasis_content(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.curre
        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.

```
7839  local function fill_emph(t, opening_index, closing_index)
7840    local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7841    t[opening_index + 1].is_active = true
7842    t[opening_index + 1].rendered = writer.emphasis(content)
7843  end
7844
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
7845  local function fill_strong(t, opening_index, closing_index)
7846    local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7847    t[opening_index + 1].is_active = true
7848    t[opening_index + 1].rendered = writer.strong(content)
7849  end
7850
```

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

```
7851  local function breaks_three_rule(opening_delimiter, closing_delimiter)
7852    return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
7853      ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0
7854      (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count
7855  end
7856
```

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`.

```
7857  local function find_emphasis_opener(t, bottom_index, latest_index, character, closi
7858    for i = latest_index, bottom_index, -1 do
7859      local value = t[i]
7860      if value.is_active and
7861         value.is_opening and
7862         value.type == "delimiter" and
7863         value.element == "emphasis" and
7864         (value.character == character) and
7865         (value.current_count > 0) then
7866        if not breaks_three_rule(value, closing_delimiter) then
7867          return i
7868        end
7869      end
7870    end
7871  end
7872
```

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

```
7873  local function process_emphasis(t, opening_index, closing_index)
```

```
  for i = opening_index, closing_index do
    local value = t[i]
    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_le

    local opener_position = find_emphasis_opener(t, current_openers_bottom, current

    if (opener_position == nil) then
      openers_bottom[character][is_opening][closing_length_modulo_three + 1] = curr
      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)
    local c
    local char_length
    for pos = start_pos, end_pos, 1 do
```

```
7964      if (start_pos < 0) then
7965        char_length = -pos
7966      else
7967        char_length = pos + 1
7968      end
7969      c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
7970      if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
7971        return i
7972      end
7973    end
7974  end
7975
7976  local function check_preceding_unicode_punctuation(s, i)
7977    return check_unicode_type(s, i, -4, -1, "%p")
7978  end
7979
7980  local function check_preceding_unicode_whitespace(s, i)
7981    return check_unicode_type(s, i, -4, -1, "%s")
7982  end
7983
7984  local function check_following_unicode_punctuation(s, i)
7985    return check_unicode_type(s, i, 0, 3, "%p")
7986  end
7987
7988  local function check_following_unicode_whitespace(s, i)
7989    return check_unicode_type(s, i, 0, 3, "%s")
7990  end
7991
7992  parsers.unicode_preceding_punctuation = B(parsers.escapable)
7993                                        + Cmt(parsers.succeed, check_preceding_unicod
7994
7995  parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_unicode
7996
7997  parsers.unicode_following_punctuation = #parsers.escapable
7998                                        + Cmt(parsers.succeed, check_following_unicod
7999
8000  parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_unicode
8001
8002  parsers.delimiter_run = function(character)
8003    return  (B(parsers.backslash * character) + -B(character))
8004          * character^1
8005          * -#character
8006  end
8007
8008  parsers.left_flanking_delimiter_run = function(character)
8009    return  (B( parsers.any)
8010              * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_wh
```

```
             + -B(parsers.any))
           * parsers.delimiter_run(character)
           * parsers.unicode_following_punctuation
           + parsers.delimiter_run(character)
           * -#(parsers.unicode_following_punctuation + parsers.unicode_following_wh
               + parsers.eof)
end

parsers.right_flanking_delimiter_run = function(character)
  return  parsers.unicode_preceding_punctuation
        * parsers.delimiter_run(character)
        * (parsers.unicode_following_punctuation + parsers.unicode_following_whites
          + parsers.eof)
        + (B(parsers.any)
          * -(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whi
        * parsers.delimiter_run(character)
end

if options.underscores then
  parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
                     + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
                         + (parsers.unicode_preceding_punctuation
                           * #parsers.right_flanking_delimiter_run(parsers.underscor
                     * parsers.left_flanking_delimiter_run(parsers.underscore)

  parsers.emph_end  = parsers.right_flanking_delimiter_run(parsers.asterisk)
                     + (-#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

```

3.1.5.7 Helpers for Links and Link Reference Definitions (local)

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

```

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

```
function self.register_link(_, tag, url, title,
                            attributes)
  local normalized_tag = self.normalize_tag(tag)
    if references[normalized_tag] == nil then
      references[normalized_tag] = {
        url = url,
        title = title,
        attributes = attributes
      }
    end
  return ""
end

```

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

```
function self.lookup_reference(tag)
  return references[self.normalize_tag(tag)]
end

parsers.title_s_direct_ref  = parsers.squote
```

```
                              * Cs((parsers.html_entities
                                  + (parsers.anyescaped - parsers.squote - parsers.bl
                              * parsers.squote

parsers.title_d_direct_ref  = parsers.dquote
                              * Cs((parsers.html_entities
                                  + (parsers.anyescaped - parsers.dquote - parsers.bl
                              * parsers.dquote

parsers.title_p_direct_ref  = parsers.lparent
                              * Cs((parsers.html_entities
                                  + (parsers.anyescaped - parsers.lparent - parsers.r
                              * 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.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
                                  * Cg(Cc(true), "is_opening")
                                  * Cg(Cc(false), "is_closing")
                                  * ( Cg(Cc("image"), "element")
                                    * parsers.image_opening
                                    * Cg(parsers.exclamation * parsers.lbracket, "con
                                    + Cg(Cc("link"), "element")
                                    * parsers.link_opening
                                    * Cg(parsers.lbracket, "content")))

parsers.link_image_closing  = Ct( Cg(Cc("delimiter"), "type")
                                  * Cg(Cc("link"), "element")
                                  * Cg(Cc(false), "is_opening")
                                  * Cg(Cc(true), "is_closing")
                                  * ( Cg(Cc(true), "is_direct")
                                    * Cg(parsers.rbracket * #parsers.inline_direct_re
                                    + Cg(Cc(false), "is_direct")
                                    * Cg(parsers.rbracket, "content")))

parsers.link_image_open_or_close  = parsers.link_image_opening
                                    + parsers.link_image_closing

if options.html then
  parsers.link_emph_precedence  = parsers.inticks
                                  + parsers.autolink
```

```
                                  + parsers.html_inline_tags
  else
    parsers.link_emph_precedence  = parsers.inticks
                                  + parsers.autolink
  end

  parsers.link_and_emph_endline = parsers.newline
                                * ((parsers.check_minimal_indent
                                  * -V("EndlineExceptions")
                                  + parsers.check_optional_indent
                                  * -V("EndlineExceptions")
                                  * -parsers.starter) / "")
                                * parsers.spacechar^0 / "\n"

  parsers.link_and_emph_content = Ct( Cg(Cc("content"), "type")
                                    * Cg(Cs(( parsers.link_emph_precedence
                                            + parsers.backslash * parsers.any
                                            + parsers.link_and_emph_endline
                                            + (parsers.linechar
                                              - parsers.blankline^2
                                              - parsers.link_image_open_or_close
                                              - parsers.emph_open_or_close))^0), "con

  parsers.link_and_emph_table = (parsers.link_image_opening + parsers.emph_open)
                              * parsers.link_and_emph_content
                              * ((parsers.link_image_open_or_close + parsers.emph_ope
                                * parsers.link_and_emph_content)^1

```

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

```
  local function collect_link_content(t, opening_index, closing_index)
    local content = {}
    for i = opening_index, closing_index do
      content[#content + 1] = t[i].content
    end
    return util.rope_to_string(content)
  end

```

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

```
  local function find_link_opener(t, bottom_index, latest_index)
    for i = latest_index, bottom_index, -1 do
      local value = t[i]
      if value.type == "delimiter" and
         value.is_opening and
         (value.element == "link" or value.element == "image")
```

```
          and not value.removed then
        if value.is_active then
          return i
        end
        value.removed = true
        return nil
      end
    end
  end

```

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

```
  local function find_next_link_closing_index(t, latest_index)
    for i = latest_index, #t do
      local value = t[i]
      if value.is_closing and
         value.element == "link" and
         not value.removed then
        return i
      end
    end
  end

```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```
  local function disable_previous_link_openers(t, opening_index)
    if t[opening_index].element == "image" then
      return
    end

    for i = opening_index, 1, -1 do
      local value = t[i]
      if value.is_active and
         value.type == "delimiter" and
         value.is_opening and
         value.element == "link" then
        value.is_active = false
      end
    end
  end

```

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

```
  local function disable_range(t, opening_index, closing_index)
    for i = opening_index, closing_index do
```

```
      local value = t[i]
      if value.is_active then
        value.is_active = false
        if value.type == "delimiter" then
          value.removed = true
        end
      end
    end
  end

```

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

```
  local function delete_parsed_content_in_range(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.

```
8308  local function render_link_or_image(t, opening_index, closing_index, content_end_in
8309    process_emphasis(t, opening_index, content_end_index)
8310    local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index -
8311
8312    local rendered = {}
8313    if (t[opening_index].element == "link") then
8314      rendered = writer.link(mapped, reference.url, reference.title, reference.attrib
8315    end
8316
8317    if (t[opening_index].element == "image") then
8318      rendered = writer.image(mapped, reference.url, reference.title, reference.attri
8319    end
8320
8321    t[opening_index].rendered = rendered
8322    delete_parsed_content_in_range(t, opening_index + 1, closing_index)
8323    empty_content_in_range(t, opening_index, closing_index)
8324    disable_previous_link_openers(t, opening_index)
8325    disable_range(t, opening_index, closing_index)
8326  end
8327
```

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.

```
8328  local function resolve_inline_following_content(t, closing_index, match_reference,
8329    local content = ""
8330    for i = closing_index + 1, #t do
8331      content = content .. t[i].content
8332    end
8333
8334    local matching_content = parsers.succeed
8335
8336    if match_reference then
8337      matching_content = matching_content * parsers.inline_direct_ref_inside
8338    end
8339
8340    if match_link_attributes then
8341      matching_content = matching_content * Cg(Ct(parsers.attributes^-
    1), "attributes")
8342    end
8343
8344    local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), conte
8345
8346    local matched_count = matched.end_position - 1
8347    for i = closing_index + 1, #t do
8348      local value = t[i]
8349
```

```
      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[32] 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
    render_link_or_image(t, opening_index, closing_index, closing_index, inline_conte
  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
      r.attributes = join_attributes(r.attributes, inline_content.attributes)
      render_link_or_image(t, opening_index, closing_index, closing_index, r)
    end
  end
```

[32] See `b`.

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_cl
  local r = self.lookup_reference(next_link_content)

  if r then
    local inline_content = resolve_inline_following_content(t, next_link_closing_in
                                                            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,
  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
    r.attributes = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
  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") then
      goto continue
```

```
8425        end
8426
8427        local opener_position = find_link_opener(t, 1, i - 1)
8428        if (opener_position == nil) then
8429          goto continue
8430        end
8431
8432        local opening_delimiter = t[opener_position]
8433        opening_delimiter.removed = true
8434
8435        local link_type = opening_delimiter.link_type
8436
8437        if (link_type == "inline") then
8438          resolve_inline_link(t, opener_position, i)
8439        end
8440        if (link_type == "shortcut") then
8441          resolve_shortcut_link(t, opener_position, i)
8442        end
8443        if (link_type == "full") then
8444          resolve_full_link(t, opener_position, i)
8445        end
8446        if (link_type == "collapsed") then
8447          resolve_collapsed_link(t, opener_position, i)
8448        end
8449
8450        ::continue::
8451      end
8452
8453      t[#t].content = t[#t].content:gsub("%s*$","")
8454
8455      process_emphasis(t, 1, #t)
8456      local final_result = collect_emphasis_content(t, 1, #t)
8457      return final_result
8458    end
8459
8460    function self.defer_link_and_emphasis_processing(delimiter_table)
8461      return writer.defer_call(function()
8462        return process_links_and_emphasis(delimiter_table)
8463      end)
8464    end
8465
```

3.1.5.8 Inline Elements (local)

```
8466    parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
8467                     / writer.string
8468
```

```
8469  parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
8470                   / writer.string
8471
8472  parsers.Ellipsis = P("...") / writer.ellipsis
8473
8474  parsers.Smart    = parsers.Ellipsis
8475
8476  parsers.Code     = parsers.inticks / writer.code
8477
8478  if options.blankBeforeBlockquote then
8479    parsers.bqstart = parsers.fail
8480  else
8481    parsers.bqstart = parsers.blockquote_start
8482  end
8483
8484  if options.blankBeforeHeading then
8485    parsers.headerstart = parsers.fail
8486  else
8487    parsers.headerstart = parsers.atx_heading
8488  end
8489
8490  if options.blankBeforeList then
8491    parsers.interrupting_bullets = parsers.fail
8492    parsers.interrupting_enumerators = parsers.fail
8493  else
8494    parsers.interrupting_bullets  = parsers.bullet(parsers.dash, true)
8495                                  + parsers.bullet(parsers.asterisk, true)
8496                                  + parsers.bullet(parsers.plus, true)
8497
8498    parsers.interrupting_enumerators  = parsers.enumerator(parsers.period, true)
8499                                      + parsers.enumerator(parsers.rparent, true)
8500  end
8501
8502  if options.html then
8503    parsers.html_interrupting = parsers.check_trail
8504                              * ( parsers.html_incomplete_open_tag
8505                                + parsers.html_incomplete_close_tag
8506                                + parsers.html_incomplete_open_special_tag
8507                                + parsers.html_comment_start
8508                                + parsers.html_cdatasection_start
8509                                + parsers.html_declaration_start
8510                                + parsers.html_instruction_start
8511                                - parsers.html_close_special_tag
8512                                - parsers.html_empty_special_tag)
8513  else
8514    parsers.html_interrupting = parsers.fail
8515  end
```

```
8516
8517  parsers.EndlineExceptions
8518                      = parsers.blankline -- paragraph break
8519                      + parsers.eof       -- end of document
8520                      + parsers.bqstart
8521                      + parsers.thematic_break_lines
8522                      + parsers.interrupting_bullets
8523                      + parsers.interrupting_enumerators
8524                      + parsers.headerstart
8525                      + parsers.html_interrupting
8526
8527  parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8528
8529  parsers.endline = parsers.newline
8530                  * (parsers.check_minimal_indent
8531                    * -V("EndlineExceptions")
8532                    + parsers.check_optional_indent
8533                    * -V("EndlineExceptions")
8534                    * -parsers.starter)
8535                  * parsers.spacechar^0
8536
8537  parsers.Endline = parsers.endline
8538                  / writer.soft_line_break
8539
8540  parsers.EndlineNoSub = parsers.endline
8541
8542  parsers.NoSoftLineBreakEndline
8543                        = parsers.newline
8544                        * (parsers.check_minimal_indent
8545                          * -V("NoSoftLineBreakEndlineExceptions")
8546                          + parsers.check_optional_indent
8547                          * -V("NoSoftLineBreakEndlineExceptions")
8548                          * -parsers.starter)
8549                        * parsers.spacechar^0
8550                        / writer.space
8551
8552  parsers.EndlineBreak = parsers.backslash * parsers.Endline
8553                                           / writer.hard_line_break
8554
8555  parsers.OptionalIndent
8556                      = parsers.spacechar^1 / writer.space
8557
8558  parsers.Space       = parsers.spacechar^2 * parsers.Endline
8559                                           / writer.hard_line_break
8560                      + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8561                      + parsers.spacechar^1 * parsers.Endline
8562                                           / 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.
                    + parsers.spacechar^1 * parsers.Endline
                                          / writer.soft_line_break
                    + parsers.spacechar^1 * -parsers.newline / self.expandtabs

  parsers.NonbreakingEndline
                    = parsers.endline
                    / writer.soft_line_break

  parsers.NonbreakingSpace
                  = parsers.spacechar^2 * parsers.Endline
                                        / writer.hard_line_break
                  + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
                  + parsers.spacechar^1 * parsers.Endline
                                        * parsers.optionalspace
                                        / writer.soft_line_break
                  + 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_commen
                        + 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
```

3.1.5.9 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 * pa
                        * ((parsers.check_minimal_indent / "") * parsers.check_code_t
                           * (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
```

3.1.5.10 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_blan
                                    * 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
                                      * 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
```

```
8784          startnum = math.floor(startnum)
8785        end
8786      else
8787        startnum = nil
8788      end
8789      return writer.orderedlist(items,tight,startnum)
8790    end
8791
8792    parsers.OrderedListOfType = function(delimiter_type)
8793      local enumerator = parsers.enumerator(delimiter_type)
8794      return  Cg(enumerator, "listtype")
8795            * (Ct( parsers.TightListItem(Cb("listtype"))
8796                 * ((parsers.check_minimal_indent / "") * parsers.TightListItem(enumera
8797            * Cc(true)
8798            * -#((parsers.list_blank^0 / "")
8799                * parsers.check_minimal_indent * enumerator)
8800            + Ct( parsers.LooseListItem(Cb("listtype"))
8801                * ((parsers.list_blank^0 / "")
8802                  * (parsers.check_minimal_indent / "") * parsers.LooseListItem(enumera
8803            * Cc(false)
8804            ) * Ct(Cb("listtype")) / ordered_list
8805    end
8806
8807    parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
8808                        + parsers.OrderedListOfType(parsers.rparent)
```

3.1.5.11 Blank (local)

```
8809    parsers.Blank        = parsers.blankline / ""
8810                         + V("Reference")
```

3.1.5.12 Headings (local)

```
8811    function parsers.parse_heading_text(s)
8812      local inlines = self.parser_functions.parse_inlines(s)
8813      local flatten_inlines = self.writer.flatten_inlines
8814      self.writer.flatten_inlines = true
8815      local flat_text = self.parser_functions.parse_inlines(s)
8816      flat_text = util.rope_to_string(flat_text)
8817      self.writer.flatten_inlines = flatten_inlines
8818      return {flat_text, inlines}
8819    end
8820
8821    -- parse atx header
8822    parsers.AtxHeading = parsers.check_trail_no_rem
8823                       * Cg(parsers.heading_start, "level")
8824                       * ((C( parsers.optionalspace
8825                            * parsers.hash^0
```

```
                          * parsers.optionalspace
                          * parsers.newline)
                        + parsers.spacechar^1
                        * C(parsers.line))
                      / strip_atx_end
                      / parsers.parse_heading_text)
                    * Cb("level")
                    / writer.heading

parsers.heading_line  = parsers.linechar^1
                      - parsers.thematic_break_lines

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

parsers.SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
                      * #(parsers.heading_text
                         * parsers.check_minimal_indent * parsers.check_trail * par
                      * Cs(parsers.heading_text)
                      / parsers.parse_heading_text
                      * parsers.check_minimal_indent_and_trail * parsers.heading_le
                      * parsers.newline
                      * parsers.unfreeze_trail
                      / writer.heading

parsers.Heading = parsers.AtxHeading + parsers.SetextHeading
```

3.1.5.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain TEX output.

```
function self.finalize_grammar(extensions)
```

Create a local writable copy of the global read-only `walkable_syntax` hash table. This table can be used by user-defined syntax extensions to insert new PEG patterns into existing rules of the PEG grammar of markdown using the `reader->insert_pattern` method. Furthermore, built-in syntax extensions can use this table to override existing rules using the `reader->update_rule` method.

```
  local walkable_syntax = (function(global_walkable_syntax)
    local local_walkable_syntax = {}
    for lhs, rule in pairs(global_walkable_syntax) do
      local_walkable_syntax[lhs] = util.table_copy(rule)
    end
    return local_walkable_syntax
  end)(walkable_syntax)
```

The `reader->insert_pattern` method adds a pattern to `walkable_syntax[`*left-hand side terminal symbol*`]` before, instead of, or after a right-hand-side terminal symbol.

```
8861    local current_extension_name = nil
8862    self.insert_pattern = function(selector, pattern, pattern_name)
8863      assert(pattern_name == nil or type(pattern_name) == "string")
8864      local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
8865      assert(lhs ~= nil,
8866        [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
8867        .. selector .. [["]])
8868      assert(walkable_syntax[lhs] ~= nil,
8869        [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
8870      assert(pos == "before" or pos == "after" or pos == "instead of",
8871        [[Expected positional specifier "before", "after", or "instead of", not "]]
8872        .. pos .. [["]])
8873      local rule = walkable_syntax[lhs]
8874      local index = nil
8875      for current_index, current_rhs in ipairs(rule) do
8876        if type(current_rhs) == "string" and current_rhs == rhs then
8877          index = current_index
8878          if pos == "after" then
8879            index = index + 1
8880          end
8881          break
8882        end
8883      end
8884      assert(index ~= nil,
8885        [[Rule ]] .. lhs .. [[ -> ]] .. rhs
8886          .. [[ does not exist in markdown grammar]])
8887      local accountable_pattern
8888      if current_extension_name then
8889        accountable_pattern = { pattern, current_extension_name, pattern_name }
8890      else
8891        assert(type(pattern) == "string",
8892          [[reader->insert_pattern() was called outside an extension with ]]
8893          .. [[a PEG pattern instead of a rule name]])
8894        accountable_pattern = pattern
8895      end
8896      if pos == "instead of" then
8897        rule[index] = accountable_pattern
8898      else
8899        table.insert(rule, index, accountable_pattern)
8900      end
8901    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.

```
8902    local syntax =
8903      { "Blocks",
8904
8905        Blocks                = V("InitializeState")
8906                              * ( V("ExpectedJekyllData")
8907                                * (V("Blank")^0 / writer.interblocksep)
8908                                )^-1
8909                              * 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.

```
8910                              * ( V("Block")
8911                                * ( V("Blank")^0 * parsers.eof
8912                                  + ( V("Blank")^2 / writer.paragraphsep
8913                                    + V("Blank")^0 / writer.interblocksep
8914                                    )
8915                                  )
8916                                + ( V("Paragraph") + V("Plain") )
8917                                * ( V("Blank")^0 * parsers.eof
8918                                  + ( V("Blank")^2 / writer.paragraphsep
8919                                    + V("Blank")^0 / writer.interblocksep
8920                                    )
8921                                  )
8922                                * V("Block")
8923                                * ( V("Blank")^0 * parsers.eof
8924                                  + ( V("Blank")^2 / writer.paragraphsep
8925                                    + V("Blank")^0 / writer.interblocksep
8926                                    )
8927                                  )
8928                                + ( V("Paragraph") + V("Plain") )
8929                                * ( V("Blank")^0 * parsers.eof
8930                                  + V("Blank")^0 / writer.paragraphsep
8931                                  )
8932                                )^0,
8933
8934        ExpectedJekyllData    = parsers.fail,
8935
8936        Blank                 = parsers.Blank,
8937        Reference             = parsers.Reference,
8938
8939        Blockquote            = parsers.Blockquote,
8940        Verbatim              = parsers.Verbatim,
8941        ThematicBreak         = parsers.ThematicBreak,
8942        BulletList            = parsers.BulletList,
8943        OrderedList           = parsers.OrderedList,
8944        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)
      assert(current_extension_name ~= nil)
      assert(syntax[rule_name] ~= nil,
        [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
      local previous_pattern
      local extension_name
      if walkable_syntax[rule_name] then
        local previous_accountable_pattern = walkable_syntax[rule_name][1]
        previous_pattern = previous_accountable_pattern[1]
        extension_name = previous_accountable_pattern[2] .. ", " .. current_extension
```

```
8986        else
8987          previous_pattern = nil
8988          extension_name = current_extension_name
8989        end
8990        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
```

```
8991        if type(get_pattern) == "function" then
8992          pattern = get_pattern(previous_pattern)
8993        else
8994          assert(previous_pattern == nil,
8995                 [[Rule ]] .. rule_name ..
8996                 [[ has already been updated by ]] .. extension_name)
8997          pattern = get_pattern
8998        end
8999        local accountable_pattern = { pattern, extension_name, rule_name }
9000        walkable_syntax[rule_name] = { accountable_pattern }
9001      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.

```
9002      local special_characters = {}
9003      self.add_special_character = function(c)
9004        table.insert(special_characters, c)
9005        syntax.SpecialChar = S(table.concat(special_characters, ""))
9006      end
9007
9008      self.add_special_character("*")
9009      self.add_special_character("[")
9010      self.add_special_character("]")
9011      self.add_special_character("<")
9012      self.add_special_character("!")
9013      self.add_special_character("\\")
```

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

```
9014      self.initialize_named_group = function(name, value)
```

```
      local pattern = Ct("")
      if value ~= nil then
        pattern = pattern / value
      end
      syntax.InitializeState = syntax.InitializeState
                             * Cg(pattern, name)
    end
```

Add a named group for indentation.

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

Apply syntax extensions.

```
    for _, extension in ipairs(extensions) do
      current_extension_name = extension.name
      extension.extend_writer(writer)
      extension.extend_reader(self)
    end
    current_extension_name = nil
```

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

```
    if options.debugExtensions then
      local sorted_lhs = {}
      for lhs, _ in pairs(walkable_syntax) do
        table.insert(sorted_lhs, lhs)
      end
      table.sort(sorted_lhs)

      local output_lines = {"{"}
      for lhs_index, lhs in ipairs(sorted_lhs) do
        local encoded_lhs = util.encode_json_string(lhs)
        table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
        local rule = walkable_syntax[lhs]
        for rhs_index, rhs in ipairs(rule) do
          local human_readable_rhs
          if type(rhs) == "string" then
            human_readable_rhs = rhs
          else
            local pattern_name
            if rhs[3] then
              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)
```

```
9056            local output_line = [[        ]] .. encoded_rhs
9057            if rhs_index < #rule then
9058              output_line = output_line .. ","
9059            end
9060            table.insert(output_lines, output_line)
9061          end
9062          local output_line = "    ]"
9063          if lhs_index < #sorted_lhs then
9064            output_line = output_line .. ","
9065          end
9066          table.insert(output_lines, output_line)
9067        end
9068        table.insert(output_lines, "}")
9069
9070        local output = table.concat(output_lines, "\n")
9071        local output_filename = options.debugExtensionsFileName
9072        local output_file = assert(io.open(output_filename, "w"),
9073          [[Could not open file "]] .. output_filename .. [[" for writing]])
9074        assert(output_file:write(output))
9075        assert(output_file:close())
9076      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`.

```
9077      for lhs, rule in pairs(walkable_syntax) do
9078        syntax[lhs] = parsers.fail
9079        for _, rhs in ipairs(rule) do
9080          local pattern
```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```
9081          if type(rhs) == "string" then
9082            pattern = V(rhs)
9083          else
9084            pattern = rhs[1]
9085            if type(pattern) == "string" then
9086              pattern = V(pattern)
9087            end
9088          end
9089          syntax[lhs] = syntax[lhs] + pattern
9090        end
9091      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
      syntax.AutoLinkRelativeReference = parsers.fail
    end

    if options.contentLevel == "inline" then
      syntax[1] = "Inlines"
      syntax.Inlines = V("InitializeState")
                       * parsers.Inline^0
                       * ( parsers.spacing^0
                         * parsers.eof / "")
      syntax.Space = parsers.Space + parsers.blankline / writer.space
    end

    local blocks_nested_t = util.table_copy(syntax)
    blocks_nested_t.ExpectedJekyllData = parsers.fail
    parsers.blocks_nested = Ct(blocks_nested_t)
```

```
9136
9137      parsers.blocks = Ct(syntax)
9138
9139      local inlines_t = util.table_copy(syntax)
9140      inlines_t[1] = "Inlines"
9141      inlines_t.Inlines = V("InitializeState")
9142                        * parsers.Inline^0
9143                        * ( parsers.spacing^0
9144                          * parsers.eof / "")
9145      parsers.inlines = Ct(inlines_t)
9146
9147      local inlines_no_inline_note_t = util.table_copy(inlines_t)
9148      inlines_no_inline_note_t.InlineNote = parsers.fail
9149      parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9150
9151      local inlines_no_html_t = util.table_copy(inlines_t)
9152      inlines_no_html_t.DisplayHtml = parsers.fail
9153      inlines_no_html_t.InlineHtml = parsers.fail
9154      inlines_no_html_t.HtmlEntity = parsers.fail
9155      parsers.inlines_no_html = Ct(inlines_no_html_t)
9156
9157      local inlines_nbsp_t = util.table_copy(inlines_t)
9158      inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9159      inlines_nbsp_t.Space = parsers.NonbreakingSpace
9160      parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9161
9162      local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9163      inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9164      inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9165      parsers.inlines_no_link_or_emphasis = Ct(inlines_no_link_or_emphasis_t)
```

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

```
9166      return function(input)
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
9167        input = input:gsub("\r\n?", "\n")
9168        if input:sub(-1) ~= "\n" then
9169          input = input .. "\n"
9170        end
```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```
9171        references = {}
9172        local opt_string = {}
9173        for k, _ in pairs(defaultOptions) do
```

```
9174          local v = options[k]
9175          if type(v) == "table" then
9176            for _, i in ipairs(v) do
9177              opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9178            end
9179          elseif k ~= "cacheDir" then
9180            opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9181          end
9182        end
9183        table.sort(opt_string)
9184        local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9185        local output
```

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

```
9186        local function convert(input)
9187          local document = self.parser_functions.parse_blocks(input)
9188          return util.rope_to_string(writer.document(document))
9189        end
9190        if options.eagerCache or options.finalizeCache then
9191          local name = util.cache(options.cacheDir, input, salt, convert,
9192                                  ".md" .. writer.suffix)
9193          output = writer.pack(name)
```

Otherwise, return the result of the conversion directly.

```
9194        else
9195          output = convert(input)
9196        end
```

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

```
9197        if options.finalizeCache then
9198          local file, mode
9199          if options.frozenCacheCounter > 0 then
9200            mode = "a"
9201          else
9202            mode = "w"
9203          end
9204          file = assert(io.open(options.frozenCacheFileName, mode),
9205            [[Could not open file "]] .. options.frozenCacheFileName
9206            .. [[" for writing]])
9207          assert(file:write([[\expandafter\global\expandafter\def\csname ]]
9208            .. [[markdownFrozenCache]] .. options.frozenCacheCounter
9209            .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
9210          assert(file:close())
9211        end
9212        return output
```

```
    end
  end
  return self
end
```

3.1.6 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

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

3.1.6.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

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

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

```
      function self.span(s, attr)
        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.rbr
                                + V("Space") + V("Endline")
                                + (parsers.any
                                  - (parsers.newline + parsers.lbracket + parsers.rbr
                                    + 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
```

3.1.6.2 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

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

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

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.
- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.
- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```
      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 "C
            "{", #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("Endline")
                         + (parsers.anyescaped
                           - (parsers.newline + parsers.rbracket + parsers.blankline^
                         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)

      local citation_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.semicolon
                            + parsers.blankline^2))
                        - (parsers.spnl * parsers.rbracket))^0)

      local citation_body_chunk
                    = citation_body_prenote
                    * parsers.spnlc * citation_name
                    * (parsers.internal_punctuation - parsers.semicolon)^-
  1
                    * parsers.spnlc * citation_body_postnote

      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.sp * 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
```

3.1.6.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```
9408 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.

```
9409   local languages_json = (function()
9410     local base, prev, curr
9411     for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
9412       local file = io.open(pathname, "r")
9413       if not file then goto continue end
9414       local input = assert(file:read("*a"))
9415       assert(file:close())
9416       local json = input:gsub('("[^\n]-"):','[%1]=')
9417       curr = load("_ENV = {}; return "..json)()
9418       if type(curr) == "table" then
9419         if base == nil then
9420           base = curr
9421         else
9422           setmetatable(prev, { __index = curr })
9423         end
9424         prev = curr
9425       end
9426       ::continue::
9427     end
9428     return base or {}
9429   end)()
9430
9431   return {
9432     name = "built-in content_blocks syntax extension",
9433     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.

```
9434       function self.contentblock(src,suf,type,tit)
9435         if not self.is_writing then return "" end
9436         src = src.."."..suf
9437         suf = suf:lower()
9438         if type == "onlineimage" then
9439           return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
9440                                   "{",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
```

3.1.6.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

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

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
                               "\\markdownRendererDlDefinitionEnd "}
        end
        retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
        return retVal
      end

      function self.definitionlist(items,tight)
        if not self.is_writing then return "" end
        local buffer = {}
        for _,item in ipairs(items) do
          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
        end
        if tight and tight_lists then
          return {"\\markdownRendererDlBeginTight\n", buffer,
            "\n\\markdownRendererDlEndTight"}
        else
          return {"\\markdownRendererDlBegin\n", buffer,
            "\n\\markdownRendererDlEnd"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local defstartchar = S("~:")

```

```
9571        local defstart  = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
9572                                        * (parsers.tab + parsers.space^-
    3)
9573                        + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
9574                                         * (parsers.tab + parsers.space^-
    2)
9575                        + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
9576                                         * (parsers.tab + parsers.space^-
    1)
9577                        + parsers.check_trail_length(3) * defstartchar * #parsers.spaci
9578
9579        local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_
9580
9581        local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option
9582
9583        local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
9584
9585        local indented_blocks = function(bl)
9586          return Cs( bl
9587                 * (blank^1 * (parsers.check_minimal_indent / "")
9588                   * parsers.check_code_trail * -parsers.blankline * bl)^0
9589                 * (blank^1 + parsers.eof))
9590        end
9591
9592        local function definition_list_item(term, defs, _)
9593          return { term = self.parser_functions.parse_inlines(term),
9594                   definitions = defs }
9595        end
9596
9597        local DefinitionListItemLoose
9598                      = C(parsers.line) * blank^0
9599                      * Ct((parsers.check_minimal_indent * (defstart
9600                           * indented_blocks(dlchunk)
9601                           / self.parser_functions.parse_blocks_nested))^1)
9602                      * Cc(false) / definition_list_item
9603
9604        local DefinitionListItemTight
9605                      = C(parsers.line)
9606                      * Ct((parsers.check_minimal_indent * (defstart * dlchunk
9607                           / self.parser_functions.parse_blocks_nested))^1)
9608                      * Cc(true) / definition_list_item
9609
9610        local DefinitionList
9611                      = ( Ct(DefinitionListItemLoose^1) * Cc(false)
9612                        + Ct(DefinitionListItemTight^1)
9613                        * (blank^0
9614                          * -DefinitionListItemLoose * Cc(true))
```

```
                      ) / writer.definitionlist

      self.insert_pattern("Block after Heading",
                          DefinitionList, "DefinitionList")
    end
  }
end
```

3.1.6.5 Fancy Lists

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

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

```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
  - `Default` – default style,
  - `OneParen` – parentheses, and
  - `Period` – periods.

```
      function self.fancylist(items,tight,startnum,numstyle,numdelim)
        if not self.is_writing then return "" end
        local buffer = {}
        local num = startnum
        for _,item in ipairs(items) do
          if item ~= "" then
```

```
9634            buffer[#buffer + 1] = self.fancyitem(item,num)
9635          end
9636          if num ~= nil and item ~= "" then
9637            num = num + 1
9638          end
9639        end
9640        local contents = util.intersperse(buffer,"\n")
9641        if tight and options.tightLists then
9642          return {"\\markdownRendererFancyOlBeginTight{",
9643                  numstyle,"}{",numdelim,"}",contents,
9644                  "\n\\markdownRendererFancyOlEndTight "}
9645        else
9646          return {"\\markdownRendererFancyOlBegin{",
9647                  numstyle,"}{",numdelim,"}",contents,
9648                  "\n\\markdownRendererFancyOlEnd "}
9649        end
9650      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.

```
9651      function self.fancyitem(s,num)
9652        if num ~= nil then
9653          return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
9654                  "\\markdownRendererFancyOlItemEnd "}
9655        else
9656          return {"\\markdownRendererFancyOlItem ",s,"\\markdownRendererFancyOlItemEn
9657        end
9658      end
9659    end, extend_reader = function(self)
9660      local parsers = self.parsers
9661      local options = self.options
9662      local writer = self.writer
9663
9664      local function combine_markers_and_delims(markers, delims)
9665        local markers_table = {}
9666        for _,marker in ipairs(markers) do
9667          local start_marker
9668          local continuation_marker
9669          if type(marker) == "table" then
9670            start_marker = marker[1]
9671            continuation_marker = marker[2]
9672          else
9673            start_marker = marker
9674            continuation_marker = marker
9675          end
9676          for _,delim in ipairs(delims) do
```

```
9677            table.insert(markers_table, {start_marker, continuation_marker, delim})
9678          end
9679        end
9680        return markers_table
9681      end
9682
9683      local function join_table_with_func(func, markers_table)
9684        local pattern = func(table.unpack(markers_table[1]))
9685        for i = 2, #markers_table do
9686          pattern = pattern + func(table.unpack(markers_table[i]))
9687        end
9688        return pattern
9689      end
9690
9691      local lowercase_letter_marker = R("az")
9692      local uppercase_letter_marker = R("AZ")
9693
9694      local roman_marker = function(chars)
9695        local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
9696        local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
9697        return  m^-3
9698                * (c*m + c*d + d^-1 * c^-3)
9699                * (x*c + x*l + l^-1 * x^-3)
9700                * (i*x + i*v + v^-1 * i^-3)
9701      end
9702
9703      local lowercase_roman_marker  = roman_marker({"m", "d", "c", "l", "x", "v", "i"
9704      local uppercase_roman_marker  = roman_marker({"M", "D", "C", "L", "X", "V", "I"
9705
9706      local lowercase_opening_roman_marker  = P("i")
9707      local uppercase_opening_roman_marker  = P("I")
9708
9709      local digit_marker = parsers.dig * parsers.dig^-8
9710
9711      local markers = {
9712        {lowercase_opening_roman_marker, lowercase_roman_marker},
9713        {uppercase_opening_roman_marker, uppercase_roman_marker},
9714        lowercase_letter_marker,
9715        uppercase_letter_marker,
9716        lowercase_roman_marker,
9717        uppercase_roman_marker,
9718        digit_marker
9719      }
9720
9721      local delims = {
9722        parsers.period,
9723        parsers.rparent
```

```
9724        }
9725
9726        local markers_table = combine_markers_and_delims(markers, delims)
9727
9728        local function enumerator(start_marker, _, delimiter_type, interrupting)
9729          local delimiter_range
9730          local allowed_end
9731          if interrupting then
9732            delimiter_range = P("1")
9733            allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9734          else
9735            delimiter_range = start_marker
9736            allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
9737          end
9738
9739          return parsers.check_trail
9740                  * Ct(C(delimiter_range) * C(delimiter_type))
9741                  * allowed_end
9742        end
9743
9744        local starter = join_table_with_func(enumerator, markers_table)
9745
9746        local TightListItem = function(starter)
9747          return  parsers.add_indent(starter, "li")
9748                  * parsers.indented_content_tight
9749        end
9750
9751        local LooseListItem = function(starter)
9752          return  parsers.add_indent(starter, "li")
9753                  * parsers.indented_content_loose
9754                  * remove_indent("li")
9755        end
9756
9757        local function roman2number(roman)
9758          local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
9759          local numeral = 0
9760
9761          local i = 1
9762          local len = string.len(roman)
9763          while i < len do
9764            local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
9765            if z1 < z2 then
9766                numeral = numeral + (z2 - z1)
9767                i = i + 2
9768            else
9769                numeral = numeral + z1
9770                i = i + 1
```

```
9771            end
9772          end
9773          if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
9774          return numeral
9775        end
9776
9777        local function sniffstyle(numstr, delimend)
9778          local numdelim
9779          if delimend == ")" then
9780            numdelim = "OneParen"
9781          elseif delimend == "." then
9782            numdelim = "Period"
9783          else
9784            numdelim = "Default"
9785          end
9786
9787          local num
9788          num = numstr:match("^([I])$")
9789          if num then
9790            return roman2number(num), "UpperRoman", numdelim
9791          end
9792          num = numstr:match("^([i])$")
9793          if num then
9794            return roman2number(string.upper(num)), "LowerRoman", numdelim
9795          end
9796          num = numstr:match("^([A-Z])$")
9797          if num then
9798            return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
9799          end
9800          num = numstr:match("^([a-z])$")
9801          if num then
9802            return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
9803          end
9804          num = numstr:match("^([IVXLCDM]+)")
9805          if num then
9806            return roman2number(num), "UpperRoman", numdelim
9807          end
9808          num = numstr:match("^([ivxlcdm]+)")
9809          if num then
9810            return roman2number(string.upper(num)), "LowerRoman", numdelim
9811          end
9812          return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
9813        end
9814
9815        local function fancylist(items,tight,start)
9816          local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
9817          return writer.fancylist(items,tight,
```

```
                                  options.startNumber and startnum or 1,
                                  numstyle or "Decimal",
                                  numdelim or "Default")
      end

      local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
        local enumerator_start = enumerator(start_marker, continuation_marker, delimi
        local enumerator_cont = enumerator(continuation_marker, continuation_marker,
        return Cg(enumerator_start, "listtype")
             * (Ct( TightListItem(Cb("listtype"))
                 * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
             * Cc(true)
             * -#((parsers.conditionally_indented_blankline^0 / "")
                 * parsers.check_minimal_indent * enumerator_cont)
             + Ct( LooseListItem(Cb("listtype"))
                 * ((parsers.conditionally_indented_blankline^0 / "")
                   * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_c
             * Cc(false)
             ) * Ct(Cb("listtype")) / fancylist
      end

      local FancyList = join_table_with_func(FancyListOfType, markers_table)

      local Endline   = parsers.newline
                      * (parsers.check_minimal_indent
                        * -parsers.EndlineExceptions
                        + parsers.check_optional_indent
                        * -parsers.EndlineExceptions
                        * -starter)
                      * parsers.spacechar^0
                      / writer.soft_line_break

      self.update_rule("OrderedList", FancyList)
      self.update_rule("Endline", Endline)
    end
  }
end
```

3.1.6.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the

syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```
M.extensions.fenced_code = function(blank_before_code_fence,
                                    allow_attributes,
                                    allow_raw_blocks)
  return {
    name = "built-in fenced_code syntax extension",
    extend_writer = function(self)
      local options = self.options

```

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

```
      function self.fencedCode(s, i, attr)
        if not self.is_writing then return "" end
        s = s:gsub("\n$", "")
        local buf = {}
        if attr ~= nil then
          table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
                             self.attributes(attr)})
        end
        local name = util.cache_verbatim(options.cacheDir, s)
        table.insert(buf, {"\\markdownRendererInputFencedCode{",
                           name,"}{",self.string(i),"}{",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
      local writer = self.writer

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

```
9894          return #a >= #b and i
9895        end
9896
9897        local function strip_enclosing_whitespaces(str)
9898          return str:gsub("^%s*(.-)%s*$", "%1")
9899        end
9900
9901        local tilde_infostring = Cs(Cs((V("HtmlEntity")
9902                                        + parsers.anyescaped
9903                                        - parsers.newline)^0)
9904                                   / strip_enclosing_whitespaces)
9905
9906        local backtick_infostring = Cs(Cs((V("HtmlEntity")
9907                                           + (-#(parsers.backslash * parsers.backtick) *
9908                                            - parsers.newline
9909                                            - parsers.backtick)^0)
9910                                      / strip_enclosing_whitespaces)
9911
9912        local fenceindent
9913
9914        local function has_trail(indent_table)
9915          return indent_table ~= nil and
9916            indent_table.trail ~= nil and
9917            next(indent_table.trail) ~= nil
9918        end
9919
9920        local function has_indents(indent_table)
9921          return indent_table ~= nil and
9922            indent_table.indents ~= nil and
9923            next(indent_table.indents) ~= nil
9924        end
9925
9926        local function get_last_indent_name(indent_table)
9927          if has_indents(indent_table) then
9928            return indent_table.indents[#indent_table.indents].name
9929          end
9930        end
9931
9932        local function count_fenced_start_indent(_, _, indent_table, trail)
9933          local last_indent_name = get_last_indent_name(indent_table)
9934          fenceindent = 0
9935          if last_indent_name ~= "li" then
9936            fenceindent = #trail
9937          end
9938          return true
9939        end
9940
```

```
9941        local fencehead      = function(char, infostring)
9942          return               Cmt(Cb("indent_info") * parsers.check_trail, count_fence
9943                             * Cg(char^3, "fencelength")
9944                             * parsers.optionalspace
9945                             * infostring
9946                             * (parsers.newline + parsers.eof)
9947        end
9948
9949        local fencetail      = function(char)
9950          return               parsers.check_trail_no_rem
9951                             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
9952                             * parsers.optionalspace * (parsers.newline + parsers.eof)
9953                             + parsers.eof
9954        end
9955
9956        local function process_fenced_line(s, i, indent_table, line_content, is_blank)
9957          local remainder = ""
9958          if has_trail(indent_table) then
9959            remainder = indent_table.trail.internal_remainder
9960          end
9961
9962          if is_blank and get_last_indent_name(indent_table) == "li" then
9963            remainder = ""
9964          end
9965
9966          local str = remainder .. line_content
9967          local index = 1
9968          local remaining = fenceindent
9969
9970          while true do
9971            local c = str:sub(index, index)
9972            if c == " " and remaining > 0 then
9973              remaining = remaining - 1
9974              index = index + 1
9975            elseif c == "\t" and remaining > 3 then
9976              remaining = remaining - 4
9977              index = index + 1
9978            else
9979              break
9980            end
9981          end
9982
9983          return true, str:sub(index)
9984        end
9985
9986        local fencedline = function(char)
9987          return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),
```

```
9988        end
9989
9990        local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true)
9991
9992        local TildeFencedCode
9993               = fencehead(parsers.tilde, tilde_infostring)
9994               * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
9995                     + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
9996               * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
9997
9998        local BacktickFencedCode
9999               = fencehead(parsers.backtick, backtick_infostring)
10000              * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10001                    + (parsers.check_minimal_indent / "") * fencedline(parsers.backtic
10002              * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10003
10004       local infostring_with_attributes
10005                       = Ct(C((parsers.linechar
10006                             - ( parsers.optionalspace
10007                               * parsers.attributes))^0)
10008                           * parsers.optionalspace
10009                           * Ct(parsers.attributes))
10010
10011       local FencedCode
10012                = ((TildeFencedCode + BacktickFencedCode)
10013                / function(infostring, code)
10014                    local expanded_code = self.expandtabs(code)
10015
10016                    if allow_raw_blocks then
10017                      local raw_attr = lpeg.match(parsers.raw_attribute,
10018                                                  infostring)
10019                      if raw_attr then
10020                        return writer.rawBlock(expanded_code, raw_attr)
10021                      end
10022                    end
10023
10024                    local attr = nil
10025                    if allow_attributes then
10026                      local match = lpeg.match(infostring_with_attributes,
10027                                               infostring)
10028                      if match then
10029                        infostring, attr = table.unpack(match)
10030                      end
10031                    end
10032                    return writer.fencedCode(expanded_code, infostring, attr)
10033                  end)
10034
```

```
10035       self.insert_pattern("Block after Verbatim",
10036                           FencedCode, "FencedCode")
10037
10038       local fencestart
10039       if blank_before_code_fence then
10040         fencestart = parsers.fail
10041       else
10042         fencestart = fencehead(parsers.backtick, backtick_infostring)
10043                    + fencehead(parsers.tilde, tilde_infostring)
10044       end
10045
10046       self.update_rule("EndlineExceptions", function(previous_pattern)
10047         if previous_pattern == nil then
10048           previous_pattern = parsers.EndlineExceptions
10049         end
10050         return previous_pattern + fencestart
10051       end)
10052
10053       self.add_special_character("`")
10054       self.add_special_character("~")
10055     end
10056   }
10057 end
```

3.1.6.7 Fenced Divs

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

```
10058 M.extensions.fenced_divs = function(blank_before_div_fence)
10059   return {
10060     name = "built-in fenced_divs syntax extension",
10061     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.

```
10062       function self.div_begin(attributes)
10063         local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",
10064                               self.attributes(attributes)}
10065         local end_output = {"\\markdownRendererFencedDivAttributeContextEnd "}
10066         return self.push_attributes("div", attributes, start_output, end_output)
10067       end
```

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

```
10068       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 `div_level` for tracking how deep we are nested in divs.

```
    self.initialize_named_group("div_level", "0")

    local function increment_div_level(increment)
      local function update_div_level(s, i, current_level) -- luacheck: ignore s i
        current_level = tonumber(current_level)
        local next_level = tostring(current_level + increment)
        return true, next_level
      end

      return Cg( Cmt(Cb("div_level"), update_div_level)
               , "div_level")
    end

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

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

    local blank = parsers.minimally_indented_blank
```

```
10112
10113       local block_separated  = parsers.block_sep_group(blank)
10114                              * non_fenced_div_block
10115
10116       local loop_body_pair  = parsers.create_loop_body_pair(block_separated,
10117                                                             non_fenced_div_paragraph,
10118                                                             parsers.block_sep_group(b
10119                                                             parsers.par_sep_group(bla
10120
10121       local content_loop  = ( non_fenced_div_block
10122                             * loop_body_pair.block^0
10123                             + non_fenced_div_paragraph
10124                             * block_separated
10125                             * loop_body_pair.block^0
10126                             + non_fenced_div_paragraph
10127                             * loop_body_pair.par^0)
10128                           * blank^0
10129
10130       local FencedDiv = fenced_div_begin
10131                       / function (infostring)
10132                           local attr = lpeg.match(Ct(parsers.attributes), infostring)
10133                           if attr == nil then
10134                             attr = {"." .. infostring}
10135                           end
10136                           return attr
10137                         end
10138                       / writer.div_begin
10139                       * increment_div_level(1)
10140                       * parsers.skipblanklines
10141                       * Ct(content_loop)
10142                       * parsers.minimally_indented_blank^0
10143                       * parsers.check_minimal_indent_and_trail * fenced_div_end * inc
    1)
10144                       * (Cc("") / writer.div_end)
10145
10146       self.insert_pattern("Block after Verbatim",
10147                           FencedDiv, "FencedDiv")
10148
10149       self.add_special_character(":")
10150
```

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.

```
10151       local function check_div_level(s, i, current_level) -- luacheck: ignore s i
10152         current_level = tonumber(current_level)
10153         return current_level > 0
```

```
      end

      local is_inside_div = Cmt(Cb("div_level"), check_div_level)
      local fencestart = is_inside_div * fenced_div_end

      if not blank_before_div_fence then
        self.update_rule("EndlineExceptions", function(previous_pattern)
          if previous_pattern == nil then
            previous_pattern = parsers.EndlineExceptions
          end
          return previous_pattern + fencestart
        end)
      end
    end
  }
end
```

3.1.6.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

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

      local 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_
                          * 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 *
                           * 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
                           * Cb("attributes")
                           * parsers.newline
                           * parsers.unfreeze_trail
                           / writer.heading

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

3.1.6.9 Inline Code Attributes

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

```
M.extensions.inline_code_attributes = function()
  return {
```

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

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

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

3.1.6.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

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

Define `writer->lineblock` as a function that will transform a line block consisted of `lines` to the output format, with all but the last newline rendered as a line break.

```
      function self.lineblock(lines)
        if not self.is_writing then return "" end
        local buffer = {}
        for i = 1, #lines - 1 do
          buffer[#buffer + 1] = { lines[i], self.hard_line_break }
        end
        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.lineblo

      self.insert_pattern("Block after Blockquote",
                           LineBlock, "LineBlock")
    end
  }
end
```

3.1.6.11 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
```

3.1.6.12 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.spnlc * parsers.url
                                    * ( parsers.spnlc_sep * parsers.title * (parsers.
                                      * parsers.only_blank
                                      + parsers.spnlc_sep * parsers.title * parsers.o
                                      + Cc("") * (parsers.spnlc * Ct(parsers.attribut
                                      + 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),
                                        / 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
```

3.1.6.13 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.

```
      function self.note(s)
        if self.flatten_inlines then return "" end
        return {"\\markdownRendererNote{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      if inline_notes then
        local InlineNote
                    = parsers.circumflex
                    * (parsers.link_label / self.parser_functions.parse_inlines_no_in
```

```
                / writer.note

    self.insert_pattern("Inline after LinkAndEmph",
                        InlineNote, "InlineNote")
  end
  if notes then
    local function strip_first_char(s)
      return s:sub(2)
    end

    local RawNoteRef
                = #(parsers.lbracket * parsers.circumflex)
                * parsers.link_label / strip_first_char

    local rawnotes = {}

    -- like indirect_link
    local function lookup_note(ref)
      return writer.defer_call(function()
        local found = rawnotes[self.normalize_tag(ref)]
        if found then
          return writer.note(
            self.parser_functions.parse_blocks_nested(found))
        else
          return {"[",
            self.parser_functions.parse_inlines("^" .. ref), "]"}
        end
      end)
    end

    local function register_note(ref,rawnote)
      local normalized_tag = self.normalize_tag(ref)
      if rawnotes[normalized_tag] == nil then
        rawnotes[normalized_tag] = rawnote
      end
      return ""
    end

    local NoteRef = RawNoteRef / lookup_note

    local optionally_indented_line = parsers.check_optional_indent_and_any_trail

    local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt

    local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)

    local indented_blocks = function(bl)
```

```
10440           return Cs( bl
10441                 * (blank^1 * (parsers.check_optional_indent / "")
10442                   * parsers.check_code_trail * -parsers.blankline * bl)^0)
10443         end
10444
10445         local NoteBlock
10446                     = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10447                     * parsers.spnlc * indented_blocks(chunk)
10448                     / register_note
10449
10450         local Reference = NoteBlock + parsers.Reference
10451
10452         self.update_rule("Reference", Reference)
10453         self.insert_pattern("Inline before LinkAndEmph",
10454                             NoteRef, "NoteRef")
10455       end
10456
10457       self.add_special_character("^")
10458     end
10459   }
10460 end
```

3.1.6.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```
10461 M.extensions.pipe_tables = function(table_captions, table_attributes)
10462
10463   local function make_pipe_table_rectangular(rows)
10464     local num_columns = #rows[2]
10465     local rectangular_rows = {}
10466     for i = 1, #rows do
10467       local row = rows[i]
10468       local rectangular_row = {}
10469       for j = 1, num_columns do
10470         rectangular_row[j] = row[j] or ""
10471       end
10472       table.insert(rectangular_rows, rectangular_row)
10473     end
10474     return rectangular_rows
10475   end
10476
10477   local function pipe_table_row(allow_empty_first_column
```

```
                                , nonempty_column
                                , column_separator
                                , column)
    local row_beginning
    if allow_empty_first_column then
      row_beginning = -- empty first column
                      #(parsers.spacechar^4
                        * column_separator)
                    * parsers.optionalspace
                    * column
                    * parsers.optionalspace
                    -- non-empty first column
                    + parsers.nonindentspace
                    * nonempty_column^-1
                    * parsers.optionalspace
    else
      row_beginning = parsers.nonindentspace
                    * nonempty_column^-1
                    * parsers.optionalspace
    end

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

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

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```
      function self.table(rows, caption, attributes)
        if not self.is_writing then return "" end
```

```
10521         local buffer = {}
10522         if attributes ~= nil then
10523           table.insert(buffer,
10524                        "\\markdownRendererTableAttributeContextBegin\n")
10525           table.insert(buffer, self.attributes(attributes))
10526         end
10527         table.insert(buffer,
10528                      {"\\markdownRendererTable{",
10529                       caption or "", "}{", #rows - 1, "}{",
10530                       #rows[1], "}"})
10531         local temp = rows[2] -- put alignments on the first row
10532         rows[2] = rows[1]
10533         rows[1] = temp
10534         for i, row in ipairs(rows) do
10535           table.insert(buffer, "{")
10536           for _, column in ipairs(row) do
10537             if i > 1 then -- do not use braces for alignments
10538               table.insert(buffer, "{")
10539             end
10540             table.insert(buffer, column)
10541             if i > 1 then
10542               table.insert(buffer, "}")
10543             end
10544           end
10545           table.insert(buffer, "}")
10546         end
10547         if attributes ~= nil then
10548           table.insert(buffer,
10549                        "\\markdownRendererTableAttributeContextEnd{}")
10550         end
10551         return buffer
10552       end
10553     end, extend_reader = function(self)
10554       local parsers = self.parsers
10555       local writer = self.writer
10556
10557       local table_hline_separator = parsers.pipe + parsers.plus
10558
10559       local table_hline_column = (parsers.dash
10560                                   - #(parsers.dash
10561                                      * (parsers.spacechar
10562                                        + table_hline_separator
10563                                        + parsers.newline)))^1
10564                                * (parsers.colon * Cc("r")
10565                                  + parsers.dash * Cc("d"))
10566                                + parsers.colon
10567                                * (parsers.dash
```

```
                                  - #(parsers.dash
                                     * (parsers.spacechar
                                       + table_hline_separator
                                       + parsers.newline)))^1
                                * (parsers.colon * Cc("c")
                                  + parsers.dash * Cc("l"))

      local table_hline = pipe_table_row(false
                                        , table_hline_column
                                        , table_hline_separator
                                        , table_hline_column)

      local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
                                        * parsers.optionalspace * parsers.newline)^0
                                      * parsers.check_minimal_indent_and_trail
                                      * (P("Table")^-1 * parsers.colon)
                                      * parsers.optionalspace

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

      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

      local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_inden
                          * table_hline * parsers.newline
                          * ((parsers.check_minimal_indent / {}) * table_row * parsers.
                        / make_pipe_table_rectangular
                        * table_caption^-1
                        / writer.table

      self.insert_pattern("Block after Blockquote",
                          PipeTable, "PipeTable")
    end
  }
end
```

3.1.6.15 Raw Attributes

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

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

```

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

```
10654       function self.rawInline(s, attr)
10655         if not self.is_writing then return "" end
10656         if self.flatten_inlines then return s end
10657         local name = util.cache_verbatim(options.cacheDir, s)
10658         return {"\\markdownRendererInputRawInline{",
10659                 name,"}{", self.string(attr),"}"}
10660       end
10661     end, extend_reader = function(self)
10662       local writer = self.writer
10663
10664       local RawInline = parsers.inticks
10665                       * parsers.raw_attribute
10666                       / writer.rawInline
10667
10668       self.insert_pattern("Inline before Code",
10669                           RawInline, "RawInline")
10670     end
10671   }
10672 end
```

3.1.6.16 Strike-Through

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

```
10673 M.extensions.strike_through = function()
10674   return {
10675     name = "built-in strike_through syntax extension",
10676     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.

```
10677       function self.strike_through(s)
10678         if self.flatten_inlines then return s end
10679         return {"\\markdownRendererStrikeThrough{",s,"}"}
10680       end
10681     end, extend_reader = function(self)
10682       local parsers = self.parsers
10683       local writer = self.writer
10684
10685       local StrikeThrough = (
10686         parsers.between(parsers.Inline, parsers.doubletildes,
10687                         parsers.doubletildes)
10688       ) / writer.strike_through
10689
10690       self.insert_pattern("Inline after LinkAndEmph",
10691                           StrikeThrough, "StrikeThrough")
10692
10693       self.add_special_character("~")
```

```
    end
  }
end
```

3.1.6.17 Subscripts

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

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

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

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

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

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

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

3.1.6.18 Superscripts

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

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

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

```
      function self.superscript(s)
        if self.flatten_inlines then return s end
```

```
10726         return {"\\markdownRendererSuperscript{",s,"}"}
10727       end
10728     end, extend_reader = function(self)
10729       local parsers = self.parsers
10730       local writer = self.writer
10731
10732       local Superscript = (
10733         parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
10734       ) / writer.superscript
10735
10736       self.insert_pattern("Inline after LinkAndEmph",
10737                           Superscript, "Superscript")
10738
10739       self.add_special_character("^")
10740     end
10741   }
10742 end
```

3.1.6.19 TEX Math

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

```
10743 M.extensions.tex_math = function(tex_math_dollars,
10744                                   tex_math_single_backslash,
10745                                   tex_math_double_backslash)
10746   return {
10747     name = "built-in tex_math syntax extension",
10748     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.

```
10749       function self.display_math(s)
10750         if self.flatten_inlines then return s end
10751         return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
10752       end
```

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

```
10753       function self.inline_math(s)
10754         if self.flatten_inlines then return s end
10755         return {"\\markdownRendererInlineMath{",self.math(s),"}"}
10756       end
10757     end, extend_reader = function(self)
10758       local parsers = self.parsers
10759       local writer = self.writer
10760
10761       local function between(p, starter, ender)
10762         return (starter * Cs(p * (p - ender)^0) * ender)
```

```
10763       end
10764
10765       local function strip_preceding_whitespaces(str)
10766         return str:gsub("^%s*(.-)$", "%1")
10767       end
10768
10769       local allowed_before_closing = B( parsers.backslash * parsers.any
10770                                       + parsers.any * (parsers.any - parsers.backslas
10771
10772       local allowed_before_closing_no_space = B( parsers.backslash * parsers.any
10773                                                + parsers.any * (parsers.nonspacechar
10774
```

The following patterns implement the Pandoc dollar math syntax extension.

```
10775       local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
10776                                  + parsers.backslash^-1
10777                                  * parsers.linechar)
10778                                  - parsers.blankline^2
10779                                  - parsers.dollar
10780
10781       local inline_math_opening_dollars = parsers.dollar
10782                                         * #(parsers.nonspacechar)
10783
10784       local inline_math_closing_dollars = allowed_before_closing_no_space
10785                                         * parsers.dollar
10786                                         * -#(parsers.digit)
10787
10788       local inline_math_dollars = between(Cs( dollar_math_content),
10789                                           inline_math_opening_dollars,
10790                                           inline_math_closing_dollars)
10791
10792       local display_math_opening_dollars  = parsers.dollar
10793                                           * parsers.dollar
10794
10795       local display_math_closing_dollars  = parsers.dollar
10796                                           * parsers.dollar
10797
10798       local display_math_dollars = between(Cs( dollar_math_content),
10799                                            display_math_opening_dollars,
10800                                            display_math_closing_dollars)
```

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

```
10801       local backslash_math_content  = (parsers.newline * (parsers.check_optional_inde
10802                                     + parsers.linechar)
10803                                     - parsers.blankline^2
```

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

```
10804       local inline_math_opening_double  = parsers.backslash
10805                                         * parsers.backslash
10806                                         * parsers.lparent
10807
10808       local inline_math_closing_double  = allowed_before_closing
10809                                         * parsers.spacechar^0
10810                                         * parsers.backslash
10811                                         * parsers.backslash
10812                                         * parsers.rparent
10813
10814       local inline_math_double  = between(Cs( backslash_math_content),
10815                                           inline_math_opening_double,
10816                                           inline_math_closing_double)
10817                                 / strip_preceding_whitespaces
10818
10819       local display_math_opening_double = parsers.backslash
10820                                         * parsers.backslash
10821                                         * parsers.lbracket
10822
10823       local display_math_closing_double = allowed_before_closing
10824                                         * parsers.spacechar^0
10825                                         * parsers.backslash
10826                                         * parsers.backslash
10827                                         * parsers.rbracket
10828
10829       local display_math_double = between(Cs( backslash_math_content),
10830                                           display_math_opening_double,
10831                                           display_math_closing_double)
10832                                 / strip_preceding_whitespaces
```

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

```
10833       local inline_math_opening_single  = parsers.backslash
10834                                         * parsers.lparent
10835
10836       local inline_math_closing_single  = allowed_before_closing
10837                                         * parsers.spacechar^0
10838                                         * parsers.backslash
10839                                         * parsers.rparent
10840
10841       local inline_math_single  = between(Cs( backslash_math_content),
10842                                           inline_math_opening_single,
10843                                           inline_math_closing_single)
10844                                 / strip_preceding_whitespaces
10845
10846       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
```

3.1.6.20 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.

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

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t`.

```
      function self.jekyllData(d, t, p)
        if not self.is_writing then return "" end

        local buf = {}

        local keys = {}
        for k, _ in pairs(d) do
          table.insert(keys, k)
        end
        table.sort(keys)

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

        if #d > 0 then
            table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
            table.insert(buf, self.identifier(p or "null"))
            table.insert(buf, "}{")
            table.insert(buf, #keys)
            table.insert(buf, "}")
        else
            table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
            table.insert(buf, self.identifier(p or "null"))
            table.insert(buf, "}{")
            table.insert(buf, #keys)
```

```
10928             table.insert(buf, "}")
10929         end
10930
10931         for _, k in ipairs(keys) do
10932           local v = d[k]
10933           local typ = type(v)
10934           k = tostring(k or "null")
10935           if typ == "table" and next(v) ~= nil then
10936             table.insert(
10937               buf,
10938               self.jekyllData(v, t, k)
10939             )
10940           else
10941             k = self.identifier(k)
10942             v = tostring(v)
10943             if typ == "boolean" then
10944               table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
10945               table.insert(buf, k)
10946               table.insert(buf, "}{")
10947               table.insert(buf, v)
10948               table.insert(buf, "}")
10949             elseif typ == "number" then
10950               table.insert(buf, "\\markdownRendererJekyllDataNumber{")
10951               table.insert(buf, k)
10952               table.insert(buf, "}{")
10953               table.insert(buf, v)
10954               table.insert(buf, "}")
10955             elseif typ == "string" then
10956               table.insert(buf, "\\markdownRendererJekyllDataString{")
10957               table.insert(buf, k)
10958               table.insert(buf, "}{")
10959               table.insert(buf, t(v))
10960               table.insert(buf, "}")
10961             elseif typ == "table" then
10962               table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
10963               table.insert(buf, k)
10964               table.insert(buf, "}")
10965             else
10966               error(format("Unexpected type %s for value of " ..
10967                            "YAML key %s", typ, k))
10968             end
10969           end
10970         end
10971
10972         if #d > 0 then
10973           table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
10974         else
```

```
10975           table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
10976         end
10977
10978         if not p then
10979           table.insert(buf, "\\markdownRendererJekyllDataEnd")
10980         end
10981
10982         return buf
10983       end
10984     end, extend_reader = function(self)
10985       local parsers = self.parsers
10986       local writer = self.writer
10987
10988       local JekyllData
10989                   = Cmt( C((parsers.line - P("---") - P("..."))^0)
10990                        , function(s, i, text) -- luacheck: ignore s i
10991                            local data
10992                            local ran_ok, _ = pcall(function()
10993                              -- TODO: Replace with `require("tinyyaml")` in TeX Liv
10994                              local tinyyaml = require("markdown-tinyyaml")
10995                              data = tinyyaml.parse(text, {timestamps=false})
10996                            end)
10997                            if ran_ok and data ~= nil then
10998                              return true, writer.jekyllData(data, function(s)
10999                                return self.parser_functions.parse_blocks_nested(s)
11000                              end, nil)
11001                            else
11002                              return false
11003                            end
11004                          end
11005                        )
11006
11007       local UnexpectedJekyllData
11008                   = P("---")
11009                   * parsers.blankline / 0
11010                   * #(-parsers.blankline)  -- if followed by blank, it's thematic b
11011                   * JekyllData
11012                   * (P("---") + P("..."))
11013
11014       local ExpectedJekyllData
11015                   = ( P("---")
11016                     * parsers.blankline / 0
11017                     * #(-parsers.blankline)  -- if followed by blank, it's thematic
11018                     )^-1
11019                   * JekyllData
11020                   * (P("---") + P("..."))^-1
11021
```

```
      self.insert_pattern("Block before Blockquote",
                          UnexpectedJekyllData, "UnexpectedJekyllData")
      if expect_jekyll_data then
        self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
      end
    end
  }
end
```

3.1.7 Conversion from Markdown to Plain TEX

The `new` function returns a conversion function that takes a markdown string and turns it into a plain TEX output. See Section 2.1.1.

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

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

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

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
      elseif 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 = {}

  if options.bracketedSpans then
    local bracketed_spans_extension = M.extensions.bracketed_spans()
    table.insert(extensions, bracketed_spans_extension)
  end

  if options.contentBlocks then
    local content_blocks_extension = M.extensions.content_blocks(
```

```
11058       options.contentBlocksLanguageMap)
11059     table.insert(extensions, content_blocks_extension)
11060   end
11061
11062   if options.definitionLists then
11063     local definition_lists_extension = M.extensions.definition_lists(
11064       options.tightLists)
11065     table.insert(extensions, definition_lists_extension)
11066   end
11067
11068   if options.fencedCode then
11069     local fenced_code_extension = M.extensions.fenced_code(
11070       options.blankBeforeCodeFence,
11071       options.fencedCodeAttributes,
11072       options.rawAttribute)
11073     table.insert(extensions, fenced_code_extension)
11074   end
11075
11076   if options.fencedDivs then
11077     local fenced_div_extension = M.extensions.fenced_divs(
11078       options.blankBeforeDivFence)
11079     table.insert(extensions, fenced_div_extension)
11080   end
11081
11082   if options.headerAttributes then
11083     local header_attributes_extension = M.extensions.header_attributes()
11084     table.insert(extensions, header_attributes_extension)
11085   end
11086
11087   if options.inlineCodeAttributes then
11088     local inline_code_attributes_extension =
11089       M.extensions.inline_code_attributes()
11090     table.insert(extensions, inline_code_attributes_extension)
11091   end
11092
11093   if options.jekyllData then
11094     local jekyll_data_extension = M.extensions.jekyll_data(
11095       options.expectJekyllData)
11096     table.insert(extensions, jekyll_data_extension)
11097   end
11098
11099   if options.linkAttributes then
11100     local link_attributes_extension =
11101       M.extensions.link_attributes()
11102     table.insert(extensions, link_attributes_extension)
11103   end
11104
```

```
11105   if options.lineBlocks then
11106     local line_block_extension = M.extensions.line_blocks()
11107     table.insert(extensions, line_block_extension)
11108   end
11109
11110   if options.mark then
11111     local mark_extension = M.extensions.mark()
11112     table.insert(extensions, mark_extension)
11113   end
11114
11115   if options.pipeTables then
11116     local pipe_tables_extension = M.extensions.pipe_tables(
11117       options.tableCaptions, options.tableAttributes)
11118     table.insert(extensions, pipe_tables_extension)
11119   end
11120
11121   if options.rawAttribute then
11122     local raw_inline_extension = M.extensions.raw_inline()
11123     table.insert(extensions, raw_inline_extension)
11124   end
11125
11126   if options.strikeThrough then
11127     local strike_through_extension = M.extensions.strike_through()
11128     table.insert(extensions, strike_through_extension)
11129   end
11130
11131   if options.subscripts then
11132     local subscript_extension = M.extensions.subscripts()
11133     table.insert(extensions, subscript_extension)
11134   end
11135
11136   if options.superscripts then
11137     local superscript_extension = M.extensions.superscripts()
11138     table.insert(extensions, superscript_extension)
11139   end
11140
11141   if options.texMathDollars or
11142      options.texMathSingleBackslash or
11143      options.texMathDoubleBackslash then
11144     local tex_math_extension = M.extensions.tex_math(
11145       options.texMathDollars,
11146       options.texMathSingleBackslash,
11147       options.texMathDoubleBackslash)
11148     table.insert(extensions, tex_math_extension)
11149   end
11150
11151   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 = util.lookup_files(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,
        [[User-defined syntax extension "]] .. pathname
        .. [[" does not specify mandatory field "api_version"]])
      assert(type(user_extension.api_version) == "number",
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "api_version" of type "]]
        .. type(user_extension.api_version)
        .. [[" but "number" was expected]])
      assert(user_extension.api_version > 0
         and user_extension.api_version <= metadata.user_extension_api_version,
        [[User-defined syntax extension "]] .. pathname
        .. [[" uses syntax extension API version "]]
```

```
        .. user_extension.api_version .. [[ but markdown.lua ]]
        .. metadata.version .. [[ uses API version ]]
        .. metadata.user_extension_api_version
        .. [[, which is incompatible]])

      assert(user_extension.grammar_version ~= nil,
        [[User-defined syntax extension "]] .. pathname
        .. [[" does not specify mandatory field "grammar_version"]])
      assert(type(user_extension.grammar_version) == "number",
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "grammar_version" of type "]]
        .. type(user_extension.grammar_version)
        .. [[" but "number" was expected]])
      assert(user_extension.grammar_version == metadata.grammar_version,
        [[User-defined syntax extension "]] .. pathname
        .. [[" uses grammar version "]] .. user_extension.grammar_version
        .. [[ but markdown.lua ]] .. metadata.version
        .. [[ uses grammar version ]] .. metadata.grammar_version
        .. [[, which is incompatible]])

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

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.6.)

```
      local extension = {
        name = [[user-defined "]] .. pathname .. [[" syntax extension]],
        extend_reader = user_extension.finalize_grammar,
        extend_writer = function() end,
      }
      return extension
    end)(user_extension_filename)
    table.insert(extensions, user_extension)
  end
```

Produce 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
```

3.1.8 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

```

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

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

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

```
local kpse
(function()
  local should_initialize = package.loaded.kpse == nil
```

```
                       or tex.initialize ~= nil
  kpse = require("kpse")
  if should_initialize then
    kpse.set_program_name("luatex")
  end
end)()
local md = require("markdown")
```

Since we are loading the rest of the Lua implementation dynamically, check that both the `markdown` module and the command line implementation are the same version.

```
if metadata.version ~= md.metadata.version then
  warn("markdown-cli.lua " .. metadata.version .. " used with " ..
       "markdown.lua " .. md.metadata.version .. ".")
end
local convert = md.new(options)
local output = convert(input)

if output_filename then
  local output_file = assert(io.open(output_filename, "w"),
    [[Could not open file "]] .. output_filename .. [[" for writing]])
  assert(output_file:write(output))
  assert(output_file:close())
else
  assert(io.write(output))
end
```

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

```
if options.cacheDir then
  lfs.rmdir(options["cacheDir"])
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. Futhermore, 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{}%
```

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_renderer_prototype:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} \unskip }
      }
  }
\cs_new:Nn
  \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
      { #1 }
      { #2 }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
      { #1 }
      { #2 }
  }
\ExplSyntaxOff
```

3.2.3.2 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 }
}
\def\markdownRendererJekyllDataStringPrototype#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
              {
                \tl_gput_right:Nx
                  \g_@@_formatted_lua_options_tl
                  { "##1" ,~ }
              }
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { \c_right_brace_str ,~ }
          }
      }
      {
        \@@_get_option_value:nN
          { #1 }
          \l_tmpa_tl
        \tl_gput_right:Nx
          \g_@@_formatted_lua_options_tl
          { #1~=~ " \l_tmpa_tl " ,~ }
      }
  }
\cs_generate_variant:Nn
  \clist_map_inline:nn
  { Vn }
\let\markdownPrepareLuaOptions=\@@_format_lua_options:
```

```
\def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
\ExplSyntaxOff
```

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

```
\def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```
  local lfs = require("lfs")
  local cacheDir = "\markdownOptionCacheDir"
  if not lfs.isdir(cacheDir) then
    assert(lfs.mkdir(cacheDir))
  end
```

Next, load the `markdown` module and create a converter function using the plain TEX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
  local md = require("markdown")
  local convert = md.new(\markdownLuaOptions)
}%
```

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.

```
  lfs.rmdir(cacheDir)
}%
```

3.2.5 Buffering 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_ε `\filecontents` macro to plain TEX.

```
\begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as

an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `stripPercentSigns` is enabled.

```
\catcode`\^^M=13%
\catcode`\^^I=13%
\catcode`|=0%
\catcode`\\=12%
|catcode`@=14%
|catcode`|%=12@
|gdef|markdownReadAndConvert#1#2{@
  |begingroup@
```

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

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

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

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

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

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

```
    }@
  }@
```

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

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

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

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

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

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

Repeat with the next line.

```
    ^^M}@
```

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

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

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

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

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

Reset the character categories back to the former state.

```
|endgroup
```

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 }
              }
          }
      }
    \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 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TEX.

```
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code. Furthermore, use the ampersand symbol to specify parameters.

```
  \catcode`|=0%
  \catcode`\\=12%
  \catcode`|&=6%
  |gdef|markdownInput#1{%
```

Change the category code of the percent sign (`%`) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
    |begingroup
    |catcode`|%=12
```

Furthermore, also change the category code of the hash sign (`#`) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```
    |catcode`|#=12
```

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

```
    |markdownIfOption{frozenCache}{%
      |ifnum|markdownOptionFrozenCacheCounter=0|relax
        |markdownInfo{Reading frozen cache from
          "|markdownOptionFrozenCacheFileName"}%
        |input|markdownOptionFrozenCacheFileName|relax
      |fi
      |markdownInfo{Including markdown document number
```

```
        "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
      |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }{%
      |markdownInfo{Including markdown document "&1"}%
```

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

```
      |openin|markdownInputFileStream&1
      |closein|markdownInputFileStream
      |markdownPrepareLuaOptions
      |markdownLuaExecute{%
        |markdownPrepare
        local file = assert(io.open("&1", "r"),
          [[Could not open file "&1" for reading]])
        local input = assert(file:read("*a"))
        assert(file:close())
        print(convert(input))
        |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 implemenation makes use of the fact that, apart from some subtle differences, LaTeX implements the majority of the plain TeX format [11, 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 Logging Facilities

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

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the LaTeX interface (see Section 2.3.2).

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

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

```
\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]* }
      {
```

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.

```
        \c { markdownReadAndConvert@markdown } { }
      }
      {
        \group_end:
        \markdownReadAndConvert@markdown { }
      }
  }
  { \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.
  }
\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@markdown#1<%
    |markdownReadAndConvert<\end{markdown#1}>%
                           <|end<markdown#1>>>%
|endgroup
```

3.3.3 Options

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

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

3.3.4 Themes

This section overrides the plain TEX implementation of the theme-loading mechanism from Section 3.2.2. Futhermore, 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
        {
```

```
          \@@_load_theme:nn
            { #1 }
            { #2 }
        }
    \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%
```

```
12155   .\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.

```
12156 \newcommand\markdown@witiko@graphicx@http@download[2]{%
12157   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:

```
12158 \begingroup
12159 \catcode`\%=12
12160 \catcode`\^^A=14
```

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

```
12161 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
12162   \begingroup
12163     \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:

```
12164     \markdownIfOption{frozenCache}{}{^^A
12165       \immediate\write18{^^A
12166         mkdir -p "\markdownOptionCacheDir";
12167         if printf '%s' "#3" | grep -q -E '^https?:';
12168         then
```

The image will be downloaded to the pathname cacheDir/⟨*the MD5 digest of the image URL*⟩.⟨*the suffix of the image URL*⟩:

```
12169           OUTPUT_PREFIX="\markdownOptionCacheDir";
12170           OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
12171           OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]//')";
12172           OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

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

```
12173           if ! [ -e "$OUTPUT" ];
12174           then
12175             \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
12176             printf '%s' "$OUTPUT" > "\filename";
12177           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:

```
12178         else
12179           printf '%s' '#3' > "\filename";
12180         fi}}^^A
```

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

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

The witiko/markdown/defaults LaTeX theme provides default definitions for token renderer prototypes. First, the LaTeX theme loads the plain TeX theme with the default definitions for plain TeX:

```
\markdownLoadPlainTeXTheme
```

Next, the LaTeX theme overrides some of the plain TeX definitions. See Section 3.3.5 for the actual definitions.

3.3.5 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}
```

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

```
\@ifclassloaded{beamer}{}{%
  \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
  \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
}
```

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

```
\ExplSyntaxOn
\@ifpackageloaded{paralist}{
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_number_style_tl
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_delimiter_style_tl
  \cs_new:Nn
    \@@_latex_fancy_list_item_label_number:nn
    {
      \str_case:nn
        { #1 }
        {
          { Decimal } { #2 }
          { LowerRoman } { \int_to_roman:n { #2 } }
```

```
          { UpperRoman } { \int_to_Roman:n { #2 } }
          { LowerAlpha } { \int_to_alph:n { #2 } }
          { UpperAlpha } { \int_to_Alph:n { #2 } }
        }
    }
  \cs_new:Nn
    \@@_latex_fancy_list_item_label_delimiter:n
    {
      \str_case:nn
        { #1 }
        {
          { Default } { . }
          { OneParen } { ) }
          { Period } { . }
        }
    }
  \cs_new:Nn
    \@@_latex_fancy_list_item_label:nnn
    {
      \@@_latex_fancy_list_item_label_number:nn
        { #1 }
        { #3 }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \cs_new:Nn
    \@@_latex_paralist_style:nn
    {
      \str_case:nn
        { #1 }
        {
          { Decimal } { 1 }
          { LowerRoman } { i }
          { UpperRoman } { I }
          { LowerAlpha } { a }
          { UpperAlpha } { A }
        }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \markdownSetup{rendererPrototypes={
```

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 }
      \tl_set:Nn
        \l_tmpa_tl
        {
          \plpartopsep=\partopsep
          \pltopsep=\topsep
        }
      \@@_if_option:nTF
        { startNumber }
        {
          \tl_put_right:Nn
            \l_tmpa_tl
            { \begin{compactenum} }
        }
        {
          \tl_put_right: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_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,[33] we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```
  codeSpan = {%
    \ifmmode
      \text{#1}%
    \else
      \texttt{#1}%
    \fi
  }}}
\ExplSyntaxOn
\markdownSetup{
  rendererPrototypes = {
    contentBlock = {
      \str_case:nnF
        { #1 }
        {
          { csv }
            {
              \begin{table}
                \begin{center}
                  \csvautotabular{#3}
                \end{center}
```

[33] 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.

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

```
\ExplSyntaxOn
\seq_new:N \l_@@_header_identifiers_seq
\markdownSetup{
  rendererPrototypes = {
    headerAttributeContextBegin = {
      \seq_clear:N \l_@@_header_identifiers_seq
      \markdownSetup
        {
          renderers = {
            attributeIdentifier = {
```

```
              \seq_put_right:Nn
                \l_@@_header_identifiers_seq
                { ##1 }
            },
          },
        }
    },
    headerAttributeContextEnd = {
      \seq_map_inline:Nn
        \l_@@_header_identifiers_seq
        { \label { ##1 } }
    },
  },
}
\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}}}}
```

3.3.5.1 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
          }
          {
```

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}}}}
}%
```

3.3.5.2 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
}
```

3.3.5.3 HTML elements

If the `html` option is enabled and we are using TEX4ht[34], we will pass HTML elements to the output HTML document unchanged.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
```

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

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

3.3.5.4 Citations

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

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\RequirePackage{gobble}
\def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#4\relax
    \ifx\relax#5\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \cite{#1#2#6}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobblethree
      \fi
    \else%  Before a postnote (#5), dump the accumulator
      \ifx\relax#1\relax\else
        \cite{#1}%
      \fi
      \cite[#5]{#6}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
      \else
        \expandafter\expandafter\expandafter
```

```
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobblethree
    \fi
  \else%  Before a prenote (#4), dump the accumulator
    \ifx\relax#1\relax\else
      \cite{#1}%
    \fi
    \ifnum\markdownLaTeXCitationsCounter>1\relax
      \space  % Insert a space before the prenote in later citations
    \fi
    #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXBasicCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobblethree
  \fi\markdownLaTeXBasicCitations{#1#2#6},}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% Natbib implementation
\def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \citep{#1,#5}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
```

```
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  Before a postnote (#4), dump the accumulator
      \ifx\relax#1\relax\else
        \citep{#1}%
      \fi
      \citep[][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  Before a prenote (#3), dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citep{#1}%
    \fi
    \citep[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibCitations{#1,#5}}
\def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \citet{#1,#5}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
```

```
        \@gobbletwo
      \fi
    \else%  After a prenote or a postnote, dump the accumulator
      \ifx\relax#1\relax\else
        \citet{#1}%
      \fi
      , \citet[#3][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
        ,
      \else
        \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
          ,
        \fi
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibTextCitations
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  After a prenote or a postnote, dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citet{#1}%
    \fi
    , \citet[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
      ,
    \else
      \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
        ,
      \fi
    \fi
    \expandafter\expandafter\expandafter
    \markdownLaTeXNatbibTextCitations
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibTextCitations{#1,#5}}

% BibLaTeX implementation
\def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
```

```
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \autocites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
\def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \textcites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}

\markdownSetup{rendererPrototypes = {
  cite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXCitations
      \expandafter{\expandafter}%
    }},
  textCite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
```

```
        \markdownLaTeXNatbibTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXTextCitations
      \expandafter{\expandafter}%
    }}}}
```

3.3.5.5 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}}}
```

3.3.5.6 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}
```

3.3.5.7 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
    },
  }}
}{}

```

3.3.5.8 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
```

3.3.5.9 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}%
      },
    }
  }
}{}
```

3.3.5.10 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}%
      },
    }
  }
}{}
```

3.3.5.11 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
```

3.3.5.12 Raw Attributes

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

```
\ExplSyntaxOn
```

```
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\ExplSyntaxOff
\fi % Closes `\markdownIfOption{plain}{\iffalse}{\iftrue}`
```

3.3.6 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. [12, sec. 31]. According to Eijkhout [13, 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. Futhermore, 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
```

```
13240       {
13241         \regex_extract_once:nnN
13242           { \w* }
13243           { #2 }
13244           \l_tmpa_seq
13245         \seq_pop_left:NN
13246           \l_tmpa_seq
13247           \l_tmpa_tl
13248         \typefile[\l_tmpa_tl][]{#1}
13249       }
13250   }
13251 \ExplSyntaxOff
13252 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
13253 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
13254 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
13255 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
13256 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
13257 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
13258 \def\markdownRendererThematicBreakPrototype{%
13259   \blackrule[height=1pt, width=\hsize]}%
13260 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
13261 \def\markdownRendererTickedBoxPrototype{$\boxtimes$}
13262 \def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
13263 \def\markdownRendererUntickedBoxPrototype{$\square$}
13264 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
13265 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
13266 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
13267 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%
```

3.4.3.2 Tables

There is a basic implementation of tables.

```
13268 \newcount\markdownConTeXtRowCounter
13269 \newcount\markdownConTeXtRowTotal
13270 \newcount\markdownConTeXtColumnCounter
13271 \newcount\markdownConTeXtColumnTotal
13272 \newtoks\markdownConTeXtTable
13273 \newtoks\markdownConTeXtTableFloat
13274 \def\markdownRendererTablePrototype#1#2#3{%
13275   \markdownConTeXtTable={}%
13276   \ifx\empty#1\empty
13277     \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_renderer_prototype:nn
              { #1 }
              { context }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { context }
          {
            \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
```

```
        \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset_eq:NN
  \markdownRendererInputRawBlockPrototype
  \markdownRendererInputRawInlinePrototype
\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] 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).

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

[11] 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).

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

[13] 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

`autoIdentifiers` *18*, 30, 71, 85

`blankBeforeBlockquote` *18*
`blankBeforeCodeFence` *19*
`blankBeforeDivFence` *19*
`blankBeforeHeading` *19*
`blankBeforeList` *20*
`bracketedSpans` *20*, 73
`breakableBlockquotes` *20*

`cacheDir` 4, 15, 17, 53, 127, 275, 341, 353
`citationNbsps` *21*
`citations` *21*, 76
`codeSpans` *22*
`contentBlocks` 17, *22*
`contentBlocksLanguageMap` 17
`contentLevel` *23*

`debugExtensions` 9, 17, *23*, 272
`debugExtensionsFileName` 17, 23
`defaultOptions` *9*, 47, 321
`definitionLists` *23*, 80

eagerCache *15*
entities.char_entity *185*
entities.dec_entity *185*
entities.hex_entity *185*
entities.hex_entity_with_x_char *185*
expandtabs *237*
expectJekyllData *24*
extensions 25, *134*, 277
extensions.bracketed_spans *277*
extensions.citations *278*
extensions.content_blocks *282*
extensions.definition_lists *285*
extensions.fancy_lists *287*
extensions.fenced_code *292*
extensions.fenced_divs *297*
extensions.header_attributes *300*
extensions.inline_code_attributes *301*
extensions.jekyll_data *318*
extensions.line_blocks *302*
extensions.link_attributes *303*
extensions.mark *303*
extensions.notes *305*
extensions.pipe_table *307*
extensions.raw_inline *311*
extensions.strike_through *312*
extensions.subscripts *313*
extensions.superscripts *313*
extensions.tex_math *314*

fancyLists *27*, 94–99, 354
fencedCode *27*, 35, 77, 83, 99, 351
fencedCodeAttributes *28*, 71
fencedDiv 84
fencedDivs *28*, 37
finalizeCache 16, 18, *29*, 29, 53, 126, 276
frozenCache 18, 29, 53, 126, 129, 130, 352, 353
frozenCacheCounter 29, 276, 345, 346
frozenCacheFileName 18, 29, 53, 276

gfmAutoIdentifiers 18, *29*, 71, 85

hashEnumerators *30*
headerAttributes *30*, 37, 71, 85

html *31*, 87, 88, 363
hybrid *31*, 36, 42, 44, 56, 64, 101, 127, 190, 238, 345

inlineCodeAttributes *31*, 71, 78
inlineNotes *32*
\input 50
\inputmarkdown *131*, 132, 376
inputTempFileName 54, 56, 342, 343
iterlines *237*

jekyllData 3, 24, 25, *32*, 107–111

languages_json *282*, 282
lineBlocks *33*, 90
linkAttributes *33*, 71, 89, 92, 255, 374

mark *34*, 92, 373
\markdown *125*
markdown *124*, 125, 347
markdown* *124*, 125, 126, 347
\markdown_jekyll_data_concatenate_address:NN *336*
\markdown_jekyll_data_pop: *337*
\markdown_jekyll_data_push:nN *337*
\markdown_jekyll_data_push_address_segment:n *335*
\markdown_jekyll_data_set_keyval:Nn *337*
\markdown_jekyll_data_set_keyvals:nn *337*
\markdown_jekyll_data_update_address_tls: *336*
\markdownBegin *49*, 49, 50, 123, 125, 132
\markdownCleanup *341*
\markdownEnd *49*, 49, 50, 123, 125, 132
\markdownError *123*, 123
\markdownEscape *49*, 51, 346
\markdownIfOption *52*
\markdownIfSnippetExists *66*
\markdownInfo *123*, 123
\markdownInput *49*, 50, 124–126, 132, 345, 347
\markdownInputFileStream *341*
\markdownInputPlainTeX *347*
\markdownLoadPlainTeXTheme *127*, 134, 330
\markdownLuaExecute *344*, 345
\markdownLuaOptions *339*, 341
\markdownMakeOther *123*, 376
\markdownOptionFinalizeCache *53*

\markdownOptionFrozenCache *53*
\markdownOptionHybrid *56*
\markdownOptionInputTempFileName *53*
\markdownOptionNoDefaults *55*
\markdownOptionOutputDir *53*, 54
\markdownOptionPlain *55*
\markdownOptionStripPercentSigns *56*
\markdownOutputFileStream *341*
\markdownPrepare *341*
\markdownPrepareLuaOptions *339*
\markdownReadAndConvert *123*, 341, 347, 348, 377
\markdownReadAndConvertProcessLine *343*, 343
\markdownReadAndConvertStripPercentSigns *342*
\markdownReadAndConvertTab *341*
\markdownRendererAttributeClassName *72*
\markdownRendererAttributeIdentifier *71*
\markdownRendererAttributeKeyValue *72*
\markdownRendererBlockQuoteBegin *73*
\markdownRendererBlockQuoteEnd *73*
\markdownRendererBracketedSpanAttributeContextBegin *73*
\markdownRendererBracketedSpanAttributeContextEnd *73*
\markdownRendererCite *76*, 76
\markdownRendererCodeSpan *77*
\markdownRendererCodeSpanAttributeContextBegin *78*
\markdownRendererCodeSpanAttributeContextEnd *78*
\markdownRendererContentBlock *78*, 79
\markdownRendererContentBlockCode *79*
\markdownRendererContentBlockOnlineImage *79*
\markdownRendererDisplayMath *106*
\markdownRendererDlBegin *80*
\markdownRendererDlBeginTight *80*
\markdownRendererDlDefinitionBegin *81*
\markdownRendererDlDefinitionEnd *81*
\markdownRendererDlEnd *82*
\markdownRendererDlEndTight *82*
\markdownRendererDlItem *80*
\markdownRendererDlItemEnd *81*
\markdownRendererDocumentBegin *93*
\markdownRendererDocumentEnd *93*
\markdownRendererEllipsis 38, *82*
\markdownRendererEmphasis *83*, 113
\markdownRendererFancyOlBegin *95*, 95

\markdownRendererFancyOlBeginTight 95
\markdownRendererFancyOlEnd 98
\markdownRendererFancyOlEndTight 98
\markdownRendererFancyOlItem 96
\markdownRendererFancyOlItemEnd 97
\markdownRendererFancyOlItemWithNumber 97
\markdownRendererFencedCodeAttributeContextBegin 83
\markdownRendererFencedCodeAttributeContextEnd 83
\markdownRendererFencedDivAttributeContextBegin 84
\markdownRendererFencedDivAttributeContextEnd 84
\markdownRendererHalfTickedBox 107
\markdownRendererHardLineBreak 91
\markdownRendererHeaderAttributeContextBegin 85
\markdownRendererHeaderAttributeContextEnd 85
\markdownRendererHeadingFive 86
\markdownRendererHeadingFour 86
\markdownRendererHeadingOne 85
\markdownRendererHeadingSix 87
\markdownRendererHeadingThree 86
\markdownRendererHeadingTwo 86
\markdownRendererImage 88
\markdownRendererImageAttributeContextBegin 89
\markdownRendererImageAttributeContextEnd 89
\markdownRendererInlineHtmlComment 87
\markdownRendererInlineHtmlTag 87
\markdownRendererInlineMath 106
\markdownRendererInputBlockHtmlElement 88
\markdownRendererInputFencedCode 77
\markdownRendererInputRawBlock 99
\markdownRendererInputRawInline 99
\markdownRendererInputVerbatim 77
\markdownRendererInterblockSeparator 89
\markdownRendererJekyllDataBegin 107
\markdownRendererJekyllDataBoolean 109
\markdownRendererJekyllDataEmpty 111
\markdownRendererJekyllDataEnd 108
\markdownRendererJekyllDataMappingBegin 108
\markdownRendererJekyllDataMappingEnd 108
\markdownRendererJekyllDataNumber 110
\markdownRendererJekyllDataSequenceBegin 109
\markdownRendererJekyllDataSequenceEnd 109
\markdownRendererJekyllDataString 110

\markdownRendererLineBlockBegin *90*
\markdownRendererLineBlockEnd *90*
\markdownRendererLink *91*, 113
\markdownRendererLinkAttributeContextBegin *92*
\markdownRendererLinkAttributeContextEnd *92*
\markdownRendererMark *92*
\markdownRendererNbsp *93*
\markdownRendererNote *94*
\markdownRendererOlBegin *94*
\markdownRendererOlBeginTight *94*
\markdownRendererOlEnd *97*
\markdownRendererOlEndTight *98*
\markdownRendererOlItem 38, *95*
\markdownRendererOlItemEnd *96*
\markdownRendererOlItemWithNumber 38, *96*
\markdownRendererParagraphSeparator *90*
\markdownRendererReplacementCharacter *100*
\markdownRendererSectionBegin *100*
\markdownRendererSectionEnd *100*
\markdownRendererSoftLineBreak *91*
\markdownRendererStrikeThrough *103*
\markdownRendererStrongEmphasis *83*
\markdownRendererSubscript *104*
\markdownRendererSuperscript *104*
\markdownRendererTable *105*
\markdownRendererTableAttributeContextBegin *104*
\markdownRendererTableAttributeContextEnd *104*
\markdownRendererTextCite *76*
\markdownRendererThematicBreak *106*
\markdownRendererTickedBox *107*
\markdownRendererUlBegin *74*
\markdownRendererUlBeginTight *74*
\markdownRendererUlEnd *75*
\markdownRendererUlEndTight *75*
\markdownRendererUlItem *74*
\markdownRendererUlItemEnd *75*
\markdownRendererUntickedBox *107*
\markdownSetup *52*, 52, 56, 126, 132, 348, 349
\markdownSetupSnippet *65*, 65
\markdownWarning *123*, 123

new *7*, 16, 321

notes *34*, 94

parsers *204*, 236, 237
pipeTables 6, *35*, 41, 105
preserveTabs *35*, 39, 237

rawAttribute *35*, 36, 99
reader *8*, 26, 134, 204, 236, 277
reader->add_special_character *8*, 9, 26, 271
reader->auto_link_email *261*
reader->auto_link_url *261*
reader->create_parser *237*
reader->finalize_grammar *267*, 325
reader->initialize_named_group *271*
reader->insert_pattern *8*, 9, 26, 267, 268, 273
reader->lookup_reference *249*
reader->normalize_tag *237*
reader->options *236*
reader->parser_functions *237*
reader->parser_functions.name *237*
reader->parsers *236*, 236, 237
reader->register_link *249*
reader->update_rule 267, 270, 273
reader->writer *236*
reader.new *236*, 236, 325
relativeReferences *36*

\setupmarkdown *132*
shiftHeadings 6, 37
singletonCache *16*
slice 6, 37, 187, 198, 199
smartEllipses *38*, 82, 127
\startmarkdown *131*, 132, 377
startNumber *38*, 95–97
\stopmarkdown *131*, 132, 377
strikeThrough *38*, 103, 373
stripIndent *39*, 238
stripPercentSigns 342
subscripts *39*, 104
superscripts *40*, 104
syntax *268*, 273

tableAttributes *40*, 104

tableCaptions 6, 40, *41*, 104
taskLists *41*, 107, 363
texComments *42*, 238
texMathDollars *42*, 106
texMathDoubleBackslash *43*, 106
texMathSingleBackslash *43*, 106
tightLists *43*, 74, 75, 80, 82, 94, 95, 98, 99, 354

underscores *44*
util.cache *135*, 135
util.cache_verbatim *135*
util.encode_json_string *136*
util.err *135*
util.escaper *138*
util.expand_tabs_in_line *136*
util.flatten *137*
util.intersperse *138*
util.lookup_files *136*
util.map *138*
util.pathname *139*
util.rope_last *137*
util.rope_to_string *137*
util.table_copy *136*
util.walk *136*, 137

walkable_syntax *8*, 17, 23, 267, 268, 270–273
writer *134*, 134, 186, 277
writer->active_attributes *197*, 197, 198
writer->attribute_type_levels *197*
writer->attributes *195*
writer->block_html_element *193*
writer->blockquote *194*
writer->bulletitem *192*
writer->bulletlist *192*
writer->citations *278*
writer->code *190*
writer->contentblock *282*
writer->defer_call *204*, 204
writer->definitionlist *285*
writer->display_math *314*
writer->div_begin *297*
writer->div_end *297*

writer->document *194*
writer->ellipsis *188*
writer->emphasis *194*
writer->escape *190*
writer->escape_minimal *189*
writer->escape_programmatic_text *189*
writer->escape_typographic_text *189*
writer->escaped_chars *189*, 189
writer->escaped_minimal_strings *189*, 189
writer->escaped_strings *189*
writer->escaped_uri_chars *189*, 189
writer->fancyitem *288*
writer->fancylist *287*
writer->fencedCode *293*
writer->flatten_inlines *186*, 186
writer->get_state *203*
writer->hard_line_break *188*
writer->heading *202*
writer->identifier *190*
writer->image *191*
writer->infostring *190*
writer->inline_html_comment *193*
writer->inline_html_tag *193*
writer->inline_math *314*
writer->interblocksep *188*
writer->is_writing *187*, 187
writer->jekyllData *318*
writer->lineblock *302*
writer->link *191*
writer->mark *303*
writer->math *190*
writer->nbsp *187*
writer->note *305*
writer->options *186*
writer->ordereditem *193*
writer->orderedlist *192*
writer->pack *188*, 276
writer->paragraph *188*
writer->paragraphsep *188*
writer->plain *187*
writer->pop_attributes *197*, 198
writer->push_attributes *197*, 198, 199

writer->rawBlock *293*
writer->rawInline *311*
writer->set_state *203*
writer->slice_begin *187*
writer->slice_end *187*
writer->soft_line_break *188*
writer->space *187*
writer->span *277*
writer->strike_through *312*
writer->string *190*
writer->strong *194*
writer->subscript *313*
writer->suffix *187*
writer->superscript *313*
writer->table *308*
writer->thematic_break *188*
writer->tickbox *194*
writer->uri *190*
writer->verbatim *194*
writer.new *186*, 186, 325