A Markdown Interpreter for TEX

Vít Novotný
witiko@mail.muni.cz

Version 2.5.0
April 10, 2017

Contents

1 Introduction

This document is a reference manual for the Markdown package. It is split into three sections. This section explains the purpose and the background of the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package. Section 3 describes the implementation of the package. It is aimed at the developer of the package and the curious user.

1.1 About Markdown

The Markdown package provides facilities for the conversion of markdown markup to plain TEX. These are provided both in the form of a Lua module and in the form of plain TEX, LATEX, and ConTEXt macro packages that enable the direct inclusion of markdown documents inside TEX documents.

Architecturally, the package consists of the Lunamark v0.5.0 Lua module by John MacFarlane, which was slimmed down and rewritten for the needs of the package. On top of Lunamark sits code for the plain TEX, LATEX, and ConTEXt formats by Vít Novotný.

```
local metadata = {
    version   = "2.5.0",
    comment   = "A module for the conversion from markdown to plain TeX",
    author    = "John MacFarlane, Hans Hagen, Vít Novotný",
    copyright = "2009-2017 John MacFarlane, Hans Hagen; " ..
```

```
                "2016-2017 Vít Novotný",
    license   = "LPPL 1.3"
}
if not modules then modules = { } end
modules['markdown'] = metadata
```

1.2 Feedback

Please use the markdown project page on GitHub[1] to report bugs and submit feature requests. Before making a feature request, please ensure that you have thoroughly studied this manual. 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 on the TEX-LATEX Stack Exchange[2].

1.3 Acknowledgements

I would like to thank the Faculty of Informatics at the Masaryk University in Brno for providing me with the opportunity to work on this package alongside my studies. I would also like to thank the creator of the Lunamark Lua module, John Macfarlane, for releasing Lunamark under a permissive license that enabled its inclusion into the package.

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

1.4 Prerequisites

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

1.4.1 Lua Prerequisites

The Lua part of the package requires that the following Lua modules are available from within the LuaTEX engine:

LPeg $\geq$ 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 $\geq$ 0.10 is included in LuaTEX $\geq$ 0.72.0 (TEXLive $\geq$ 2013).

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

[1] https://github.com/witiko/markdown/issues
[2] https://tex.stackexchange.com

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 footnote tags to the lower case. Selene Unicode is included in all releases of LuaTEX (TEXLive $\geq$ 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 (TEXLive $\geq$ 2008).

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

All the abovelisted modules are statically linked into the current version of the LuaTEX engine (see [1, Section 3.3]).

1.4.2 Plain TEX Prerequisites

The plain TEX part of the package requires that the plain TEX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.4.1) and 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 `\markdownOptionCacheDir` macro before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTEX (TEXLive $\geq$ 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 (see [1, Section 3.2]).

The Lua File System module is statically linked into the LuaTEX engine (see [1, Section 3.3]).

The plain TEX part of the package also requires that either the LuaTEX `\directlua` primitive or the shell access file stream 18 is available in your TEX engine. If only the shell access file stream is available in your TEX engine (as is the case with pdfTEX and XƎTEX) or if you enforce the use of shell using the `\markdownMode` macro, then note the following:

- 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.

- You will need to avoid the use of the `-output-directory` TEX parameter when typesetting a document. The parameter causes auxiliary files to be written to a specified output directory, but the shell will be executed in the current directory. Things will not work out.

1.4.3 LaTeX Prerequisites

The LaTeX part of the package requires that the LaTeX 2_ε format is loaded,

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

all the plain TeX prerequisites (see Section 1.4.2), and the following LaTeX 2_ε packages:

keyval A package that enables the creation of parameter sets. This package is used to provide the `\markdownSetup` macro, the package options processing, as well as the parameters of the `markdown*` LaTeX environment.

url A package that provides the `\url` macro for the typesetting of URLs. It is used to provide the default token renderer prototype (see Section 2.2.4) for links.

graphicx A package that provides the `\includegraphics` macro for the typesetting of images. It is used to provide the corresponding default token renderer prototype (see Section 2.2.4).

paralist A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists. It is used to provide the corresponding default token renderer prototypes (see Section 2.2.4).

ifthen A package that provides a concise syntax for the inspection of macro values. It is used to determine whether or not the paralist package should be loaded based on the user options.

fancyvrb A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code. It is used to provide the corresponding default token renderer prototype (see Section 2.2.4).

csvsimple A package that provides the default token renderer prototype for iA Writer content blocks with the CSV filename extension (see Section 2.2.4).

1.4.4 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 1.4.2), and the following modules:

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the CSV filename extension (see Section 2.2.4).

2 User Guide

This part of the manual 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 purely abstract. It serves as a means of structuring this manual and as a promise to the user that if they only access the package through the interfaces, the future versions of the package should remain backwards compatible.

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.

```
local M = {}
```

2.1.1 Conversion from Markdown to Plain TeX

The Lua interface exposes the `new(options)` method. This method creates converter functions that perform the conversion from markdown to plain TeX according to the table `options` that contains options recognized by the Lua interface. (see Section 2.1.2). 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 Options

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

```
local defaultOptions = {}
```

`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.

```
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.

```
defaultOptions.blankBeforeCodeFence = 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.

```
defaultOptions.blankBeforeHeading = false
```

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

`true`	A blank line separates block quotes.
`false`	Blank lines in the middle of a block quote are ignored.

```
defaultOptions.breakableBlockquotes = false
```

`cacheDir`=⟨*directory name*⟩ default: .

The path to the directory containing auxiliary cache files.

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.

```
defaultOptions.cacheDir = "."
```

citationNbsps=true, false default: false

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

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

```
defaultOptions.citationNbsps = true
```

citations=true, false default: false

true
: Enable the pandoc citation syntax extension:

```
Here is a simple parenthetical citation [@doe99] and here
is a string of several [see @doe99, pp. 33-35; also
@smith04, chap. 1].

A parenthetical citation can have a [prenote @doe99] and
a [@smith04 postnote]. The name of the author can be
suppressed by inserting a dash before the name of an
author as follows [-@smith04].

Here is a simple text citation @doe99 and here is
a string of several @doe99 [pp. 33-35; also @smith04,
chap. 1]. Here is one with the name of the author
suppressed -@doe99.
```

false
: Disable the pandoc citation syntax extension.

```
defaultOptions.citations = false
```

codeSpans=true, false default: true

true
: Enable the code span syntax:

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

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

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

```
defaultOptions.codeSpans = true
```

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

`true` Enable the iA Writer content blocks syntax extension [2]:

```
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.

```
defaultOptions.contentBlocks = false
```

`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. See Section 2.2.3.9 for more information.

```
defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`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.

```
defaultOptions.definitionLists = false
```

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

`true` Enable the commonmark fenced code block extension:

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

  ‘‘‘ html
  <pre>
    <code>
      // Some comments
      line 1 of code
      line 2 of code
      line 3 of code
    </code>
  </pre>
  ‘‘‘
```

`false` Disable the commonmark fenced code block extension.

```
defaultOptions.fencedCode = false
```

`footnotes`=`true`, `false` default: `false`

`true` Enable the pandoc footnote syntax extension:

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

[^1]: Here is the footnote.

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

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

        { some.code }

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

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

`false` Disable the pandoc footnote syntax extension.

```
defaultOptions.footnotes = 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.

```
defaultOptions.hashEnumerators = false
```

`html`=`true`, `false` default: `false`

`true` Enable the recognition of HTML tags, block elements, comments, HTML instructions, and entities in the input. Tags, block elements (along with contents), HTML instructions, and comments 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.

```
defaultOptions.html = false
```

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

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

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

```
defaultOptions.hybrid = false
```

inlineFootnotes=true, false default: false

true
: Enable the pandoc inline footnote syntax extension:

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

false
: Disable the pandoc inline footnote syntax extension.

```
defaultOptions.inlineFootnotes = false
```

preserveTabs=true, false default: false

true
: Preserve all tabs in the input.

false
: Convert any tabs in the input to spaces.

```
defaultOptions.preserveTabs = false
```

smartEllipses=true, false default: false

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

false
: Preserve all ellipses in the input.

```
defaultOptions.smartEllipses = false
```

startNumber=true, false default: true

true
: Make the number in the first item in ordered lists significant. The item numbers will be passed to the \markdownRendererOlItemWithNumber TEX macro.

false
: Ignore the number in the items of ordered lists. Each item will only produce a \markdownRendererOlItem TEX macro.

```
defaultOptions.startNumber = true
```

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

`true`	Lists whose bullets do not consist of multiple paragraphs will be detected and passed to the `\markdownRendererOlBeginTight`, `\markdownRendererOlEndTight`, `\markdownRendererUlBeginTight`, `\markdownRendererUlEndTight`, `\markdownRendererDlBeginTight`, and `\markdownRendererDlEndTight` macros.
`false`	Lists whose bullets do not consist of multiple paragraphs will be treated the same way as lists that do.

```
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.

```
defaultOptions.underscores = true
```

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 2.1.2) used during the conversion from markdown to plain TEX, and for changing the way markdown the tokens are rendered.

```
\def\markdownLastModified{2017/04/10}%
\def\markdownVersion{2.5.0}%
```

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, and \markdownInput macros.

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

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

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

There are several limitations to the macros you need to be aware of. The first limitation concerns the \markdownEnd macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string otherwise. 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 (see [3, p. 46]). As a corrolary, the \markdownBegin macro also ignores them.

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

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

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

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

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

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

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

This macro is not subject to the 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
```

2.2.2 Options

The plain TeX options are represented by TeX macros. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.2), while some of them are specific to the plain TeX interface.

2.2.2.1 File and directory names

The \markdownOptionHelperScriptFileName macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain TeX in TeX engines without the \directlua primitive. It defaults to \jobname.markdown.lua, where \jobname is the base name of the document being typeset.

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

```
\def\markdownOptionHelperScriptFileName{\jobname.markdown.lua}%
```

The \markdownOptionInputTempFileName macro sets the filename of the temporary input file that is created during the conversion from markdown to plain TeX in TeX engines without the \directlua primitive. It defaults to \jobname.markdown.out. The same limitations as in the case of the \markdownOptionHelperScriptFileName macro apply here.

```
\def\markdownOptionInputTempFileName{\jobname.markdown.in}%
```

The \markdownOptionOutputTempFileName macro sets the filename of the temporary output file that is created during the conversion from markdown to plain TeX in TeX engines without the \directlua primitive. It defaults to \jobname.markdown.out. The same limitations apply here as in the case of the \markdownOptionHelperScriptFileName macro.

```
\def\markdownOptionOutputTempFileName{\jobname.markdown.out}%
```

The \markdownOptionCacheDir macro corresponds to the Lua interface cacheDir option that sets the name of the directory that will contain the produced cache files. The option defaults to _markdown_\jobname, which is a similar naming scheme to the one used by the minted LaTeX package. The same limitations apply here as in the case of the \markdownOptionHelperScriptFileName macro.

```
\def\markdownOptionCacheDir{./_markdown_\jobname}%
```

2.2.2.2 Lua Interface Options

The following macros map directly to the options recognized by the Lua interface (see Section 2.1.2) and are not processed by the plain TeX implementation, only passed along to Lua. They are undefined, which makes them fall back to the default values provided by the Lua interface.

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 \markdownOptionHelperScriptFileName macro.

```
\let\markdownOptionBlankBeforeBlockquote\undefined
\let\markdownOptionBlankBeforeCodeFence\undefined
\let\markdownOptionBlankBeforeHeading\undefined
\let\markdownOptionBreakableBlockquotes\undefined
\let\markdownOptionCitations\undefined
\let\markdownOptionCitationNbsps\undefined
\let\markdownOptionContentBlocks\undefined
\let\markdownOptionContentBlocksLanguageMap\undefined
\let\markdownOptionDefinitionLists\undefined
\let\markdownOptionFootnotes\undefined
\let\markdownOptionFencedCode\undefined
\let\markdownOptionHashEnumerators\undefined
\let\markdownOptionHtml\undefined
\let\markdownOptionHybrid\undefined
\let\markdownOptionInlineFootnotes\undefined
\let\markdownOptionPreserveTabs\undefined
\let\markdownOptionSmartEllipses\undefined
\let\markdownOptionStartNumber\undefined
\let\markdownOptionTightLists\undefined
```

2.2.3 Token Renderers

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

2.2.3.1 Interblock Separator Renderer

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

```
\def\markdownRendererInterblockSeparator{%
  \markdownRendererInterblockSeparatorPrototype}%
```

2.2.3.2 Line Break Renderer

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

```
\def\markdownRendererLineBreak{%
  \markdownRendererLineBreakPrototype}%
```

2.2.3.3 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 true. The macro receives no arguments.

```
\def\markdownRendererEllipsis{%
  \markdownRendererEllipsisPrototype}%
```

2.2.3.4 Non-breaking Space Renderer

The \markdownRendererNbsp macro represents a non-breaking space.

```
\def\markdownRendererNbsp{%
  \markdownRendererNbspPrototype}%
```

2.2.3.5 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}%
\def\markdownRendererRightBrace{%
  \markdownRendererRightBracePrototype}%
\def\markdownRendererDollarSign{%
  \markdownRendererDollarSignPrototype}%
\def\markdownRendererPercentSign{%
  \markdownRendererPercentSignPrototype}%
\def\markdownRendererAmpersand{%
  \markdownRendererAmpersandPrototype}%
\def\markdownRendererUnderscore{%
  \markdownRendererUnderscorePrototype}%
\def\markdownRendererHash{%
  \markdownRendererHashPrototype}%
\def\markdownRendererCircumflex{%
  \markdownRendererCircumflexPrototype}%
\def\markdownRendererBackslash{%
  \markdownRendererBackslashPrototype}%
```

```
\def\markdownRendererTilde{%
  \markdownRendererTildePrototype}%
\def\markdownRendererPipe{%
  \markdownRendererPipePrototype}%
```

2.2.3.6 Code Span Renderer

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

```
\def\markdownRendererCodeSpan{%
  \markdownRendererCodeSpanPrototype}%
```

2.2.3.7 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}%
```

2.2.3.8 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}%
```

2.2.3.9 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}%
```

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

```
\def\markdownRendererContentBlockOnlineImage{%
  \markdownRendererContentBlockOnlineImagePrototype}%
```

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[3] contains a

[3] Local files take precedence. Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

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 four 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 [2] is a good starting point.

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

2.2.3.10 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}%
```

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 `false`. The macro receives no arguments.

```
\def\markdownRendererUlBeginTight{%
  \markdownRendererUlBeginTightPrototype}%
```

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

```
\def\markdownRendererUlItem{%
  \markdownRendererUlItemPrototype}%
```

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

```
\def\markdownRendererUlItemEnd{%
  \markdownRendererUlItemEndPrototype}%
```

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}%
```

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 `false`. The macro receives no arguments.

```
\def\markdownRendererUlEndTight{%
  \markdownRendererUlEndTightPrototype}%
```

2.2.3.11 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). The macro receives no arguments.

```
\def\markdownRendererOlBegin{%
  \markdownRendererOlBeginPrototype}%
```

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 false. The macro receives no arguments.

```
\def\markdownRendererOlBeginTight{%
  \markdownRendererOlBeginTightPrototype}%
```

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

```
\def\markdownRendererOlItem{%
  \markdownRendererOlItemPrototype}%
```

The \markdownRendererOlItemEnd macro represents the end of an item in an ordered list. The macro receives no arguments.

```
\def\markdownRendererOlItemEnd{%
  \markdownRendererOlItemEndPrototype}%
```

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

```
\def\markdownRendererOlItemWithNumber{%
  \markdownRendererOlItemWithNumberPrototype}%
```

The \markdownRendererOlEnd macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
\def\markdownRendererOlEnd{%
  \markdownRendererOlEndPrototype}%
```

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 false. The macro receives no arguments.

```
\def\markdownRendererOlEndTight{%
  \markdownRendererOlEndTightPrototype}%
```

2.2.3.12 Definition List Renderers

The following macros are only produces, when the `definitionLists` option is `true`.

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}%
```

The `\markdownRendererDlBeginTight` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```
\def\markdownRendererDlBeginTight{%
  \markdownRendererDlBeginTightPrototype}%
```

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}%
```

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

```
\def\markdownRendererDlItemEnd{%
  \markdownRendererDlItemEndPrototype}%
```

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}%
```

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}%
```

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}%
```

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 `false`. The macro receives no arguments.

```
\def\markdownRendererDlEndTight{%
  \markdownRendererDlEndTightPrototype}%
```

2.2.3.13 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}%
```

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}%
```

2.2.3.14 Block Quote Renderers

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

```
\def\markdownRendererBlockQuoteBegin{%
  \markdownRendererBlockQuoteBeginPrototype}%
```

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

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

2.2.3.15 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}%
```

The \markdownRendererInputFencedCode macro represents a fenced code block. This macro will only be produced, when the fencedCode option is true. The macro receives two arguments that correspond to the filename of a file contaning the code block contents and to the code fence infostring.

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

2.2.3.16 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}%
```

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

```
\def\markdownRendererHeadingTwo{%
  \markdownRendererHeadingTwoPrototype}%
```

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

```
\def\markdownRendererHeadingThree{%
  \markdownRendererHeadingThreePrototype}%
```

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

```
\def\markdownRendererHeadingFour{%
  \markdownRendererHeadingFourPrototype}%
```

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

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

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

```
\def\markdownRendererHeadingSix{%
  \markdownRendererHeadingSixPrototype}%
```

2.2.3.17 Horizontal Rule Renderer The \markdownRendererHorizontalRule macro represents a horizontal rule. The macro receives no arguments.

```
\def\markdownRendererHorizontalRule{%
  \markdownRendererHorizontalRulePrototype}%
```

2.2.3.18 Footnote Renderer The \markdownRendererFootnote macro represents a footnote. This macro will only be produced, when the footnotes option is true. The macro receives a single argument that corresponds to the footnote text.

```
\def\markdownRendererFootnote{%
  \markdownRendererFootnotePrototype}%
```

2.2.3.19 Parenthesized Citations Renderer The \markdownRendererCite macro represents a string of one or more parenthetical citations. This macro will only be produced, when the citations option is true. 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}%
```

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

```
\def\markdownRendererTextCite{%
  \markdownRendererTextCitePrototype}%
```

2.2.4 Token Renderer Prototypes

The following TeX macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the LaTeX and ConTeXt implementations (see sections 3.3 and 3.4).

```
\def\markdownRendererInterblockSeparatorPrototype{}%
\def\markdownRendererLineBreakPrototype{}%
\def\markdownRendererEllipsisPrototype{}%
\def\markdownRendererNbspPrototype{}%
\def\markdownRendererLeftBracePrototype{}%
\def\markdownRendererRightBracePrototype{}%
\def\markdownRendererDollarSignPrototype{}%
\def\markdownRendererPercentSignPrototype{}%
\def\markdownRendererAmpersandPrototype{}%
\def\markdownRendererUnderscorePrototype{}%
\def\markdownRendererHashPrototype{}%
\def\markdownRendererCircumflexPrototype{}%
\def\markdownRendererBackslashPrototype{}%
\def\markdownRendererTildePrototype{}%
\def\markdownRendererPipePrototype{}%
\def\markdownRendererCodeSpanPrototype#1{}%
\def\markdownRendererLinkPrototype#1#2#3#4{}%
\def\markdownRendererImagePrototype#1#2#3#4{}%
\def\markdownRendererContentBlockPrototype#1#2#3#4{}%
\def\markdownRendererContentBlockOnlineImagePrototype#1#2#3#4{}%
\def\markdownRendererContentBlockCodePrototype#1#2#3#4{}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
```

```
\def\markdownRendererDlBeginPrototype{}%
\def\markdownRendererDlBeginTightPrototype{}%
\def\markdownRendererDlItemPrototype#1{}%
\def\markdownRendererDlItemEndPrototype{}%
\def\markdownRendererDlDefinitionBeginPrototype{}%
\def\markdownRendererDlDefinitionEndPrototype{}%
\def\markdownRendererDlEndPrototype{}%
\def\markdownRendererDlEndTightPrototype{}%
\def\markdownRendererEmphasisPrototype#1{}%
\def\markdownRendererStrongEmphasisPrototype#1{}%
\def\markdownRendererBlockQuoteBeginPrototype{}%
\def\markdownRendererBlockQuoteEndPrototype{}%
\def\markdownRendererInputVerbatimPrototype#1{}%
\def\markdownRendererInputFencedCodePrototype#1#2{}%
\def\markdownRendererHeadingOnePrototype#1{}%
\def\markdownRendererHeadingTwoPrototype#1{}%
\def\markdownRendererHeadingThreePrototype#1{}%
\def\markdownRendererHeadingFourPrototype#1{}%
\def\markdownRendererHeadingFivePrototype#1{}%
\def\markdownRendererHeadingSixPrototype#1{}%
\def\markdownRendererHorizontalRulePrototype{}%
\def\markdownRendererFootnotePrototype#1{}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
```

2.2.5 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros provide access to logging to the rest of the macros. Their first argument specifies the text of the info, warning, or error message.

```
\def\markdownInfo#1{}%
\def\markdownWarning#1{}%
```

The `\markdownError` macro receives a second argument that provides a help text suggesting a remedy to the error.

```
\def\markdownError#1#2{}%
```

You may redefine these macros to redirect and process the info, warning, and error messages.

2.2.6 Miscellanea

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

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

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain TEX special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

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

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

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

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

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

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

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

```
\ifx\markdownMode\undefined
  \ifx\directlua\undefined
    \def\markdownMode{0}%
  \else
    \def\markdownMode{2}%
  \fi
\fi
```

The following macros are no longer a part of the plain TeX interface and are only defined for backwards compatibility:

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

2.3 LaTeX Interface

The LaTeX interface provides LaTeX environments for the typesetting of markdown input from within LaTeX, facilities for setting Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain TeX, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain TeX interface (see Section 2.2).

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

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

where ⟨*options*⟩ are the LaTeX interface options (see Section 2.3.2). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.2) and `markdownRendererPrototypes` (see Section 2.3.2.3) keys. This limitation is due to the way LaTeX 2$_\varepsilon$ parses package options.

2.3.1 Typesetting Markdown

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

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

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

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

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

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

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

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

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

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

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

2.3.2 Options

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

The LaTeX options map directly to the options recognized by the plain TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain TeX interface (see Sections 2.2.3 and 2.2.4).

The LaTeX options may be specified when loading the LaTeX package (see Section 2.3), when using the `markdown*` LaTeX environment, or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument.

```
\newcommand\markdownSetup[1]{%
  \setkeys{markdownOptions}{#1}}%
```

2.3.2.1 Plain TeX Interface Options

The following options map directly to the option macros exposed by the plain TeX interface (see Section 2.2.2).

```
\RequirePackage{keyval}
\define@key{markdownOptions}{helperScriptFileName}{%
  \def\markdownOptionHelperScriptFileName{#1}}%
\define@key{markdownOptions}{inputTempFileName}{%
  \def\markdownOptionInputTempFileName{#1}}%
\define@key{markdownOptions}{outputTempFileName}{%
  \def\markdownOptionOutputTempFileName{#1}}%
\define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
  \def\markdownOptionBlankBeforeBlockquote{#1}}%
\define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
  \def\markdownOptionBlankBeforeCodeFence{#1}}%
\define@key{markdownOptions}{blankBeforeHeading}[true]{%
  \def\markdownOptionBlankBeforeHeading{#1}}%
\define@key{markdownOptions}{breakableBlockquotes}[true]{%
  \def\markdownOptionBreakableBlockquotes{#1}}%
\define@key{markdownOptions}{cacheDir}{%
  \def\markdownOptionCacheDir{#1}}%
\define@key{markdownOptions}{citations}[true]{%
  \def\markdownOptionCitations{#1}}%
\define@key{markdownOptions}{citationNbsps}[true]{%
  \def\markdownOptionCitationNbsps{#1}}%
\define@key{markdownOptions}{contentBlocks}[true]{%
  \def\markdownOptionContentBlocks{#1}}%
\define@key{markdownOptions}{codeSpans}[true]{%
  \def\markdownOptionCodeSpans{#1}}%
\define@key{markdownOptions}{contentBlocksLanguageMap}{%
  \def\markdownOptionContentBlocksLanguageMap{#1}}%
\define@key{markdownOptions}{definitionLists}[true]{%
  \def\markdownOptionDefinitionLists{#1}}%
\define@key{markdownOptions}{footnotes}[true]{%
  \def\markdownOptionFootnotes{#1}}%
\define@key{markdownOptions}{fencedCode}[true]{%
  \def\markdownOptionFencedCode{#1}}%
\define@key{markdownOptions}{hashEnumerators}[true]{%
  \def\markdownOptionHashEnumerators{#1}}%
\define@key{markdownOptions}{html}[true]{%
  \def\markdownOptionHtml{#1}}%
\define@key{markdownOptions}{hybrid}[true]{%
  \def\markdownOptionHybrid{#1}}%
\define@key{markdownOptions}{inlineFootnotes}[true]{%
  \def\markdownOptionInlineFootnotes{#1}}%
\define@key{markdownOptions}{preserveTabs}[true]{%
  \def\markdownOptionPreserveTabs{#1}}%
\define@key{markdownOptions}{smartEllipses}[true]{%
  \def\markdownOptionSmartEllipses{#1}}%
```

```
\define@key{markdownOptions}{startNumber}[true]{%
  \def\markdownOptionStartNumber{#1}}%
\define@key{markdownOptions}{tightLists}[true]{%
  \def\markdownOptionTightLists{#1}}%
\define@key{markdownOptions}{underscores}[true]{%
  \def\markdownOptionUnderscores{#1}}%
```

The following example LaTeX code showcases a possible configuration of plain TeX interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

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

2.3.2.2 Plain TeX Markdown Token Renderers

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

```
\define@key{markdownRenderers}{interblockSeparator}{%
  \renewcommand\markdownRendererInterblockSeparator{#1}}%
\define@key{markdownRenderers}{lineBreak}{%
  \renewcommand\markdownRendererLineBreak{#1}}%
\define@key{markdownRenderers}{ellipsis}{%
  \renewcommand\markdownRendererEllipsis{#1}}%
\define@key{markdownRenderers}{nbsp}{%
  \renewcommand\markdownRendererNbsp{#1}}%
\define@key{markdownRenderers}{leftBrace}{%
  \renewcommand\markdownRendererLeftBrace{#1}}%
\define@key{markdownRenderers}{rightBrace}{%
  \renewcommand\markdownRendererRightBrace{#1}}%
\define@key{markdownRenderers}{dollarSign}{%
  \renewcommand\markdownRendererDollarSign{#1}}%
\define@key{markdownRenderers}{percentSign}{%
  \renewcommand\markdownRendererPercentSign{#1}}%
\define@key{markdownRenderers}{ampersand}{%
  \renewcommand\markdownRendererAmpersand{#1}}%
\define@key{markdownRenderers}{underscore}{%
  \renewcommand\markdownRendererUnderscore{#1}}%
\define@key{markdownRenderers}{hash}{%
  \renewcommand\markdownRendererHash{#1}}%
\define@key{markdownRenderers}{circumflex}{%
  \renewcommand\markdownRendererCircumflex{#1}}%
```

```
\define@key{markdownRenderers}{backslash}{%
  \renewcommand\markdownRendererBackslash{#1}}%
\define@key{markdownRenderers}{tilde}{%
  \renewcommand\markdownRendererTilde{#1}}%
\define@key{markdownRenderers}{pipe}{%
  \renewcommand\markdownRendererPipe{#1}}%
\define@key{markdownRenderers}{codeSpan}{%
  \renewcommand\markdownRendererCodeSpan[1]{#1}}%
\define@key{markdownRenderers}{link}{%
  \renewcommand\markdownRendererLink[4]{#1}}%
\define@key{markdownRenderers}{contentBlock}{%
  \renewcommand\markdownRendererContentBlock[4]{#1}}%
\define@key{markdownRenderers}{contentBlockOnlineImage}{%
  \renewcommand\markdownRendererContentBlockOnlineImage[4]{#1}}%
\define@key{markdownRenderers}{contentBlockCode}{%
  \renewcommand\markdownRendererContentBlockCode[5]{#1}}%
\define@key{markdownRenderers}{image}{%
  \renewcommand\markdownRendererImage[4]{#1}}%
\define@key{markdownRenderers}{ulBegin}{%
  \renewcommand\markdownRendererUlBegin{#1}}%
\define@key{markdownRenderers}{ulBeginTight}{%
  \renewcommand\markdownRendererUlBeginTight{#1}}%
\define@key{markdownRenderers}{ulItem}{%
  \renewcommand\markdownRendererUlItem{#1}}%
\define@key{markdownRenderers}{ulItemEnd}{%
  \renewcommand\markdownRendererUlItemEnd{#1}}%
\define@key{markdownRenderers}{ulEnd}{%
  \renewcommand\markdownRendererUlEnd{#1}}%
\define@key{markdownRenderers}{ulEndTight}{%
  \renewcommand\markdownRendererUlEndTight{#1}}%
\define@key{markdownRenderers}{olBegin}{%
  \renewcommand\markdownRendererOlBegin{#1}}%
\define@key{markdownRenderers}{olBeginTight}{%
  \renewcommand\markdownRendererOlBeginTight{#1}}%
\define@key{markdownRenderers}{olItem}{%
  \renewcommand\markdownRendererOlItem{#1}}%
\define@key{markdownRenderers}{olItemWithNumber}{%
  \renewcommand\markdownRendererOlItemWithNumber[1]{#1}}%
\define@key{markdownRenderers}{olItemEnd}{%
  \renewcommand\markdownRendererOlItemEnd{#1}}%
\define@key{markdownRenderers}{olEnd}{%
  \renewcommand\markdownRendererOlEnd{#1}}%
\define@key{markdownRenderers}{olEndTight}{%
  \renewcommand\markdownRendererOlEndTight{#1}}%
\define@key{markdownRenderers}{dlBegin}{%
  \renewcommand\markdownRendererDlBegin{#1}}%
\define@key{markdownRenderers}{dlBeginTight}{%
```

```
  \renewcommand\markdownRendererDlBeginTight{#1}}%
\define@key{markdownRenderers}{dlItem}{%
  \renewcommand\markdownRendererDlItem[1]{#1}}%
\define@key{markdownRenderers}{dlItemEnd}{%
  \renewcommand\markdownRendererDlItemEnd{#1}}%
\define@key{markdownRenderers}{dlDefinitionBegin}{%
  \renewcommand\markdownRendererDlDefinitionBegin{#1}}%
\define@key{markdownRenderers}{dlDefinitionEnd}{%
  \renewcommand\markdownRendererDlDefinitionEnd{#1}}%
\define@key{markdownRenderers}{dlEnd}{%
  \renewcommand\markdownRendererDlEnd{#1}}%
\define@key{markdownRenderers}{dlEndTight}{%
  \renewcommand\markdownRendererDlEndTight{#1}}%
\define@key{markdownRenderers}{emphasis}{%
  \renewcommand\markdownRendererEmphasis[1]{#1}}%
\define@key{markdownRenderers}{strongEmphasis}{%
  \renewcommand\markdownRendererStrongEmphasis[1]{#1}}%
\define@key{markdownRenderers}{blockQuoteBegin}{%
  \renewcommand\markdownRendererBlockQuoteBegin{#1}}%
\define@key{markdownRenderers}{blockQuoteEnd}{%
  \renewcommand\markdownRendererBlockQuoteEnd{#1}}%
\define@key{markdownRenderers}{inputVerbatim}{%
  \renewcommand\markdownRendererInputVerbatim[1]{#1}}%
\define@key{markdownRenderers}{inputFencedCode}{%
  \renewcommand\markdownRendererInputFencedCode[2]{#1}}%
\define@key{markdownRenderers}{headingOne}{%
  \renewcommand\markdownRendererHeadingOne[1]{#1}}%
\define@key{markdownRenderers}{headingTwo}{%
  \renewcommand\markdownRendererHeadingTwo[1]{#1}}%
\define@key{markdownRenderers}{headingThree}{%
  \renewcommand\markdownRendererHeadingThree[1]{#1}}%
\define@key{markdownRenderers}{headingFour}{%
  \renewcommand\markdownRendererHeadingFour[1]{#1}}%
\define@key{markdownRenderers}{headingFive}{%
  \renewcommand\markdownRendererHeadingFive[1]{#1}}%
\define@key{markdownRenderers}{headingSix}{%
  \renewcommand\markdownRendererHeadingSix[1]{#1}}%
\define@key{markdownRenderers}{horizontalRule}{%
  \renewcommand\markdownRendererHorizontalRule{#1}}%
\define@key{markdownRenderers}{footnote}{%
  \renewcommand\markdownRendererFootnote[1]{#1}}%
\define@key{markdownRenderers}{cite}{%
  \renewcommand\markdownRendererCite[1]{#1}}%
\define@key{markdownRenderers}{textCite}{%
  \renewcommand\markdownRendererTextCite[1]{#1}}%
```

The following example LaTeX code showcases a possible configuration of the

\markdownRendererLink and \markdownRendererEmphasis markdown token renderers.

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

2.3.2.3 Plain TeX Markdown Token Renderer Prototypes

The LaTeX interface recognizes an option with the rendererPrototypes key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain TeX interface (see Section 2.2.4).

```
\define@key{markdownRendererPrototypes}{interblockSeparator}{%
  \renewcommand\markdownRendererInterblockSeparatorPrototype{#1}}%
\define@key{markdownRendererPrototypes}{lineBreak}{%
  \renewcommand\markdownRendererLineBreakPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ellipsis}{%
  \renewcommand\markdownRendererEllipsisPrototype{#1}}%
\define@key{markdownRendererPrototypes}{nbsp}{%
  \renewcommand\markdownRendererNbspPrototype{#1}}%
\define@key{markdownRendererPrototypes}{leftBrace}{%
  \renewcommand\markdownRendererLeftBracePrototype{#1}}%
\define@key{markdownRendererPrototypes}{rightBrace}{%
  \renewcommand\markdownRendererRightBracePrototype{#1}}%
\define@key{markdownRendererPrototypes}{dollarSign}{%
  \renewcommand\markdownRendererDollarSignPrototype{#1}}%
\define@key{markdownRendererPrototypes}{percentSign}{%
  \renewcommand\markdownRendererPercentSignPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ampersand}{%
  \renewcommand\markdownRendererAmpersandPrototype{#1}}%
\define@key{markdownRendererPrototypes}{underscore}{%
  \renewcommand\markdownRendererUnderscorePrototype{#1}}%
\define@key{markdownRendererPrototypes}{hash}{%
  \renewcommand\markdownRendererHashPrototype{#1}}%
\define@key{markdownRendererPrototypes}{circumflex}{%
  \renewcommand\markdownRendererCircumflexPrototype{#1}}%
\define@key{markdownRendererPrototypes}{backslash}{%
  \renewcommand\markdownRendererBackslashPrototype{#1}}%
\define@key{markdownRendererPrototypes}{tilde}{%
  \renewcommand\markdownRendererTildePrototype{#1}}%
\define@key{markdownRendererPrototypes}{pipe}{%
  \renewcommand\markdownRendererPipePrototype{#1}}%
\define@key{markdownRendererPrototypes}{codeSpan}{%
```

```
  \renewcommand\markdownRendererCodeSpanPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{link}{%
  \renewcommand\markdownRendererLinkPrototype[4]{#1}}%
\define@key{markdownRendererPrototypes}{contentBlock}{%
  \renewcommand\markdownRendererContentBlockPrototype[4]{#1}}%
\define@key{markdownRendererPrototypes}{contentBlockOnlineImage}{%
  \renewcommand\markdownRendererContentBlockOnlineImagePrototype[4]{#1}}%
\define@key{markdownRendererPrototypes}{contentBlockCode}{%
  \renewcommand\markdownRendererContentBlockCodePrototype[5]{#1}}%
\define@key{markdownRendererPrototypes}{image}{%
  \renewcommand\markdownRendererImagePrototype[4]{#1}}%
\define@key{markdownRendererPrototypes}{ulBegin}{%
  \renewcommand\markdownRendererUlBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulBeginTight}{%
  \renewcommand\markdownRendererUlBeginTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulItem}{%
  \renewcommand\markdownRendererUlItemPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulItemEnd}{%
  \renewcommand\markdownRendererUlItemEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulEnd}{%
  \renewcommand\markdownRendererUlEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulEndTight}{%
  \renewcommand\markdownRendererUlEndTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olBegin}{%
  \renewcommand\markdownRendererOlBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olBeginTight}{%
  \renewcommand\markdownRendererOlBeginTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olItem}{%
  \renewcommand\markdownRendererOlItemPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olItemWithNumber}{%
  \renewcommand\markdownRendererOlItemWithNumberPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{olItemEnd}{%
  \renewcommand\markdownRendererOlItemEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olEnd}{%
  \renewcommand\markdownRendererOlEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olEndTight}{%
  \renewcommand\markdownRendererOlEndTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlBegin}{%
  \renewcommand\markdownRendererDlBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlBeginTight}{%
  \renewcommand\markdownRendererDlBeginTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlItem}{%
  \renewcommand\markdownRendererDlItemPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{dlItemEnd}{%
  \renewcommand\markdownRendererDlItemEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlDefinitionBegin}{%
  \renewcommand\markdownRendererDlDefinitionBeginPrototype{#1}}%
```

```
\define@key{markdownRendererPrototypes}{dlDefinitionEnd}{%
  \renewcommand\markdownRendererDlDefinitionEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlEnd}{%
  \renewcommand\markdownRendererDlEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlEndTight}{%
  \renewcommand\markdownRendererDlEndTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{emphasis}{%
  \renewcommand\markdownRendererEmphasisPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{strongEmphasis}{%
  \renewcommand\markdownRendererStrongEmphasisPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{blockQuoteBegin}{%
  \renewcommand\markdownRendererBlockQuoteBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{blockQuoteEnd}{%
  \renewcommand\markdownRendererBlockQuoteEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{inputVerbatim}{%
  \renewcommand\markdownRendererInputVerbatimPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{inputFencedCode}{%
  \renewcommand\markdownRendererInputFencedCodePrototype[2]{#1}}%
\define@key{markdownRendererPrototypes}{headingOne}{%
  \renewcommand\markdownRendererHeadingOnePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingTwo}{%
  \renewcommand\markdownRendererHeadingTwoPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingThree}{%
  \renewcommand\markdownRendererHeadingThreePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingFour}{%
  \renewcommand\markdownRendererHeadingFourPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingFive}{%
  \renewcommand\markdownRendererHeadingFivePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingSix}{%
  \renewcommand\markdownRendererHeadingSixPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{horizontalRule}{%
  \renewcommand\markdownRendererHorizontalRulePrototype{#1}}%
\define@key{markdownRendererPrototypes}{footnote}{%
  \renewcommand\markdownRendererFootnotePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{cite}{%
  \renewcommand\markdownRendererCitePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{textCite}{%
  \renewcommand\markdownRendererTextCitePrototype[1]{#1}}%
```

The following example LaTeX code showcases a possible configuration of the \markdownRendererImagePrototype and \markdownRendererCodeSpanPrototype markdown token renderer prototypes.

```
\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}},    % Render inline code via `\texttt`.
  }
}
```

2.4 ConTEXt Interface

The ConTEXt interface provides a start-stop macro pair for the typesetting of markdown input from within ConTEXt. The rest of the interface is inherited from the plain TEX interface (see Section 2.2).

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

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.

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

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

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
```

3 Technical Documentation

This part of the manual 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.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects that provide the conversion from markdown to plain TeX.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain TeX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

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

3.1.1 Utility Functions

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

```
local util = {}
```

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

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

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

```
function util.cache(dir, string, salt, transform, suffix)
  local digest = md5.sumhexa(string .. (salt or ""))
  local name = util.pathname(dir, digest .. suffix)
  local file = io.open(name, "r")
  if file == nil then -- If no cache entry exists, then create a new one.
    local file = assert(io.open(name, "w"))
    local result = string
    if transform ~= nil then
      result = transform(result)
    end
    assert(file:write(result))
    assert(file:close())
  end
  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.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 [4, 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 $1 \leq$ `i` $\leq$ `#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 $1 \leq$ `i` $\leq$ `#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

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

capture that replaces any occurance of the string `k` with the string `v` for each $(\texttt{k}, \texttt{v}) \in$ `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 = {
  ["quot"] = 0x0022,
  ["amp"] = 0x0026,
  ["apos"] = 0x0027,
  ["lt"] = 0x003C,
  ["gt"] = 0x003E,
  ["nbsp"] = 160,
  ["iexcl"] = 0x00A1,
  ["cent"] = 0x00A2,
  ["pound"] = 0x00A3,
  ["curren"] = 0x00A4,
  ["yen"] = 0x00A5,
  ["brvbar"] = 0x00A6,
  ["sect"] = 0x00A7,
  ["uml"] = 0x00A8,
  ["copy"] = 0x00A9,
  ["ordf"] = 0x00AA,
  ["laquo"] = 0x00AB,
  ["not"] = 0x00AC,
  ["shy"] = 173,
  ["reg"] = 0x00AE,
  ["macr"] = 0x00AF,
  ["deg"] = 0x00B0,
  ["plusmn"] = 0x00B1,
  ["sup2"] = 0x00B2,
  ["sup3"] = 0x00B3,
  ["acute"] = 0x00B4,
  ["micro"] = 0x00B5,
  ["para"] = 0x00B6,
  ["middot"] = 0x00B7,
  ["cedil"] = 0x00B8,
  ["sup1"] = 0x00B9,
  ["ordm"] = 0x00BA,
  ["raquo"] = 0x00BB,
  ["frac14"] = 0x00BC,
  ["frac12"] = 0x00BD,
  ["frac34"] = 0x00BE,
  ["iquest"] = 0x00BF,
  ["Agrave"] = 0x00C0,
  ["Aacute"] = 0x00C1,
  ["Acirc"] = 0x00C2,
  ["Atilde"] = 0x00C3,
  ["Auml"] = 0x00C4,
  ["Aring"] = 0x00C5,
  ["AElig"] = 0x00C6,
  ["Ccedil"] = 0x00C7,
  ["Egrave"] = 0x00C8,
  ["Eacute"] = 0x00C9,
  ["Ecirc"] = 0x00CA,
  ["Euml"] = 0x00CB,
  ["Igrave"] = 0x00CC,
  ["Iacute"] = 0x00CD,
  ["Icirc"] = 0x00CE,
  ["Iuml"] = 0x00CF,
  ["ETH"] = 0x00D0,
  ["Ntilde"] = 0x00D1,
  ["Ograve"] = 0x00D2,
  ["Oacute"] = 0x00D3,
  ["Ocirc"] = 0x00D4,
  ["Otilde"] = 0x00D5,
  ["Ouml"] = 0x00D6,
  ["times"] = 0x00D7,
  ["Oslash"] = 0x00D8,
  ["Ugrave"] = 0x00D9,
  ["Uacute"] = 0x00DA,
  ["Ucirc"] = 0x00DB,
  ["Uuml"] = 0x00DC,
  ["Yacute"] = 0x00DD,
  ["THORN"] = 0x00DE,
  ["szlig"] = 0x00DF,
  ["agrave"] = 0x00E0,
  ["aacute"] = 0x00E1,
  ["acirc"] = 0x00E2,
  ["atilde"] = 0x00E3,
  ["auml"] = 0x00E4,
  ["aring"] = 0x00E5,
  ["aelig"] = 0x00E6,
  ["ccedil"] = 0x00E7,
  ["egrave"] = 0x00E8,
  ["eacute"] = 0x00E9,
  ["ecirc"] = 0x00EA,
  ["euml"] = 0x00EB,
  ["igrave"] = 0x00EC,
  ["iacute"] = 0x00ED,
  ["icirc"] = 0x00EE,
  ["iuml"] = 0x00EF,
  ["eth"] = 0x00F0,
  ["ntilde"] = 0x00F1,
  ["ograve"] = 0x00F2,
  ["oacute"] = 0x00F3,
  ["ocirc"] = 0x00F4,
  ["otilde"] = 0x00F5,
  ["ouml"] = 0x00F6,
  ["divide"] = 0x00F7,
  ["oslash"] = 0x00F8,
  ["ugrave"] = 0x00F9,
  ["uacute"] = 0x00FA,
  ["ucirc"] = 0x00FB,
  ["uuml"] = 0x00FC,
  ["yacute"] = 0x00FD,
  ["thorn"] = 0x00FE,
  ["yuml"] = 0x00FF,
  ["OElig"] = 0x0152,
  ["oelig"] = 0x0153,
  ["Scaron"] = 0x0160,
  ["scaron"] = 0x0161,
  ["Yuml"] = 0x0178,
  ["fnof"] = 0x0192,
  ["circ"] = 0x02C6,
  ["tilde"] = 0x02DC,
  ["Alpha"] = 0x0391,
  ["Beta"] = 0x0392,
  ["Gamma"] = 0x0393,
  ["Delta"] = 0x0394,
  ["Epsilon"] = 0x0395,
  ["Zeta"] = 0x0396,
  ["Eta"] = 0x0397,
  ["Theta"] = 0x0398,
  ["Iota"] = 0x0399,
  ["Kappa"] = 0x039A,
  ["Lambda"] = 0x039B,
  ["Mu"] = 0x039C,
  ["Nu"] = 0x039D,
  ["Xi"] = 0x039E,
  ["Omicron"] = 0x039F,
  ["Pi"] = 0x03A0,
  ["Rho"] = 0x03A1,
  ["Sigma"] = 0x03A3,
  ["Tau"] = 0x03A4,
  ["Upsilon"] = 0x03A5,
  ["Phi"] = 0x03A6,
  ["Chi"] = 0x03A7,
  ["Psi"] = 0x03A8,
  ["Omega"] = 0x03A9,
  ["alpha"] = 0x03B1,
  ["beta"] = 0x03B2,
  ["gamma"] = 0x03B3,
  ["delta"] = 0x03B4,
  ["epsilon"] = 0x03B5,
  ["zeta"] = 0x03B6,
  ["eta"] = 0x03B7,
  ["theta"] = 0x03B8,
  ["iota"] = 0x03B9,
  ["kappa"] = 0x03BA,
  ["lambda"] = 0x03BB,
  ["mu"] = 0x03BC,
  ["nu"] = 0x03BD,
  ["xi"] = 0x03BE,
  ["omicron"] = 0x03BF,
  ["pi"] = 0x03C0,
  ["rho"] = 0x03C1,
  ["sigmaf"] = 0x03C2,
  ["sigma"] = 0x03C3,
  ["tau"] = 0x03C4,
  ["upsilon"] = 0x03C5,
  ["phi"] = 0x03C6,
  ["chi"] = 0x03C7,
  ["psi"] = 0x03C8,
  ["omega"] = 0x03C9,
  ["thetasym"] = 0x03D1,
  ["upsih"] = 0x03D2,
  ["piv"] = 0x03D6,
  ["ensp"] = 0x2002,
  ["emsp"] = 0x2003,
  ["thinsp"] = 0x2009,
  ["ndash"] = 0x2013,
  ["mdash"] = 0x2014,
  ["lsquo"] = 0x2018,
  ["rsquo"] = 0x2019,
  ["sbquo"] = 0x201A,
  ["ldquo"] = 0x201C,
  ["rdquo"] = 0x201D,
  ["bdquo"] = 0x201E,
  ["dagger"] = 0x2020,
  ["Dagger"] = 0x2021,
  ["bull"] = 0x2022,
  ["hellip"] = 0x2026,
  ["permil"] = 0x2030,
  ["prime"] = 0x2032,
  ["Prime"] = 0x2033,
  ["lsaquo"] = 0x2039,
  ["rsaquo"] = 0x203A,
  ["oline"] = 0x203E,
  ["frasl"] = 0x2044,
  ["euro"] = 0x20AC,
  ["image"] = 0x2111,
  ["weierp"] = 0x2118,
  ["real"] = 0x211C,
  ["trade"] = 0x2122,
  ["alefsym"] = 0x2135,
  ["larr"] = 0x2190,
  ["uarr"] = 0x2191,
  ["rarr"] = 0x2192,
  ["darr"] = 0x2193,
  ["harr"] = 0x2194,
  ["crarr"] = 0x21B5,
  ["lArr"] = 0x21D0,
  ["uArr"] = 0x21D1,
  ["rArr"] = 0x21D2,
  ["dArr"] = 0x21D3,
  ["hArr"] = 0x21D4,
  ["forall"] = 0x2200,
  ["part"] = 0x2202,
  ["exist"] = 0x2203,
  ["empty"] = 0x2205,
  ["nabla"] = 0x2207,
  ["isin"] = 0x2208,
  ["notin"] = 0x2209,
  ["ni"] = 0x220B,
  ["prod"] = 0x220F,
  ["sum"] = 0x2211,
  ["minus"] = 0x2212,
  ["lowast"] = 0x2217,
  ["radic"] = 0x221A,
  ["prop"] = 0x221D,
  ["infin"] = 0x221E,
  ["ang"] = 0x2220,
  ["and"] = 0x2227,
  ["or"] = 0x2228,
  ["cap"] = 0x2229,
  ["cup"] = 0x222A,
  ["int"] = 0x222B,
  ["there4"] = 0x2234,
  ["sim"] = 0x223C,
  ["cong"] = 0x2245,
  ["asymp"] = 0x2248,
  ["ne"] = 0x2260,
  ["equiv"] = 0x2261,
  ["le"] = 0x2264,
  ["ge"] = 0x2265,
  ["sub"] = 0x2282,
  ["sup"] = 0x2283,
  ["nsub"] = 0x2284,
  ["sube"] = 0x2286,
  ["supe"] = 0x2287,
  ["oplus"] = 0x2295,
  ["otimes"] = 0x2297,
  ["perp"] = 0x22A5,
  ["sdot"] = 0x22C5,
  ["lceil"] = 0x2308,
  ["rceil"] = 0x2309,
  ["lfloor"] = 0x230A,
  ["rfloor"] = 0x230B,
  ["lang"] = 0x27E8,
  ["rang"] = 0x27E9,
  ["loz"] = 0x25CA,
  ["spades"] = 0x2660,
  ["clubs"] = 0x2663,
  ["hearts"] = 0x2665,
  ["diams"] = 0x2666,
}
```

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

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

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

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

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

```
function entities.char_entity(s)
  local n = character_entities[s]
  return unicode.utf8.char(n)
end
```

3.1.3 Plain TeX Writer

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

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

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

The `writer.new` method creates and returns a new TeX writer object associated with the Lua interface options (see Section 2.1.2) `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*⟩.

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

```
  local self = {}
  options = options or {}
```

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

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

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

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

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

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

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

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

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

```
  function self.plain(s)
    return s
  end
```

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

```
  function self.paragraph(s)
    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.

```
  self.interblocksep = "\\markdownRendererInterblockSeparator\n{}"
```

Define `writer->eof` as the end of file marker in the output format.

```
  self.eof = [[\relax]]
```

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

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

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

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

Define `writer->hrule` as the output format of a horizontal rule.

```
  self.hrule = "\\markdownRendererHorizontalRule{}"
```

Define a table escaped_chars containing the mapping from special plain TeX characters (including the active pipe character (|) of ConTeXt) to their escaped variants. Define tables escaped_minimal_chars and escaped_minimal_strings containing the mapping from special plain characters and character strings that need to be escaped even in content that will not be typeset.

```
local escaped_chars = {
   ["{"] = "\\markdownRendererLeftBrace{}",
   ["}"] = "\\markdownRendererRightBrace{}",
   ["$"] = "\\markdownRendererDollarSign{}",
   ["%"] = "\\markdownRendererPercentSign{}",
   ["&"] = "\\markdownRendererAmpersand{}",
   ["_"] = "\\markdownRendererUnderscore{}",
   ["#"] = "\\markdownRendererHash{}",
   ["^"] = "\\markdownRendererCircumflex{}",
   ["\\"] = "\\markdownRendererBackslash{}",
   ["~"] = "\\markdownRendererTilde{}",
   ["|"] = "\\markdownRendererPipe{}", }
 local escaped_minimal_chars = {
   ["{"] = "\\markdownRendererLeftBrace{}",
   ["}"] = "\\markdownRendererRightBrace{}",
   ["%"] = "\\markdownRendererPercentSign{}",
   ["\\"] = "\\markdownRendererBackslash{}", }
 local escaped_minimal_strings = {
   ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ", }
```

Use the escaped_chars table to create an escaper function escape and the escaped_minimal_chars and escaped_minimal_strings tables to create an escaper function escape_minimal.

```
local escape = util.escaper(escaped_chars)
local escape_minimal = util.escaper(escaped_minimal_chars,
  escaped_minimal_strings)
```

Define writer->string as a function that will transform an input plain text span s to the output format and writer->uri as a function that will transform an input URI u to the output format. If the hybrid option is true, use identity functions. Otherwise, use the escape and escape_minimal functions.

```
if options.hybrid then
  self.string = function(s) return s end
  self.uri = function(u) return u end
else
  self.string = escape
  self.uri = escape_minimal
end
```

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

```
function self.code(s)
```

```
    return {"\\markdownRendererCodeSpan{",escape(s),"}"}
  end
```

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

```
  function self.link(lab,src,tit)
    return {"\\markdownRendererLink{",lab,"}",
                          "{",self.string(src),"}",
                          "{",self.uri(src),"}",
                          "{",self.string(tit or ""),"}"}
  end
```

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

```
  function self.image(lab,src,tit)
    return {"\\markdownRendererImage{",lab,"}",
                           "{",self.string(src),"}",
                           "{",self.uri(src),"}",
                           "{",self.string(tit or ""),"}"}
  end
```

The `languages_json` table maps programming language filename extensions to fence infostrings. All `options.contentBlocksLanguageMap` files located by kpathsea are loaded into a chain of tables. `languages_json` corresponds to the first table and is chained with the rest via Lua metatables.

```
local languages_json = (function()
  local kpse = require('kpse')
  kpse.set_program_name('luatex')
  local base, prev, curr
  for _, file in ipairs{kpse.lookup(options.contentBlocksLanguageMap,
                                    { all=true })} do
    json = assert(io.open(file, "r")):read("*all")
                                     :gsub('("[^\n]-"):','[%1]=')
    curr = (function()
      local _ENV={ json=json, load=load } -- run in sandbox
      return load("return "..json)()
    end)()
    if type(curr) == "table" then
      if base == nil then
        base = curr
      else
        setmetatable(prev, { __index = curr })
      end
      prev = curr
    end
  end
  return base or {}
end)()
```

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

```
  function self.contentblock(src,suf,type,tit)
    src = src.."."..suf
    suf = suf:lower()
    if type == "onlineimage" then
      return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
                             "{",self.string(src),"}",
                             "{",self.uri(src),"}",
                             "{",self.string(tit or ""),"}"}
    elseif languages_json[suf] then
      return {"\\markdownRendererContentBlockCode{",suf,"}",
                             "{",self.string(languages_json[suf]),"}",
                             "{",self.string(src),"}",
                             "{",self.uri(src),"}",
                             "{",self.string(tit or ""),"}"}
    else
      return {"\\markdownRendererContentBlock{",suf,"}",
                             "{",self.string(src),"}",
                             "{",self.uri(src),"}",
                             "{",self.string(tit or ""),"}"}
    end
  end
```

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.

```
  local function ulitem(s)
    return {"\\markdownRendererUlItem ",s,
            "\\markdownRendererUlItemEnd "}
  end

  function self.bulletlist(items,tight)
    local buffer = {}
    for _,item in ipairs(items) do
      buffer[#buffer + 1] = ulitem(item)
    end
    local contents = util.intersperse(buffer,"\n")
    if tight and options.tightLists then
      return {"\\markdownRendererUlBeginTight\n",contents,
        "\n\\markdownRendererUlEndTight "}
    else
      return {"\\markdownRendererUlBegin\n",contents,
        "\n\\markdownRendererUlEnd "}
    end
  end
```

Define `writer->ollist` 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 should be used as the number of the first list item.

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

  function self.orderedlist(items,tight,startnum)
    local buffer = {}
    local num = startnum
    for _,item in ipairs(items) do
      buffer[#buffer + 1] = olitem(item,num)
      if num ~= nil then
        num = num + 1
      end
    end
    local contents = util.intersperse(buffer,"\n")
    if tight and options.tightLists then
      return {"\\markdownRendererOlBeginTight\n",contents,
        "\n\\markdownRendererOlEndTight "}
    else
      return {"\\markdownRendererOlBegin\n",contents,
        "\n\\markdownRendererOlEnd "}
    end
  end
```

Define `writer->inline_html` and `writer->display_html` as functions that will transform an inline or block HTML element respectively to the output format, where `html` is the HTML input.

```
  function self.inline_html(html)  return "" end
  function self.display_html(html) return "" end
```

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)
  local buffer = {}
  for _,item in ipairs(items) do
    buffer[#buffer + 1] = dlitem(item.term, item.definitions)
  end
  if tight and options.tightLists then
    return {"\\markdownRendererDlBeginTight\n", buffer,
      "\n\\markdownRendererDlEndTight"}
  else
    return {"\\markdownRendererDlBegin\n", buffer,
      "\n\\markdownRendererDlEnd"}
  end
end
```

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

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

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

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

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

```
function self.blockquote(s)
  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)
```

```
    local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
    return {"\\markdownRendererInputVerbatim{",name,"}"}
  end
```

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

```
  function self.fencedCode(i, s)
    local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
    return {"\\markdownRendererInputFencedCode{",name,"}{",i,"}"}
  end
```

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

```
  function self.heading(s,level)
    local cmd
    if level == 1 then
      cmd = "\\markdownRendererHeadingOne"
    elseif level == 2 then
      cmd = "\\markdownRendererHeadingTwo"
    elseif level == 3 then
      cmd = "\\markdownRendererHeadingThree"
    elseif level == 4 then
      cmd = "\\markdownRendererHeadingFour"
    elseif level == 5 then
      cmd = "\\markdownRendererHeadingFive"
    elseif level == 6 then
      cmd = "\\markdownRendererHeadingSix"
    else
      cmd = ""
    end
    return {cmd,"{",s,"}"}
  end
```

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

```
  function self.note(s)
    return {"\\markdownRendererFootnote{",s,"}"}
  end
```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is `true`, 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 = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
      "{", #cites, "}"}
    for _,cite in ipairs(cites) do
      buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
        cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}"}
    end
    return buffer
  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.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.circumflex             = P("^")
parsers.slash                  = P("/")
parsers.equal                  = P("=")
parsers.colon                  = P(":")
parsers.semicolon              = P(";")
parsers.exclamation            = P("!")
parsers.tilde                  = P("~")
parsers.tab                    = P("\t")
parsers.newline                = P("\n")
parsers.tightblocksep          = P("\001")

parsers.digit                  = R("09")
parsers.hexdigit               = R("09","af","AF")
parsers.letter                 = R("AZ","az")
parsers.alphanumeric           = R("AZ","az","09")
parsers.keyword                = parsers.letter
                               * parsers.alphanumeric^0
parsers.internal_punctuation   = S(":;,.#$%&-+?<>~/")

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

parsers.any                    = P(1)
parsers.fail                   = parsers.any - 1

parsers.escapable              = S("\\`*_{}[]()+_.!<>#-~:^@;")
parsers.anyescaped             = P("\\") / "" * 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.specialchar            = S("*_`&[]<!\\.@-^")

parsers.normalchar             = parsers.any - (parsers.specialchar
                                                + parsers.spacing
                                                + parsers.tightblocksep)
parsers.eof                    = -parsers.any
parsers.nonindentspace         = parsers.space^-3 * - parsers.spacechar
parsers.indent                 = parsers.space^-3 * parsers.tab
                               + parsers.fourspaces / ""
parsers.linechar               = P(1 - parsers.newline)

parsers.blankline              = parsers.optionalspace
                               * parsers.newline / "\n"
```

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

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

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

3.1.4.2 Parsers Used for Markdown Lists

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

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

3.1.4.3 Parsers Used for Markdown Code Spans

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

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

parsers.closeticks  = parsers.space^-1
```

```
                    * Cmt(C(parsers.backtick^1)
                         * Cb("ticks"), captures_equal_length)

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

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

3.1.4.4 Parsers Used for Fenced Code Blocks

```
local function captures_geq_length(s,i,a,b)
  return #a >= #b and i
end

parsers.infostring     = (parsers.linechar - (parsers.backtick
                       + parsers.space^1 * (parsers.newline + parsers.eof)))^0

local fenceindent
parsers.fencehead    = function(char)
  return               C(parsers.nonindentspace) / function(s) fenceindent = #s end
                     * Cg(char^3, "fencelength")
                     * parsers.optionalspace * C(parsers.infostring)
                     * parsers.optionalspace * (parsers.newline + parsers.eof)
end

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

parsers.fencedline   = function(char)
  return               C(parsers.line - parsers.fencetail(char))
                     / function(s)
                           return s:gsub("^" .. string.rep(" ?", fenceindent), "")
                       end
end
```

3.1.4.5 Parsers Used for Markdown Tags and Links

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

-- content in balanced brackets, parentheses, or quotes:
parsers.bracketed   = P{ parsers.lbracket
```

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

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

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

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

-- bracketed tag for markdown links, allowing nested brackets:
parsers.tag         = parsers.lbracket
                    * Cs((parsers.alphanumeric^1
                        + parsers.bracketed
                        + parsers.inticks
                        + (parsers.anyescaped
                          - (parsers.rbracket + parsers.blankline^2)))^0)
                    * parsers.rbracket

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

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

```

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

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

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

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

3.1.4.6 Parsers Used for iA Writer Content Blocks

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

-- case insensitive online image suffix:
parsers.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:
parsers.onlineimageurl
```

```
                    = (parsers.less
                      * Cs((parsers.anyescaped
                           - parsers.more
                           - #(parsers.period
                              * parsers.onlineimagesuffix
                              * parsers.more
                              * parsers.contentblock_tail))^0)
                      * parsers.period
                      * Cs(parsers.onlineimagesuffix)
                      * parsers.more
                      + (Cs((parsers.inparens
                            + (parsers.anyescaped
                              - parsers.spacing
                              - parsers.rparent
                              - #(parsers.period
                                 * parsers.onlineimagesuffix
                                 * parsers.contentblock_tail)))^0)
                        * parsers.period
                        * Cs(parsers.onlineimagesuffix))
                      ) * Cc("onlineimage")

-- filename for iA Writer content blocks with mandatory suffix:
parsers.localfilepath
                    = parsers.slash
                    * Cs((parsers.anyescaped
                         - parsers.tab
                         - parsers.newline
                         - #(parsers.period
                            * parsers.alphanumeric^1
                            * parsers.contentblock_tail))^1)
                    * parsers.period
                    * Cs(parsers.alphanumeric^1)
                    * Cc("localfile")
```

3.1.4.7 Parsers Used for Citations

```
parsers.citation_name = Cs(parsers.dash^-1) * parsers.at
                      * Cs(parsers.alphanumeric
                          * (parsers.alphanumeric + parsers.internal_punctuation
                            - parsers.comma - parsers.semicolon)^0)

parsers.citation_body_prenote
                    = Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + (parsers.anyescaped
                           - (parsers.rbracket + parsers.blankline^2))
                         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)

parsers.citation_body_postnote
                    = Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + (parsers.anyescaped
                           - (parsers.rbracket + parsers.semicolon
                             + parsers.blankline^2))
                         - (parsers.spnl * parsers.rbracket))^0)

parsers.citation_body_chunk
                    = parsers.citation_body_prenote
                    * parsers.spnl * parsers.citation_name
                    * (parsers.comma * parsers.spnl)^-1
                    * parsers.citation_body_postnote

parsers.citation_body
                    = parsers.citation_body_chunk
                    * (parsers.semicolon * parsers.spnl
                      * parsers.citation_body_chunk)^0

parsers.citation_headless_body_postnote
                    = Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + (parsers.anyescaped
                           - (parsers.rbracket + parsers.at
                             + parsers.semicolon + parsers.blankline^2))
                         - (parsers.spnl * parsers.rbracket))^0)

parsers.citation_headless_body
                    = parsers.citation_headless_body_postnote
                    * (parsers.sp * parsers.semicolon * parsers.spnl
                      * parsers.citation_body_chunk)^0
```

3.1.4.8 Parsers Used for Footnotes

```
local function strip_first_char(s)
  return s:sub(2)
end

parsers.RawNoteRef = #(parsers.lbracket * parsers.circumflex)
                   * parsers.tag / strip_first_char
```

3.1.4.9 Parsers Used for HTML

```
-- case-insensitive match (we assume s is lowercase). must be single byte encoding
```

```
parsers.keyword_exact = function(s)
  local parser = P(0)
  for i=1,#s do
    local c = s:sub(i,i)
    local m = c .. upper(c)
    parser = parser * S(m)
  end
  return parser
end

parsers.block_keyword =
    parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
    parsers.keyword_exact("center") + parsers.keyword_exact("del") +
    parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
    parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
    parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
    parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
    parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
    parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
    parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
    parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
    parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
    parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
    parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
    parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
    parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
    parsers.keyword_exact("td") + parsers.keyword_exact("tr")

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

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

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

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

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

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

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

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

parsers.closeelt_exact = function(s)
  return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
       * parsers.sp * parsers.more
end

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

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

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

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

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

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

parsers.displayhtml = parsers.htmlcomment
```

```
                     + parsers.emptyelt_block
                     + parsers.openelt_exact("hr")
                     + parsers.in_matched_block_tags
                     + parsers.htmlinstruction

parsers.inlinehtml   = parsers.emptyelt_any
                     + parsers.htmlcomment
                     + parsers.htmlinstruction
                     + parsers.openelt_any
                     + parsers.closeelt_any
```

3.1.4.10 Parsers Used for HTML entities

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

3.1.4.11 Helpers for References

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

3.1.4.12 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

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

3.1.4.13 Block Elements

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

parsers.OnlineImageURL
                     = parsers.leader
                     * parsers.onlineimageurl
                     * parsers.optionaltitle

```

```
parsers.LocalFilePath
                      = parsers.leader
                      * parsers.localfilepath
                      * parsers.optionaltitle

parsers.TildeFencedCode
                      = parsers.fencehead(parsers.tilde)
                      * Cs(parsers.fencedline(parsers.tilde)^0)
                      * parsers.fencetail(parsers.tilde)

parsers.BacktickFencedCode
                      = parsers.fencehead(parsers.backtick)
                      * Cs(parsers.fencedline(parsers.backtick)^0)
                      * parsers.fencetail(parsers.backtick)

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

3.1.4.14 Lists

```
parsers.defstartchar = S("~:")
parsers.defstart     = ( parsers.defstartchar * #parsers.spacing
                                             * (parsers.tab + parsers.space^-3)
                      + parsers.space * parsers.defstartchar * #parsers.spacing
                                      * (parsers.tab + parsers.space^-2)
                      + parsers.space * parsers.space * parsers.defstartchar
                                      * #parsers.spacing
                                      * (parsers.tab + parsers.space^-1)
                      + parsers.space * parsers.space * parsers.space
                                      * parsers.defstartchar * #parsers.spacing
                      )

parsers.dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
```

3.1.4.15 Headings

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

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

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

```
end
```

3.1.5 Markdown Reader

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

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

The `reader.new` method creates and returns a new TeX reader object associated with the Lua interface options (see Section 2.1.2) `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 = {}
  options = options or {}
```

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

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

3.1.5.1 Top-Level Helper Functions

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

```
  local function normalize_tag(tag)
    return unicode.utf8.lower(
      gsub(util.rope_to_string(tag), "[ \n\r\t]+", " "))
  end
```

Define `expandtabs` either as an identity function, when the `preserveTabs` Lua inrerface option is `true`, or to a function that expands tabs into spaces otherwise.

```
  local expandtabs
  if options.preserveTabs then
    expandtabs = function(s) return s end
  else
    expandtabs = function(s)
                   if s:find("\t") then
                     return s:gsub("[^\n]*", util.expand_tabs_in_line)
```

```
                    else
                      return s
                    end
                  end
end
```

The larsers (as in “local parsers”) hash table stores PEG patterns that depend on the received options, which impedes their reuse between different reader objects.

```
local larsers    = {}
```

3.1.5.2 Top-Level Parser Functions

```
local function create_parser(name, grammar)
  return function(str)
    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

local parse_blocks
  = create_parser("parse_blocks",
                  function()
                    return larsers.blocks
                  end)

local parse_blocks_toplevel
  = create_parser("parse_blocks_toplevel",
                  function()
                    return larsers.blocks_toplevel
                  end)

local parse_inlines
  = create_parser("parse_inlines",
                  function()
                    return larsers.inlines
                  end)

local parse_inlines_no_link
  = create_parser("parse_inlines_no_link",
                  function()
                    return larsers.inlines_no_link
                  end)

local parse_inlines_no_inline_note
```

```
  = create_parser("parse_inlines_no_inline_note",
                  function()
                    return larsers.inlines_no_inline_note
                  end)

local parse_inlines_nbsp
  = create_parser("parse_inlines_nbsp",
                  function()
                    return larsers.inlines_nbsp
                  end)
```

3.1.5.3 Parsers Used for Markdown Lists (local)

```
if options.hashEnumerators then
  larsers.dig = parsers.digit + parsers.hash
else
  larsers.dig = parsers.digit
end

larsers.enumerator = C(larsers.dig^3 * parsers.period) * #parsers.spacing
                   + C(larsers.dig^2 * parsers.period) * #parsers.spacing
                                     * (parsers.tab + parsers.space^1)
                   + C(larsers.dig * parsers.period) * #parsers.spacing
                                   * (parsers.tab + parsers.space^-2)
                   + parsers.space * C(larsers.dig^2 * parsers.period)
                                   * #parsers.spacing
                   + parsers.space * C(larsers.dig * parsers.period)
                                   * #parsers.spacing
                                   * (parsers.tab + parsers.space^-1)
                   + parsers.space * parsers.space * C(larsers.dig^1
                                   * parsers.period) * #parsers.spacing
```

3.1.5.4 Parsers Used for Blockquotes (local)

```
-- strip off leading > and indents, and run through blocks
larsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-1)/""
                           * parsers.linechar^0 * parsers.newline)^1
                          * (-(parsers.leader * parsers.more
                              + parsers.blankline) * parsers.linechar^1
                            * parsers.newline)^0

if not options.breakableBlockquotes then
  larsers.blockquote_body = larsers.blockquote_body
                          * (parsers.blankline^0 / "")
end
```

3.1.5.5 Parsers Used for Citations (local)

```
larsers.citations = function(text_cites, raw_cites)
    local function normalize(str)
        if str == "" then
            str = nil
        else
            str = (options.citationNbsps and parse_inlines_nbsp or
              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.string(raw_cites[i+2]),
            postnote = normalize(raw_cites[i+3]),
        }
    end
    return writer.citations(text_cites, cites)
end
```

3.1.5.6 Parsers Used for Footnotes (local)

```
local rawnotes = {}

-- like indirect_link
local function lookup_note(ref)
  return function()
    local found = rawnotes[normalize_tag(ref)]
    if found then
      return writer.note(parse_blocks_toplevel(found))
    else
      return {"[", parse_inlines("^" .. ref), "]"}
    end
  end
end

local function register_note(ref,rawnote)
  rawnotes[normalize_tag(ref)] = rawnote
  return ""
end

larsers.NoteRef    = parsers.RawNoteRef / lookup_note


larsers.NoteBlock  = parsers.leader * parsers.RawNoteRef * parsers.colon
                   * parsers.spnl * parsers.indented_blocks(parsers.chunk)
                   / register_note

larsers.InlineNote = parsers.circumflex
                   * (parsers.tag / parse_inlines_no_inline_note) -- no notes inside
                   / writer.note
```

3.1.5.7 Helpers for Links and References (local)

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

-- add a reference to the list
local function register_link(tag,url,title)
    references[normalize_tag(tag)] = { url = url, title = title }
    return ""
end

-- lookup link reference and return either
-- the link or nil and fallback text.
local function lookup_reference(label,sps,tag)
    local tagpart
    if not tag then
        tag = label
        tagpart = ""
    elseif tag == "" then
        tag = label
        tagpart = "[]"
    else
        tagpart = {"[", parse_inlines(tag), "]"}
    end
    if sps then
      tagpart = {sps, tagpart}
    end
    local r = references[normalize_tag(tag)]
    if r then
      return r
    else
      return nil, {"[", parse_inlines(label), "]", tagpart}
    end
end

-- lookup link reference and return a link, if the reference is found,
-- or a bracketed label otherwise.
local function indirect_link(label,sps,tag)
  return function()
```

```
    local r,fallback = lookup_reference(label,sps,tag)
    if r then
      return writer.link(parse_inlines_no_link(label), r.url, r.title)
    else
      return fallback
    end
  end
end

-- lookup image reference and return an image, if the reference is found,
-- or a bracketed label otherwise.
local function indirect_image(label,sps,tag)
  return function()
    local r,fallback = lookup_reference(label,sps,tag)
    if r then
      return writer.image(writer.string(label), r.url, r.title)
    else
      return {"!", fallback}
    end
  end
end
```

3.1.5.8 Inline Elements (local)

```
larsers.Str      = parsers.normalchar^1 / writer.string

larsers.Symbol   = (parsers.specialchar - parsers.tightblocksep)
                 / writer.string

larsers.Ellipsis = P("...") / writer.ellipsis

larsers.Smart    = larsers.Ellipsis

larsers.Code     = parsers.inticks / writer.code

if options.blankBeforeBlockquote then
  larsers.bqstart = parsers.fail
else
  larsers.bqstart = parsers.more
end

if options.blankBeforeHeading then
  larsers.headerstart = parsers.fail
else
  larsers.headerstart = parsers.hash
                      + (parsers.line * (parsers.equal^1 + parsers.dash^1)
                      * parsers.optionalspace * parsers.newline)
```

```
end

if not options.fencedCode or options.blankBeforeCodeFence then
  larsers.fencestart = parsers.fail
else
  larsers.fencestart = parsers.fencehead(parsers.backtick)
                     + parsers.fencehead(parsers.tilde)
end

larsers.Endline   = parsers.newline * -( -- newline, but not before...
                      parsers.blankline -- paragraph break
                    + parsers.tightblocksep  -- nested list
                    + parsers.eof       -- end of document
                    + larsers.bqstart
                    + larsers.headerstart
                    + larsers.fencestart
                  ) * parsers.spacechar^0 / writer.space

larsers.Space      = parsers.spacechar^2 * larsers.Endline / writer.linebreak
                   + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
                   + parsers.spacechar^1 * larsers.Endline^-1
                                         * parsers.optionalspace / writer.space

larsers.NonbreakingEndline
                  = parsers.newline * -( -- newline, but not before...
                      parsers.blankline -- paragraph break
                    + parsers.tightblocksep  -- nested list
                    + parsers.eof       -- end of document
                    + larsers.bqstart
                    + larsers.headerstart
                    + larsers.fencestart
                  ) * parsers.spacechar^0 / writer.nbsp

larsers.NonbreakingSpace
                = parsers.spacechar^2 * larsers.Endline / writer.linebreak
                + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
                + parsers.spacechar^1 * larsers.Endline^-1
                                      * parsers.optionalspace / writer.nbsp

if options.underscores then
  larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
                                     parsers.doubleasterisks)
                   + parsers.between(parsers.Inline, parsers.doubleunderscores,
                                     parsers.doubleunderscores)
                   ) / writer.strong

  larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
                                     parsers.asterisk)
                   + parsers.between(parsers.Inline, parsers.underscore,
                                     parsers.underscore)
                   ) / writer.emphasis
else
  larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
                                     parsers.doubleasterisks)
                   ) / writer.strong

  larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
                                     parsers.asterisk)
                   ) / writer.emphasis
end

larsers.AutoLinkUrl    = parsers.less
                       * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
                       * parsers.more
                       / function(url)
                           return writer.link(writer.string(url), url)
                         end

larsers.AutoLinkEmail = parsers.less
                      * C((parsers.alphanumeric + S("-._+"))^1
                      * P("@") * parsers.urlchar^1)
                      * parsers.more
                      / function(email)
                          return writer.link(writer.string(email),
                                             "mailto:"..email)
                        end

larsers.DirectLink    = (parsers.tag / parse_inlines_no_link)  -- no links inside lin
                      * parsers.spnl
                      * parsers.lparent
                      * (parsers.url + Cc(""))  -- link can be empty [foo]()
                      * parsers.optionaltitle
                      * parsers.rparent
                      / writer.link

larsers.IndirectLink  = parsers.tag * (C(parsers.spnl) * parsers.tag)^-1
                      / indirect_link

-- parse a link or image (direct or indirect)
larsers.Link          = larsers.DirectLink + larsers.IndirectLink

larsers.DirectImage   = parsers.exclamation
                      * (parsers.tag / parse_inlines)
                      * parsers.spnl
```

```
                             * parsers.lparent
                             * (parsers.url + Cc(""))  -- link can be empty [foo]()
                             * parsers.optionaltitle
                             * parsers.rparent
                             / writer.image

larsers.IndirectImage = parsers.exclamation * parsers.tag
                             * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image

larsers.Image         = larsers.DirectImage + larsers.IndirectImage

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

larsers.ParenthesizedCitations
                             = Ct(parsers.lbracket
                             * parsers.citation_body
                             * parsers.rbracket)
                             / function(raw_cites)
                                 return larsers.citations(false, raw_cites)
                               end

larsers.Citations     = larsers.TextCitations + larsers.ParenthesizedCitations

-- avoid parsing long strings of * or _ as emph/strong
larsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
                             / writer.string

larsers.EscapedChar   = S("\\") * C(parsers.escapable) / writer.string

larsers.InlineHtml    = C(parsers.inlinehtml) / writer.inline_html

larsers.HtmlEntity    = parsers.hexentity / entities.hex_entity  / writer.string
                             + parsers.decentity / entities.dec_entity  / writer.string
                             + parsers.tagentity / entities.char_entity / writer.string
```

3.1.5.9 Block Elements (local)

```
larsers.ContentBlock = parsers.leader
                            * (parsers.localfilepath + parsers.onlineimageurl)
```

```
                        * parsers.contentblock_tail
                        / writer.contentblock

larsers.DisplayHtml  = C(parsers.displayhtml)
                     / expandtabs / writer.display_html

larsers.Verbatim     = Cs( (parsers.blanklines
                         * ((parsers.indentedline - parsers.blankline))^1)^1
                         ) / expandtabs / writer.verbatim

larsers.FencedCode   = (parsers.TildeFencedCode
                       + parsers.BacktickFencedCode)
                     / function(infostring, code)
                         return writer.fencedCode(writer.string(infostring),
                                                  expandtabs(code))
                       end

larsers.Blockquote   = Cs(larsers.blockquote_body^1)
                     / parse_blocks_toplevel / writer.blockquote

larsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
                         + parsers.lineof(parsers.dash)
                         + parsers.lineof(parsers.underscore)
                         ) / writer.hrule

larsers.Reference    = parsers.define_reference_parser / register_link

larsers.Paragraph    = parsers.nonindentspace * Ct(parsers.Inline^1)
                     * parsers.newline
                     * ( parsers.blankline^1
                       + #parsers.hash
                       + #(parsers.leader * parsers.more * parsers.space^-1)
                       )
                     / writer.paragraph

larsers.ToplevelParagraph
                     = parsers.nonindentspace * Ct(parsers.Inline^1)
                     * ( parsers.newline
                     * ( parsers.blankline^1
                       + #parsers.hash
                       + #(parsers.leader * parsers.more * parsers.space^-1)
                       + parsers.eof
                       )
                     + parsers.eof )
                     / writer.paragraph

larsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
```

```
                        / writer.plain
```

3.1.5.10 Lists (local)

```
larsers.starter = parsers.bullet + larsers.enumerator

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

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

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

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

larsers.TightListItem = function(starter)
    return -larsers.HorizontalRule
           * (Cs(starter / "" * larsers.ListBlock * larsers.NestedList^-1)
             / parse_blocks)
           * -(parsers.blanklines * parsers.indent)
end

larsers.LooseListItem = function(starter)
    return -larsers.HorizontalRule
           * Cs( starter / "" * larsers.ListBlock * Cc("\n")
             * (larsers.NestedList + larsers.ListContinuationBlock^0)
             * (parsers.blanklines / "\n\n")
             ) / parse_blocks
end

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

local function ordered_list(items,tight,startNumber)
  if options.startNumber then
    startNumber = tonumber(startNumber) or 1  -- fallback for '#'
  else
    startNumber = nil
  end
  return writer.orderedlist(items,tight,startNumber)
end

larsers.OrderedList = Cg(larsers.enumerator, "listtype") *
                      ( Ct(larsers.TightListItem(Cb("listtype"))
                          * larsers.TightListItem(larsers.enumerator)^0)
                      * Cc(true) * parsers.skipblanklines * -larsers.enumerator
                      + Ct(larsers.LooseListItem(Cb("listtype"))
                          * larsers.LooseListItem(larsers.enumerator)^0)
                      * Cc(false) * parsers.skipblanklines
                      ) * Cb("listtype") / ordered_list

local function definition_list_item(term, defs, tight)
  return { term = parse_inlines(term), definitions = defs }
end

larsers.DefinitionListItemLoose = C(parsers.line) * parsers.skipblanklines
                                * Ct((parsers.defstart
                                     * parsers.indented_blocks(parsers.dlchunk)
                                     / parse_blocks_toplevel)^1)
                                * Cc(false) / definition_list_item

larsers.DefinitionListItemTight = C(parsers.line)
                                * Ct((parsers.defstart * parsers.dlchunk
                                     / parse_blocks)^1)
                                * Cc(true) / definition_list_item

larsers.DefinitionList = ( Ct(larsers.DefinitionListItemLoose^1) * Cc(false)
                         + Ct(larsers.DefinitionListItemTight^1)
                         * (parsers.skipblanklines
                           * -larsers.DefinitionListItemLoose * Cc(true))
                         ) / writer.definitionlist
```

3.1.5.11 Blank (local)

```
larsers.Blank        = parsers.blankline / ""
                     + larsers.NoteBlock
                     + larsers.Reference
                     + (parsers.tightblocksep / "\n")
```

3.1.5.12 Headings (local)

```
-- parse atx header
larsers.AtxHeading = Cg(parsers.HeadingStart,"level")
                   * parsers.optionalspace
                   * (C(parsers.line) / strip_atx_end / parse_inlines)
                   * Cb("level")
```

```
                    / writer.heading

-- parse setext header
larsers.SetextHeading = #(parsers.line * S("=-"))
                      * Ct(parsers.line / parse_inlines)
                      * parsers.HeadingLevel
                      * parsers.optionalspace * parsers.newline
                      / writer.heading

larsers.Heading = larsers.AtxHeading + larsers.SetextHeading
```

3.1.5.13 Syntax Specification

```
local syntax =
  { "Blocks",

    Blocks                = larsers.Blank^0 * parsers.Block^-1
                          * (larsers.Blank^0 / function()
                                                 return writer.interblocksep
                                               end
                            * parsers.Block)^0
                          * larsers.Blank^0 * parsers.eof,

    Blank                 = larsers.Blank,

    Block                 = V("ContentBlock")
                          + V("Blockquote")
                          + V("Verbatim")
                          + V("FencedCode")
                          + V("HorizontalRule")
                          + V("BulletList")
                          + V("OrderedList")
                          + V("Heading")
                          + V("DefinitionList")
                          + V("DisplayHtml")
                          + V("Paragraph")
                          + V("Plain"),

    ContentBlock          = larsers.ContentBlock,
    Blockquote            = larsers.Blockquote,
    Verbatim              = larsers.Verbatim,
    FencedCode            = larsers.FencedCode,
    HorizontalRule        = larsers.HorizontalRule,
    BulletList            = larsers.BulletList,
    OrderedList           = larsers.OrderedList,
    Heading               = larsers.Heading,
    DefinitionList        = larsers.DefinitionList,
    DisplayHtml           = larsers.DisplayHtml,
    Paragraph             = larsers.Paragraph,
    Plain                 = larsers.Plain,

    Inline                = V("Str")
                          + V("Space")
                          + V("Endline")
                          + V("UlOrStarLine")
                          + V("Strong")
                          + V("Emph")
                          + V("InlineNote")
                          + V("NoteRef")
                          + V("Citations")
                          + V("Link")
                          + V("Image")
                          + V("Code")
                          + V("AutoLinkUrl")
                          + V("AutoLinkEmail")
                          + V("InlineHtml")
                          + V("HtmlEntity")
                          + V("EscapedChar")
                          + V("Smart")
                          + V("Symbol"),

    Str                   = larsers.Str,
    Space                 = larsers.Space,
    Endline               = larsers.Endline,
    UlOrStarLine          = larsers.UlOrStarLine,
    Strong                = larsers.Strong,
    Emph                  = larsers.Emph,
    InlineNote            = larsers.InlineNote,
    NoteRef               = larsers.NoteRef,
    Citations             = larsers.Citations,
    Link                  = larsers.Link,
    Image                 = larsers.Image,
    Code                  = larsers.Code,
    AutoLinkUrl           = larsers.AutoLinkUrl,
    AutoLinkEmail         = larsers.AutoLinkEmail,
    InlineHtml            = larsers.InlineHtml,
    HtmlEntity            = larsers.HtmlEntity,
    EscapedChar           = larsers.EscapedChar,
    Smart                 = larsers.Smart,
    Symbol                = larsers.Symbol,
  }

if not options.citations then
  syntax.Citations = parsers.fail
```

```
end

if not options.contentBlocks then
  syntax.ContentBlock = parsers.fail
end

if not options.codeSpans then
  syntax.Code = parsers.fail
end

if not options.definitionLists then
  syntax.DefinitionList = parsers.fail
end

if not options.fencedCode then
  syntax.FencedCode = parsers.fail
end

if not options.footnotes then
  syntax.NoteRef = parsers.fail
end

if not options.html then
  syntax.DisplayHtml = parsers.fail
  syntax.InlineHtml = parsers.fail
  syntax.HtmlEntity  = parsers.fail
end

if not options.inlineFootnotes then
  syntax.InlineNote = parsers.fail
end

if not options.smartEllipses then
  syntax.Smart = parsers.fail
end

local blocks_toplevel_t = util.table_copy(syntax)
blocks_toplevel_t.Paragraph = larsers.ToplevelParagraph
larsers.blocks_toplevel = Ct(blocks_toplevel_t)

larsers.blocks = Ct(syntax)

local inlines_t = util.table_copy(syntax)
inlines_t[1] = "Inlines"
inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
larsers.inlines = Ct(inlines_t)

```

```
  local inlines_no_link_t = util.table_copy(inlines_t)
  inlines_no_link_t.Link = parsers.fail
  larsers.inlines_no_link = Ct(inlines_no_link_t)

  local inlines_no_inline_note_t = util.table_copy(inlines_t)
  inlines_no_inline_note_t.InlineNote = parsers.fail
  larsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)

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

3.1.5.14 Exported Conversion Function

Define `reader->convert` as a function that converts markdown string `input` into a plain TEX output and returns it. Note that the converter assumes that the input has UNIX line endings.

```
  function self.convert(input)
    references = {}
```

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 2.1.2). The `cacheDir` option is disregarded.

```
    local opt_string = {}
    for k,_ in pairs(defaultOptions) do
      local v = options[k]
      if k ~= "cacheDir" then
        opt_string[#opt_string+1] = k .. "=" .. tostring(v)
      end
    end
    table.sort(opt_string)
    local salt = table.concat(opt_string, ",") .. "," .. metadata.version
```

Produce the cache file, transform its filename via the `writer->pack` method, and return the result.

```
    local name = util.cache(options.cacheDir, input, salt, function(input)
        return util.rope_to_string(parse_blocks_toplevel(input)) .. writer.eof
      end, ".md" .. writer.suffix)
    return writer.pack(name)
  end
  return self
end
```

3.1.6 Conversion from Markdown to Plain TeX

The `new` method returns the `reader->convert` function of a reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object associated with `options`.

```
function M.new(options)
  local writer = M.writer.new(options)
  local reader = M.reader.new(writer, options)
  return reader.convert
end

return M
```

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

```
\def\markdownInfo#1{%
  \message{(l.\the\inputlineno) markdown.tex info: #1.}}%
\def\markdownWarning#1{%
  \message{(l.\the\inputlineno) markdown.tex warning: #1}}%
\def\markdownError#1#2{%
  \errhelp{#2.}%
  \errmessage{(l.\the\inputlineno) markdown.tex error: #1}}%
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererLineBreakPrototype{\hfil\break}%
\let\markdownRendererEllipsisPrototype\dots
\def\markdownRendererNbspPrototype{~}%
\def\markdownRendererLeftBracePrototype{\char`{}%
\def\markdownRendererRightBracePrototype{\char`}}%
\def\markdownRendererDollarSignPrototype{\char`$}%
\def\markdownRendererPercentSignPrototype{\char`\%}%
\def\markdownRendererAmpersandPrototype{\char`&}%
\def\markdownRendererUnderscorePrototype{\char`_}%
\def\markdownRendererHashPrototype{\char`\#}%
\def\markdownRendererCircumflexPrototype{\char`^}%
\def\markdownRendererBackslashPrototype{\char`\\}%
```

```
\def\markdownRendererTildePrototype{\char`~}%
\def\markdownRendererPipePrototype{|}%
\def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
\def\markdownRendererLinkPrototype#1#2#3#4{#2}%
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \markdownInput{#3}}%
\def\markdownRendererContentBlockOnlineImagePrototype{%
  \markdownRendererImage}%
\def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
  \markdownRendererInputFencedCode{#3}{#2}}%
\def\markdownRendererImagePrototype#1#2#3#4{#2}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
\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{\par\begingroup\it}%
\def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
\def\markdownRendererInputVerbatimPrototype#1{%
  \par{\tt\input"#1"\relax}\par}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \markdownRendererInputVerbatimPrototype{#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\markdownRendererHorizontalRulePrototype{}%
```

```
\def\markdownRendererFootnotePrototype#1{#1}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
```

3.2.3 Lua Snippets

The \markdownLuaOptions macro expands to a Lua table that contains the plain TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.2).

```
\def\markdownLuaOptions{{%
\ifx\markdownOptionBlankBeforeBlockquote\undefined\else
  blankBeforeBlockquote = \markdownOptionBlankBeforeBlockquote,
\fi
\ifx\markdownOptionBlankBeforeCodeFence\undefined\else
  blankBeforeCodeFence = \markdownOptionBlankBeforeCodeFence,
\fi
\ifx\markdownOptionBlankBeforeHeading\undefined\else
  blankBeforeHeading = \markdownOptionBlankBeforeHeading,
\fi
\ifx\markdownOptionBreakableBlockquotes\undefined\else
  breakableBlockquotes = \markdownOptionBreakableBlockquotes,
\fi
\ifx\markdownOptionCacheDir\undefined\else
  cacheDir = "\markdownOptionCacheDir",
\fi
\ifx\markdownOptionCitations\undefined\else
  citations = \markdownOptionCitations,
\fi
\ifx\markdownOptionCitationNbsps\undefined\else
  citationNbsps = \markdownOptionCitationNbsps,
\fi
\ifx\markdownOptionCodeSpans\undefined\else
  codeSpans = \markdownOptionCodeSpans,
\fi
\ifx\markdownOptionContentBlocks\undefined\else
  contentBlocks = \markdownOptionContentBlocks,
\fi
\ifx\markdownOptionContentBlocksLanguageMap\undefined\else
  contentBlocksLanguageMap =
    "\markdownOptionContentBlocksLanguageMap",
\fi
\ifx\markdownOptionDefinitionLists\undefined\else
  definitionLists = \markdownOptionDefinitionLists,
\fi
\ifx\markdownOptionFootnotes\undefined\else
  footnotes = \markdownOptionFootnotes,
\fi
\ifx\markdownOptionFencedCode\undefined\else
  fencedCode = \markdownOptionFencedCode,
\fi
\ifx\markdownOptionHashEnumerators\undefined\else
  hashEnumerators = \markdownOptionHashEnumerators,
\fi
\ifx\markdownOptionHtml\undefined\else
  html = \markdownOptionHtml,
\fi
\ifx\markdownOptionHybrid\undefined\else
  hybrid = \markdownOptionHybrid,
\fi
\ifx\markdownOptionInlineFootnotes\undefined\else
  inlineFootnotes = \markdownOptionInlineFootnotes,
\fi
\ifx\markdownOptionPreserveTabs\undefined\else
  preserveTabs = \markdownOptionPreserveTabs,
\fi
\ifx\markdownOptionSmartEllipses\undefined\else
  smartEllipses = \markdownOptionSmartEllipses,
\fi
\ifx\markdownOptionStartNumber\undefined\else
  startNumber = \markdownOptionStartNumber,
\fi
\ifx\markdownOptionTightLists\undefined\else
  tightLists = \markdownOptionTightLists,
\fi
\ifx\markdownOptionUnderscores\undefined\else
  underscores = \markdownOptionUnderscores,
\fi}
}%
```

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 `\markdownOptionCacheDir` directory exists.

```
local lfs = require("lfs")
local cacheDir = "\markdownOptionCacheDir"
if lfs.isdir(cacheDir) == true then else
  assert(lfs.mkdir(cacheDir))
end
```

Next, load the `markdown` module and create a converter function using the plain TEX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
local md = require("markdown")
local convert = md.new(\markdownLuaOptions)
```

```
}%
```

3.2.4 Buffering Markdown Input

The macro `\markdownLuaExecuteFileStream` contains the number of the output file stream that will be used to store the helper Lua script in the file named `\markdownOptionHelperScriptFileName` during the expansion of the macro `\markdownLuaExecute` when the Lua shell escape bridge is in use, and to store the markdown input in the file named `\markdownOptionInputTempFileName` during the expansion of the macro `\markdownReadAndConvert`.

```
\csname newwrite\endcsname\markdownLuaExecuteFileStream
```

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 2ε `\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.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |gdef|markdownReadAndConvert#1#2{%
    |begingroup%
```

Open the `\markdownOptionInputTempFileName` file for writing.

```
    |immediate|openout|markdownLuaExecuteFileStream%
      |markdownOptionInputTempFileName%
    |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 \markdownReadAndConvertProcessLine macro will process the individual lines of output. Note the use of the comments to ensure that the entire macro is at a single line and therefore no (active) newline symbols are produced.

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

When the ending token sequence does not appear in the line, store the line in the \markdownOptionInputTempFileName file.

```
      |ifx|relax##3|relax%
        |immediate|write|markdownLuaExecuteFileStream{##1}%
      |else%
```

When the ending token sequence appears in the line, make the next newline character close the \markdownOptionInputTempFileName file, return the character categories back to the former state, convert the \markdownOptionInputTempFileName 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}%
          |immediate|closeout|markdownLuaExecuteFileStream%
          |endgroup%
          |markdownInput|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{%
        |markdownReadAndConvertProcessLine####1#1#1|relax}%
      ^^M}%
    ^^M}%
```

Reset the character categories back to the former state.

```
|endgroup
```

3.2.5 Lua Shell Escape Bridge

The following TEX code is intended for TEX engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the LuaTEX engine, their TeX distribution contains it, and uses shell access to produce and execute Lua scripts using the TEXLua interpreter (see [1, Section 3.1.1]).

```
\ifnum\markdownMode<2\relax
\ifnum\markdownMode=0\relax
  \markdownInfo{Using mode 0: Shell escape via write18}%
\else
  \markdownInfo{Using mode 1: Shell escape via os.execute}%
\fi
```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (`1`), disabled (`0`), or restricted (`2`).

Inherit the value of the the `\pdfshellescape` (LuaTEX, PdfTEX) or the `\shellescape` (XƎTEX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```
\ifx\pdfshellescape\undefined
  \ifx\shellescape\undefined
    \ifnum\markdownMode=0\relax
      \def\markdownExecuteShellEscape{1}%
    \else
      \def\markdownExecuteShellEscape{%
        \directlua{tex.sprint(status.shell_escape or "1")}}%
    \fi
  \else
    \let\markdownExecuteShellEscape\shellescape
  \fi
\else
  \let\markdownExecuteShellEscape\pdfshellescape
\fi
```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```
\ifnum\markdownMode=0\relax
  \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
\else
  \def\markdownExecuteDirect#1{%
    \directlua{os.execute("\luaescapestring{#1}")}}%
```

```
\fi
```

The \markdownExecute macro is a wrapper on top of \markdownExecuteDirect that checks the value of \markdownExecuteShellEscape and prints an error message if the shell is inaccessible.

```
\def\markdownExecute#1{%
  \ifnum\markdownExecuteShellEscape=1\relax
    \markdownExecuteDirect{#1}%
  \else
    \markdownError{I can not access the shell}{Either run the TeX
      compiler with the --shell-escape or the --enable-write18 flag,
      or set shell_escape=t in the texmf.cnf file}%
  \fi}%
```

The \markdownLuaExecute macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TEX engine, but it can use the print function in the same manner it would use the tex.print method.

```
\def\markdownLuaExecute#1{%
```

Create the file \markdownOptionHelperScriptFileName and fill it with the input Lua code prepended with kpathsea initialization, so that Lua modules from the TEX distribution are available.

```
  \immediate\openout\markdownLuaExecuteFileStream=%
    \markdownOptionHelperScriptFileName
  \markdownInfo{Writing a helper Lua script to the file
    "\markdownOptionHelperScriptFileName"}%
  \immediate\write\markdownLuaExecuteFileStream{%
    local kpse = require('kpse')
    kpse.set_program_name('luatex') #1}%
  \immediate\closeout\markdownLuaExecuteFileStream
```

Execute the generated \markdownOptionHelperScriptFileName Lua script using the TEXLua binary and store the output in the \markdownOptionOutputTempFileName file.

```
  \markdownInfo{Executing a helper Lua script from the file
    "\markdownOptionHelperScriptFileName" and storing the result in the
    file "\markdownOptionOutputTempFileName"}%
  \markdownExecute{texlua "\markdownOptionHelperScriptFileName" >
    "\markdownOptionOutputTempFileName"}%
```

\input the generated \markdownOptionOutputTempFileName file.

```
  \input\markdownOptionOutputTempFileName\relax}%
```

3.2.6 Direct Lua Access

The following TEX code is intended for TEX engines that provide direct access to Lua (LuaTEX). The macro \markdownLuaExecute defined here and in Section 3.2.5

are meant to be indistinguishable to the remaining code. This corresponds to the \markdownMode value of 2.

```
\else
\markdownInfo{Using mode 2: Direct Lua access}%
```

The direct Lua access version of the \markdownLuaExecute macro is defined in terms of the \directlua primitive. The print function is set as an alias to the \tex.print method in order to mimic the behaviour of the \markdownLuaExecute definition from Section 3.2.5,

```
\def\markdownLuaExecute#1{\directlua{local print = tex.print #1}}%
\fi
```

3.2.7 Typesetting Markdown

The \markdownInput macro uses an implementation of the \markdownLuaExecute macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TEX.

```
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```
  \catcode`|=0%
  \catcode`\\=12%
  |gdef|markdownInput#1{%
    |markdownInfo{Including markdown document "#1"}%
    |markdownLuaExecute{%
      |markdownPrepare
      local input = assert(io.open("#1","r")):read("*a")
```

Since the Lua converter expects UNIX line endings, normalize the input.

```
      print(convert(input:gsub("\r\n?", "\n")))}}%
|endgroup
```

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 (see [5, Section 9]). As a consequence, we can directly reuse the existing plain TEX implementation.

```
\input markdown
\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.

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

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the LaTeX interface (see Section 2.3.2).

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

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

```
\renewenvironment{markdown}{%
  \markdownReadAndConvert@markdown{}}\relax
\renewenvironment{markdown*}[1]{%
  \markdownSetup{#1}%
  \markdownReadAndConvert@markdown*}\relax
\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
```

After processing the options, activate the `renderers` and `rendererPrototypes` keys.

```
\define@key{markdownOptions}{renderers}{%
  \setkeys{markdownRenderers}{#1}%
  \def\KV@prefix{KV@markdownOptions@}}%
\define@key{markdownOptions}{rendererPrototypes}{%
  \setkeys{markdownRendererPrototypes}{#1}%
  \def\KV@prefix{KV@markdownOptions@}}%
```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
\RequirePackage{url}
\RequirePackage{graphicx}
```

If the `\markdownOptionTightLists` macro expands to `false`, do not load the paralist package. This is necessary for LaTeX 2_ε document classes that do not play nice with paralist, such as beamer. If the `\markdownOptionTightLists` is undefined and the beamer document class is in use, then do not load the paralist package either.

```
\RequirePackage{ifthen}
\ifx\markdownOptionTightLists\undefined
  \@ifclassloaded{beamer}{}{
    \RequirePackage{paralist}}
\else
  \ifthenelse{\equal{\markdownOptionTightLists}{false}}{}{
    \RequirePackage{paralist}}
\fi
```

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.

```
\@ifpackageloaded{paralist}{
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\begin{compactitem}},
    ulEndTight = {\end{compactitem}},
    olBeginTight = {\begin{compactenum}},
    olEndTight = {\end{compactenum}},
    dlBeginTight = {\begin{compactdesc}},
    dlEndTight = {\end{compactdesc}}}}
}{
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\markdownRendererUlBegin},
    ulEndTight = {\markdownRendererUlEnd},
    olBeginTight = {\markdownRendererOlBegin},
    olEndTight = {\markdownRendererOlEnd},
    dlBeginTight = {\markdownRendererDlBegin},
    dlEndTight = {\markdownRendererDlEnd}}}}
\RequirePackage{fancyvrb}
\RequirePackage{csvsimple}
\markdownSetup{rendererPrototypes={
  lineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  pipe = {\textbar},
  codeSpan = {\texttt{#1}},
  link = {#1\footnote{\ifx\empty#4\empty\else#4:
    \fi\texttt<\url{#3}\texttt>}},
  contentBlock = {%
    \ifthenelse{\equal{#1}{csv}}{%
      \begin{table}%
        \begin{center}%
          \csvautotabular{#3}%
        \end{center}
        \ifx\empty#4\empty\else
          \caption{#4}%
        \fi
        \label{tab:#1}%
      \end{table}}{%
      \markdownInput{#3}}},
  image = {%
    \begin{figure}%
      \begin{center}%
        \includegraphics{#3}%
      \end{center}%
      \ifx\empty#4\empty\else
        \caption{#4}%
      \fi
      \label{fig:#1}%
    \end{figure}},
  ulBegin = {\begin{itemize}},
  ulItem = {\item},
  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}},
  blockQuoteBegin = {\begin{quotation}},
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  inputFencedCode = {%
    \ifx\relax#2\relax
      \VerbatimInput{#1}%
    \else
      \ifx\minted@jobname\undefined
        \ifx\lst@version\undefined
          \markdownRendererInputFencedCode{#1}{}%
```

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

```
        \else
          \lstinputlisting[language=#2]{#1}%
        \fi
```

When the minted package is loaded, use it for syntax highlighting. The minted package is preferred over listings.

```
      \else
        \inputminted{#2}{#1}%
      \fi
    \fi},
  horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  footnote = {\footnote{#1}}}}
```

Support the nesting of strong emphasis.

```
\newif\ifmarkdownLATEXStrongEmphasisNested
\markdownLATEXStrongEmphasisNestedfalse
\markdownSetup{rendererPrototypes={
  strongEmphasis = {%
    \ifmarkdownLATEXStrongEmphasisNested
      \markdownLATEXStrongEmphasisNestedfalse
      \textmd{#1}%
      \markdownLATEXStrongEmphasisNestedtrue
    \else
      \markdownLATEXStrongEmphasisNestedtrue
      \textbf{#1}%
      \markdownLATEXStrongEmphasisNestedfalse
    \fi}}}
```

Support LaTeX document classes that do not provide chapters.

```
\ifx\chapter\undefined
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}},
    headingFive = {\subparagraph{#1}}}}
\else
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}},
    headingSix = {\subparagraph{#1}}}}
\fi
```

There is a basic implementation for citations that uses the LaTeX `\cite` macro. There is also a more advanced implementation that uses the BibLaTeX `\autocites` and `\textcites` macros. This implementation will be used, when BibLaTeX is loaded.

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\def\markdownLaTeXBasicCitations#1#2#3#4{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#2\relax\else#2~\fi\cite[#3]{#4}%
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \expandafter\@gobble
  \fi\markdownLaTeXBasicCitations}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% 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}%
    \ifx\autocites\undefined
      \expandafter
      \markdownLaTeXBasicCitations
    \else
```

```
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXCitations
      \expandafter{\expandafter}%
    \fi},
  textCite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \ifx\textcites\undefined
      \expandafter
      \markdownLaTeXBasicTextCitations
    \else
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXTextCitations
      \expandafter{\expandafter}%
    \fi}}}
```

3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the inputenc package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents package.

```
\newcommand\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat}%
```

3.4 ConTEXt Implementation

The ConTEXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTEXt formats *seem* to implement (the documentation is scarce) the majority of the plain TEX format required by the plain TEX implementation. As a consequence, we can directly reuse the existing plain TEX implementation after supplying the missing plain TEX macros.

```
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\input markdown
```

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 Logging Facilities

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

```
\def\markdownInfo#1{\writestatus{markdown}{#1.}}%
\def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
```

3.4.2 Typesetting Markdown

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

```
\begingroup
```

Locally swap the category code of the backslash symbol with the pipe symbol. 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`\\=12%
  |gdef|startmarkdown{%
    |markdownReadAndConvert{\stopmarkdown}%
                           {|stopmarkdown}}%
|endgroup
```

3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
\def\markdownRendererLineBreakPrototype{\blank}%
\def\markdownRendererLeftBracePrototype{\textbraceleft}%
\def\markdownRendererRightBracePrototype{\textbraceright}%
\def\markdownRendererDollarSignPrototype{\textdollar}%
\def\markdownRendererPercentSignPrototype{\percent}%
\def\markdownRendererUnderscorePrototype{\textunderscore}%
\def\markdownRendererCircumflexPrototype{\textcircumflex}%
\def\markdownRendererBackslashPrototype{\textbackslash}%
\def\markdownRendererTildePrototype{\textasciitilde}%
```

```
\def\markdownRendererPipePrototype{\char`|}%
\def\markdownRendererLinkPrototype#1#2#3#4{%
  \useURL[#1][#3][][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
  \fi\tt<\hyphenatedurl{#3}>}}%
\usemodule[database]
\defineseparatedlist
  [MarkdownConTeXtCSV]
  [separator={,},
   before=\bTABLE,after=\eTABLE,
   first=\bTR,last=\eTR,
   left=\bTD,right=\eTD]
\def\markdownConTeXtCSV{csv}
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \def\markdownConTeXtCSV@arg{#1}%
\ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
    \placetable[][tab:#1]{#4}{%
      \processseparatedfile[MarkdownConTeXtCSV][#3]}%
\else
\markdownInput{#3}%
\fi}%
\def\markdownRendererImagePrototype#1#2#3#4{%
  \placefigure[][fig:#1]{#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\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \ifx\relax#2\relax
    \typefile{#1}%
  \else
```

The code fence infostring is used as a name from the ConTEXt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```
% Map the ‘TEX‘ syntax highlighter to the ‘latex‘ infostring.
\definetyping [latex]
\setuptyping  [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext
```

```
    \typefile[#2][]{#1}%
  \fi}%
\def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
\def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
\def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
\def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
\def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
```

```
\def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
\def\markdownRendererHorizontalRulePrototype{%
  \blackrule[height=1pt, width=\hsize]}%
\def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
\stopmodule\protect
```

References

1. LUATEX DEVELOPMENT TEAM. *LuaTEX reference manual* [online]. 2016 [visited on 2016-11-27]. Available from: http://www.luatex.org/svn/trunk/manual/luatex.pdf.
2. SOTKOV, Anton. *File transclusion syntax for Markdown* [online]. 2017 [visited on 2017-03-18]. Available from: https://github.com/iainc/Markdown-Content-Blocks.
3. KNUTH, Donald Ervin. *The TEXbook*. 3rd ed. Addison-Westley, 1986. ISBN 0-201-13447-0.
4. IERUSALIMSCHY, Roberto. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. ISBN 978-85-903798-5-0.
5. BRAAMS, Johannes; CARLISLE, David; JEFFREY, Alan; LAMPORT, Leslie; MITTELBACH, Frank; ROWLEY, Chris; SCHÖPF, Rainer. *The LATEX 2$_\varepsilon$ Sources* [online]. 2016 [visited on 2016-09-27]. Available from: http://mirrors.ctan.org/macros/latex/base/source2e.pdf.