

A Markdown Interpreter for T_EX

Vít Novotný (based on the work of
John MacFarlane and Hans Hagen)
witiko@mail.muni.cz

Version 2.1.1
September 6, 2016

Contents

1	Introduction	1		
1.1	About Markdown	1	2.3	ℳT _E X Interface
1.2	Feedback	2	2.4	ConT _E Xt Interface
1.3	Acknowledgements	2		
1.4	Prerequisites	2	3	Technical Documentation
2	User Guide	4	3.1	Lua Implementation
2.1	Lua Interface	4	3.2	Plain T _E X Implementation
2.2	Plain T _E X Interface	10	3.3	ℳT _E X Implementation
			3.4	ConT _E Xt Implementation

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 T_EX. These are provided both in the form of a Lua module and in the form of plain T_EX, ℳT_EX, and ConT_EXt macro packages that enable the direct inclusion of markdown documents inside T_EX documents.

Architecturally, the package consists of the Lunamark v1.4.0 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 T_EX, ℳT_EX, and ConT_EXt formats by Vít Novotný.

```
1 local metadata = {
2   version   = "2.1.1",
3   comment   = "A module for the conversion from markdown to plain TeX",
4   author    = "John MacFarlane, Hans Hagen, Vít Novotný",
```

```

5     copyright = "2009-2016 John MacFarlane, Hans Hagen; 2016 Vít Novotný",
6     license   = "LPPL 1.3"
7 }
8 if not modules then modules = { } end
9 modules['markdown'] = metadata

```

1.2 Feedback

Please use the markdown project page on GitHub¹ 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 T_EX-~~E~~_TX Stack Exchange².

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 T_EX part of the package draws inspiration from several sources including the source code of ~~E~~_TX 2_ε, the minted package by Geoffrey M. Poore – which likewise tackles the issue of interfacing with an external interpreter from T_EX, 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 LuaT_EX 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 LuaT_EX \geq 0.72.0 (T_EXLive \geq 2013).

```

10     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 Lua \TeX (\TeX Live \geq 2008).

```
11      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 Lua \TeX (\TeX Live \geq 2008).

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

All the abovelisted modules are statically linked into the current version of the Lua \TeX 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 Lua \TeX (\TeX Live \geq 2008).

The plain \TeX code makes use of the `isdir` method that was added to the Lua File System library by the Lua \TeX engine developers (see [1, Section 3.2]).

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

The plain \TeX part of the package also requires that either the Lua \TeX `\directlua` primitive or the shell access file stream 18 is available.

1.4.3 \LaTeX Prerequisites

The \LaTeX part of the package requires that the $\LaTeX 2_{\epsilon}$ format is loaded,

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

all the plain \TeX prerequisites (see Section 1.4.2), and the following $\LaTeX 2_{\epsilon}$ 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).

1.4.4 ConT_EXt prerequisites

The ConT_EXt part of the package requires that either the Mark II or the Mark IV format is loaded and all the plain T_EX prerequisites (see Section 1.4.2).

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 T_EX 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 T_EX. This interface is used by the plain T_EX 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.

```
14 local M = {}
```

2.1.1 Conversion from Markdown to Plain T_EX

The Lua interface exposes the `new(options)` method. This method creates converter functions that perform the conversion from markdown to plain T_EX 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 T_EX output using the default options and prints the T_EX 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.

```
15 local defaultOptions = {}
```

`blankBeforeBlockquote=true, false` default: false

<code>true</code>	Require a blank line between a paragraph and the following blockquote.
<code>false</code>	Do not require a blank line between a paragraph and the following blockquote.

```
16 defaultOptions.blankBeforeBlockquote = false
```

`blankBeforeCodeFence=true, false` default: false

<code>true</code>	Require a blank line between a paragraph and the following fenced code block.
<code>false</code>	Do not require a blank line between a paragraph and the following fenced code block.

```
17 defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeHeading=true, false` default: false

<code>true</code>	Require a blank line between a paragraph and the following header.
<code>false</code>	Do not require a blank line between a paragraph and the following header.

```
18 defaultOptions.blankBeforeHeading = false
```

`cacheDir`=*<directory>* 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.

```
19 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.

```
20 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.

```
21 defaultOptions.citations = false
```

`definitionLists=true, false`

default: false

`true`

Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

Third paragraph of definition 2.
```

`false`

Disable the pandoc definition list syntax extension.

```
22 defaultOptions.definitionLists = false
```

`hashEnumerators=true, false`

default: false

`true`

Enable the use of hash symbols (#) as ordered item list markers.

`false`

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

```
23 defaultOptions.hashEnumerators = 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.

```
24 defaultOptions.hybrid = 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>
'''
```

true

Disable the commonmark fenced code block extension.

```
25 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.
```



```
26 defaultOptions.footnotes = false
```

`false` Convert any tabs in the input to spaces.

<code>false</code>	Preserve all ellipses in the input.
--------------------	-------------------------------------

false	Ignore the number in the items of ordered lists. Each item will only produce a <code>\markdownRenderer01Item</code> TeX macro.
-------	--

`false` Lists whose bullets do not consist of multiple paragraphs will be treated the same way as lists that do.

9

2.2 Plain T_EX Interface

The plain T_EX interface provides macros for the typesetting of markdown input from within plain T_EX, for setting the Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain T_EX, and for changing the way markdown the tokens are rendered.

```
31 \def\markdownLastModified{2016/09/06}%  
32 \def\markdownVersion{2.1.1}%
```

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

```
\input markdown
```

It is expected that the special plain T_EX characters have the expected category codes, when `\input`ing 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.

```
33 \let\markdownBegin\relax  
34 \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 corollary, 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 T_EX (see [2, p. 46]). As a corollary, 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 T_EX 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 .

```
35 \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.

```
36 \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.

```
37 \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.

```
38 \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 \TeX package. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
39 \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.

```
40 \let\markdownOptionBlankBeforeBlockquote\undefined
41 \let\markdownOptionBlankBeforeCodeFence\undefined
42 \let\markdownOptionBlankBeforeHeading\undefined
43 \let\markdownOptionCitations\undefined
44 \let\markdownOptionCitationNbsps\undefined
45 \let\markdownOptionDefinitionLists\undefined
46 \let\markdownOptionFootnotes\undefined
47 \let\markdownOptionFencedCode\undefined
48 \let\markdownOptionHashEnumerators\undefined
49 \let\markdownOptionHybrid\undefined
50 \let\markdownOptionPreserveTabs\undefined
51 \let\markdownOptionSmartEllipses\undefined
52 \let\markdownOptionStartNumber\undefined
53 \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.

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

2.2.3.2 Line Break Renderer The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

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

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

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

2.2.3.4 Non-breaking Space Renderer The `\markdownRendererNbsp` macro represents a non-breaking space.

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

2.2.3.5 Special Character Renderers The following macros replace any special plain \TeX characters (including the active pipe character (`|`) of $\text{Con}\TeX$ t) in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```
62 \def\markdownRendererLeftBrace{%
63   \markdownRendererLeftBracePrototype}%
64 \def\markdownRendererRightBrace{%
65   \markdownRendererRightBracePrototype}%
66 \def\markdownRendererDollarSign{%
67   \markdownRendererDollarSignPrototype}%
68 \def\markdownRendererPercentSign{%
69   \markdownRendererPercentSignPrototype}%
70 \def\markdownRendererAmpersand{%
71   \markdownRendererAmpersandPrototype}%
72 \def\markdownRendererUnderscore{%
73   \markdownRendererUnderscorePrototype}%
74 \def\markdownRendererHash{%
```

```

75 \markdownRendererHashPrototype}%
76 \def\markdownRendererCircumflex{%
77 \markdownRendererCircumflexPrototype}%
78 \def\markdownRendererBackslash{%
79 \markdownRendererBackslashPrototype}%
80 \def\markdownRendererTilde{%
81 \markdownRendererTildePrototype}%
82 \def\markdownRendererPipe{%
83 \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.

```

84 \def\markdownRendererCodeSpan{%
85 \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.

```

86 \def\markdownRendererLink{%
87 \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.

```

88 \def\markdownRendererImage{%
89 \markdownRendererImagePrototype}%

```

2.2.3.9 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.

```

90 \def\markdownRendererUlBegin{%
91 \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.

```

92 \def\markdownRendererUlBeginTight{%
93 \markdownRendererUlBeginTightPrototype}%

```

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

```
94 \def\markdownRendererUListItem{%
95   \markdownRendererUListItemPrototype}%
```

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

```
96 \def\markdownRendererUListItemEnd{%
97   \markdownRendererUListItemEndPrototype}%
```

The `\markdownRendererUListEnd` 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.

```
98 \def\markdownRendererUListEnd{%
99   \markdownRendererUListEndPrototype}%
```

The `\markdownRendererUListEndTight` 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.

```
100 \def\markdownRendererUListEndTight{%
101   \markdownRendererUListEndTightPrototype}%
```

2.2.3.10 Ordered List Renderers The `\markdownRendererOListBegin` 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.

```
102 \def\markdownRendererOListBegin{%
103   \markdownRendererOListBeginPrototype}%
```

The `\markdownRendererOListBeginTight` 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.

```
104 \def\markdownRendererOListBeginTight{%
105   \markdownRendererOListBeginTightPrototype}%
```

The `\markdownRendererOListItem` 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.

```
106 \def\markdownRendererOListItem{%
107   \markdownRendererOListItemPrototype}%
```

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

```
108 \def\markdownRendererOListItemEnd{%
109   \markdownRendererOListItemEndPrototype}%
```

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.

```
110 \def\markdownRendererOlItemWithNumber{%
111   \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.

```
112 \def\markdownRendererOlEnd{%
113   \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.

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

2.2.3.11 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.

```
116 \def\markdownRendererDlBegin{%
117   \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.

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

The `\markdownRendererDlItem` macro represents a term in a definition list. The macro receives a single argument that corresponds to the term being defined.

```
120 \def\markdownRendererDlItem{%
121   \markdownRendererDlItemPrototype}%
```

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

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

The `\markdownRendererDlDefinitionBegin` macro represents the beginning of a definition in a definition list. There can be several definitions for a single term.


```

124 \def\markdownRendererDlDefinitionBegin{%
125   \markdownRendererDlDefinitionBeginPrototype}%

```

The `\markdownRendererDlDefinitionEnd` macro represents the end of a definition in a definition list. There can be several definitions for a single term.

```

126 \def\markdownRendererDlDefinitionEnd{%
127   \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.

```

128 \def\markdownRendererDlEnd{%
129   \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.

```

130 \def\markdownRendererDlEndTight{%
131   \markdownRendererDlEndTightPrototype}%

```

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

```

132 \def\markdownRendererEmphasis{%
133   \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.

```

134 \def\markdownRendererStrongEmphasis{%
135   \markdownRendererStrongEmphasisPrototype}%

```

2.2.3.13 Block Quote Renderers The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```

136 \def\markdownRendererBlockQuoteBegin{%
137   \markdownRendererBlockQuoteBeginPrototype}%

```

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

```

138 \def\markdownRendererBlockQuoteEnd{%
139   \markdownRendererBlockQuoteEndPrototype}%

```

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

```
140 \def\markdownRendererInputVerbatim{%  
141   \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 containing the code block contents and to the code fence infostring.

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

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

```
144 \def\markdownRendererHeadingOne{%  
145   \markdownRendererHeadingOnePrototype}%
```

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

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

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

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

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

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

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

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

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

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

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

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

2.2.3.17 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.

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

2.2.3.18 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.

```
160 \def\markdownRendererCite{%
161   \markdownRendererCitePrototype}%
```

2.2.3.19 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.

```
162 \def\markdownRendererTextCite{%
163   \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).

```
164 \def\markdownRendererInterblockSeparatorPrototype{%
165 \def\markdownRendererLineBreakPrototype}%
166 \def\markdownRendererEllipsisPrototype}%
167 \def\markdownRendererNbspPrototype}%
168 \def\markdownRendererLeftBracePrototype}%
169 \def\markdownRendererRightBracePrototype}%
170 \def\markdownRendererDollarSignPrototype}%
171 \def\markdownRendererPercentSignPrototype}%
```

```

172 \def\markdownRendererAmpersandPrototype{}%
173 \def\markdownRendererUnderscorePrototype{}%
174 \def\markdownRendererHashPrototype{}%
175 \def\markdownRendererCircumflexPrototype{}%
176 \def\markdownRendererBackslashPrototype{}%
177 \def\markdownRendererTildePrototype{}%
178 \def\markdownRendererPipePrototype{}%
179 \def\markdownRendererCodeSpanPrototype#1{}%
180 \def\markdownRendererLinkPrototype#1#2#3#4{}%
181 \def\markdownRendererImagePrototype#1#2#3#4{}%
182 \def\markdownRendererU1BeginPrototype{}%
183 \def\markdownRendererU1BeginTightPrototype{}%
184 \def\markdownRendererU1ItemPrototype{}%
185 \def\markdownRendererU1ItemEndPrototype{}%
186 \def\markdownRendererU1EndPrototype{}%
187 \def\markdownRendererU1EndTightPrototype{}%
188 \def\markdownRendererO1BeginPrototype{}%
189 \def\markdownRendererO1BeginTightPrototype{}%
190 \def\markdownRendererO1ItemPrototype{}%
191 \def\markdownRendererO1ItemWithNumberPrototype#1{}%
192 \def\markdownRendererO1ItemEndPrototype{}%
193 \def\markdownRendererO1EndPrototype{}%
194 \def\markdownRendererO1EndTightPrototype{}%
195 \def\markdownRendererD1BeginPrototype{}%
196 \def\markdownRendererD1BeginTightPrototype{}%
197 \def\markdownRendererD1ItemPrototype#1{}%
198 \def\markdownRendererD1ItemEndPrototype{}%
199 \def\markdownRendererD1DefinitionBeginPrototype{}%
200 \def\markdownRendererD1DefinitionEndPrototype{}%
201 \def\markdownRendererD1EndPrototype{}%
202 \def\markdownRendererD1EndTightPrototype{}%
203 \def\markdownRendererEmphasisPrototype#1{}%
204 \def\markdownRendererStrongEmphasisPrototype#1{}%
205 \def\markdownRendererBlockQuoteBeginPrototype{}%
206 \def\markdownRendererBlockQuoteEndPrototype{}%
207 \def\markdownRendererInputVerbatimPrototype#1{}%
208 \def\markdownRendererInputFencedCodePrototype#1#2{}%
209 \def\markdownRendererHeadingOnePrototype#1{}%
210 \def\markdownRendererHeadingTwoPrototype#1{}%
211 \def\markdownRendererHeadingThreePrototype#1{}%
212 \def\markdownRendererHeadingFourPrototype#1{}%
213 \def\markdownRendererHeadingFivePrototype#1{}%
214 \def\markdownRendererHeadingSixPrototype#1{}%
215 \def\markdownRendererHorizontalRulePrototype{}%
216 \def\markdownRendererFootnotePrototype#1{}%
217 \def\markdownRendererCitePrototype#1{}%
218 \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.

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

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

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

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

2.2.6 Miscellanea

The `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros specify the Lua code for registering and unregistering a callback for changing the contents of the line input buffer before a \TeX engine that supports direct Lua access via the `\directlua` macro starts looking at it. The first argument of the `\markdownLuaRegisterIBCallback` macro corresponds to the callback function being registered.

Local members defined within `\markdownLuaRegisterIBCallback` are guaranteed to be visible from `\markdownLuaUnregisterIBCallback` and the execution of the two macros alternates, so it is not necessary to consider the case, when one of the macros is called twice in a row.

```
222 \def\markdownLuaRegisterIBCallback#1{%  
223   local old_callback = callback.find("process_input_buffer")  
224   callback.register("process_input_buffer", #1)}%  
225 \def\markdownLuaUnregisterIBCallback{%  
226   callback.register("process_input_buffer", old_callback)}%
```

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*.

```
227 \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.

```

228 \let\markdownReadAndConvert\relax
229 \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*.

```

230 \catcode'\|=0\catcode'\=12%
231 \gdef\markdownBegin{%
232     \markdownReadAndConvert{\markdownEnd}%
233     {\markdownEnd}}%
234 \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 3.2.5), 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.

```

235 \ifx\markdownMode\undefined
236     \ifx\directlua\undefined
237         \def\markdownMode{0}%
238     \else
239         \def\markdownMode{2}%
240     \fi
241 \fi

```

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 $\langle options \rangle$ are the \LaTeX interface options (see Section 2.3.2). Note that $\langle options \rangle$ 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 $\LaTeX 2_\epsilon$ parses package options.

2.3.1 Typesetting Markdown

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

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

```
242 \newenvironment{markdown}\relax\relax
243 \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:

<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown} _Hello_ **world** ... \end{markdown} % ... \end{document}</pre>	<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown*}{smartEllipses} _Hello_ **world** ... \end{markdown*} % ... \end{document}</pre>
--	---

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 influence this markdown document.

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

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

2.3.2 Options

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

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.

```
244 \newcommand\markdownSetup[1]{%
245   \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).

```
246 \RequirePackage{keyval}
247 \define@key{markdownOptions}{helperScriptFileName}{%
248   \def\markdownOptionHelperScriptFileName{#1}}%
249 \define@key{markdownOptions}{inputTempFileName}{%
250   \def\markdownOptionInputTempFileName{#1}}%
251 \define@key{markdownOptions}{outputTempFileName}{%
252   \def\markdownOptionOutputTempFileName{#1}}%
253 \define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
254   \def\markdownOptionBlankBeforeBlockquote{#1}}%
255 \define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
256   \def\markdownOptionBlankBeforeCodeFence{#1}}%
257 \define@key{markdownOptions}{blankBeforeHeading}[true]{%
258   \def\markdownOptionBlankBeforeHeading{#1}}%
259 \define@key{markdownOptions}{citations}[true]{%
```



```

260 \def\markdownOptionCitations{#1}}%
261 \define@key{markdownOptions}{citationNbsps}[true]{%
262 \def\markdownOptionCitationNbsps{#1}}%
263 \define@key{markdownOptions}{cacheDir}{%
264 \def\markdownOptionCacheDir{#1}}%
265 \define@key{markdownOptions}{definitionLists}[true]{%
266 \def\markdownOptionDefinitionLists{#1}}%
267 \define@key{markdownOptions}{footnotes}[true]{%
268 \def\markdownOptionFootnotes{#1}}%
269 \define@key{markdownOptions}{fencedCode}[true]{%
270 \def\markdownOptionFencedCode{#1}}%
271 \define@key{markdownOptions}{hashEnumerators}[true]{%
272 \def\markdownOptionHashEnumerators{#1}}%
273 \define@key{markdownOptions}{hybrid}[true]{%
274 \def\markdownOptionHybrid{#1}}%
275 \define@key{markdownOptions}{preserveTabs}[true]{%
276 \def\markdownOptionPreserveTabs{#1}}%
277 \define@key{markdownOptions}{smartEllipses}[true]{%
278 \def\markdownOptionSmartEllipses{#1}}%
279 \define@key{markdownOptions}{startNumber}[true]{%
280 \def\markdownOptionStartNumber{#1}}%

```

If the `tightLists=false` option is specified, when loading the package, then the `paralist` package for typesetting tight lists will not be automatically loaded. This precaution is meant to minimize the footprint of this package, since some document-classes (beamer) experience clashes with the `paralist` package.

```

281 \define@key{markdownOptions}{tightLists}[true]{%
282 \def\markdownOptionTightLists{#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).

```

283 \define@key{markdownRenderers}{interblockSeparator}{%
284 \renewcommand\markdownRendererInterblockSeparator{#1}}%

```

```

285 \define@key{markdownRenderers}{lineBreak}{%
286   \renewcommand\markdownRendererLineBreak{#1}}%
287 \define@key{markdownRenderers}{ellipsis}{%
288   \renewcommand\markdownRendererEllipsis{#1}}%
289 \define@key{markdownRenderers}{nbsp}{%
290   \renewcommand\markdownRendererNbsp{#1}}%
291 \define@key{markdownRenderers}{leftBrace}{%
292   \renewcommand\markdownRendererLeftBrace{#1}}%
293 \define@key{markdownRenderers}{rightBrace}{%
294   \renewcommand\markdownRendererRightBrace{#1}}%
295 \define@key{markdownRenderers}{dollarSign}{%
296   \renewcommand\markdownRendererDollarSign{#1}}%
297 \define@key{markdownRenderers}{percentSign}{%
298   \renewcommand\markdownRendererPercentSign{#1}}%
299 \define@key{markdownRenderers}{ampersand}{%
300   \renewcommand\markdownRendererAmpersand{#1}}%
301 \define@key{markdownRenderers}{underscore}{%
302   \renewcommand\markdownRendererUnderscore{#1}}%
303 \define@key{markdownRenderers}{hash}{%
304   \renewcommand\markdownRendererHash{#1}}%
305 \define@key{markdownRenderers}{circumflex}{%
306   \renewcommand\markdownRendererCircumflex{#1}}%
307 \define@key{markdownRenderers}{backslash}{%
308   \renewcommand\markdownRendererBackslash{#1}}%
309 \define@key{markdownRenderers}{tilde}{%
310   \renewcommand\markdownRendererTilde{#1}}%
311 \define@key{markdownRenderers}{pipe}{%
312   \renewcommand\markdownRendererPipe{#1}}%
313 \define@key{markdownRenderers}{codeSpan}{%
314   \renewcommand\markdownRendererCodeSpan[1]{#1}}%
315 \define@key{markdownRenderers}{link}{%
316   \renewcommand\markdownRendererLink[4]{#1}}%
317 \define@key{markdownRenderers}{image}{%
318   \renewcommand\markdownRendererImage[4]{#1}}%
319 \define@key{markdownRenderers}{ulBegin}{%
320   \renewcommand\markdownRendererUlBegin{#1}}%
321 \define@key{markdownRenderers}{ulBeginTight}{%
322   \renewcommand\markdownRendererUlBeginTight{#1}}%
323 \define@key{markdownRenderers}{ulItem}{%
324   \renewcommand\markdownRendererUlItem{#1}}%
325 \define@key{markdownRenderers}{ulItemEnd}{%
326   \renewcommand\markdownRendererUlItemEnd{#1}}%
327 \define@key{markdownRenderers}{ulEnd}{%
328   \renewcommand\markdownRendererUlEnd{#1}}%
329 \define@key{markdownRenderers}{ulEndTight}{%
330   \renewcommand\markdownRendererUlEndTight{#1}}%
331 \define@key{markdownRenderers}{olBegin}{%

```

```

332 \renewcommand\markdownRendererOlBegin{#1}}%
333 \define@key{markdownRenderers}{olBeginTight}{%
334 \renewcommand\markdownRendererOlBeginTight{#1}}%
335 \define@key{markdownRenderers}{olItem}{%
336 \renewcommand\markdownRendererOlItem{#1}}%
337 \define@key{markdownRenderers}{olItemWithNumber}{%
338 \renewcommand\markdownRendererOlItemWithNumber[1]{#1}}%
339 \define@key{markdownRenderers}{olItemEnd}{%
340 \renewcommand\markdownRendererOlItemEnd{#1}}%
341 \define@key{markdownRenderers}{olEnd}{%
342 \renewcommand\markdownRendererOlEnd{#1}}%
343 \define@key{markdownRenderers}{olEndTight}{%
344 \renewcommand\markdownRendererOlEndTight{#1}}%
345 \define@key{markdownRenderers}{dlBegin}{%
346 \renewcommand\markdownRendererDlBegin{#1}}%
347 \define@key{markdownRenderers}{dlBeginTight}{%
348 \renewcommand\markdownRendererDlBeginTight{#1}}%
349 \define@key{markdownRenderers}{dlItem}{%
350 \renewcommand\markdownRendererDlItem[1]{#1}}%
351 \define@key{markdownRenderers}{dlItemEnd}{%
352 \renewcommand\markdownRendererDlItemEnd{#1}}%
353 \define@key{markdownRenderers}{dlDefinitionBegin}{%
354 \renewcommand\markdownRendererDlDefinitionBegin{#1}}%
355 \define@key{markdownRenderers}{dlDefinitionEnd}{%
356 \renewcommand\markdownRendererDlDefinitionEnd{#1}}%
357 \define@key{markdownRenderers}{dlEnd}{%
358 \renewcommand\markdownRendererDlEnd{#1}}%
359 \define@key{markdownRenderers}{dlEndTight}{%
360 \renewcommand\markdownRendererDlEndTight{#1}}%
361 \define@key{markdownRenderers}{emphasis}{%
362 \renewcommand\markdownRendererEmphasis[1]{#1}}%
363 \define@key{markdownRenderers}{strongEmphasis}{%
364 \renewcommand\markdownRendererStrongEmphasis[1]{#1}}%
365 \define@key{markdownRenderers}{blockquoteBegin}{%
366 \renewcommand\markdownRendererBlockQuoteBegin{#1}}%
367 \define@key{markdownRenderers}{blockquoteEnd}{%
368 \renewcommand\markdownRendererBlockQuoteEnd{#1}}%
369 \define@key{markdownRenderers}{inputVerbatim}{%
370 \renewcommand\markdownRendererInputVerbatim[1]{#1}}%
371 \define@key{markdownRenderers}{inputFencedCode}{%
372 \renewcommand\markdownRendererInputFencedCode[2]{#1}}%
373 \define@key{markdownRenderers}{headingOne}{%
374 \renewcommand\markdownRendererHeadingOne[1]{#1}}%
375 \define@key{markdownRenderers}{headingTwo}{%
376 \renewcommand\markdownRendererHeadingTwo[1]{#1}}%
377 \define@key{markdownRenderers}{headingThree}{%
378 \renewcommand\markdownRendererHeadingThree[1]{#1}}%

```

```

379 \define@key{markdownRenderers}{headingFour}{%
380   \renewcommand\markdownRendererHeadingFour[1]{#1}}%
381 \define@key{markdownRenderers}{headingFive}{%
382   \renewcommand\markdownRendererHeadingFive[1]{#1}}%
383 \define@key{markdownRenderers}{headingSix}{%
384   \renewcommand\markdownRendererHeadingSix[1]{#1}}%
385 \define@key{markdownRenderers}{horizontalRule}{%
386   \renewcommand\markdownRendererHorizontalRule{#1}}%
387 \define@key{markdownRenderers}{footnote}{%
388   \renewcommand\markdownRendererFootnote[1]{#1}}%
389 \define@key{markdownRenderers}{cite}{%
390   \renewcommand\markdownRendererCite[1]{#1}}%
391 \define@key{markdownRenderers}{textCite}{%
392   \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).

```

393 \define@key{markdownRendererPrototypes}{interblockSeparator}{%
394   \renewcommand\markdownRendererInterblockSeparatorPrototype{#1}}%
395 \define@key{markdownRendererPrototypes}{lineBreak}{%
396   \renewcommand\markdownRendererLineBreakPrototype{#1}}%
397 \define@key{markdownRendererPrototypes}{ellipsis}{%
398   \renewcommand\markdownRendererEllipsisPrototype{#1}}%
399 \define@key{markdownRendererPrototypes}{nbsp}{%
400   \renewcommand\markdownRendererNbspPrototype{#1}}%
401 \define@key{markdownRendererPrototypes}{leftBrace}{%
402   \renewcommand\markdownRendererLeftBracePrototype{#1}}%
403 \define@key{markdownRendererPrototypes}{rightBrace}{%
404   \renewcommand\markdownRendererRightBracePrototype{#1}}%
405 \define@key{markdownRendererPrototypes}{dollarSign}{%
406   \renewcommand\markdownRendererDollarSignPrototype{#1}}%
407 \define@key{markdownRendererPrototypes}{percentSign}{%

```

```

408 \renewcommand\markdownRendererPercentSignPrototype{#1}}%
409 \define@key{markdownRendererPrototypes}{ampersand}{%
410 \renewcommand\markdownRendererAmpersandPrototype{#1}}%
411 \define@key{markdownRendererPrototypes}{underscore}{%
412 \renewcommand\markdownRendererUnderscorePrototype{#1}}%
413 \define@key{markdownRendererPrototypes}{hash}{%
414 \renewcommand\markdownRendererHashPrototype{#1}}%
415 \define@key{markdownRendererPrototypes}{circumflex}{%
416 \renewcommand\markdownRendererCircumflexPrototype{#1}}%
417 \define@key{markdownRendererPrototypes}{backslash}{%
418 \renewcommand\markdownRendererBackslashPrototype{#1}}%
419 \define@key{markdownRendererPrototypes}{tilde}{%
420 \renewcommand\markdownRendererTildePrototype{#1}}%
421 \define@key{markdownRendererPrototypes}{pipe}{%
422 \renewcommand\markdownRendererPipePrototype{#1}}%
423 \define@key{markdownRendererPrototypes}{codeSpan}{%
424 \renewcommand\markdownRendererCodeSpanPrototype[1]{#1}}%
425 \define@key{markdownRendererPrototypes}{link}{%
426 \renewcommand\markdownRendererLinkPrototype[4]{#1}}%
427 \define@key{markdownRendererPrototypes}{image}{%
428 \renewcommand\markdownRendererImagePrototype[4]{#1}}%
429 \define@key{markdownRendererPrototypes}{ulBegin}{%
430 \renewcommand\markdownRendererUlBeginPrototype{#1}}%
431 \define@key{markdownRendererPrototypes}{ulBeginTight}{%
432 \renewcommand\markdownRendererUlBeginTightPrototype{#1}}%
433 \define@key{markdownRendererPrototypes}{ulItem}{%
434 \renewcommand\markdownRendererUlItemPrototype{#1}}%
435 \define@key{markdownRendererPrototypes}{ulItemEnd}{%
436 \renewcommand\markdownRendererUlItemEndPrototype{#1}}%
437 \define@key{markdownRendererPrototypes}{ulEnd}{%
438 \renewcommand\markdownRendererUlEndPrototype{#1}}%
439 \define@key{markdownRendererPrototypes}{ulEndTight}{%
440 \renewcommand\markdownRendererUlEndTightPrototype{#1}}%
441 \define@key{markdownRendererPrototypes}{olBegin}{%
442 \renewcommand\markdownRendererOlBeginPrototype{#1}}%
443 \define@key{markdownRendererPrototypes}{olBeginTight}{%
444 \renewcommand\markdownRendererOlBeginTightPrototype{#1}}%
445 \define@key{markdownRendererPrototypes}{olItem}{%
446 \renewcommand\markdownRendererOlItemPrototype{#1}}%
447 \define@key{markdownRendererPrototypes}{olItemWithNumber}{%
448 \renewcommand\markdownRendererOlItemWithNumberPrototype[1]{#1}}%
449 \define@key{markdownRendererPrototypes}{olItemEnd}{%
450 \renewcommand\markdownRendererOlItemEndPrototype{#1}}%
451 \define@key{markdownRendererPrototypes}{olEnd}{%
452 \renewcommand\markdownRendererOlEndPrototype{#1}}%
453 \define@key{markdownRendererPrototypes}{olEndTight}{%
454 \renewcommand\markdownRendererOlEndTightPrototype{#1}}%

```

```

455 \define@key{markdownRendererPrototypes}{dlBegin}{%
456   \renewcommand\markdownRendererDlBeginPrototype{#1}}%
457 \define@key{markdownRendererPrototypes}{dlBeginTight}{%
458   \renewcommand\markdownRendererDlBeginTightPrototype{#1}}%
459 \define@key{markdownRendererPrototypes}{dlItem}{%
460   \renewcommand\markdownRendererDlItemPrototype[1]{#1}}%
461 \define@key{markdownRendererPrototypes}{dlItemEnd}{%
462   \renewcommand\markdownRendererDlItemEndPrototype{#1}}%
463 \define@key{markdownRendererPrototypes}{dlDefinitionBegin}{%
464   \renewcommand\markdownRendererDlDefinitionBeginPrototype{#1}}%
465 \define@key{markdownRendererPrototypes}{dlDefinitionEnd}{%
466   \renewcommand\markdownRendererDlDefinitionEndPrototype{#1}}%
467 \define@key{markdownRendererPrototypes}{dlEnd}{%
468   \renewcommand\markdownRendererDlEndPrototype{#1}}%
469 \define@key{markdownRendererPrototypes}{dlEndTight}{%
470   \renewcommand\markdownRendererDlEndTightPrototype{#1}}%
471 \define@key{markdownRendererPrototypes}{emphasis}{%
472   \renewcommand\markdownRendererEmphasisPrototype[1]{#1}}%
473 \define@key{markdownRendererPrototypes}{strongEmphasis}{%
474   \renewcommand\markdownRendererStrongEmphasisPrototype[1]{#1}}%
475 \define@key{markdownRendererPrototypes}{blockquoteBegin}{%
476   \renewcommand\markdownRendererBlockQuoteBeginPrototype{#1}}%
477 \define@key{markdownRendererPrototypes}{blockquoteEnd}{%
478   \renewcommand\markdownRendererBlockQuoteEndPrototype{#1}}%
479 \define@key{markdownRendererPrototypes}{inputVerbatim}{%
480   \renewcommand\markdownRendererInputVerbatimPrototype[1]{#1}}%
481 \define@key{markdownRendererPrototypes}{inputFencedCode}{%
482   \renewcommand\markdownRendererInputFencedCodePrototype[2]{#1}}%
483 \define@key{markdownRendererPrototypes}{headingOne}{%
484   \renewcommand\markdownRendererHeadingOnePrototype[1]{#1}}%
485 \define@key{markdownRendererPrototypes}{headingTwo}{%
486   \renewcommand\markdownRendererHeadingTwoPrototype[1]{#1}}%
487 \define@key{markdownRendererPrototypes}{headingThree}{%
488   \renewcommand\markdownRendererHeadingThreePrototype[1]{#1}}%
489 \define@key{markdownRendererPrototypes}{headingFour}{%
490   \renewcommand\markdownRendererHeadingFourPrototype[1]{#1}}%
491 \define@key{markdownRendererPrototypes}{headingFive}{%
492   \renewcommand\markdownRendererHeadingFivePrototype[1]{#1}}%
493 \define@key{markdownRendererPrototypes}{headingSix}{%
494   \renewcommand\markdownRendererHeadingSixPrototype[1]{#1}}%
495 \define@key{markdownRendererPrototypes}{horizontalRule}{%
496   \renewcommand\markdownRendererHorizontalRulePrototype{#1}}%
497 \define@key{markdownRendererPrototypes}{footnote}{%
498   \renewcommand\markdownRendererFootnotePrototype[1]{#1}}%
499 \define@key{markdownRendererPrototypes}{cite}{%
500   \renewcommand\markdownRendererCitePrototype[1]{#1}}%
501 \define@key{markdownRendererPrototypes}{textCite}{%

```

```
502 \renewcommand\markdownRendererTextCitePrototype[1]{#1}}%
```

The following example \TeX 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 Con \TeX t Interface

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

```
503 \writestatus{loading}{Con $\text{\TeX}$ t User Module / markdown}%
504 \unprotect
```

The Con \TeX t interface is implemented by the `t-markdown.tex` Con \TeX t 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.

```
505 \let\startmarkdown\relax
506 \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 Con \TeX t 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).

```

507 local upper, gsub, format, length =
508   string.upper, string.gsub, string.format, string.len
509 local concat = table.concat
510 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
511   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
512   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 Lua code. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```

513 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.

```

514 function util.err(msg, exit_code)
515   io.stderr:write("markdown.lua: " .. msg .. "\n")
516   os.exit(exit_code or 1)
517 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.

```

518 function util.cache(dir, string, salt, transform, suffix)
519   local digest = md5.sumhexa(string .. (salt or ""))
520   local name = util.pathname(dir, digest .. suffix)
521   local file = io.open(name, "r")
522   if file == nil then -- If no cache entry exists, then create a new one.
523     local file = assert(io.open(name, "w"))
524     local result = string
525     if transform ~= nil then
526       result = transform(result)
527     end
528     assert(file:write(result))
529     assert(file:close())
530   end
531   return name
532 end

```

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

```

533 function util.table_copy(t)
534   local u = { }
535   for k, v in pairs(t) do u[k] = v end
536   return setmetatable(u, getmetatable(t))
537 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 [3, Chapter 21].

```

538 function util.expand_tabs_in_line(s, tabstop)
539   local tab = tabstop or 4
540   local corr = 0
541   return (s:gsub("\t", function(p)
542     local sp = tab - (p - 1 + corr) % tab
543     corr = corr - 1 + sp
544     return string.rep(" ", sp)
545   end))
546 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.

```

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

```

```

550     f(t)
551 elseif typ == "table" then
552     local i = 1
553     local n
554     n = t[i]
555     while n do
556         util.walk(n, f)
557         i = i + 1
558         n = t[i]
559     end
560 elseif typ == "function" then
561     local ok, val = pcall(t)
562     if ok then
563         util.walk(val, f)
564     end
565 else
566     f(tostring(t))
567 end
568 end

```

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

```

569 function util.flatten(ary)
570     local new = {}
571     for _,v in ipairs(ary) do
572         if type(v) == "table" then
573             for _,w in ipairs(util.flatten(v)) do
574                 new[#new + 1] = w
575             end
576         else
577             new[#new + 1] = v
578         end
579     end
580     return new
581 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.

```

582 function util.rope_to_string(rope)
583     local buffer = {}
584     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
585     return table.concat(buffer)
586 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.

```

587 function util.rope_last(rope)
588     if #rope == 0 then

```

```

589     return nil
590 else
591     local l = rope[#rope]
592     if type(l) == "table" then
593         return util.rope_last(l)
594     else
595         return l
596     end
597 end
598 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 $1 \leq i \leq \#ary$.

```

599 function util.intersperse(ary, x)
600     local new = {}
601     local l = #ary
602     for i,v in ipairs(ary) do
603         local n = #new
604         new[n + 1] = v
605         if i ~= l then
606             new[n + 2] = x
607         end
608     end
609     return new
610 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 $1 \leq i \leq \#ary$.

```

611 function util.map(ary, f)
612     local new = {}
613     for i,v in ipairs(ary) do
614         new[i] = f(v)
615     end
616     return new
617 end

```

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

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

```

618 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

619     local char_escapes_list = ""
620     for i,_ in pairs(char_escapes) do
621         char_escapes_list = char_escapes_list .. i

```

```
622 end
```

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

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

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

$$\sum_{(k,v) \in \text{string_escapes}} P(k) / v + \text{escapable}$$

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

```
624 if string_escapes then
625   for k,v in pairs(string_escapes) do
626     escapable = P(k) / v + escapable
627   end
628 end
```

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

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

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

```
630 return function(s)
631   return lpeg.match(escape_string, s)
632 end
633 end
```

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

```
634 function util.pathname(dir, file)
635   if #dir == 0 then
636     return file
637   else
638     return dir .. "/" .. file
639   end
640 end
```

3.1.2 Plain T_EX Writer

This section documents the `writer` object, which implements the routines for producing the T_EX output. The object is an amalgamate of the generic, T_EX, \LaTeX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark 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.

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

The `writer.new` method creates and returns a new T_EX 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`.

```
642 function M.writer.new(options)
643     local self = {}
644     options = options or {}
```

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

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

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

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

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

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

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

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

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

```
650     function self.plain(s)
651         return s
652     end
```

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

```
653     function self.paragraph(s)
654         return s
655     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.

```
656     function self.pack(name)
657         return [[\input"']] .. name .. [[\"relax]]
658     end
```

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

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

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

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

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

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

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

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

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

```
663 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.

```
664 local escaped_chars = {
665   ["{"] = "\\markdownRendererLeftBrace{"",
666   ["}"] = "\\markdownRendererRightBrace{"",
667   ["$"] = "\\markdownRendererDollarSign{"",
668   ["%"] = "\\markdownRendererPercentSign{"",
669   ["&"] = "\\markdownRendererAmpersand{"",
670   ["_"] = "\\markdownRendererUnderscore{"",
671   ["#"] = "\\markdownRendererHash{"",
672   ["^"] = "\\markdownRendererCircumflex{"",
673   ["\\"] = "\\markdownRendererBackslash{"",
674   ["~"] = "\\markdownRendererTilde{"",
675   ["|"] = "\\markdownRendererPipe{"", }
676 local escaped_minimal_chars = {
677   ["{"] = "\\markdownRendererLeftBrace{"",
678   ["}"] = "\\markdownRendererRightBrace{"",
679   ["%"] = "\\markdownRendererPercentSign{"",
680   ["\\"] = "\\markdownRendererBackslash{"", }
681 local escaped_minimal_strings = {
682   ["^^"] = "\\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`.

```
683 local escape = util.escaper(escaped_chars)
684 local escape_minimal = util.escaper(escaped_minimal_chars,
685   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.

```

686   if options.hybrid then
687       self.string = function(s) return s end
688       self.uri = function(u) return u end
689   else
690       self.string = escape
691       self.uri = escape_minimal
692   end

```

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

```

693   function self.code(s)
694       return {"\\markdownRendererCodeSpan{",escape(s),""}
695   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.

```

696   function self.link(lab,src,tit)
697       return {"\\markdownRendererLink{",lab,""},
698               "{",self.string(src),""},
699               "{",self.uri(src),""},
700               "{",self.string(tit or ""),""}
701   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.

```

702   function self.image(lab,src,tit)
703       return {"\\markdownRendererImage{",lab,""},
704               "{",self.string(src),""},
705               "{",self.uri(src),""},
706               "{",self.string(tit or ""),""}
707   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.

```

708   local function ulitem(s)
709       return {"\\markdownRendererULItem ",s,
710               "\\markdownRendererULItemEnd "}
711   end
712
713   function self.bulletlist(items,tight)
714       local buffer = {}
715       for _,item in ipairs(items) do
716           buffer[#buffer + 1] = ulitem(item)
717       end
718       local contents = util.intersperse(buffer,"\n")

```

```

719     if tight and options.tightLists then
720         return {"\\markdownRendererUlBeginTight\n",contents,
721             "\n\\markdownRendererUlEndTight "}
722     else
723         return {"\\markdownRendererUlBegin\n",contents,
724             "\n\\markdownRendererUlEnd "}
725     end
726 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.

```

727 local function olitem(s,num)
728     if num ~= nil then
729         return {"\\markdownRendererOlItemWithNumber{" ,num,"}",s,
730             "\\markdownRendererOlItemEnd "}
731     else
732         return {"\\markdownRendererOlItem ",s,
733             "\\markdownRendererOlItemEnd "}
734     end
735 end
736
737 function self.orderedlist(items,tight,startnum)
738     local buffer = {}
739     local num = startnum
740     for _,item in ipairs(items) do
741         buffer[#buffer + 1] = olitem(item,num)
742         if num ~= nil then
743             num = num + 1
744         end
745     end
746     local contents = util.intersperse(buffer,"\n")
747     if tight and options.tightLists then
748         return {"\\markdownRendererOlBeginTight\n",contents,
749             "\n\\markdownRendererOlEndTight "}
750     else
751         return {"\\markdownRendererOlBegin\n",contents,
752             "\n\\markdownRendererOlEnd "}
753     end
754 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.

```

755 local function dliem(term, defs)

```



```

756     local retVal = {"\\markdownRendererDlItem{" ,term,"}"}
757     for _, def in ipairs(defs) do
758         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin " ,def,
759                               "\\markdownRendererDlDefinitionEnd "}
760     end
761     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
762     return retVal
763 end
764
765 function self.definitionlist(items,tight)
766     local buffer = {}
767     for _,item in ipairs(items) do
768         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
769     end
770     if tight and options.tightLists then
771         return {"\\markdownRendererDlBeginTight\\n", buffer,
772               "\\n\\markdownRendererDlEndTight"}
773     else
774         return {"\\markdownRendererDlBegin\\n", buffer,
775               "\\n\\markdownRendererDlEnd"}
776     end
777 end

```

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

```

778 function self.emphasis(s)
779     return {"\\markdownRendererEmphasis{" ,s,"}"}
780 end

```

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

```

781 function self.strong(s)
782     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
783 end

```

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

```

784 function self.blockquote(s)
785     return {"\\markdownRendererBlockQuoteBegin\\n",s,
786           "\\n\\markdownRendererBlockQuoteEnd "}
787 end

```

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

```

788 function self.verbatim(s)
789     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
790     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
791 end

```

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

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

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

```
796 function self.heading(s,level)
797   local cmd
798   if level == 1 then
799     cmd = "\\markdownRendererHeadingOne"
800   elseif level == 2 then
801     cmd = "\\markdownRendererHeadingTwo"
802   elseif level == 3 then
803     cmd = "\\markdownRendererHeadingThree"
804   elseif level == 4 then
805     cmd = "\\markdownRendererHeadingFour"
806   elseif level == 5 then
807     cmd = "\\markdownRendererHeadingFive"
808   elseif level == 6 then
809     cmd = "\\markdownRendererHeadingSix"
810   else
811     cmd = ""
812   end
813   return {cmd,"{" ,s,"}"}
814 end
```

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

```
815 function self.note(s)
816   return {"\\markdownRendererFootnote{" ,s,"}"}
817 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.

```

818 function self.citations(text_cites, cites)
819     local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
820         "{", #cites, "}"}
821     for _,cite in ipairs(cites) do
822         buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
823             cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
824     end
825     return buffer
826 end
827
828 return self
829 end

```

3.1.3 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`.

```

830 M.reader = {}
831 function M.reader.new(writer, options)
832     local self = {}
833     options = options or {}

```

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

```

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

```

3.1.3.1 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.

```

836     local function normalize_tag(tag)

```

```

837     return unicode.utf8.lower(
838         gsub(util.ropetostring(tag), "[\n\r\t]+", " "))
839 end

```

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

```

840 local expandtabs
841 if options.preserveTabs then
842     expandtabs = function(s) return s end
843 else
844     expandtabs = function(s)
845         if s:find("\t") then
846             return s:gsub("[^\n]*", util.expand_tabs_in_line)
847         else
848             return s
849         end
850     end
851 end

```

3.1.3.2 Top Level Parsing Functions

```

852 local syntax
853 local blocks_toplevel
854 local blocks
855 local inlines, inlines_no_link, inlines_nbsp
856
857 local function create_parser(name, grammar)
858     return function(str)
859         local res = lpeg.match(grammar(), str)
860         if res == nil then
861             error(format("%s failed on:\n%s", name, str:sub(1,20)))
862         else
863             return res
864         end
865     end
866 end
867
868 local parse_blocks = create_parser("parse_blocks",
869     function() return blocks end)
870 local parse_blocks_toplevel = create_parser("parse_blocks_toplevel",
871     function() return blocks_toplevel end)
872 local parse_inlines = create_parser("parse_inlines",
873     function() return inlines end)
874 local parse_inlines_no_link = create_parser("parse_inlines_no_link",
875     function() return inlines_no_link end)
876 local parse_inlines_nbsp = create_parser("parse_inlines_nbsp",
877     function() return inlines_nbsp end)

```

3.1.3.3 Generic PEG Patterns

```
878 local percent = P("%")
879 local at = P("@")
880 local comma = P(",")
881 local asterisk = P("*")
882 local dash = P("-")
883 local plus = P("+")
884 local underscore = P("_")
885 local period = P(".")
886 local hash = P("#")
887 local ampersand = P("&")
888 local backtick = P("`")
889 local less = P("<")
890 local more = P(">")
891 local space = P(" ")
892 local squote = P("'")
893 local dquote = P('"')
894 local lparent = P("(")
895 local rparent = P(")")
896 local lbracket = P("[")
897 local rbracket = P("]")
898 local circumflex = P("^")
899 local slash = P("/")
900 local equal = P("=")
901 local colon = P(":")
902 local semicolon = P(";")
903 local exclamation = P("!")
904 local tilde = P("~")
905
906 local digit = R("09")
907 local hexdigit = R("09", "af", "AF")
908 local letter = R("AZ", "az")
909 local alphanumeric = R("AZ", "az", "09")
910 local keyword = letter * alphanumeric^0
911 local internal_punctuation = S(".;,.$%&-+?<>~/")
912
913 local doubleasterisks = P("**")
914 local doubleunderscores = P("__")
915 local fourspaces = P("    ")
916
917 local any = P(1)
918 local fail = any - 1
919 local always = P("")
920
921 local escapable = S("\\'*_{}[]() +_ . !<>#--:~@;")
922
923 local anyescaped = P("\\") / " " * escapable
```

```

924                                     + any
925
926 local tab                           = P("\t")
927 local spacechar                     = S("\t ")
928 local spacing                       = S(" \n\r\t")
929 local newline                       = P("\n")
930 local nonspacechar                  = any - spacing
931 local tightblocksep                 = P("\001")
932
933 local specialchar                   = S("*_ '&[]<!\.\@-")
934
935 local normalchar                    = any -
936                                     (specialchar + spacing + tightblocksep)
937 local optionalspace                  = spacechar^0
938 local eof                           = - any
939 local nonindentspace                 = space^3 * - spacechar
940 local indent                        = space^3 * tab
941                                     + fourspace / ""
942 local linechar                       = P(1 - newline)
943
944 local blankline                      = optionalspace * newline / "\n"
945 local blanklines                     = blankline^0
946 local skipblanklines                 = (optionalspace * newline)^0
947 local indentedline                  = indent / "" * C(linechar^1 * newline^-1)
948 local optionallyindentedline         = indent^-1 / "" * C(linechar^1 * newline^-1)
949 local sp                             = spacing^0
950 local spnl                           = optionalspace * (newline * optionalspace)^-1
951 local line                           = linechar^0 * newline
952                                     + linechar^1 * eof
953 local nonemptyline                  = line - blankline
954
955 local chunk = line * (optionallyindentedline - blankline)^0
956
957 -- block followed by 0 or more optionally
958 -- indented blocks with first line indented.
959 local function indented_blocks(bl)
960     return Cs( bl
961                 * (blankline^1 * indent * -blankline * bl)^0
962                 * (blankline^1 + eof) )
963 end

```

3.1.3.4 List PEG Patterns

```

964 local bulletchar = C(plus + asterisk + dash)
965
966 local bullet      = ( bulletchar * #spacing * (tab + space^3)
967                      + space * bulletchar * #spacing * (tab + space^2)

```

```

968             + space * space * bulletchar * #spacing * (tab + space^-1)
969             + space * space * space * bulletchar * #spacing
970             ) * -bulletchar
971
972     if options.hashEnumerators then
973         dig = digit + hash
974     else
975         dig = digit
976     end
977
978     local enumerator = C(dig^3 * period) * #spacing
979                     + C(dig^2 * period) * #spacing * (tab + space^1)
980                     + C(dig * period) * #spacing * (tab + space^-2)
981                     + space * C(dig^2 * period) * #spacing
982                     + space * C(dig * period) * #spacing * (tab + space^-1)
983                     + space * space * C(dig^1 * period) * #spacing

```

3.1.3.5 Code Span PEG Patterns

```

984     local openticks    = Cg(backtick^1, "ticks")
985
986     local function captures_equal_length(s,i,a,b)
987         return #a == #b and i
988     end
989
990     local closeticks    = space^-1 *
991                     Cmt(C(backtick^1) * Cb("ticks"), captures_equal_length)
992
993     local intickschar = (any - S(" \n\r'"))
994                     + (newline * -blankline)
995                     + (space - closeticks)
996                     + (backtick^1 - closeticks)
997
998     local inticks      = openticks * space^-1 * C(intickschar^0) * closeticks
999 % \paragraph{Fenced Code \acro{peg} Patterns}
1000 % \begin{macrocode}
1001     local function captures_geq_length(s,i,a,b)
1002         return #a >= #b and i
1003     end
1004
1005     local infostring    = (linechar - (backtick + space^1 * (newline + eof)))^0
1006
1007     local fenceindent
1008     local function fencehead(char)
1009         return
1010             C(nonindentSPACE) / function(s) fenceindent = #s end
1011             * Cg(char^3, "fencelength")
1012             * optionalSPACE * C(infostring) * optionalSPACE

```

```

1012             * (newline + eof)
1013 end
1014
1015 local function fencetail(char)
1016     return          nonindentSPACE
1017             * Cmt(C(char^3) * Cb("fencelength"),
1018                 captures_geq_length)
1019             * optionalSPACE * (newline + eof)
1020             + eof
1021 end
1022
1023 local function fencedline(char)
1024     return          C(line - fencetail(char))
1025             / function(s)
1026                 return s:gsub("^" .. string.rep(" ?",
1027                     fenceindent), "")
1028             end
1029 end

```

3.1.3.6 Tag PEG Patterns

```

1030 local leader      = space^-3
1031
1032 -- in balanced brackets, parentheses, quotes:
1033 local bracketed    = P{ lbracket
1034             * ((anyescaped - (lbracket + rbracket
1035                 + blankline^2)) + V(1))^0
1036             * rbracket }
1037
1038 local inparens     = P{ lparent
1039             * ((anyescaped - (lparent + rparent
1040                 + blankline^2)) + V(1))^0
1041             * rparent }
1042
1043 local squoted       = P{ squote * alphanumeric
1044             * ((anyescaped - (squote + blankline^2))
1045                 + V(1))^0
1046             * squote }
1047
1048 local dquoted       = P{ dquote * alphanumeric
1049             * ((anyescaped - (dquote + blankline^2))
1050                 + V(1))^0
1051             * dquote }
1052
1053 -- bracketed 'tag' for markdown links, allowing nested brackets:
1054 local tag           = lbracket
1055             * Cs((alphanumeric^1

```



```

1056             + bracketed
1057             + inticks
1058             + (anyescaped - (rbracket + blankline^2)))^0)
1059         * rbracket
1060
1061     -- url for markdown links, allowing balanced parentheses:
1062     local url      = less * Cs((anyescaped-more)^0) * more
1063                   + Cs((inparens + (anyescaped-spacing-rparent))^1)
1064
1065     -- quoted text possibly with nested quotes:
1066     local title_s   = quote * Cs(((anyescaped-quote) + quoted)^0) *
1067                       quote
1068
1069     local title_d   = dquote * Cs(((anyescaped-dquote) + dquoted)^0) *
1070                       dquote
1071
1072     local title_p   = lparen
1073                   * Cs((inparens + (anyescaped-rparent))^0)
1074                   * rparen
1075
1076     local title     = title_d + title_s + title_p
1077
1078     local optionaltitle = spnl * title * spacechar^0
1079                   + Cc("")

```

3.1.3.7 Citation PEG Patterns

```

1080     local citation_name = Cs(dash^-1) * at
1081                       * Cs(alphanumeric
1082                           * (alphanumeric + internal_punctuation
1083                             - comma - semicolon)^0)
1084
1085     local citation_body_prenote
1086           = Cs((alphanumeric^1
1087               + bracketed
1088               + inticks
1089               + (anyescaped
1090                 - (rbracket + blankline^2))
1091               - (spnl * dash^-1 * at))^0)
1092
1093     local citation_body_postnote
1094           = Cs((alphanumeric^1
1095               + bracketed
1096               + inticks
1097               + (anyescaped
1098                 - (rbracket + semicolon + blankline^2))
1099               - (spnl * rbracket))^0)

```

```

1100
1101 local citation_body_chunk
1102         = citation_body_prenote
1103         * spnl * citation_name
1104         * (comma * spnl)^-1
1105         * citation_body_postnote
1106
1107 local citation_body = citation_body_chunk
1108         * (semicolon * spnl * citation_body_chunk)^0
1109
1110 local citation_headless_body_postnote
1111         = Cs((alphanumeric^1
1112             + bracketed
1113             + inticks
1114             + (anyescaped
1115                 - (rbracket + at + semicolon + blankline^2))
1116             - (spnl * rbracket))^0)
1117
1118 local citation_headless_body
1119         = citation_headless_body_postnote
1120         * (sp * semicolon * spnl * citation_body_chunk)^0

```

3.1.3.8 Footnote PEG Patterns

```

1121 local rawnotes = {}
1122
1123 local function strip_first_char(s)
1124     return s:sub(2)
1125 end
1126
1127 -- like indirect_link
1128 local function lookup_note(ref)
1129     return function()
1130         local found = rawnotes[normalize_tag(ref)]
1131         if found then
1132             return writer.note(parse_blocks_toplevel(found))
1133         else
1134             return {"[", parse_inlines("^" .. ref), "]" }
1135         end
1136     end
1137 end
1138
1139 local function register_note(ref, rawnote)
1140     rawnotes[normalize_tag(ref)] = rawnote
1141     return ""
1142 end
1143

```

```

1144 local RawNoteRef = #(lbracket * circumflex) * tag / strip_first_char
1145
1146 local NoteRef      = RawNoteRef / lookup_note
1147
1148 local NoteBlock
1149
1150 if options.footnotes then
1151     NoteBlock = leader * RawNoteRef * colon * spnl *
1152                 indented_blocks(chunk) / register_note
1153 else
1154     NoteBlock = fail
1155 end

```

3.1.3.9 Link and Image PEG Patterns

```

1156 -- List of references defined in the document
1157 local references
1158
1159 -- add a reference to the list
1160 local function register_link(tag,url,title)
1161     references[normalize_tag(tag)] = { url = url, title = title }
1162     return ""
1163 end
1164
1165 -- parse a reference definition:  [foo]: /bar "title"
1166 local define_reference_parser =
1167     leader * tag * colon * spacechar^0 * url * optionaltitle * blankline^1
1168
1169 -- lookup link reference and return either
1170 -- the link or nil and fallback text.
1171 local function lookup_reference(label,sps,tag)
1172     local tagpart
1173     if not tag then
1174         tag = label
1175         tagpart = ""
1176     elseif tag == "" then
1177         tag = label
1178         tagpart = "[]"
1179     else
1180         tagpart = {"[", parse_inlines(tag), "]" }
1181     end
1182     if sps then
1183         tagpart = {sps, tagpart}
1184     end
1185     local r = references[normalize_tag(tag)]
1186     if r then
1187         return r

```

```

1188     else
1189         return nil, {"[", parse_inlines(label), "]", tagpart}
1190     end
1191 end
1192
1193 -- lookup link reference and return a link, if the reference is found,
1194 -- or a bracketed label otherwise.
1195 local function indirect_link(label,sps,tag)
1196     return function()
1197         local r,fallback = lookup_reference(label,sps,tag)
1198         if r then
1199             return writer.link(parse_inlines_no_link(label), r.url, r.title)
1200         else
1201             return fallback
1202         end
1203     end
1204 end
1205
1206 -- lookup image reference and return an image, if the reference is found,
1207 -- or a bracketed label otherwise.
1208 local function indirect_image(label,sps,tag)
1209     return function()
1210         local r,fallback = lookup_reference(label,sps,tag)
1211         if r then
1212             return writer.image(writer.string(label), r.url, r.title)
1213         else
1214             return {"!", fallback}
1215         end
1216     end
1217 end

```

3.1.3.10 Spacing PEG Patterns

```

1218 local bqstart      = more
1219 local headerstart  = hash
1220                  + (line * (equal^1 + dash^1) * optionalspace * newline)
1221 local fencestart   = fencehead(backtick) + fencehead(tilde)
1222
1223 if options.blankBeforeBlockquote then
1224     bqstart = fail
1225 end
1226
1227 if options.blankBeforeHeading then
1228     headerstart = fail
1229 end
1230
1231 if not options.fencedCode or options.blankBeforeCodeFence then

```

```

1232     fencestart = fail
1233 end

```

3.1.3.11 String PEG Rules

```

1234 local Inline    = V("Inline")
1235
1236 local Str        = normalchar^1 / writer.string
1237
1238 local Symbol     = (specialchar - tightblocksep) / writer.string

```

3.1.3.12 Ellipsis PEG Rules

```

1239 local Ellipsis  = P("...") / writer.ellipsis
1240
1241 local Smart      = Ellipsis

```

3.1.3.13 Inline Code Block PEG Rules

```

1242 local Code      = inticks / writer.code

```

3.1.3.14 Spacing PEG Rules

```

1243 local Endline   = newline * -( -- newline, but not before...
1244                     blankline -- paragraph break
1245                     + tightblocksep -- nested list
1246                     + eof          -- end of document
1247                     + bqstart
1248                     + headerstart
1249                     + fencestart
1250                     ) * spacechar^0 / writer.space
1251
1252 local Space      = spacechar^2 * Endline / writer.linebreak
1253                 + spacechar^1 * Endline^-1 * eof / ""
1254                 + spacechar^1 * Endline^-1 * optionalspace / writer.space
1255
1256 local NonbreakingEndline
1257     = newline * -( -- newline, but not before...
1258                     blankline -- paragraph break
1259                     + tightblocksep -- nested list
1260                     + eof          -- end of document
1261                     + bqstart
1262                     + headerstart
1263                     + fencestart
1264                     ) * spacechar^0 / writer.nbsp
1265
1266 local NonbreakingSpace
1267     = spacechar^2 * Endline / writer.linebreak

```

```

1268             + spacechar^1 * Endline^-1 * eof / ""
1269             + spacechar^1 * Endline^-1 * optionalspace / writer.nbsp
1270
1271 -- parse many p between starter and ender
1272 local function between(p, starter, ender)
1273     local ender2 = B(nonspacechar) * ender
1274     return (starter * #nonspacechar * Ct(p * (p - ender2)^0) * ender2)
1275 end

```

3.1.3.15 Emphasis PEG Rules

```

1276 local Strong = ( between(Inline, doubleasterisks, doubleasterisks)
1277                 + between(Inline, doubleunderscores, doubleunderscores)
1278                 ) / writer.strong
1279
1280 local Emph    = ( between(Inline, asterisk, asterisk)
1281                 + between(Inline, underscore, underscore)
1282                 ) / writer.emphasis

```

3.1.3.16 Link PEG Rules

```

1283 local urlchar = anyescaped - newline - more
1284
1285 local AutoLinkUrl = less
1286                 * C(alphanumeric^1 * P(":/") * urlchar^1)
1287                 * more
1288                 / function(url)
1289                     return writer.link(writer.string(url), url)
1290                 end
1291
1292 local AutoLinkEmail = less
1293                 * C((alphanumeric + S("-._+"))^1 * P("@") * urlchar^1)
1294                 * more
1295                 / function(email)
1296                     return writer.link(writer.string(email),
1297                                         "mailto:".email)
1298                 end
1299
1300 local DirectLink = (tag / parse_inlines_no_link) -- no links inside links
1301                 * spnl
1302                 * lparent
1303                 * (url + Cc("")) -- link can be empty [foo]()
1304                 * optionaltitle
1305                 * rparent
1306                 / writer.link
1307
1308 local IndirectLink = tag * (C(spnl) * tag)^-1 / indirect_link
1309

```

```

1310 -- parse a link or image (direct or indirect)
1311 local Link      = DirectLink + IndirectLink

```

3.1.3.17 Image PEG Rules

```

1312 local DirectImage = exclamation
1313                      * (tag / parse_inlines)
1314                      * spnl
1315                      * lparent
1316                      * (url + Cc("")) -- link can be empty [foo]()
1317                      * optionaltitle
1318                      * rparent
1319                      / writer.image
1320
1321 local IndirectImage = exclamation * tag * (C(spnl) * tag)^-1 /
1322                      indirect_image
1323
1324 local Image      = DirectImage + IndirectImage

```

3.1.3.18 Miscellaneous Inline PEG Rules

```

1325 -- avoid parsing long strings of * or _ as emph/strong
1326 local U1OrStarLine = asterisk^4 + underscore^4 / writer.string
1327
1328 local EscapedChar  = S("\\") * C(escapable) / writer.string

```

3.1.3.19 Citations PEG Rules

```

1329 local function citations(text_cites, raw_cites)
1330     local function normalize(str)
1331         if str == "" then
1332             str = nil
1333         else
1334             str = (options.citationNbsps and parse_inlines_nbsp or
1335                 parse_inlines)(str)
1336         end
1337         return str
1338     end
1339
1340     local cites = {}
1341     for i = 1,#raw_cites,4 do
1342         cites[#cites+1] = {
1343             prenote = normalize(raw_cites[i]),
1344             suppress_author = raw_cites[i+1] == "-",
1345             name = writer.string(raw_cites[i+2]),
1346             postnote = normalize(raw_cites[i+3]),
1347         }
1348     end

```

```

1349     return writer.citations(text_cites, cites)
1350 end
1351
1352 local TextCitations = Ct(Cc("")
1353     * citation_name
1354     * ((spnl
1355         * lbracket
1356         * citation_headless_body
1357         * rbracket) + Cc(""))) /
1358     function(raw_cites)
1359         return citations(true, raw_cites)
1360     end
1361
1362 local ParenthesizedCitations
1363     = Ct(lbracket
1364         * citation_body
1365         * rbracket) /
1366         function(raw_cites)
1367             return citations(false, raw_cites)
1368         end
1369
1370 local Citations      = TextCitations + ParenthesizedCitations

```

3.1.3.20 Code Block PEG Rules

```

1371 local Block          = V("Block")
1372
1373 local Verbatim       = Cs( (blanklines
1374     * ((indentedline - blankline))^1)^1
1375     ) / expandtabs / writer.verbatim
1376
1377 local TildeFencedCode
1378     = fencehead(tilde)
1379     * Cs(fencedline(tilde)^0)
1380     * fencetail(tilde)
1381
1382 local BacktickFencedCode
1383     = fencehead(backtick)
1384     * Cs(fencedline(backtick)^0)
1385     * fencetail(backtick)
1386
1387 local FencedCode     = (TildeFencedCode + BacktickFencedCode)
1388     / function(infostring, code)
1389         return writer.fencedCode(
1390             writer.string(infostring),
1391             expandtabs(code))
1392     end

```


3.1.3.21 Blockquote PEG Patterns

```
1393 -- strip off leading > and indents, and run through blocks
1394 local Blockquote = Cs((
1395     ((leader * more * space^-1) / "" * linechar^0 * newline)^1
1396     * (-blankline * linechar^1 * newline)^0
1397     * (blankline^0 / "")
1398     )^1) / parse_blocks_toplevel / writer.blockquote
1399
1400 local function lineof(c)
1401     return (leader * (P(c) * optionalspace)^3 * (newline * blankline^1
1402     + newline^-1 * eof))
1403 end
```

3.1.3.22 Horizontal Rule PEG Rules

```
1404 local HorizontalRule = ( lineof(asterisk)
1405     + lineof(dash)
1406     + lineof(underscore)
1407     ) / writer.hrule
```

3.1.3.23 List PEG Rules

```
1408 local starter = bullet + enumerator
1409
1410 -- we use \001 as a separator between a tight list item and a
1411 -- nested list under it.
1412 local NestedList = Cs((optionallyindentedline - starter)^1)
1413     / function(a) return "\001"..a end
1414
1415 local ListBlockLine = optionallyindentedline
1416     - blankline - (indent^-1 * starter)
1417
1418 local ListBlock = line * ListBlockLine^0
1419
1420 local ListContinuationBlock = blanklines * (indent / "") * ListBlock
1421
1422 local function TightListItem(starter)
1423     return -HorizontalRule
1424     * (Cs(starter / "" * ListBlock * NestedList^-1) /
1425     parse_blocks)
1426     * -(blanklines * indent)
1427 end
1428
1429 local function LooseListItem(starter)
1430     return -HorizontalRule
1431     * Cs( starter / "" * ListBlock * Cc("\n")
1432     * (NestedList + ListContinuationBlock^0)
```

```

1433         * (blanklines / "\n\n")
1434     ) / parse_blocks
1435 end
1436
1437 local BulletList = ( Ct(TightListItem(bullet)^1)
1438     * Cc(true) * skipblanklines * -bullet
1439     + Ct(LooseListItem(bullet)^1)
1440     * Cc(false) * skipblanklines ) /
1441     writer.bulletlist
1442
1443 local function orderedlist(items,tight,startNumber)
1444     if options.startNumber then
1445         startNumber = tonumber(startNumber) or 1 -- fallback for '#'
1446     else
1447         startNumber = nil
1448     end
1449     return writer.orderedlist(items,tight,startNumber)
1450 end
1451
1452 local OrderedList = Cg(enumerator, "listtype") *
1453     ( Ct(TightListItem(Cb("listtype")) *
1454         TightListItem(enumerator)^0)
1455     * Cc(true) * skipblanklines * -enumerator
1456     + Ct(LooseListItem(Cb("listtype")) *
1457         LooseListItem(enumerator)^0)
1458     * Cc(false) * skipblanklines
1459     ) * Cb("listtype") / orderedlist
1460
1461 local defstartchar = S("~:")
1462 local defstart = ( defstartchar * #spacing * (tab + space^-3)
1463     + space * defstartchar * #spacing * (tab + space^-2)
1464     + space * space * defstartchar * #spacing *
1465     (tab + space^-1)
1466     + space * space * space * defstartchar * #spacing
1467     )
1468
1469 local dlchunk = Cs(line * (indentedline - blankline)^0)
1470
1471 local function definition_list_item(term, defs, tight)
1472     return { term = parse_inlines(term), definitions = defs }
1473 end
1474
1475 local DefinitionListItemLoose = C(line) * skipblanklines
1476     * Ct((defstart *
1477         indented_blocks(dlchunk) /
1478         parse_blocks_toplevel)^1)
1479     * Cc(false)

```

```

1480             / definition_list_item
1481
1482     local DefinitionListItemTight = C(line)
1483             * Ct((defstart * dlchunk /
1484                   parse_blocks)^1)
1485             * Cc(true)
1486             / definition_list_item
1487
1488     local DefinitionList = ( Ct(DefinitionListItemLoose^1) * Cc(false)
1489                             + Ct(DefinitionListItemTight^1)
1490                             * (skipblanklines *
1491                               -DefinitionListItemLoose * Cc(true))
1492     ) / writer.definitionlist

```

3.1.3.24 Blank Line PEG Rules

```

1493     local Reference      = define_reference_parser / register_link
1494     local Blank          = blankline / ""
1495                             + NoteBlock
1496                             + Reference
1497                             + (tightblocksep / "\n")

```

3.1.3.25 Paragraph PEG Rules

```

1498     local Paragraph      = nonindentospace * Ct(Inline^1) * newline
1499                             * ( blankline^1
1500                               + #hash
1501                               + #(leader * more * space^-1)
1502                               )
1503                             / writer.paragraph
1504
1505     local ToplevelParagraph
1506         = nonindentospace * Ct(Inline^1) * (newline
1507         * ( blankline^1
1508           + #hash
1509           + #(leader * more * space^-1)
1510           + eof
1511           )
1512         + eof )
1513         / writer.paragraph
1514
1515     local Plain           = nonindentospace * Ct(Inline^1) / writer.plain

```

3.1.3.26 Heading PEG Rules

```

1516     -- parse Atx heading start and return level
1517     local HeadingStart = #hash * C(hash^-6) * -hash / length
1518

```

```

1519 -- parse setext header ending and return level
1520 local HeadingLevel = equal^1 * Cc(1) + dash^1 * Cc(2)
1521
1522 local function strip_atx_end(s)
1523     return s:gsub("#%s]*\n$", "")
1524 end
1525
1526 -- parse atx header
1527 local AtxHeading = Cg(HeadingStart, "level")
1528     * optionalspace
1529     * (C(line) / strip_atx_end / parse_inlines)
1530     * Cb("level")
1531     / writer.heading
1532
1533 -- parse setext header
1534 local SetextHeading = #(line * S("--"))
1535     * Ct(line / parse_inlines)
1536     * HeadingLevel
1537     * optionalspace * newline
1538     / writer.heading
1539
1540 local Heading = AtxHeading + SetextHeading

```

3.1.3.27 Top Level PEG Specification

```

1541 syntax =
1542     { "Blocks",
1543
1544         Blocks                = Blank^0 *
1545                               Block^-1 *
1546                               (Blank^0 / function()
1547                                   return writer.interblocksep
1548                                   end * Block)^0 *
1549                               Blank^0 *
1550                               eof,
1551
1552         Blank                 = Blank,
1553
1554         Block                  = V("Blockquote")
1555                               + V("Verbatim")
1556                               + V("FencedCode")
1557                               + V("HorizontalRule")
1558                               + V("BulletList")
1559                               + V("OrderedList")
1560                               + V("Heading")
1561                               + V("DefinitionList")
1562                               + V("Paragraph")

```

1563		+ V("Plain"),
1564		
1565	Blockquote	= Blockquote,
1566	Verbatim	= Verbatim,
1567	FencedCode	= FencedCode,
1568	HorizontalRule	= HorizontalRule,
1569	BulletList	= BulletList,
1570	OrderedList	= OrderedList,
1571	Heading	= Heading,
1572	DefinitionList	= DefinitionList,
1573	DisplayHtml	= DisplayHtml,
1574	Paragraph	= Paragraph,
1575	Plain	= Plain,
1576		
1577	Inline	= V("Str")
1578		+ V("Space")
1579		+ V("Endline")
1580		+ V("U1OrStarLine")
1581		+ V("Strong")
1582		+ V("Emph")
1583		+ V("NoteRef")
1584		+ V("Citations")
1585		+ V("Link")
1586		+ V("Image")
1587		+ V("Code")
1588		+ V("AutoLinkUrl")
1589		+ V("AutoLinkEmail")
1590		+ V("EscapedChar")
1591		+ V("Smart")
1592		+ V("Symbol"),
1593		
1594	Str	= Str,
1595	Space	= Space,
1596	Endline	= Endline,
1597	U1OrStarLine	= U1OrStarLine,
1598	Strong	= Strong,
1599	Emph	= Emph,
1600	NoteRef	= NoteRef,
1601	Citations	= Citations,
1602	Link	= Link,
1603	Image	= Image,
1604	Code	= Code,
1605	AutoLinkUrl	= AutoLinkUrl,
1606	AutoLinkEmail	= AutoLinkEmail,
1607	InlineHtml	= InlineHtml,
1608	HtmlEntity	= HtmlEntity,
1609	EscapedChar	= EscapedChar,

```

1610         Smart                = Smart,
1611         Symbol                = Symbol,
1612     }
1613
1614     if not options.definitionLists then
1615         syntax.DefinitionList = fail
1616     end
1617
1618     if not options.fencedCode then
1619         syntax.FencedCode = fail
1620     end
1621
1622     if not options.citations then
1623         syntax.Citations = fail
1624     end
1625
1626     if not options.footnotes then
1627         syntax.NoteRef = fail
1628     end
1629
1630     if not options.smartEllipses then
1631         syntax.Smart = fail
1632     end
1633
1634     local blocks_toplevel_t = util.table_copy(syntax)
1635     blocks_toplevel_t.Paragraph = ToplevelParagraph
1636     blocks_toplevel = Ct(blocks_toplevel_t)
1637
1638     blocks = Ct(syntax)
1639
1640     local inlines_t = util.table_copy(syntax)
1641     inlines_t[1] = "Inlines"
1642     inlines_t.Inlines = Inline^0 * (spacing^0 * eof / "")
1643     inlines = Ct(inlines_t)
1644
1645     local inlines_no_link_t = util.table_copy(inlines_t)
1646     inlines_no_link_t.Link = fail
1647     inlines_no_link = Ct(inlines_no_link_t)
1648
1649     local inlines_nbsp_t = util.table_copy(inlines_t)
1650     inlines_nbsp_t.Endline = NonbreakingEndline
1651     inlines_nbsp_t.Space = NonbreakingSpace
1652     inlines_nbsp = Ct(inlines_nbsp_t)

```

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

```
1653 function self.convert(input)
1654     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.

```
1655     local opt_string = {}
1656     for k,_ in pairs(defaultOptions) do
1657         local v = options[k]
1658         if k ~= "cacheDir" then
1659             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
1660         end
1661     end
1662     table.sort(opt_string)
1663     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
```

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

```
1664     local name = util.cache(options.cacheDir, input, salt, function(input)
1665         return util.rope_to_string(parse_blocks_toplevel(input)) .. writer.eof
1666     end, ".md" .. writer.suffix)
1667     return writer.pack(name)
1668 end
1669 return self
1670 end
```

3.1.4 Conversion from Markdown to Plain T_EX

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`.

```
1671 function M.new(options)
1672     local writer = M.writer.new(options)
1673     local reader = M.reader.new(writer, options)
1674     return reader.convert
1675 end
1676
1677 return M
```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX 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

```

1678 \def\markdownInfo#1{%
1679   \message{(1.\the\inputlineno) markdown.tex info: #1.}}%
1680 \def\markdownWarning#1{%
1681   \message{(1.\the\inputlineno) markdown.tex warning: #1.}}%
1682 \def\markdownError#1#2{%
1683   \errhelp{#2.}}%
1684   \errmessage{(1.\the\inputlineno) markdown.tex error: #1.}}%

```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

1685 \def\markdownRendererInterblockSeparatorPrototype{\par}%
1686 \def\markdownRendererLineBreakPrototype{\hfil\break}%
1687 \let\markdownRendererEllipsisPrototype\dots
1688 \def\markdownRendererNbspPrototype{~}%
1689 \def\markdownRendererLeftBracePrototype{\char'{}%
1690 \def\markdownRendererRightBracePrototype{\char'}%
1691 \def\markdownRendererDollarSignPrototype{\char'$}%
1692 \def\markdownRendererPercentSignPrototype{\char'\}%
1693 \def\markdownRendererAmpersandPrototype{\char'&}%
1694 \def\markdownRendererUnderscorePrototype{\char'_}%
1695 \def\markdownRendererHashPrototype{\char'\#}%
1696 \def\markdownRendererCircumflexPrototype{\char'^}%
1697 \def\markdownRendererBackslashPrototype{\char'\}%
1698 \def\markdownRendererTildePrototype{\char'~}%
1699 \def\markdownRendererPipePrototype{|}%
1700 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
1701 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
1702 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
1703 \def\markdownRendererULBeginPrototype{}%
1704 \def\markdownRendererULBeginTightPrototype{}%
1705 \def\markdownRendererULItemPrototype{}%
1706 \def\markdownRendererULItemEndPrototype{}%
1707 \def\markdownRendererULEndPrototype{}%
1708 \def\markdownRendererULEndTightPrototype{}%
1709 \def\markdownRendererOLBeginPrototype{}%
1710 \def\markdownRendererOLBeginTightPrototype{}%
1711 \def\markdownRendererOLItemPrototype{}%
1712 \def\markdownRendererOLItemWithNumberPrototype#1{}%
1713 \def\markdownRendererOLItemEndPrototype{}%
1714 \def\markdownRendererOLEndPrototype{}%
1715 \def\markdownRendererOLEndTightPrototype{}%

```



```

1716 \def\markdownRendererDlBeginPrototype{%
1717 \def\markdownRendererDlBeginTightPrototype{%
1718 \def\markdownRendererDlItemPrototype#1{#1}%
1719 \def\markdownRendererDlItemEndPrototype{%
1720 \def\markdownRendererDlDefinitionBeginPrototype{%
1721 \def\markdownRendererDlDefinitionEndPrototype{\par}%
1722 \def\markdownRendererDlEndPrototype{%
1723 \def\markdownRendererDlEndTightPrototype{%
1724 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
1725 \def\markdownRendererStrongEmphasisPrototype#1{{\it#1}}%
1726 \def\markdownRendererBlockQuoteBeginPrototype{\par\begin{group}\it}%
1727 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
1728 \def\markdownRendererInputVerbatimPrototype#1{%
1729 \par{\tt\input"#1"\relax}\par}%
1730 \def\markdownRendererInputFencedCodePrototype#1#2{%
1731 \markdownRendererInputVerbatimPrototype{#1}}%
1732 \def\markdownRendererHeadingOnePrototype#1{#1}%
1733 \def\markdownRendererHeadingTwoPrototype#1{#1}%
1734 \def\markdownRendererHeadingThreePrototype#1{#1}%
1735 \def\markdownRendererHeadingFourPrototype#1{#1}%
1736 \def\markdownRendererHeadingFivePrototype#1{#1}%
1737 \def\markdownRendererHeadingSixPrototype#1{#1}%
1738 \def\markdownRendererHorizontalRulePrototype{%
1739 \def\markdownRendererFootnotePrototype#1{#1}%
1740 \def\markdownRendererCitePrototype#1{%
1741 \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). Note that the boolean options are not sanitized and expect the plain \TeX option macros to expand to either `true` or `false`.

```

1742 \def\markdownLuaOptions{%
1743 \ifx\markdownOptionBlankBeforeBlockquote\undefined\else
1744 blankBeforeBlockquote = \markdownOptionBlankBeforeBlockquote,
1745 \fi
1746 \ifx\markdownOptionBlankBeforeCodeFence\undefined\else
1747 blankBeforeCodeFence = \markdownOptionBlankBeforeCodeFence,
1748 \fi
1749 \ifx\markdownOptionBlankBeforeHeading\undefined\else
1750 blankBeforeHeading = \markdownOptionBlankBeforeHeading,
1751 \fi
1752 \ifx\markdownOptionCacheDir\undefined\else
1753 cacheDir = "\markdownOptionCacheDir",
1754 \fi

```

```

1755 \ifx\markdownOptionCitations\undefined\else
1756   citations = \markdownOptionCitations,
1757 \fi
1758 \ifx\markdownOptionCitationNbsps\undefined\else
1759   citationNbsps = \markdownOptionCitationNbsps,
1760 \fi
1761 \ifx\markdownOptionDefinitionLists\undefined\else
1762   definitionLists = \markdownOptionDefinitionLists,
1763 \fi
1764 \ifx\markdownOptionFootnotes\undefined\else
1765   footnotes = \markdownOptionFootnotes,
1766 \fi
1767 \ifx\markdownOptionFencedCode\undefined\else
1768   fencedCode = \markdownOptionFencedCode,
1769 \fi
1770 \ifx\markdownOptionHashEnumerators\undefined\else
1771   hashEnumerators = \markdownOptionHashEnumerators,
1772 \fi
1773 \ifx\markdownOptionHybrid\undefined\else
1774   hybrid = \markdownOptionHybrid,
1775 \fi
1776 \ifx\markdownOptionPreserveTabs\undefined\else
1777   preserveTabs = \markdownOptionPreserveTabs,
1778 \fi
1779 \ifx\markdownOptionSmartEllipses\undefined\else
1780   smartEllipses = \markdownOptionSmartEllipses,
1781 \fi
1782 \ifx\markdownOptionStartNumber\undefined\else
1783   startNumber = \markdownOptionStartNumber,
1784 \fi
1785 \ifx\markdownOptionTightLists\undefined\else
1786   tightLists = \markdownOptionTightLists,
1787 \fi}
1788 }%

```

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.

```

1789 \def\markdownPrepare{%

```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```

1790 local lfs = require("lfs")
1791 local cacheDir = "\markdownOptionCacheDir"
1792 if lfs.isdir(cacheDir) == true then else
1793   assert(lfs.mkdir(cacheDir))
1794 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.

```
1795 local md = require("markdown")
1796 local convert = md.new(\markdownLuaOptions)
1797 }%
```

3.2.4 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` and `\markdownReadAndConvert` macros defined here and in Section 3.2.5 are meant to be transparent to the remaining code.

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

```
1798
1799 \ifnum\markdownMode<2\relax
1800 \ifnum\markdownMode=0\relax
1801   \markdownInfo{Using mode 0: Shell escape via write18}%
1802 \else
1803   \markdownInfo{Using mode 1: Shell escape via os.execute}%
1804 \fi
```

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`, and to store the markdown input in the file named `\markdownOptionInputTempFileName` during the expansion of the macro `\markdownReadAndConvert`.

```
1805 \csname newwrite\endcsname\markdownLuaExecuteFileStream
```

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

Inherit the value of the `\pdfshellescape` (Lua \TeX , Pdf \TeX) or the `\shellescape` (X \TeX) 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.

```
1806 \ifx\pdfshellescape\undefined
1807   \ifx\shellescape\undefined
1808     \ifnum\markdownMode=0\relax
1809       \def\markdownExecuteShellEscape{1}%
1810     \else
1811       \def\markdownExecuteShellEscape{%
1812         \directlua{tex.sprint(status.shell_escape or "1")}}%
```

```

1813     \fi
1814   \else
1815     \let\markdownExecuteShellEscape\shellescape
1816   \fi
1817 \else
1818   \let\markdownExecuteShellEscape\pdfshellescape
1819 \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 `markdown.execute` method otherwise.

```

1820 \ifnum\markdownMode=0\relax
1821   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
1822 \else
1823   \def\markdownExecuteDirect#1{%
1824     \directlua{os.execute("\luaescapestring{#1}")}}%
1825 \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.

```

1826 \def\markdownExecute#1{%
1827   \ifnum\markdownExecuteShellEscape=1\relax
1828     \markdownExecuteDirect{#1}%
1829   \else
1830     \markdownError{I can not access the shell}{Either run the TeX
1831       compiler with the --shell-escape or the --enable-write18 flag,
1832       or set shell_escape=t in the texmf.cnf file}%
1833   \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.

```

1834 \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.

```

1835   \immediate\openout\markdownLuaExecuteFileStream=%
1836     \markdownOptionHelperScriptFileName
1837   \markdownInfo{Writing a helper Lua script to the file
1838     "\markdownOptionHelperScriptFileName"}%
1839   \immediate\write\markdownLuaExecuteFileStream{%
1840     local kpse = require('kpse')
1841     kpse.set_program_name('luatex') #1}%
1842   \immediate\closeout\markdownLuaExecuteFileStream

```

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

```
1843 \markdownInfo{Executing a helper Lua script from the file
1844     "\markdownOptionHelperScriptFileName" and storing the result in the
1845     file "\markdownOptionOutputTempFileName"}%
1846 \markdownExecute{texlua "\markdownOptionHelperScriptFileName" >
1847     "\markdownOptionOutputTempFileName"}%
```

`\input` the generated `\markdownOptionOutputTempFileName` file.

```
1848 \input\markdownOptionOutputTempFileName\relax}%
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
1849 \begingroup
1850 \catcode'\^^I=12%
1851 \gdef\markdownReadAndConvertTab{^^I}%
1852 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the `MTEX2ε` `\filecontents` macro to plain `TeX`.

```
1853 \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.

```
1854 \catcode'\^^M=13%
1855 \catcode'\^^I=13%
1856 \catcode'|=0%
1857 \catcode'\=12%
1858 |gdef|markdownReadAndConvert#1#2{%
1859     |begingroup%
```

Open the `\markdownOptionInputTempFileName` file for writing.

```
1860 |immediate|openout|markdownLuaExecuteFileStream%
1861 |markdownOptionInputTempFileName%
1862 |markdownInfo{Buffering markdown input into the temporary %
1863     input file "|markdownOptionInputTempFileName" and scanning %
1864     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.

```
1865 |def|do##1{|catcode'##1=12}|dospecials%
1866 |catcode'|=12%
1867 |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.

```

1868      |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.
1869      |ifx|relax##3|relax%
1870      |immediate|write|markdownLuaExecuteFileStream{##1}%
1871      |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.

1872      |def^^M{%
1873      |markdownInfo{The ending token sequence was found}%
1874      |immediate|closeout|markdownLuaExecuteFileStream%
1875      |endgroup%
1876      |markdownInput|markdownOptionInputTempFileName%
1877      #2}%
1878      |fi%

Repeat with the next line.
1879      ^^M}%

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

1880      |catcode'|^^I=13%
1881      |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.

1882      |catcode'|^^M=13%
1883      |def^^M##1^^M{%
1884      |def^^M####1^^M{%
1885      |markdownReadAndConvertProcessLine####1#1#1|relax}%
1886      ^^M}%
1887      ^^M}%

Reset the character categories back to the former state.
1888 |endgroup

```

3.2.5 Direct Lua Access

The following T_EX code is intended for T_EX engines that provide direct access to Lua (LuaT_EX). The `\markdownLuaExecute` and `\markdownReadAndConvert` defined here and in Section 3.2.4 are meant to be transparent to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

1889 \else
1890 \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 3.2.4,

```

1891 \def\markdownLuaExecute#1{\directlua{local print = tex.print #1}}%

```

In the definition of the direct Lua access version of the `\markdownReadAndConvert` macro, we will be using the hash symbol (`#`), the underscore symbol (`_`), the circumflex symbol (`^`), the dollar sign (`$`), the backslash symbol (`\`), the percent sign (`%`), and the braces (`{}`) as a part of the Lua syntax.

```

1892 \begingroup

```

To this end, we will make the underscore symbol, the dollar sign, and circumflex symbols ordinary characters,

```

1893 \catcode'\_ =12%
1894 \catcode'\$ =12%
1895 \catcode'\^ =12%

```

swap the category code of the hash symbol with the slash symbol (`/`).

```

1896 \catcode'\/=6%
1897 \catcode'\#=12%

```

swap the category code of the percent sign with the at symbol (`@`).

```

1898 \catcode'\@ =14%
1899 \catcode'\% =12%

```

swap the category code of the backslash symbol with the pipe symbol (`|`),

```

1900 \catcode'| =0@
1901 \catcode'\ =12@

```

Braces are a part of the plain \TeX syntax, but they are not removed during expansion, so we do not need to bother with changing their category codes.

```

1902 |gdef|markdownReadAndConvert/1/2{@

```

Make the `\markdownReadAndConvertAfter` macro store the token sequence that will be inserted into the document after the ending token sequence has been found.

```

1903 |def|markdownReadAndConvertAfter{/2}@
1904 |markdownInfo{Buffering markdown input and scanning for the
1905 closing token sequence "/1"}@
1906 |directlua{@

```

Set up an empty Lua table that will serve as our buffer.

```

1907 |markdownPrepare
1908 local buffer = {}

```

Create a regex that will match the ending input sequence. Escape any special regex characters (like a star inside `\end{markdown*}`) inside the input.

```
1909     local ending_sequence = "^.-" .. ([[/1]]):gsub(
1910         "([%(%)%.%%%+%-%%*%?[%%]%^%$])", "%%1")
```

Register a callback that will notify you about new lines of input.

```
1911     |markdownLuaRegisterIBCallback{function(line)
```

When the ending token sequence appears on a line, unregister the callback, convert the contents of our buffer from markdown to plain \TeX , and insert the result into the input line buffer of \TeX .

```
1912         if line:match(ending_sequence) then
1913             |markdownLuaUnregisterIBCallback;
1914             local input = table.concat(buffer, "\n") .. "\n"
1915             local output = convert(input)
1916             return [[\markdownInfo{The ending token sequence was found}]] ..
1917                 output .. [[\markdownReadAndConvertAfter]]
```

When the ending token sequence does not appear on a line, store the line in our buffer, and insert either `\fi`, if this is the first line of input, or an empty token list to the input line buffer of \TeX .

```
1918         else
1919             buffer[#buffer+1] = line
1920             return [[\]] .. (#buffer == 1 and "fi" or "relax")
1921         end
1922     end}}@
```

Insert `\iffalse` after the `\markdownReadAndConvert` macro in order to consume the rest of the first line of input.

```
1923     |iffalse}@
```

Reset the character categories back to the former state.

```
1924     |endgroup
1925     \fi
```

3.2.6 Typesetting Markdown

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

```
1926 \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.

```
1927     \catcode'\=0%
1928     \catcode'\=12%
1929     |gdef|markdownInput#1{%
```



```

1930 |markdownInfo{Including markdown document "#1"}%
1931 |markdownLuaExecute{%
1932 |markdownPrepare
1933 |local input = assert(io.open("#1","r")):read("*a")

```

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

```

1934 |print(convert(input:gsub("\r\n?", "\n")))}%
1935 |endgroup

```

3.3 L^AT_EX Implementation

The L^AT_EX implementation makes use of the fact that, apart from some subtle differences, L^AT_EX implements the majority of the plain T_EX format (see [4, Section 9]). As a consequence, we can directly reuse the existing plain T_EX implementation.

```

1936 \input markdown
1937 \def\markdownVersionSpace{ }%
1938 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
1939 \markdownVersion\markdownVersionSpace markdown renderer]%

```

3.3.1 Logging Facilities

The L^AT_EX implementation redefines the plain T_EX logging macros (see Section 3.2.1) to use the L^AT_EX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

1940 \renewcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
1941 \renewcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
1942 \renewcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%

```

3.3.2 Typesetting Markdown

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

```

1943 \let\markdownInputPlainTeX\markdownInput
1944 \renewcommand\markdownInput[2][{}]{%
1945 |begingroup
1946 |markdownSetup{#1}%
1947 |markdownInputPlainTeX{#2}%
1948 |endgroup}%

```

The `markdown`, and `markdown*` L^AT_EX environments are implemented using the `\markdownReadAndConvert` macro.

```

1949 \renewenvironment{markdown}{%
1950 |markdownReadAndConvert@markdown{}}\relax
1951 \renewenvironment{markdown*}[1]{%

```

```

1952 \markdownSetup{#1}%
1953 \markdownReadAndConvert@markdown*}\relax
1954 \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*.

```

1955 \catcode'\|=0\catcode'\<=1\catcode'\>=2%
1956 \catcode'\|=12\catcode'\{=12\catcode'\}=12%
1957 \gdef\markdownReadAndConvert@markdown#1<%
1958     \markdownReadAndConvert<\end{markdown#1}>%
1959     <|\end<markdown#1>>>%
1960 \endgroup

```

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

1961 \DeclareOption*{%
1962     \expandafter\markdownSetup\expandafter{\CurrentOption}}%
1963 \ProcessOptions\relax

```

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

```

1964 \define@key{markdownOptions}{renderers}{%
1965     \setkeys{markdownRenderers}{#1}%
1966     \def\KV@prefix{KV@markdownOptions@}}%
1967 \define@key{markdownOptions}{rendererPrototypes}{%
1968     \setkeys{markdownRendererPrototypes}{#1}%
1969     \def\KV@prefix{KV@markdownOptions@}}%

```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

1970 \RequirePackage{url}
1971 \RequirePackage{graphicx}

```

If the `\markdownOptionTightLists` macro expands to `false`, do not load the `paralist` package. This is necessary for $\text{\LaTeX 2}_{\epsilon}$ document classes that do not play nice with `paralist`, such as `beamer`.

```

1972 \RequirePackage{ifthen}
1973 \ifx\markdownOptionTightLists\undefined
1974     \RequirePackage{paralist}
1975 \else
1976     \ifthenelse{\equal{\markdownOptionTightLists}{false}}{ }{
1977         \RequirePackage{paralist}}
1978 \fi

```

```

1979 \RequirePackage{fancyvrb}
1980 \markdownSetup{rendererPrototypes={
1981   lineBreak = {\},
1982   leftBrace = {\textbraceleft},
1983   rightBrace = {\textbraceright},
1984   dollarSign = {\textdollar},
1985   underscore = {\textunderscore},
1986   circumflex = {\textasciicircum},
1987   backslash = {\textbackslash},
1988   tilde = {\textasciitilde},
1989   pipe = {\textbar},
1990   codeSpan = {\texttt{#1}},
1991   link = {#1\footnote{\ifx\empty#4\empty\else#4:
1992     \fi\texttt{<url{#3}\texttt>}},
1993   image = {\begin{figure}
1994     \begin{center}%
1995     \includegraphics{#3}%
1996     \end{center}%
1997     \ifx\empty#4\empty\else
1998     \caption{#4}%
1999     \fi
2000     \label{fig:#1}%
2001     \end{figure}},
2002   ulBegin = {\begin{itemize}},
2003   ulBeginTight = {\begin{compactitem}},
2004   ulItem = {\item},
2005   ulEnd = {\end{itemize}},
2006   ulEndTight = {\end{compactitem}},
2007   olBegin = {\begin{enumerate}},
2008   olBeginTight = {\begin{compactenum}},
2009   olItem = {\item},
2010   olItemWithNumber = {\item[#1.]},
2011   olEnd = {\end{enumerate}},
2012   olEndTight = {\end{compactenum}},
2013   dlBegin = {\begin{description}},
2014   dlBeginTight = {\begin{compactdesc}},
2015   dlItem = {\item[#1]},
2016   dlEnd = {\end{description}},
2017   dlEndTight = {\end{compactdesc}},
2018   emphasis = {\emph{#1}},
2019   strongEmphasis = {%
2020     \ifx\alert\undefined
2021     \textbf{\emph{#1}}%
2022     \else % Beamer support
2023     \alert{\emph{#1}}%
2024     \fi},
2025   blockQuoteBegin = {\begin{quotation}},

```

```

2026 blockQuoteEnd = {\end{quotation}},
2027 inputVerbatim = {\VerbatimInput{#1}},
2028 inputFencedCode = {%
2029     \ifx\relax#2\relax
2030     \VerbatimInput{#1}%
2031     \else
2032     \ifx\minted@jobname\undefined
2033     \ifx\lst@version\undefined
2034     \markdownRendererInputFencedCode{#1}{}%

```

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

```

2035     \else
2036     \lstinputlisting[language=#2]{#1}%
2037     \fi

```

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

```

2038     \else
2039     \inputminted{#2}{#1}%
2040     \fi
2041 \fi},
2042 horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
2043 footnote = {\footnote{#1}}}}
2044
2045 \ifx\chapter\undefined
2046 \markdownSetup{rendererPrototypes = {
2047     headingOne = {\section{#1}},
2048     headingTwo = {\subsection{#1}},
2049     headingThree = {\subsubsection{#1}},
2050     headingFour = {\paragraph{#1}},
2051     headingFive = {\subparagraph{#1}}}}
2052 \else
2053 \markdownSetup{rendererPrototypes = {
2054     headingOne = {\chapter{#1}},
2055     headingTwo = {\section{#1}},
2056     headingThree = {\subsection{#1}},
2057     headingFour = {\subsubsection{#1}},
2058     headingFive = {\paragraph{#1}},
2059     headingSix = {\subparagraph{#1}}}}
2060 \fi

```

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

```

2061 \newcount\markdownLaTeXCitationsCounter
2062
2063 % Basic implementation
2064 \def\markdownLaTeXBasicCitations#1#2#3#4{%

```

```

2065 \advance\markdownLaTeXCitationsCounter by 1\relax
2066 \ifx\relax#2\relax\else#2~\fi\cite[#3]{#4}%
2067 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2068 \expandafter\@gobble
2069 \fi\markdownLaTeXBasicCitations}
2070 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
2071
2072 % BibLaTeX implementation
2073 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
2074 \advance\markdownLaTeXCitationsCounter by 1\relax
2075 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2076 \autocites#1[#3][#4]{#5}%
2077 \expandafter\@gobbletwo
2078 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
2079 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
2080 \advance\markdownLaTeXCitationsCounter by 1\relax
2081 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2082 \textcites#1[#3][#4]{#5}%
2083 \expandafter\@gobbletwo
2084 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
2085
2086 \markdownSetup{rendererPrototypes = {
2087 cite = {%
2088 \markdownLaTeXCitationsCounter=1%
2089 \def\markdownLaTeXCitationsTotal{#1}%
2090 \ifx\autocites\undefined
2091 \expandafter
2092 \markdownLaTeXBasicCitations
2093 \else
2094 \expandafter\expandafter\expandafter
2095 \markdownLaTeXBibLaTeXCitations
2096 \expandafter{\expandafter}%
2097 \fi},
2098 textCite = {%
2099 \markdownLaTeXCitationsCounter=1%
2100 \def\markdownLaTeXCitationsTotal{#1}%
2101 \ifx\textcites\undefined
2102 \expandafter
2103 \markdownLaTeXBasicTextCitations
2104 \else
2105 \expandafter\expandafter\expandafter
2106 \markdownLaTeXBibLaTeXTextCitations
2107 \expandafter{\expandafter}%
2108 \fi}}}

```

3.3.5 Miscellanea

Unlike base Lua \TeX , which only allows for a single registered function per a callback (see [1, Section 8.1]), the $\text{\TeX}2_{\epsilon}$ format disables the `callback.register` method and exposes the `luatexbase.add_to_callback` and `luatexbase.remove_from_callback` methods that enable the user code to hook several functions on a single callback (see [4, Section 73.4]).

To make our code function with the $\text{\TeX}2_{\epsilon}$ format, we need to redefine the `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros accordingly.

```
2109 \let\markdownLuaRegisterIBCallbackPrevious
2110 \markdownLuaRegisterIBCallback
2111 \let\markdownLuaUnregisterIBCallbackPrevious
2112 \markdownLuaUnregisterIBCallback
2113 \renewcommand\markdownLuaRegisterIBCallback[1]{%
2114   if luatexbase and luatexbase.add_to_callback then
2115     luatexbase.add_to_callback("process_input_buffer", #1, %
2116       "The markdown input processor")
2117   else
2118     \markdownLuaRegisterIBCallbackPrevious{#1}
2119   end}
2120 \renewcommand\markdownLuaUnregisterIBCallback{%
2121   if luatexbase and luatexbase.add_to_callback then
2122     luatexbase.remove_from_callback("process_input_buffer",%
2123       "The markdown input processor")
2124   else
2125     \markdownLuaUnregisterIBCallbackPrevious;
2126   end}
```

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.

```
2127 \newcommand\markdownMakeOther{%
2128   \count0=128\relax
2129   \loop
2130     \catcode\count0=11\relax
2131     \advance\count0 by 1\relax
2132   \ifnum\count0<256\repeat}%
```

3.4 Con \TeX t Implementation

The Con \TeX t implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV Con \TeX t 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.

```
2133 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2134 \do\#\do\^\do\_ \do\% \do\~}%
```

When there is no Lua support, then just load the plain TeX implementation.

```
2135 \ifx\directlua\undefined
2136 \input markdown
2137 \else
```

When there is Lua support, check if we can set the `process_input_buffer` LuaTeX callback.

```
2138 \directlua{%
2139     local function unescape(str)
2140         return (str:gsub("|", string.char(92))) end
2141     local old_callback = callback.find("process_input_buffer")
2142     callback.register("process_input_buffer", function() end)
2143     local new_callback = callback.find("process_input_buffer")
```

If we can not, we are probably using ConTeXt Mark IV. In ConTeXt Mark IV, the `process_input_buffer` callback is currently frozen (inaccessible from the user code) and, due to the lack of available documentation, it is unclear to me how to emulate it. As a workaround, we will force the plain TeX implementation to use the Lua shell escape bridge (see Section 3.2.4) by setting the `\markdownMode` macro to the value of 1.

```
2144     if new_callback == false then
2145         tex.print(unescape([[def|markdownMode{1}|input markdown]]))
```

If we can set the `process_input_buffer` LuaTeX callback, then just load the plain TeX implementation.

```
2146     else
2147         callback.register("process_input_buffer", old_callback)
2148         tex.print(unescape("|input markdown"))
2149     end}%
2150 \fi
```

If the shell escape bridge is being used, define the `\markdownMakeOther` macro, so that the pipe character (`|`) is inactive during the scanning. This is necessary, since the character is active in ConTeXt.

```
2151 \ifnum\markdownMode<2\relax
2152 \def\markdownMakeOther{%
2153     \catcode'|=12}%
2154 \fi
```

3.4.1 Logging Facilities

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

```
2155 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2156 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
```

3.4.2 Typesetting Markdown

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

```
2157 \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*.

```
2158 \catcode'\|=0%
2159 \catcode'\|=12%
2160 |gdef\startmarkdown{%
2161     |markdownReadAndConvert{\stopmarkdown}%
2162                               {\|stopmarkdown}}%
2163 |endgroup
```

3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
2164 \def\markdownRendererLineBreakPrototype{\blank}%
2165 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
2166 \def\markdownRendererRightBracePrototype{\textbraceright}%
2167 \def\markdownRendererDollarSignPrototype{\textdollar}%
2168 \def\markdownRendererPercentSignPrototype{\percent}%
2169 \def\markdownRendererUnderscorePrototype{\textunderscore}%
2170 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
2171 \def\markdownRendererBackslashPrototype{\textbackslash}%
2172 \def\markdownRendererTildePrototype{\textasciitilde}%
2173 \def\markdownRendererPipePrototype{\char'|}%
2174 \def\markdownRendererLinkPrototype#1#2#3#4{%
2175     \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
2176     \fi\texttt<\hyphenatedurl{#3}>}}%
2177 \def\markdownRendererImagePrototype#1#2#3#4{%
2178     \placefigure[] [fig:#1]{#4}{\externalfigure[#3]}}%
2179 \def\markdownRendererULBeginPrototype{\startitemize}%
2180 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
2181 \def\markdownRendererULItemPrototype{\item}%
2182 \def\markdownRendererULEndPrototype{\stopitemize}%
2183 \def\markdownRendererULEndTightPrototype{\stopitemize}%

```



```

2184 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
2185 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
2186 \def\markdownRendererOlItemPrototype{\item}%
2187 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
2188 \def\markdownRendererOlEndPrototype{\stopitemize}%
2189 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
2190 \definedescription
2191   [MarkdownConTeXtDlItemPrototype]
2192   [location=hanging,
2193    margin=standard,
2194    headstyle=bold]%
2195 \definestartstop
2196   [MarkdownConTeXtDlPrototype]
2197   [before=\blank,
2198    after=\blank]%
2199 \definestartstop
2200   [MarkdownConTeXtDlTightPrototype]
2201   [before=\blank\startpacked,
2202    after=\stoppacked\blank]%
2203 \def\markdownRendererDlBeginPrototype{%
2204   \startMarkdownConTeXtDlPrototype}%
2205 \def\markdownRendererDlBeginTightPrototype{%
2206   \startMarkdownConTeXtDlTightPrototype}%
2207 \def\markdownRendererDlItemPrototype#1{%
2208   \startMarkdownConTeXtDlItemPrototype{#1}}%
2209 \def\markdownRendererDlItemEndPrototype{%
2210   \stopMarkdownConTeXtDlItemPrototype}%
2211 \def\markdownRendererDlEndPrototype{%
2212   \stopMarkdownConTeXtDlPrototype}%
2213 \def\markdownRendererDlEndTightPrototype{%
2214   \stopMarkdownConTeXtDlTightPrototype}%
2215 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
2216 \def\markdownRendererStrongEmphasisPrototype#1{{\bf\em#1}}%
2217 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
2218 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
2219 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
2220 \def\markdownRendererInputFencedCodePrototype#1#2{%
2221   \ifx\relax#2\relax
2222     \typefile{#1}%
2223   \else

```

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

```

% Map the 'TEX' syntax highlighter to the 'latex' infostring.
\definetying [latex]
\setuptyping [latex] [option=TEX]

```

```

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

```

```

2224   \typefile[#2] []{#1}%
2225   \fi}%
2226 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
2227 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
2228 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
2229 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
2230 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
2231 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
2232 \def\markdownRendererHorizontalRulePrototype{%
2233   \blackrule[height=1pt, width=\hsize]}%
2234 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
2235 \stopmodule\protect

```

References

1. L^AT_EX DEVELOPMENT TEAM. *LuaT_EX reference manual (0.98.3)* [online]. 2016 [visited on 2016-08-30]. Available from: <http://www.luatex.org/svn/trunk/manual/luatex.pdf>.
2. KNUTH, Donald Ervin. *The T_EXbook*. 3rd ed. Addison-Westley, 1986. ISBN 0-201-13447-0.
3. IERUSALIMSKY, Roberto. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. ISBN 978-85-903798-5-0.
4. BRAAMS, Johannes; CARLISLE, David; JEFFREY, Alan; LAMPORT, Leslie; MITTELBACH, Frank; ROWLEY, Chris; SCHÖPF, Rainer. *The L^AT_EX 2_ε Sources* [online]. 2016 [visited on 2016-06-02]. Available from: <http://mirrors.ctan.org/macros/latex/base/source2e.pdf>.