The *cloze* package*

Josef Friedrich
josef@friedrich.rocks
github.com/Josef-Friedrich/cloze

v1.4 from 2020/05/20

*Many thanks to Robert-Michael Huber for his advice and to Paul Isambert for his article *"Three things you can do with LuaTEX that would be extremely painful otherwise"* in TUGboat, Volume 31 (2010), No. 3. This article helped a lot to write this package.

Contents

1 Introduction

cloze is a LaTeX package to generate cloze texts. It uses the capabilities of the modern TeX engine *LuaTeX*. Therefore, you must use LuaLaTeX to create documents containing gaps.

```
lualatex cloze-text.tex
```

The main feature of the package is that the formatting doesn't change when using the `hide` and `show` (→ 2.2.9) options.

Lorem ipsum *dolor sit* amet, consectetur *adipisicing* elit, sed do eiusmod tempor incididunt ut labore et *dolore magna* aliqua. Ut enim ad minim veniam, quis nostrud *exercitation* ullamco laboris nisi ut *aliquip* ex ea commodo consequat.

The command `\clozeset{hide}` only shows gaps. When you put both texts on top of each other you will see that they perfectly match.

Lorem ipsum _____ amet, consectetur _____ elit, sed do eiusmod tempor incididunt ut labore et _____ aliqua. Ut enim ad minim veniam, quis nostrud _____ ullamco laboris nisi ut _____ ex ea commodo consequat.

2 Usage

There are the commands `\cloze`, `\clozefix`, `\clozefil`, `\clozenol`, `\clozestrike` and the environments `clozepar` and `clozebox` to generate cloze texts.

2.1 The commands and environments

2.1.1 `\cloze`

`\cloze` `\cloze[`⟨*options*⟩`]{`⟨*some text*⟩`}`: The command `\cloze` is similar to a command that offers the possibility to underline the texts. `\cloze` does not prevent line breaks. The width of a gap depends on the number of letters and the font used. The only option which affects the widths of a gap is the option `margin` (→ 2.2.11).

Lorem ipsum *dolor* sit amet, *consectetur* adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore *magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi* ut aliquip ex ea commodo consequat.

It is possible to convert a complete paragraph into a 'gap'. But don't forget: There is a special environment for this: `clozepar` (→ 2.1.7).

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

The command `\cloze` doesn't change the behavior of the hyphenation. Let's try some long German words:

es *Telekommunikationsüberwachung* geht *Unternehmenssteuerfortentwicklungsgesetz* *Abteilungsleiterin* *Oberkommisarin* auch *Fillialleiterin* kurz *Oberkommisarin* *Unternehmenssteuerfortentwicklungsgesetz* *Fillialleiterin* *Metzgermeisterin* in *Abteilungsleiterin* der *Oberkommisarin* *Hochleistungsflüssigkeitschromatographie* *Fillialleiterin* Kürze *Unternehmenssteuerfortentwicklungsgesetz* *Metzgermeisterin* liegt *Abteilungsleiterin* die *Metzgermeisterin* *Abteilungsleiterin* Würze *Oberkommisarin*

2.1.2 `\clozesetfont`

`\clozesetfont` The gap font can be changed by using the command `\clozesetfont`. `\clozesetfont` redefines the command `\clozefont` which contains the font definition. Thus, the command `\clozesetfont{\Large}` has the same effect as `\renewcommand{\clozefont}{\Large}`.

Excepteur *sint* occaecat *cupidatat* non proident.

Please do not put any color definitions in `\clozesetfont`, as it won't work. Use the option `textcolor` instead (→ 2.2.10).

`\clozesetfont{\ttfamily\normalsize}` changes the gap text for example into a normal sized typewriter font.

Excepteur *sint* occaecat *cupidatat* non proident.

2.1.3 `\clozefix`

`\clozefix` `\clozefix[`⟨*options*⟩`]{`⟨*some text*⟩`}`: The command `\clozefix` creates gaps with a fixed width. The clozes are default concering the width `2cm`.

Lorem ipsum dolor sit amet:

1. *consectetur*
2. *adipisicing*
3. *elit*

sed do eiusmod.

Gaps with a fixed width are much harder to solve.

Lorem ipsum dolor *sit* amet, *consectetur* adipisicing elit, sed do eiusmod tempor incididunt *ut* labore et dolore magna aliqua.

Using the option `align` you can make nice tabulars like this:

Composer	Life span
Joseph Haydn	*1723-1809*
Wolfgang Amadeus Mozart	*1756-1791*
Ludwig van Beethoven	*1770-1827*

2.1.4 `\clozenol`

`\clozenol` `\clozenol[`⟨*options*⟩`]{`⟨*some text*⟩`}`: The macro name `clozenol` stands for *"cloze no line"*. As the the name suggests this macro typesets cloze texts without a line.

```
Lorem \clozenol{ipsum dolor} sit amet.
```

Lorem *ipsum dolor* sit amet.

```
Lorem \clozenol[textcolor=green]{ipsum dolor} sit amet.
```

Lorem *ipsum dolor* sit amet.

2.1.5 `\clozefil`

`\clozefil` `\clozefil[`⟨*options*⟩`]{`⟨*some text*⟩`}`: The name of the command is inspired by `\hfil`, `\hfill`, and `\hfilll`. Only `\clozefil` fills out all available horizontal spaces with a line.

Lorem ipsum dolor sit amet, *consectetur adipisicing elit, sed do eiusmod.*
Ut enim *ad minim veniam* exercitation.

2.1.6 `\clozeextend`

`\clozeextend` `\clozeextend[`⟨*spaces*⟩`]`: The command `\clozeextend` adds some invisible placeholders to extend some cloze texts with blank space.

```
\begin{itemize}
\item \clozefil{Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.}
\item \clozefil{Ut enim ad minim veniam \clozeextend[20]}
```

```
\item \clozefil{quis nostrud \clozeextend[20]}
\end{itemize}
```

- *Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*
- *Ut enim ad minim veniam*
- *quis nostrud*

2.1.7 clozepar

clozepar `\begin{clozepar}`[⟨*options*⟩] …*some text* …`\end{clozepar}`: The environment `clozepar` transforms a complete paragraph into a cloze text. The options `align`, `margin` and `width` have no effect on this environment.

Lorem ipsum dolor sit amet, consectetur adipisicing elit ullamco laboris nisi. *Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum.* Excepteur sint occaecat cupidatat non proident.

2.1.8 clozebox

clozebox `\begin{clozebox}`*[⟨*options*⟩] …*some text* …`\end{clozebox}`: The environment `clozebox` surrounds a text with a box. The starred version omits the line around the box. Use the options `boxwidth` and `boxheight` to specify the dimensions of the box. By default the width of the box is `\linewidth`.

```
\begin{clozebox}
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
\end{clozebox}
```

`\clozehide`

`\clozeshow`

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

2.1.8.1 Option boxwidth

See the documentation about the option (→ 2.2.7).

```
\begin{clozebox}[boxwidth=5cm]
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
\end{clozebox}
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

2.1.8.2 Option boxheight

See the documentation about the option (→ 2.2.6).

```
\begin{clozebox}[boxwidth=5cm,boxheight=5cm]
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
\end{clozebox}
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

2.1.8.3 starred

```
\begin{clozebox}*[boxwidth=5cm,boxheight=5cm]
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
\end{clozebox}
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

2.1.9 clozespace

clozespace `\begin{clozespace}[`⟨*options*⟩`]` …*some text* …`\end{clozespace}`

```
\begin{clozespace}[spacing=2]
\clozesetfont{\ttfamily\Huge}
Lorem \cloze{ipsum} dolor sit \cloze{amet}, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis \cloze{nostrud}
exercitation ullamco \cloze{laboris} nisi ut aliquip ex ea commodo
consequat.
\end{clozespace}
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

2.1.10 \clozeline

\clozeline `\clozeline[`⟨*options*⟩`]`: To create a cloze line of a certain width, use the command `\clozeline`. The default width of the line is `2cm`. In combination with the other cloze commands you can create for example an irregular alignment of the cloze text.

```
Ut enim ad
\clozeline[width=1cm]\cloze{minim}\clozeline[width=3cm]
minim veniam
```

Ut enim ad _____*minim*_____ minim veniam,

2.1.11 \clozelinefil

\clozelinefil `\clozelinefil[`⟨*options*⟩`]`: This command `\clozelinefil` fills the complete available horizontal space with a line. Moreover, `\clozelinefil` was used to create `\clozefil`.

Lorem_____

2.1.12 \clozestrike

\clozestrike `\clozestrike[`⟨*options*⟩`]{`⟨*wrong text*⟩`}{`⟨*correct text*⟩`}`

```
Lorem \clozestrike{ipsum}{dolor} sit amet.
```

dolor
Lorem ~~ipsum~~ sit amet.

```
Lorem \clozestrike[textcolor=red]{ipsum}{dolor} sit amet.
```

dolor
Lorem ~~ipsum~~ sit amet.

et dolore magna aliquyam erat *et accusam et justo*
invidunt ut ~~labore~~, sed diam voluptua. At ~~vero eos~~ duo dolores et ea *clita kasd gubergren*
~~rebum. Stet~~, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem *sadipscing elitr, sed diam*
ipsum ~~dolor sit amet, consetetur~~ nonumy eirmod tempor invidunt ut labore *At vero*
et dolore magna aliquyam erat, ~~sed diam voluptua~~.

invidunt ut labore, sed diam voluptua. At vero eos duo dolores et ea rebum. Stet, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur nonumy eirmod tempor invidunt ut labore

et dolore magna aliquyam erat, sed diam voluptua.

2.2 The options

2.2.1 Local and global options

The *cloze* package distinguishs between *local* and *global* options. Besides the possiblity to set *global* options in the `\usepackage[`⟨*global options*⟩`]{`⟨*cloze*⟩`}` declaration, the cloze package offers a special command to set *global* options: `\clozeset{`⟨*global options*⟩`}`

2.2.2 `\clozeset`

`\clozeset` `\clozeset{`⟨*global options*⟩`}`: The command can set *global* options for each paragraph.

```
\clozeset{textcolor=red} Lorem \cloze{ipsum} dolor \par
\clozeset{textcolor=green} Lorem \cloze{ipsum} dolor
```

Lorem _ipsum_ dolor
Lorem _ipsum_ dolor

`\clozeset` does not change the options within a paragraph. As you can see in the example below the last `\clozeset` applies the color green for both gaps.

```
\clozeset{textcolor=red} Lorem \cloze{ipsum} dolor
\clozeset{textcolor=green} Lorem \cloze{ipsum} dolor
```

Lorem _ipsum_ dolor Lorem _ipsum_ dolor

2.2.3 `\clozereset`

`\clozereset` `\clozereset`: The command resets all *global* options to the default values. It has no effect on the *local* options.

```
\clozeset{
  thickness=3mm,
  linecolor=yellow,
  textcolor=magenta,
  margin=-2pt
}
```

Very *silly* global *options*.

```
\clozereset
```

Relax! We can reset *those* options.

2.2.4 `\clozeshow` and `\clozehide`

`\clozeshow` `\clozeshow` and `\clozehide`: This commands are shortcuts for `\clozeset{`⟨*show*⟩`}` and `\clozeset{`⟨*hide*⟩`}`.
`\clozehide`

```
\clozehide
```

Lorem _____ amet, consectetur _____ elit.

```
\clozeshow
```

Lorem *ipsum dolor sit* amet, consectetur *adipisicing* elit.

2.2.5 `align`

`[align=`⟨*left/center/right*⟩`]`: Only the macro `\clozefix` (→ 2.1.3) takes the option `align` into account. Possible values are `left`, `center` and `right`. This option only makes sense, if the width of the line is larger than the width of the text.

Lorem ipsum (`left`)
Lorem ipsum (`center`)
Lorem ipsum (`right`)

2.2.6 `boxheight`

`boxheight` specifies the height of a cloze box. This option has only an effect on the environment `clozebox` (→ 2.1.8). An example can be found in the section about the environment (→ 2.1.8.2).

2.2.7 `boxwidth`

`boxwidth` specifies the width of a cloze box. This option has only an effect on the environment `clozebox` (→ 2.1.8). An example can be found in the section about the environment (→ 2.1.8.1).

2.2.8 `distance`

`[distance=`⟨*dimen*⟩`]`: The option `distance` specifies the spacing between the baseline of the text and the gap line. The larger the dimension of the option `distance`, the more moves the line down. Negative values cause the line to appear above the baseline. The default value is `1.5pt`.

Lorem ipsum dolor sit amet. (`1.5pt`)
Lorem ipsum dolor sit amet. (`3pt`)
Lorem ipsum dolor sit amet. (`-3pt`)

2.2.9 `hide` and `show`

`[hide]` and `[show]`: By default the cloze text is displayed. Use the option `hide` to remove the cloze text from the output. If you accidentally specify both options – `hide` and `show` – the last option "wins".

Lorem ipsum _____, consectetur _____ elit. (`hide`)
Lorem ipsum *dolor sit amet*, consectetur *adipisicing* elit. (`show`)
Lorem ipsum _____, consectetur _____ elit. (`show,hide`)
Lorem ipsum *dolor sit amet*, consectetur *adipisicing* elit. (`hide,show`)

2.2.10 `linecolor` and `textcolor`

`[linecolor=`⟨*color name*⟩`]` and `[textcolor=`⟨*color name*⟩`]`: Values for both color options are color names used by the xcolor package. To define your own color use the following command:

```
\definecolor{myclozecolor}{rgb}{0.1,0.4,0.6}
\cloze[textcolor=myclozecolor]{Lorem ipsum}
```

Lorem ipsum dolor sit amet, consectetur (`myclozecolor`)
Lorem ipsum dolor sit amet, consectetur (`red`)
Lorem ipsum dolor sit amet, consectetur (`green`)

You can use the same color names to colorize the cloze lines.

Lorem ipsum dolor sit amet, consectetur (`myclozecolor`)
Lorem ipsum dolor sit amet, consectetur (`red`)
Lorem ipsum dolor sit amet, consectetur (`green`)

2.2.11 `margin`

`[margin=`⟨*dimen*⟩`]`: The option `margin` indicates how far the line sticks up from the text. The option can be used with the commands `\cloze`, `\clozefix` and `\clozefil`. The default value of the option is `3pt`.

Lorem ipsum *dolor* sit amet. (0pt)
Lorem ipsum *dolor* sit amet. (5mm)
Lorem ipsum *dolor* sit amet. (1cm)
Lorem ipsum *dolor* sit amet. (6em)
Lorem ipsum*dolor*sit amet. (-4pt)

Is a punctation mark placed directly after a gap, then the line breaks after this punctation mark. Even the most large value of `margin` does not affect this behavior.

Lorem , *ipsum* . *dolor* ; *sit* : *amet* , *consectetur* . *adipisicing* ; *elit* : *sed* , *do* . *eiusmod* ; *tempor* .

2.2.12 `spacing`

`[spacing=`⟨*number*⟩`]`: This option provides support for setting the spacing between lines. A larger font used for the cloze texts needs more line space to avoid unsteady line distances. This option only affects the environment `clozespace` (→ 2.1.9).

2.2.13 `thickness`

`[thickness=`⟨*dimen*⟩`]`: The option `thickness` indicates how thick the line is. The option `distance` (→ 2.2.8) is not affected by this option, because the bottom of the line moves down. The default value of this option is `0.4pt`.

Lorem *ipsum dolor sit* amet. (0.01pt)
Lorem *ipsum dolor sit* amet. (1pt)
Lorem *ipsum dolor sit* amet. (2pt)

2.2.14 `width`

`[width=`⟨*dimen*⟩`]`: The only command which can be changed by the option `width` is `\clozefix` (→ 2.1.3). The default value of the option is `2cm`.

Lorem *dolor* amet. (3cm)
Lorem *dolor* amet. (5cm)
Lorem *dolor* amet. (7cm)

2.3 Special application areas

2.3.1 The `tabbing` environment

```
\begin{tabbing}
col1 \hspace{1cm} \= col2 \hspace{1cm} \= col3 \hspace{1cm} \= col4 \\
\cloze{col1} \> \> \clozefix{col3} \\
\end{tabbing}
```

col1 col2 col3 col4
col1 *col3*

2.3.2 The picture environment

```
\begin{picture}(320,100)
\put(80,25){\cloze{Lorem}}
\put(160,50){\clozefix{ipsum}}
\put(240,75){\clozefil{dolor}}
\end{picture}
```

dolor

ipsum

Lorem

2.3.3 The tabular environment

```
\begin{tabular}{l c}
\cloze{Lorem} & \cloze{ipsum} \\
\clozefix{amet} & \clozefix{consectetur} \\
\cloze{sed} & \cloze{do} \\
\end{tabular}
```

Lorem *ipsum*
amet *consectetur*
sed *do*

3 Implementation

3.1 The file `cloze.sty`

This packages are used to build *cloze*:

3.1.1 Dependencies

The package fontspec is not necessarily required. When using LuaLaTeX it is good form to load it. Apart from this the package supplies helpful messages, when you compile a LuaLaTeX document with pdfLaTeX.

```
\RequirePackage{fontspec}
```

The package luatexbase allows to register multiple Lua callbacks.

```
\RequirePackage{luatexbase-mcb}
```

The package kvoptions takes the handling of the options.

```
\RequirePackage{kvoptions}
```

The package setspace is used by the environment `clozespace`.

```
\RequirePackage{setspace}
```

The package xcolor is required to colorize the text and the line of a gap.

```
\RequirePackage{xcolor}
```

The package xparse is used by the environment `clozebox`.

```
\RequirePackage{xparse}
```

The package stackengine is required by the command `\clozestrike{}{}`.

```
\RequirePackage{stackengine}
```

The package ulem is required by the command `\clozestrike{}{}`.

```
\RequirePackage{ulem}
\normalem
```

```
\RequirePackage{transparent}
```

Load the cloze lua module and put all return values in the variable `cloze`.

```
\directlua{
  cloze = require('cloze')
}
```

```
\newif\ifclozeshow\clozeshowtrue
```

3.1.2 Internal macros

`\cloze@set@to@global` Set the Lua variable `registry.is_global` to `true`. All options are then stored in the variable `registry.global_options`.

```
\def\cloze@set@to@global{%
 \directlua{cloze.set_is_global(true)}%
}
```

`\cloze@set@to@local` First unset the variable `registry.local_options`. Now set the Lua variable `registry.is_global` to `false`. All options are then stored in the variable `registry.local_options`.

```
\def\cloze@set@to@local{%
  \directlua{
    cloze.unset_local_options()
    cloze.set_is_global(false)
  }%
}
```

`\cloze@set@option` `\cloze@set@option` is a wrapper for the Lua function `registry.set_option`. `\cloze@set@option[`⟨*key*⟩`]{`⟨*value*⟩`}` sets a key ⟨*key*⟩ to the value ⟨*value*⟩.

```
\def\cloze@set@option[#1]#2{%
  \directlua{cloze.set_option('#1', '#2')}%
}
```

`\cloze@color` Convert a color definiton name to a PDF colorstack string, for example convert the color name `blue` to the colorstack string `0 0 1 rg 0 0 1 RG`. The macro definition `\cloze@color{blue}` builds itself the macro `\color@blue`, which expands to the PDF colorstack string. The colorstack string is necessary to generate a PDF colorstack whatsit.

```
\def\cloze@color#1{\csname\string\color@#1\endcsname}
```

`\cloze@set@local@options` This macro is used in all cloze commands to handle the optional arguments. First it sets the option storage to local and then it commits the options to the package *kvoptions* via the macro `\kvsetkeys{CLZ}{}`.

```
\def\cloze@set@local@options#1{%
  \cloze@set@to@local%
  \kvsetkeys{CLZ}{#1}%
}
```

`\cloze@start@marker` At the begining `\cloze@start@marker` registers the required Lua callbacks. Then it inserts a whatsit marker which marks the begin of a gap.

```
\def\cloze@start@marker#1{%
  \strut\directlua{
```

```
    cloze.register('#1')
    cloze.marker('#1', 'start')
  }%
}
```

`\cloze@stop@marker` `\cloze@stop@marker` inserts a whatsit marker that marks the end of gap.

```
\def\cloze@stop@marker#1{%
  \strut\directlua{
    cloze.marker('#1', 'stop')
  }%
}
```

`\cloze@margin` `\cloze@margin` surrounds a text in a gap with two `kerns`.

```
\def\cloze@margin#1{%
  \directlua{cloze.margin()}%
  #1%
  \directlua{cloze.margin()}%
}
```

3.1.3 Options

cloze offers key-value pairs to use as options. For processing the key-value pairs we use the package kvoptions. To make all key-value pairs accessibly to Lua code, we use the declaration `\define@key{`⟨*CLZ*⟩`}{`⟨*option*⟩`}[`⟨*/*⟩`]{`⟨*…*⟩`}`. This declaration comes from the package keyval.

At start all values are declared as global options. At the Lua side all values are now stored in the `registry.global_options` table.

```
\cloze@set@to@global
```

We use the abbreviation `CLZ` for *cloze* as family name and prefix.

```
\SetupKeyvalOptions{
  family=CLZ,
  prefix=CLZ@
}
```

3.1.3.1 `align`

Please read the section (→ 2.2.5) how to use the option `align`. `align` affects only the command `\clozefix` (→ 2.1.3).

```
\DeclareStringOption{align}
\define@key{CLZ}{align}[]{\cloze@set@option[align]{#1}}
```

3.1.3.2 `boxheight`

Please read the section (→ **??**) how to use the option `boxheight`. `boxheight` affects only the environment `\lozebox`. (→ 2.1.3).

```
\DeclareStringOption{boxheight}
\define@key{CLZ}{boxheight}[]{\cloze@set@option[boxheight]{#1}}
```

3.1.3.3 `boxwidth`

Please read the section (→ **??**) how to use the option `boxwidth`. `boxwidth` affects only the environment `\lozebox`. (→ 2.1.3).

```
\DeclareStringOption{boxwidth}
\define@key{CLZ}{boxwidth}[]{\cloze@set@option[boxwidth]{#1}}
```

3.1.3.4 `distance`

Please read the section (→ 2.2.8) how to use the option `distance`.

```
\DeclareStringOption{distance}
\define@key{CLZ}{distance}[]{\cloze@set@option[distance]{#1}}
```

3.1.3.5 `hide`

If the option `hide` appears in the commands, `hide` will be set to *true* and `show` to *false* on the Lua side. Please read the section (→ 2.2.9) how to use the option `hide`.

```
\DeclareVoidOption{hide}{%
  \clozeshowfalse%
  \cloze@set@option[hide]{true}%
  \cloze@set@option[show]{false}%
}
```

3.1.3.6 `linecolor`

Please read the section (→ 2.2.10) how to use the option `linecolor`.

```
\DeclareStringOption{linecolor}
\define@key{CLZ}{linecolor}[]{%
  \cloze@set@option[linecolor]{\cloze@color{#1}}%
  \cloze@set@option[linecolor_name]{#1}%
}
```

3.1.3.7 `margin`

Please read the section (→ 2.2.11) how to use the option `margin`.

```
\DeclareStringOption{margin}
\define@key{CLZ}{margin}[]{\cloze@set@option[margin]{#1}}
```

3.1.3.8 `show`

If the option `show` appears in the commands, `show` will be set to *true* and `true` to *false* on the Lua side. Please read the section (→ 2.2.9) how to use the option `show`.

```
\DeclareVoidOption{show}{%
  \clozeshowtrue%
  \cloze@set@option[show]{true}%
  \cloze@set@option[hide]{false}%
}
```

3.1.3.9 `spacing`

Please read the section (→ 2.2.12) how to use the option `spacing`.

```
\DeclareStringOption{spacing}
\define@key{CLZ}{spacing}[]{\cloze@set@option[spacing]{#1}}
```

3.1.3.10 `textcolor`

Please read the section (→ 2.2.10) how to use the option `textcolor`.

```
\DeclareStringOption{textcolor}
\define@key{CLZ}{textcolor}[]{%
  \cloze@set@option[textcolor]{\cloze@color{#1}}%
  \cloze@set@option[textcolor_name]{#1}%
}
```

3.1.3.11 `thickness`

Please read the section (→ 2.2.13) how to use the option `thickness`.

```
\DeclareStringOption{thickness}
\define@key{CLZ}{thickness}[]{\cloze@set@option[thickness]{#1}}
```

3.1.3.12 `width`

Please read the section (→ 2.2.14) how to use the option `width`. `width` affects only the command `\clozefix` (→ 2.1.3).

```
\DeclareStringOption{width}
\define@key{CLZ}{width}[]{\cloze@set@option[width]{#1}}

\ProcessKeyvalOptions{CLZ}
```

3.1.4 Public macros

All public macros are prefixed with `\cloze`.

`\clozeset` The usage of the command `\clozeset` is described in detail in section (→ 2.2.2).

```
\newcommand{\clozeset}[1]{%
  \cloze@set@to@global%
  \kvsetkeys{CLZ}{#1}%
}
```

`\clozereset` The usage of the command `\clozereset` is described in detail in section (→ 2.2.3).

```
\newcommand{\clozereset}{%
  \directlua{cloze.reset()}
}
```

`\clozeshow` The usage of the command `\clozeshow` is described in detail in section (→ 2.2.4).

```
\newcommand{\clozeshow}{%
  \clozeset{show}
}
```

`\clozehide` The usage of the command `\clozehide` is described in detail in section (→ 2.2.4).

```
\newcommand{\clozehide}{%
  \clozeset{hide}
}
```

`\clozefont` The usage of the command `\clozefont` is described in detail in section (→ 2.1.2).

```
\newcommand{\clozefont}{\itshape}
```

`\clozesetfont` The usage of the command `\clozesetfont` is described in detail in section (→ 2.1.2).

```
\newcommand{\clozesetfont}[1]{%
  \renewcommand{\clozefont}[1]{%
    #1%
  }%
}
```

`\cloze` The usage of the command `\cloze` is described in detail in section (→ 2.1.1).

```
\newcommand{\cloze}[2][]{%
  \cloze@set@local@options{#1}%
  \cloze@start@marker{basic}%
  {%
    \clozefont\relax%
    \cloze@margin{#2}%
  }%
  \cloze@stop@marker{basic}%
}
```

`\clozefix` The usage of the command `\clozefix` is described in detail in section (→ 2.1.3).

```
\newcommand{\clozefix}[2][]{%
  \cloze@set@local@options{#1}%
  \cloze@start@marker{fix}%
  {%
    \clozefont\relax%
    \cloze@margin{#2}%
  }%
  \cloze@stop@marker{fix}%
}
```

`\clozenol` The usage of the command `\clozenol` is described in detail in section (→ 2.1.4). TODO: Realize this macro with lua code, without ugly `\color{white}`. command.

```
\newcommand{\clozenol}[2][]{%
  {%
    \cloze@set@local@options{#1}%
    \clozefont\relax%
    \ifclozeshow%
      \color{\directlua{tex.print(cloze.get_value('textcolor_name'))}}%
    \else%
      \color{white}%
    \fi%
    #2%
  }%
}
```

`clozepar` The usage of the environment `clozepar` is described in detail in section (→ 2.1.7).

```
\newenvironment{clozepar}[1][]%
{%
  \par%
  \cloze@set@local@options{#1}%
  \cloze@start@marker{par}%
  \clozefont\relax%
}%
```

```
{%
  \cloze@stop@marker{par}%
  \par%
  \directlua{cloze.unregister('par')}%
}
```

`\cloze@get@value`

```
\newcommand{\cloze@get@value}[1]{%
  \directlua{
    tex.print(cloze.get_value('#1'))
  }%
}
```

`clozebox` The usage of the environment `clozebox` is described in detail in section (→ 2.1.8). TODO: Realize this macro with lua code, without ugly `\color{white}` command.

```
\newsavebox{\cloze@box}
\NewDocumentEnvironment{clozebox}{ s O{} +b}{%
  \cloze@set@local@options{#2}%
  \noindent%
  \begin{lrbox}{\cloze@box}%
  \directlua{
    local boxheight = cloze.get_value('boxheight')
    local boxwidth = cloze.get_value('boxwidth')
    if boxheight then
      tex.print('\\begin{minipage}[t][' .. boxheight .. '][t]{' .. boxwidth .. '}')
    else
      tex.print('\\begin{minipage}[t]{' .. boxwidth .. '}')
    end
  }
    \clozefont\relax%
    \ifclozeshow%
      \color{\directlua{tex.print(cloze.get_value('textcolor_name'))}}#3%
    \else%
      \color{white}#3%
    \fi%
  \end{minipage}%
  \end{lrbox}%
  \IfBooleanTF{#1}%
    {\usebox{\cloze@box}}%
    {\fbox{\usebox{\cloze@box}}}%
}{}
```

`clozespace` The usage of the environment `clozespace` is described in detail in section (→ 2.1.9). TODO: Realization without setspace package.

```
\newenvironment{clozespace}[1][]%
{%
  \cloze@set@local@options{#1}%
```

```
  \begin{spacing}{\directlua{tex.print(cloze.get_value('spacing'))}}%
}{\end{spacing}}
```

`\clozefil` The usage of the command `\clozefil` is described in detail in section (→ 2.1.5).

```
\newcommand{\clozefil}[2][]{%
  \cloze[#1]{#2}\clozelinefil[#1]%
}
```

`\clozeextend` TODO: Use node library to create kern nodes.

```
\newcommand{\clozeextend}[1][1]{%
  \directlua{
    local loop = #1
    for variable = 1, loop do
      tex.print(' \string\\hspace{1em} \string\\strut')
    end
  }
}
```

`\clozeline` The usage of the command `\clozeline` is described in detail in section (→ 2.1.10).

```
\newcommand{\clozeline}[1][]{%
  \cloze@set@local@options{#1}%
  \directlua{cloze.line()}%
}
```

`\clozelinefil` The usage of the command `\clozelinefil` is described in detail in section (→ 2.1.12).

```
\newcommand{\clozelinefil}[1][]{%
  \cloze@set@local@options{#1}%
  \strut%
  \directlua{cloze.linefil()}%
  \strut%
}
```

`\cloze@text@color`

```
\newcommand{\cloze@text@color}[1]{%
  \textcolor%
    {\directlua{tex.print(cloze.get_value('textcolor_name'))}}%
    {#1}%
}
```

`\cloze@strike@line`

```
\newcommand\cloze@strike@line{%
  \bgroup%
```

```
  \markoverwith{%
    \cloze@text@color{%
      \rule[0.5ex]{2pt}{1pt}%
    }%
  }%
  \ULon%
}
```

\clozestrike

```
\newcommand{\clozestrike}[3][]{%
  \cloze@set@local@options{#1}%
  \ifclozeshow%
    \stackengine%
      {\Sstackgap}% \Sstackgap or \Lstackgap or \stackgap or stacklength
      {\cloze@strike@line{#2}}% anchor
      {\cloze@text@color{\clozefont{}#3}}% item
      {O}% O or U
      {c}% \stackalignment or l or c or r
      {\quietstack}% \quietstack or T or F
      {T}% \useanchorwidth or T or F
      {\stacktype}% \stacktype or S or L
  \else%
    \stackengine%
      {\Sstackgap}% \Sstackgap or \Lstackgap or \stackgap or stacklength
      {#2}% anchor
      {\texttransparent{0}{\clozefont{}#3}}% item
      {O}% O or U
      {c}% \stackalignment or l or c or r
      {\quietstack}% \quietstack or T or F
      {T}% \useanchorwidth or T or F
      {\stacktype}% \stacktype or S or L
  \fi%
}
```

3.2 The file `cloze.lua`

3.2.0.1 Initialisation of the function tables

It is good form to provide some background informations about this Lua module.

```
if not modules then modules = { } end modules ['cloze'] = {
  version   = '1.4',
  comment   = 'cloze',
  author    = 'Josef Friedrich, R.-M. Huber',
  copyright = 'Josef Friedrich, R.-M. Huber',
  license   = 'The LaTeX Project Public License Version 1.3c 2008-05-04'
}
```

`nodex` is a abbreviation for *node eXtended*.

```
local nodex = {}
```

All values and functions, which are related to the option management, are stored in a table called `registry`.

```
local registry = {}
```

I didn't know what value I should take as `user_id`. Therefore I took my birthday and transformed it to a large number.

```
registry.user_id = 3121978
registry.storage = {}
registry.defaults = {
  ['align'] = 'l',
  ['boxheight'] = false,
  ['boxwidth'] = '\\linewidth',
  ['distance'] = '1.5pt',
  ['hide'] = false,
  ['linecolor'] = '0 0 0 rg 0 0 0 RG', -- black
  ['linecolor_name'] = 'black',
  ['margin'] = '3pt',
  ['resetcolor'] = '0 0 0 rg 0 0 0 RG', -- black
  ['resetcolor_name'] = 'black',
  ['show_text'] = true,
  ['show'] = true,
  ['spacing'] = '1.6',
  ['textcolor'] = '0 0 1 rg 0 0 1 RG', -- blue
  ['textcolor_name'] = 'blue', -- blue
  ['thickness'] = '0.4pt',
  ['width'] = '2cm',
}
registry.global_options = {}
registry.local_options = {}
```

All those functions are stored in the table `cloze` that are registered as callbacks to the pre and post linebreak filters.

```
local cloze = {}
```

In the status table are stored state information, which are necessary for the recursive cloze generation.

```
cloze.status = {}
```

The `base` table contains some basic functions. `base` is the only table of this Lua module that will be exported.

```
local base = {}
base.is_registered = {}
```

3.2.1 Node precessing (nodex)

All functions in this section are stored in a table called `nodex`. `nodex` is a abbreviation for *node eXtended*. The `nodex` table bundles all functions, which extend the built-in `node` library.

3.2.1.1 Color handling (color)

`create_colorstack`

Create a whatsit node of the subtype `pdf_colorstack`. `data` is a PDF colorstack string like `0 0 0 rg 0 0 0 RG`.

```
37 function nodex.create_colorstack(data)
38   if not data then
39     data = '0 0 0 rg 0 0 0 RG' -- black
40   end
41   local whatsit = node.new('whatsit', 'pdf_colorstack')
42   whatsit.stack = 0
43   whatsit.data = data
44   return whatsit
45 end
```

`create_color`

`nodex.create_color()` is a wrapper for the function `nodex.create_colorstack()`. It queries the current values of the options `linecolor` and `textcolor`. The argument `option` accepts the strings `line`, `text` and `reset`.

```
46 function nodex.create_color(option)
47   local data
48   if option == 'line' then
49     data = registry.get_value('linecolor')
50   elseif option == 'text' then
51     data = registry.get_value('textcolor')
52   elseif option == 'reset' then
53     data = nil
54   else
55     data = nil
56   end
57   return nodex.create_colorstack(data)
58 end
```

3.2.1.2 Line handling (line)

`create_line`

Create a rule node, which is used as a line for the cloze texts. The `depth` and the `height` of the rule are calculated form the options `thickness` and `distance`. The argument `width` must have the length unit *scaled points*.

```
59 function nodex.create_line(width)
60   local rule = node.new(node.id('rule'))
61   local thickness = tex.sp(registry.get_value('thickness'))
62   local distance = tex.sp(registry.get_value('distance'))
```

```
  rule.depth = distance + thickness
  rule.height = - distance
  rule.width = width
  return rule
end
```

`insert_list`

Insert a `list` of nodes after or before the `current`. The `head` argument is optional. In some edge cases it is unfortately necessary. if `head` is omitted the `current` node is used. The argument `position` can take the values `'after'` or `'before'`.

```
function nodex.insert_list(position, current, list, head)
  if not head then
    head = current
  end
  for i, insert in ipairs(list) do
    if position == 'after' then
      head, current = node.insert_after(head, current, insert)
    elseif position == 'before' then
      head, current = node.insert_before(head, current, insert)
    end
  end
  return current
end
```

`insert_line`

Enclose a rule node (cloze line) with two PDF colorstack whatsits. The first colorstack node dyes the line, the seccond resets the color.

Node list:

Variable name	Node type	Node subtype	Parameter
`n.color_line`	`whatsit`	`pdf_colorstack`	Line color
`n.line`	`rule`		`width`
`n.color_reset`	`whatsit`	`pdf_colorstack`	Reset color

```
function nodex.insert_line(current, width)
  return nodex.insert_list(
    'after',
    current,
    {
      nodex.create_color('line'),
      nodex.create_line(width),
      nodex.create_color('reset')
    }
  )
end
```

`write_line`

This function enclozes a rule node with color nodes as it the function `nodex.insert_line` does. In contrast to `nodex.insert_line` the three nodes are appended to TEX's 'current list'. They are not inserted in a node list, which is accessed by a Lua callback.

Node list:

Variable name	Node type	Node subtype	Parameter
-	whatsit	pdf_colorstack	Line color
-	rule		width
-	whatsit	pdf_colorstack	Reset color

```
92 function nodex.write_line()
93   node.write(nodex.create_color('line'))
94   node.write(nodex.create_line(tex.sp(registry.get_value('width'))))
95   node.write(nodex.create_color('reset'))
96 end
```

3.2.1.3 Handling of extendable lines (linefil)

create_linefil

This function creates a line which stretchs indefinitely in the horizontal direction.

```
97 function nodex.create_linefil()
98   local glue = node.new('glue')
99   glue.subtype = 100
100  glue.stretch = 65536
101  glue.stretch_order = 3
102  local rule = nodex.create_line(0)
103  rule.dir = 'TLT'
104  glue.leader = rule
105  return glue
106 end
```

write_linefil

The function `nodex.write_linefil` surrounds a indefinitely strechable line with color whatsits and puts it to TEX's 'current (node) list'.

```
107 function nodex.write_linefil()
108   node.write(nodex.create_color('line'))
109   node.write(nodex.create_linefil())
110   node.write(nodex.create_color('reset'))
111 end
```

3.2.1.4 Kern handling (kern)

create_kern

This function creates a kern node with a given width. The argument `width` had to be specified in scaled points.

```
112 function nodex.create_kern(width)
113   local kern = node.new(node.id('kern'))
114   kern.kern = width
115   return kern
116 end
```

strut_to_hlist

To make life easier: We add at the beginning of each hlist a strut. Now we can add line, color etc. nodes after the first node of a hlist not before - after is much

more easier.

```
function nodex.strut_to_hlist(hlist)
  local n = {} -- node
  n.head = hlist.head
  n.kern = nodex.create_kern(0)
  n.strut = node.insert_before(n.head, n.head, n.kern)
  hlist.head = n.head.prev
  return hlist, n.strut, n.head
end
```

`write_margin`

Write kern nodes to the current node list. This kern nodes can be used to build a margin.

```
function nodex.write_margin()
  local kern = nodex.create_kern(tex.sp(registry.get_value('margin')))
  node.write(kern)
end
```

`search_hlist`

Search for a `hlist` (subtype `line`). Return false, if no `hlist` can be found.

```
function nodex.search_hlist(head)
  while head do
    if head.id == node.id('hlist') and head.subtype == 1 then
      return nodex.strut_to_hlist(head)
    end
    head = head.next
  end
  return false
end
```

3.2.2 Option handling (registry)

The table `registry` bundels functions that deal with option handling.

3.2.2.1 Marker processing (marker)

A marker is a whatsit node of the subtype `user_defined`. A marker has two purposes:

1. Mark the begin and the end of a gap.
2. Store a index number, that points to a Lua table, which holds some additional data like the local options.

`create_marker`

We create a user defined whatsit node that can store a number (type = 100). In order to distinguish this node from other user defined whatsit nodes we set the `user_id` to a large number. We call this whatsit node a marker. The argument `index` is a number, which is associated to values in the `registry.storage` table.

```
function registry.create_marker(index)
  local marker = node.new('whatsit','user_defined')
  marker.type = 100 -- number
  marker.user_id = registry.user_id
  marker.value = index
  return marker
end
```

write_marker

Write a marker node to TEX's current node list. The argument `mode` accepts the string values `basic`, `fix` and `par`. The argument `position`. The argument `position` is either set to `start` or to `stop`.

```
function registry.write_marker(mode, position)
  local index = registry.set_storage(mode, position)
  local marker = registry.create_marker(index)
  node.write(marker)
end
```

is_marker

This functions checks if the given node `item` is a marker.

```
function registry.is_marker(item)
  if item.id == node.id('whatsit')
    and item.subtype == node.subtype('user_defined')
    and item.user_id == registry.user_id then
    return true
  else
    return false
  end
end
```

check_marker

This functions tests, whether the given node `item` is a marker. The argument `item` is a node. The argument `mode` accepts the string values `basic`, `fix` and `par`. The argument `position` is either set to `start` or to `stop`.

```
function registry.check_marker(item, mode, position)
  local data = registry.get_marker_data(item)
  if data and data.mode == mode and data.position == position then
    return true
  else
    return false
  end
end
```

get_marker

`registry.get_marker` returns the given marker. The argument `item` is a node. The argument `mode` accepts the string values `basic`, `fix` and `par`. The argument `position` is either set to `start` or to `stop`.

```
function registry.get_marker(item, mode, position)
  local out
  if registry.check_marker(item, mode, position) then
```

```
    out = item
  else
    out = false
  end
  if out and position == 'start' then
    registry.get_marker_values(item)
  end
  return out
end
```

get_marker_data

registry.get_marker_data tests whether the node item is a marker. The argument item is a node of unspecified type.

```
function registry.get_marker_data(item)
  if item.id == node.id('whatsit')
    and item.subtype == node.subtype('user_defined')
    and item.user_id == registry.user_id then
    return registry.get_storage(item.value)
  else
    return false
  end
end
```

get_marker_values

First this function saves the associatied values of a marker to the local options table. Second it returns this values. The argument marker is a whatsit node.

```
function registry.get_marker_values(marker)
  local data = registry.get_marker_data(marker)
  registry.local_options = data.values
  return data.values
end
```

remove_marker

This function removes a given whatsit marker. It only deletes a node, if a marker is given.

```
function registry.remove_marker(marker)
  if registry.is_marker(marker) then node.remove(marker, marker) end
end
```

3.2.2.2 Storage functions (storage)

get_index

registry.index is a counter. The functions registry.get_index() increases the counter by one and then returns it.

```
function registry.get_index()
  if not registry.index then
    registry.index = 0
  end
  registry.index = registry.index + 1
```

```
201   return registry.index
202 end
```

set_storage

`registry.set_storage()` stores the local options in the Lua table `registry.storage`. It returns a numeric index number. This index number is the key, where the local options in the Lua table are stored. The argument `mode` accepts the string values `basic`, `fix` and `par`.

```
203 function registry.set_storage(mode, position)
204   local index = registry.get_index()
205   local data = {
206     ['mode'] = mode,
207     ['position'] = position
208   }
209   data.values = registry.local_options
210   registry.storage[index] = data
211   return index
212 end
```

get_storage

The function `registry.get_storage()` retrieves values which belong to a whatsit marker. The argument `index` is a numeric value.

```
213 function registry.get_storage(index)
214   return registry.storage[index]
215 end
```

3.2.2.3 Option processing (option)

set_option

This function stores a value `value` and his associated key `key` either to the global (`registry.global_options`) or to the local (`registry.local_options`) option table. The global boolean variable `registry.local_options` controls in which table the values are stored.

```
216 function registry.set_option(key, value)
217   if value == '' or value == '\\color@ ' then
218     return false
219   end
220   if registry.is_global == true then
221     registry.global_options[key] = value
222   else
223     registry.local_options[key] = value
224   end
225 end
```

set_is_global

`registry.set_is_global()` sets the variable `registry.is_global` to the value `value`. It is intended, that the variable takes boolean values.

```
226 function registry.set_is_global(value)
227   registry.is_global = value
```

```
228 end
```

unset_local_options

This function unsets the local options.

```
229 function registry.unset_local_options()
230   registry.local_options = {}
231 end
```

unset_global_options

`registry.unset_global_options` empties the global options storage.

```
232 function registry.unset_global_options()
233   registry.global_options = {}
234 end
```

get_value

Retrieve a value from a given key. First search for the value in the local options, then in the global options. If both option storages are empty, the default value will be returned.

```
235 function registry.get_value(key)
236   if registry.has_value(registry.local_options[key]) then
237     return registry.local_options[key]
238   end
239   if registry.has_value(registry.global_options[key]) then
240     return registry.global_options[key]
241   end
242   return registry.defaults[key]
243 end
```

get_value_show

The function `registry.get_value_show()` returns the boolean value `true` if the option `show` is true. In contrast to the function `registry.get_value()` it converts the string value 'true' to a boolean value.

```
244 function registry.get_value_show()
245   if
246     registry.get_value('show') == true
247   or
248     registry.get_value('show') == 'true'
249   then
250     return true
251   else
252     return false
253   end
254 end
```

has_value

This function tests whether the value `value` is not empty and has a value.

```
255 function registry.has_value(value)
256   if value == nil or value == '' or value == '\\color@ ' then
257     return false
258   else
```

```
    return true
  end
end
```

`get_defaults`

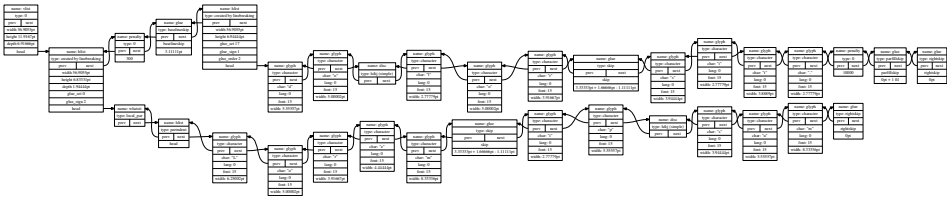
`registry.get_defaults(option)` returns a the default value of the given option.

```
function registry.get_defaults(option)
  return registry.defaults[option]
end
```

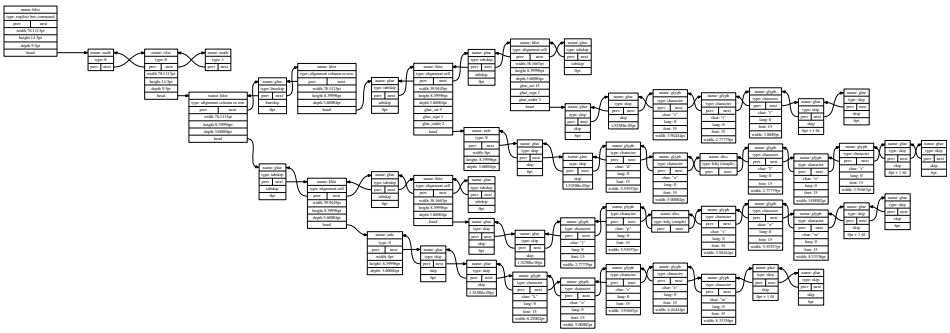
3.2.3 Assembly to cloze texts (cloze)

Some graphics for better understanding of the node tree:

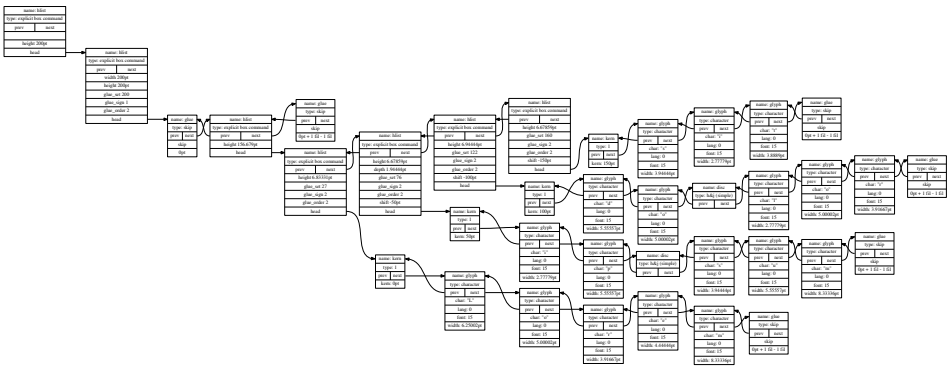
3.2.3.1 Paragraph



3.2.3.2 Tabular environment



3.2.3.3 Picture environment



basic_make

The function `cloze.basic_make()` makes one gap. The argument `start` is the node, where the gap begins. The argument `stop` is the node, where the gap ends.

```
function cloze.basic_make(start, stop)
  local n = {}
  local l = {}
  n.head = start
  if not start or not stop then
    return
  end
  n.start = start
  n.stop = stop
  l.width = node.dimensions(
    cloze.status.hlist.glue_set,
    cloze.status.hlist.glue_sign,
    cloze.status.hlist.glue_order,
    n.start,
    n.stop
  )
  n.line = nodex.insert_line(n.start, l.width)
  n.color_text = nodex.insert_list('after', n.line, {nodex.create_color('text')})
  if registry.get_value_show() then
    nodex.insert_list('after', n.color_text, {nodex.create_kern(-l.width)})
    nodex.insert_list('before', n.stop, {nodex.create_color('reset')}, n.head)
  else
    n.line.next = n.stop.next
    n.stop.prev = n.line.prev
  end
```

I some edge cases the lua callbacks get fired up twice. After the cloze has been created, the start and stop whatsit markers can be deleted.

```
  registry.remove_marker(n.start)
  registry.remove_marker(n.stop)
end
```

basic_search_stop

Search for a stop marker.

```
293 function cloze.basic_search_stop(head)
294   local stop
295   while head do
296     cloze.status.continue = true
297     stop = head
298     if registry.check_marker(stop, 'basic', 'stop') then
299       cloze.status.continue = false
300       break
301     end
302     head = head.next
303   end
304   return stop
305 end
```

basic_search_start

Search for a start marker. Also begin a new cloze, if the boolean value `cloze.status.continue` is true. The knowledge of the last hlist node is a requirement to begin a cloze.

```
306 function cloze.basic_search_start(head)
307   local start
308   local stop
309   local n = {}
310   if cloze.status.continue then
311     n.hlist = nodex.search_hlist(head)
312     if n.hlist then
313       cloze.status.hlist = n.hlist
314       start = cloze.status.hlist.head
315     end
316   elseif registry.check_marker(head, 'basic', 'start') then
317     start = head
318   end
319   if start then
320     stop = cloze.basic_search_stop(start)
321     cloze.basic_make(start, stop)
322   end
323 end
```

basic_recursion

Parse recursivley the node tree. Start and stop markers can be nested deeply into the node tree.

```
324 function cloze.basic_recursion(head)
325   while head do
326     if head.head then
327       cloze.status.hlist = head
328       cloze.basic_recursion(head.head)
329     else
330       cloze.basic_search_start(head)
```

```
331     end
332       head = head.next
333   end
334 end
```

basic

The corresponding LaTeX command to this lua function is `\cloze` (→ 2.1.1). The argument `head` is the head node of a node list.

```
335 function cloze.basic(head)
336   cloze.status.continue = false
337   cloze.basic_recursion(head)
338   return head
339 end
```

fix_length

Calculate the length of the whitespace before (`l.kern_start`) and after (`l.kern_stopt`) the text.

```
340 function cloze.fix_length(start, stop)
341   local l = {}
342   l.width = tex.sp(registry.get_value('width'))
343   l.text_width = node.dimensions(start, stop)
344   l.align = registry.get_value('align')
345   if l.align == 'right' then
346     l.kern_start = - l.text_width
347     l.kern_stop = 0
348   elseif l.align == 'center' then
349     l.half = (l.width - l.text_width) / 2
350     l.kern_start = - l.half - l.text_width
351     l.kern_stop = l.half
352   else
353     l.kern_start = - l.width
354     l.kern_stop = l.width - l.text_width
355   end
356   return l.width, l.kern_start, l.kern_stop
357 end
```

fix_make

The function `cloze.fix_make` generates a gap of fixed width.

Node lists
Show text:

Variable name	Node type	Node subtype	Parameter
`n.start`	`whatsit`	`user_definded`	`index`
`n.line`	`rule`		`l.width`
`n.kern_start`	`kern`		Depends on `align`
`n.color_text`	`whatsit`	`pdf_colorstack`	Text color
	`glyphs`		Text to show
`n.color_reset`	`whatsit`	`pdf_colorstack`	Reset color
`n.kern_stop`	`kern`		Depends on `align`
`n.stop`	`whatsit`	`user_definded`	`index`

Hide text:

Variable name	Node type	Node subtype	Parameter
n.start	whatsit	user_definded	index
n.line	rule		l.width
n.stop	whatsit	user_definded	index

The argument start is the node, where the gap begins. The argument stop is the node, where the gap ends.

```
358 function cloze.fix_make(start, stop)
359   local l = {} -- length
360   local n = {} -- node
361   l.width, l.kern_start, l.kern_stop = cloze.fix_length(start, stop)
362   n.line = nodex.insert_line(start, l.width)
363   if registry.get_value_show() then
364     nodex.insert_list(
365       'after',
366       n.line,
367       {
368         nodex.create_kern(l.kern_start),
369         nodex.create_color('text')
370       }
371     )
372     nodex.insert_list(
373       'before',
374       stop,
375       {
376         nodex.create_color('reset'),
377         nodex.create_kern(l.kern_stop)
378       },
379       start
380     )
381   else
382     n.line.next = stop.next
383   end
384   registry.remove_marker(start)
385   registry.remove_marker(stop)
386 end
```

fix_recursion

Function to recurse the node list and search after marker. head is the head node of a node list.

```
387 function cloze.fix_recursion(head)
388   local n = {} -- node
389   n.start, n.stop = false
390   while head do
391     if head.head then
392       cloze.fix_recursion(head.head)
393     else
394       if not n.start then
395         n.start = registry.get_marker(head, 'fix', 'start')
```

```
      end
      if not n.stop then
        n.stop = registry.get_marker(head, 'fix', 'stop')
      end
      if n.start and n.stop then
        cloze.fix_make(n.start, n.stop)
        n.start, n.stop = false
      end
    end
    head = head.next
  end
end
```

fix

The corresponding LaTeX command to this Lua function is `\clozefix` (→ 2.1.3). The argument `head` is the head node of a node list.

```
function cloze.fix(head)
  cloze.fix_recursion(head)
  return head
end
```

par

The corresponding LaTeX environment to this lua function is `clozepar` (→ 2.1.7).

Node lists
Show text:

Variable name	Node type	Node subtype	Parameter
`n.strut`	`kern`		width = 0
`n.line`	`rule`		`l.width` (Width from hlist)
`n.kern`	`kern`		`-l.width`
`n.color_text`	`whatsit`	`pdf_colorstack`	Text color
	`glyphs`		Text to show
`n.tail`	`glyph`		Last glyph in hlist
`n.color_reset`	`whatsit`	`pdf_colorstack`	Reset color

Hide text:

Variable name	Node type	Node subtype	Parameter
`n.strut`	`kern`		width = 0
`n.line`	`rule`		`l.width` (Width from hlist)

The argument `head` is the head node of a node list.

```
function cloze.par(head)
  local l = {} -- length
  local n = {} -- node
  for hlist in node.traverse_id(node.id('hlist'), head) do
    for whatsit in node.traverse_id(node.id('whatsit'), hlist.head) do
      registry.get_marker(whatsit, 'par', 'start')
    end
    l.width = hlist.width
    hlist, n.strut, n.head = nodex.strut_to_hlist(hlist)
    n.line = nodex.insert_line(n.strut, l.width)
    if registry.get_value_show() then
      nodex.insert_list(
```

```
424         'after',
425         n.line,
426         {
427           nodex.create_kern(-l.width),
428           nodex.create_color('text')
429         }
430       )
431       nodex.insert_list(
432         'after',
433         node.tail(head),
434         {nodex.create_color('reset')}
435       )
436     else
437       n.line.next = nil
438     end
439   end
440   return head
441 end
```

3.2.4 Basic module functions (base)

`register` The `base` table contains functions which are published to the `cloze.sty` file. This function registers the functions `cloze.par`, `cloze.basic` and `cloze.fix` the Lua callbacks. `cloze.par` and `cloze.basic` are registered to the callback `post_linebreak_filter` and `cloze.fix` to the callback `pre_linebreak_filter`. The argument `mode` accepts the string values `basic`, `fix` and `par`.

```
442 function base.register(mode)
443   local basic
444   if mode == 'par' then
445     luatexbase.add_to_callback(
446       'post_linebreak_filter',
447       cloze.par,
448       mode
449     )
450     return true
451   end
452   if not base.is_registered[mode] then
453     if mode == 'basic' then
454       luatexbase.add_to_callback(
455         'post_linebreak_filter',
456         cloze.basic,
457         mode
458       )
459     elseif mode == 'fix' then
460       luatexbase.add_to_callback(
461         'pre_linebreak_filter',
462         cloze.fix,
463         mode
```

```
      )
    else
      return false
    end
    base.is_registered[mode] = true
  end
end
```

unregister

`base.unregister(mode)` deletes the registered functions from the Lua callbacks. The argument `mode` accepts the string values `basic`, `fix` and `par`.

```
function base.unregister(mode)
  if mode == 'basic' then
    luatexbase.remove_from_callback('post_linebreak_filter', mode)
  elseif mode == 'fix' then
    luatexbase.remove_from_callback('pre_linebreak_filter', mode)
  else
    luatexbase.remove_from_callback('post_linebreak_filter', mode)
  end
end
```

Publish some functions to the `cloze.sty` file.

```
base.linefil = nodex.write_linefil
base.line = nodex.write_line
base.margin = nodex.write_margin
base.set_option = registry.set_option
base.set_is_global = registry.set_is_global
base.unset_local_options = registry.unset_local_options
base.reset = registry.unset_global_options
base.get_defaults = registry.get_defaults
base.get_value = registry.get_value
base.marker = registry.write_marker

return base
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

T

U