

The *cloze* package*

Josef Friedrich

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

v1.1 from 2016/06/13

*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	4
2	Usage	4
2.1	The commands and environments	4
2.1.1	<code>\cloze</code>	4
2.1.2	<code>\clozesetfont</code>	5
2.1.3	<code>\clozefix</code>	5
2.1.4	<code>\clozefil</code>	5
2.1.5	<code>clozepar</code>	5
2.1.6	<code>\clozeline</code>	6
2.1.7	<code>\clozelinefil</code>	6
2.2	The options	6
2.2.1	Local and global options	6
2.2.2	<code>\clozeset</code>	6
2.2.3	<code>\clozereset</code>	7
2.2.4	<code>\clozeshow</code> and <code>\clozhide</code>	7
2.2.5	<code>align</code>	7
2.2.6	<code>distance</code>	8
2.2.7	<code>hide</code> and <code>show</code>	8
2.2.8	<code>linecolor</code> and <code>textcolor</code>	8
2.2.9	<code>margin</code>	8
2.2.10	<code>thickness</code>	9
2.2.11	<code>width</code>	9
2.3	Special application areas	9
2.3.1	The <code>picture</code> environment	9
2.3.2	The <code>tabular</code> environment	10
3	Implementation	11
3.1	The file <code>cloze.sty</code>	11
3.1.1	Internal macros	11
3.1.2	Options	12
3.1.2.1	<code>align</code>	13
3.1.2.2	<code>distance</code>	13
3.1.2.3	<code>hide</code>	13
3.1.2.4	<code>linecolor</code>	13
3.1.2.5	<code>margin</code>	14
3.1.2.6	<code>show</code>	14
3.1.2.7	<code>textcolor</code>	14
3.1.2.8	<code>thickness</code>	14
3.1.2.9	<code>width</code>	14
3.1.3	Public macros	15
3.2	The file <code>cloze.lua</code>	17
3.2.0.1	Initialisation of the function tables	17

3.2.1	Node precessing (nodex)	18
3.2.1.1	Color handling (color)	18
3.2.1.2	Line handling (line)	19
3.2.1.3	Handling of extendable lines (linefil)	20
3.2.1.4	Kern handling (kern)	20
3.2.2	Option handling (registry)	21
3.2.2.1	Marker processing (marker)	21
3.2.2.2	Storage functions (storage)	23
3.2.2.3	Option processing (option)	23
3.2.3	Assembly to cloze texts (cloze)	25
3.2.4	Node trees	26
3.2.4.1	Paragraph	26
3.2.4.2	Tabular environment	26
3.2.4.3	Picture environment	27
3.2.5	Basic module functions (base)	30

1 Introduction

cloze is a L^AT_EX package to generate cloze texts. It uses the capabilities of the modern T_EX engine *LuaT_EX*. Therefore, you must use LuaL^AT_EX 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` (→ 2.2.7) options.

Lorem ipsum dolor sit amet, consectetur adipiscing 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 three commands and one environment to generate cloze texts: `\cloze`, `\clozefix`, `\clozefil` and `clozepar`.

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` (→ 2.2.9).

Lorem ipsum dolor sit amet, consectetur adipiscing 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` (→ 2.1.5).

Lorem ipsum dolor sit amet, consectetur adipiscing 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 Oberkommissarin auch Fillialleiterin kurz Oberkommissarin Unternehmenssteuerfortentwicklungsgesetz Fillialleiterin Metzgermeisterin in Abteilungsleiterin der Oberkommissarin Hochleistungsflüssigkeitschromatographie Fillialleiterin Kürze Unternehmenssteuerfortentwicklungsgesetz Metzgermeisterin liegt Abteilungsleiterin die Metzgermeisterin Abteilungsleiterin Würze Oberkommissarin

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 (\rightarrow 2.2.8).

`\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 concerning the width 2cm.

Lorem ipsum dolor sit amet:
 1. consectetur
 2. adipiscing
 3. elit
 sed do eiusmod.

2.1.4 \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 adipiscing elit, sed do eiusmod.
 Ut enim ad minim veniam exercitation.

2.1.5 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. <i>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.</i> Excepteur sint occaecat cupidatat non proident.
--

2.1.6 `\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 <u> <i>minim</i> </u> minim veniam,
--

2.1.7 `\clozelinefil`

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

Lorem_____

2.2 The options

2.2.1 Local and global options

The *cloze* package distinguishes between *local* and *global* options. Besides the possibility 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 <u><i>ipsum</i></u> dolor Lorem <u><i>ipsum</i></u> 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
```

Lorem _____ amet, consectetur _____ elit.

```
\clozeshow
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

2.2.5 align

[`align=<left/center/right>`]: Only the macro `\clozefix` (\rightarrow 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.

<u>Lorem ipsum</u>	(left)
<u> Lorem ipsum </u>	(center)
<u> Lorem ipsum </u>	(right)

2.2.6 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.

<u> Lorem ipsum dolor sit amet. </u>	(1.5pt)
<u> Lorem ipsum dolor sit amet. </u>	(3pt)
<u> Lorem ipsum dolor sit amet. </u>	(-3pt)

2.2.7 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		consecetur		elit.	(hide)
Lorem ipsum	<u>dolor sit amet</u>	consecetur	<u>adipiscing</u>	elit.	(show)
Lorem ipsum		consecetur		elit.	(show,hide)
Lorem ipsum	<u>dolor sit amet</u>	consecetur	<u>adipiscing</u>	elit.	(hide,show)

2.2.8 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}
```

<u> Lorem ipsum dolor sit amet, consecetur </u>	(myclozecolor)
<u> Lorem ipsum dolor sit amet, consecetur </u>	(red)
<u> Lorem ipsum dolor sit amet, consecetur </u>	(green)

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

<u> Lorem ipsum dolor sit amet, consecetur </u>	(myclozecolor)
<u> Lorem ipsum dolor sit amet, consecetur </u>	(red)
<u> Lorem ipsum dolor sit amet, consecetur </u>	(green)

2.2.9 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 <u>dolor</u> sit amet. Lorem ipsum <u> dolor </u> sit amet. Lorem ipsum <u> dolor </u> sit amet. Lorem ipsum <u> dolor </u> sit amet. Lorem ipsum <u>dolor</u> sit amet.	(0pt) (5mm) (1cm) (6mm) (-4pt)
--	--

		<i>dolor</i>
	<i>ipsum</i>	
<i>Lorem</i>		

2.3.2 The tabular environment

<i>Lorem</i>	<i>ipsum</i>	<i>dolor</i>	<i>sit</i>
<i>amet</i>	<i>consectetur</i>	<i>adipiscing</i>	<i>elit</i>
<i>sed</i>	<i>do</i>	<i>eiusmod</i>	<i>tempor</i>

3 Implementation

3.1 The file `cloze.sty`

This four packages are used to build *cloze*:

- `fontspec` is not necessarily required. When using Lua^AT_EX it is good form to load it. Apart from this the package supplies helpful messages, when you compile a Lua^AT_EX document with pdf^AT_EX.
- `luatexbase` allows to register multiple Lua callbacks.
- `kvoptions` takes the handling of the options.
- `xcolor` is required to colorize the text and the line of a gap.

```
26 \RequirePackage{fontspec,luatexbase-mcb,kvoptions,xcolor}
```

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

```
27 \directlua{
28   cloze = require('cloze')
29 }
```

3.1.1 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`.

```
30 \def\cloze@set@to@global{%
31   \directlua{cloze.set_is_global(true)}%
32 }
```

`\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`.

```
33 \def\cloze@set@to@local{%
34   \directlua{
35     cloze.unset_local_options()
36     cloze.set_is_global(false)
37   }%
38 }
```

`\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>*.

```
39 \def\cloze@set@option[#1]#2{%
40   \directlua{cloze.set_option('#1', '#2')}%
41 }
```

`\cloze@color` Convert a color definition 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.

```
42 \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}{}`.

```
43 \def\cloze@set@local@options#1{%
44   \cloze@set@to@local%
45   \kvsetkeys{CLZ}{#1}%
46 }
```

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

```
47 \def\cloze@start@marker#1{%
48   \strut\directlua{
49     cloze.register('#1')
50     cloze.marker('#1', 'start')
51   }%
52 }
```

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

```
53 \def\cloze@stop@marker#1{%
54   \strut\directlua{
55     cloze.marker('#1', 'stop')
56   }%
57 }
```

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

```
58 \def\cloze@margin#1{%
59   \directlua{cloze.margin()}%
60   #1%
61   \directlua{cloze.margin()}%
62 }
```

3.1.2 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.

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

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

```
64 \SetupKeyvalOptions{
65   family=CLZ,
66   prefix=CLZ@
67 }
```

3.1.2.1 align

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

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

3.1.2.2 distance

Please read the section (→ 2.2.6) how to use the option `distance`.

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

3.1.2.3 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 (→ 2.2.7) how to use the option `hide`.

```
72 \DeclareVoidOption{hide}{%
73   \cloze@set@option[hide]{true}%
74   \cloze@set@option[show]{false}%
75 }
```

3.1.2.4 linecolor

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

```
76 \DeclareStringOption{linecolor}
77 \define@key{CLZ}{linecolor}[]{%
78   \cloze@set@option[linecolor]{\cloze@color{#1}}%
79 }
```

3.1.2.5 margin

Please read the section (→ 2.2.9) how to use the option `margin`.

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

3.1.2.6 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 (→ 2.2.7) how to use the option `show`.

```
82 \DeclareVoidOption{show}{%
83   \cloze@set@option[show]{true}%
84   \cloze@set@option[hide]{false}%
85 }
```

3.1.2.7 textcolor

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

```
86 \DeclareStringOption{textcolor}
87 \define@key{CLZ}{textcolor}[]{%
88   \cloze@set@option[textcolor]{\cloze@color{#1}}%
89 }
```

3.1.2.8 thickness

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

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

3.1.2.9 width

Please read the section (→ 2.2.11) how to use the option `width`. `width` affects only the command `\clozefix` (→ 2.1.3).

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

94 \ProcessKeyvalOptions{CLZ}
```

3.1.3 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](#)).

```
95 \newcommand{\clozeset}[1]{%
96   \cloze@set@to@global%
97   \kvsetkeys{CLZ}{#1}%
98 }
```

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

```
99 \newcommand{\clozereset}{%
100   \directlua{cloze.reset()}
101 }
```

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

```
102 \newcommand{\clozeshow}{%
103   \clozeset{show}
104 }
```

`\clozhide` The usage of the command `\clozhide` is described in detail in section (→ [2.2.4](#)).

```
105 \newcommand{\clozhide}{%
106   \clozeset{hide}
107 }
```

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

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

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

```
109 \newcommand{\clozesetfont}[1]{%
110   \renewcommand{\clozefont}[1]{%
111     #1%
112   }%
113 }
```

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

```
114 \newcommand{\cloze}[2][ ]{%
115   \cloze@set@local@options{#1}%
116   \cloze@start@marker{basic}%
117   {%
118     \clozefont\relax%
119     \cloze@margin{#2}%
120   }
121 }
```

```

120 }%
121 \cloze@stop@marker{basic}%
122 }

```

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

```

123 \newcommand{\clozefix}[2] [] {%
124   \cloze@set@local@options{#1}%
125   \cloze@start@marker{fix}%
126   {%
127     \clozefont\relax%
128     \cloze@margin{#2}%
129   }%
130   \cloze@stop@marker{fix}%
131 }

```

`clozepar` The usage of the environment `clozepar` is described in detail in section (→ 2.1.5).

```

132 \newenvironment{clozepar}[1] [] {%
133   {%
134     \par%
135     \cloze@set@local@options{#1}%
136     \cloze@start@marker{par}%
137     \clozefont\relax%
138   }%
139   {%
140     \cloze@stop@marker{par}%
141     \par%
142     \directlua{cloze.unregister('par')}%
143   }

```

`\clozefil` The usage of the command `\clozefil` is described in detail in section (→ 2.1.4).

```

144 \newcommand{\clozefil}[2] [] {%
145   \cloze[#1]{#2}\clozelinefil[#1]%
146 }

```

`\clozeline` The usage of the command `\clozeline` is described in detail in section (→ 2.1.6).

```

147 \newcommand{\clozeline}[1] [] {%
148   \cloze@set@local@options{#1}%
149   \directlua{cloze.line()}%
150 }

```

`\clozelinefil` The usage of the command `\clozelinefil` is described in detail in section (→ 2.1.7).

```

151 \newcommand{\clozelinefil}[1] [] {%
152   \cloze@set@local@options{#1}%

```



```

153 \strut%
154 \directlua{cloze.linefil()}%
155 \strut%
156 }

```

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.

```

1 if not modules then modules = { } end modules ['cloze'] = {
2   version   = '0.1',
3   comment   = 'cloze',
4   author    = 'Josef Friedrich, R.-M. Huber',
5   copyright = 'Josef Friedrich, R.-M. Huber',
6   license   = 'The LaTeX Project Public License Version 1.3c 2008-05-04'
7 }

```

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

```

8 local nodex = {}

```

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

```

9 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.

```

10 registry.user_id = 3121978
11 registry.storage = {}
12 registry.defaults = {
13   ['align'] = 'l',
14   ['distance'] = '1.5pt',
15   ['hide'] = false,
16   ['linecolor'] = '0 0 0 rg 0 0 0 RG', -- black
17   ['margin'] = '3pt',
18   ['resetcolor'] = '0 0 0 rg 0 0 0 RG', -- black
19   ['show_text'] = true,
20   ['show'] = true,
21   ['textcolor'] = '0 0 1 rg 0 0 1 RG', -- blue
22   ['thickness'] = '0.4pt',
23   ['width'] = '2cm',
24 }
25 registry.global_options = {}
26 registry.local_options = {}

```

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

```
27 local cloze = {}
```

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

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

3.2.1 Node preprocessing (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`.

```
30 function nodex.create_colorstack(data)
31   if not data then
32     data = '0 0 0 rg 0 0 0 RG' -- black
33   end
34   local whatsit = node.new('whatsit', 'pdf_colorstack')
35   whatsit.stack = 0
36   whatsit.data = data
37   return whatsit
38 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`.

```
39 function nodex.create_color(option)
40   local data
41   if option == 'line' then
42     data = registry.get_value('linecolor')
43   elseif option == 'text' then
44     data = registry.get_value('textcolor')
45   elseif option == 'reset' then
46     data = nil
47   else
48     data = nil
49   end
50   return nodex.create_colorstack(data)
51 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 from the options **thickness** and **distance**. The argument **width** must have the length unit *scaled points*.

```
52 function nodex.create_line(width)
53   local rule = node.new(node.id('rule'))
54   local thickness = tex.sp(registry.get_value('thickness'))
55   local distance = tex.sp(registry.get_value('distance'))
56   rule.depth = distance + thickness
57   rule.height = - distance
58   rule.width = width
59   return rule
60 end
```

`insert_line`

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

```
61 function nodex.insert_line(head, current, width)
62   local n = {} -- node
63   n.color_line = nodex.create_color('line')
64   head, n.color_line = node.insert_after(head, current, n.color_line)
65   n.line = nodex.create_line(width)
66   head, n.line = node.insert_after(head, n.color_line, n.line)
67   n.color_reset = nodex.create_color('reset')
68   return node.insert_after(head, n.line, n.color_reset)
69 end
```

`write_line`

This function encloses 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

```
70 function nodex.write_line()
71   node.write(nodex.create_color('line'))
72   node.write(nodex.create_line(tex.sp(registry.get_value('width'))))
73   node.write(nodex.create_color('reset'))
74 end
```

3.2.1.3 Handling of extendable lines (linefil)

create_linefil

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

```
75 function nodex.create_linefil()
76   local glue = node.new('glue')
77   glue.subtype = 100
78   glue.stretch = 65536
79   glue.stretch_order = 3
80   local rule = nodex.create_line(0)
81   rule.dir = 'TLT'
82   glue.leader = rule
83   return glue
84 end
```

write_linefil

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

```
85 function nodex.write_linefil()
86   node.write(nodex.create_color('line'))
87   node.write(nodex.create_linefil())
88   node.write(nodex.create_color('reset'))
89 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.

```
90 function nodex.create_kern(width)
91   local kern = node.new(node.id('kern'))
92   kern.kern = width
93   return kern
94 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.

```
95 function nodex.strut_to_hlist(hlist)
96   local n = {} -- node
97   n.head = hlist.head
98   n.kern = nodex.create_kern(0)
99   n.strut = node.insert_before(n.head, n.head, n.kern)
100   hlist.head = n.head.prev
101   return hlist, n.strut, n.head
102 end
```

write_margin

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

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

3.2.2 Option handling (registry)

The table `registry` bundles 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.

```
107 function registry.create_marker(index)
108   local marker = node.new('whatsit','user_defined')
109   marker.type = 100 -- number
110   marker.user_id = registry.user_id
111   marker.value = index
112   return marker
113 end
```

`write_marker`

Write a marker node to T_EX'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`.

```
114 function registry.write_marker(mode, position)
115   local index = registry.set_storage(mode, position)
116   local marker = registry.create_marker(index)
117   node.write(marker)
118 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`.

```

119 function registry.check_marker(item, mode, position)
120   local data = registry.get_marker_data(item)
121   if data and data.mode == mode and data.position == position then
122     return true
123   else
124     return false
125   end
126 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**.

```

127 function registry.get_marker(item, mode, position)
128   local out
129   if registry.check_marker(item, mode, position) then
130     out = item
131   else
132     out = false
133   end
134   if out and position == 'start' then
135     registry.get_marker_values(item)
136   end
137   return out
138 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.

```

139 function registry.get_marker_data(item)
140   if item.id == node.id('whatsit')
141     and item.subtype == node.subtype('user_defined')
142     and item.user_id == registry.user_id then
143     return registry.get_storage(item.value)
144   else
145     return false
146   end
147 end

```

get_marker_values

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

```

148 function registry.get_marker_values(marker)
149   local data = registry.get_marker_data(marker)
150   registry.local_options = data.values
151   return data.values
152 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.

```
153 function registry.get_index()
154   if not registry.index then
155     registry.index = 0
156   end
157   registry.index = registry.index + 1
158   return registry.index
159 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`.

```
160 function registry.set_storage(mode, position)
161   local index = registry.get_index()
162   local data = {
163     ['mode'] = mode,
164     ['position'] = position
165   }
166   data.values = registry.local_options
167   registry.storage[index] = data
168   return index
169 end
```

`get_storage`

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

```
170 function registry.get_storage(index)
171   return registry.storage[index]
172 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.

```
173 function registry.set_option(key, value)
174   if value == '' or value == '\\color@ ' then
175     return false
176   end
177   if registry.is_global == true then
178     registry.global_options[key] = value
```

```

179     else
180         registry.local_options[key] = value
181     end
182 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.

183 function registry.set_is_global(value)
184     registry.is_global = value
185 end
unset_local_options
    This function unsets the local options.

186 function registry.unset_local_options()
187     registry.local_options = {}
188 end
unset_global_options
    registry.unset_global_options empties the global options storage.

189 function registry.unset_global_options()
190     registry.global_options = {}
191 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.

192 function registry.get_value(key)
193     if registry.has_value(registry.local_options[key]) then
194         return registry.local_options[key]
195     end
196     if registry.has_value(registry.global_options[key]) then
197         return registry.global_options[key]
198     end
199     return registry.defaults[key]
200 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.

201 function registry.get_value_show()
202     if
203         registry.get_value('show') == true
204     or
205         registry.get_value('show') == 'true'
206     then
207         return true
208     else

```



```

209     return false
210 end
211 end

```

has_value

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

```

212 function registry.has_value(value)
213   if value == nil or value == '' or value == '\\color@ ' then
214     return false
215   else
216     return true
217   end
218 end

```

get_defaults

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

```

219 function registry.get_defaults(option)
220   return registry.defaults[option]
221 end

```

3.2.3 Assembly to cloze texts (cloze)

basic_make

The function **cloze.basic_make()** makes one gap. The argument **head** is the head node of a node list. The argument **start** is the node, where the gap begins. The argument **stop** is the node, where the gap ends.

```

222 function cloze.basic_make(head, hlist, start, stop)
223   local n = {}
224   local l = {}
225   if not start or not stop then
226     return
227   end
228   n.start = start
229   n.stop = stop
230   l.width = node.dimensions(
231     hlist.glue_set,
232     hlist.glue_sign,
233     hlist.glue_order,
234     n.start,
235     n.stop
236   )
237   head, n.line = nodex.insert_line(head, n.start, l.width)
238   n.color_text = nodex.create_color('text')
239   head, n.color_text = node.insert_after(
240     hlist.head,
241     n.line,
242     n.color_text
243   )
244   if registry.get_value_show() then
245     n.kern = nodex.create_kern(-l.width)

```

```

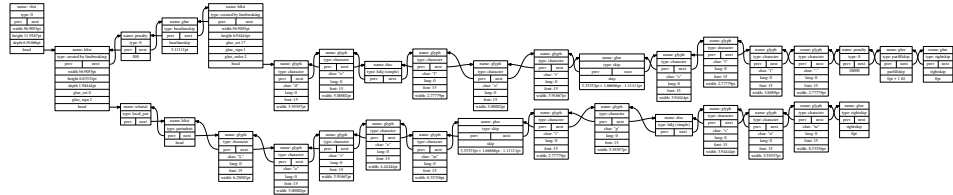
246     node.insert_after(head, n.color_text, n.kern)
247     n.color_reset = nodex.create_color('reset')
248     node.insert_after(head, n.stop, n.color_reset)
249 else
250     n.line.next = n.stop.next
251     n.stop.prev = n.line.prev
252 end
253
254 end

```

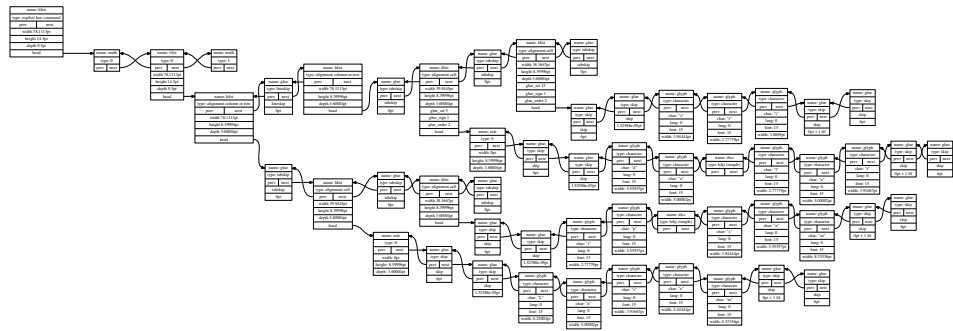
3.2.4 Node trees

Some graphics for better understanding of the node tree:

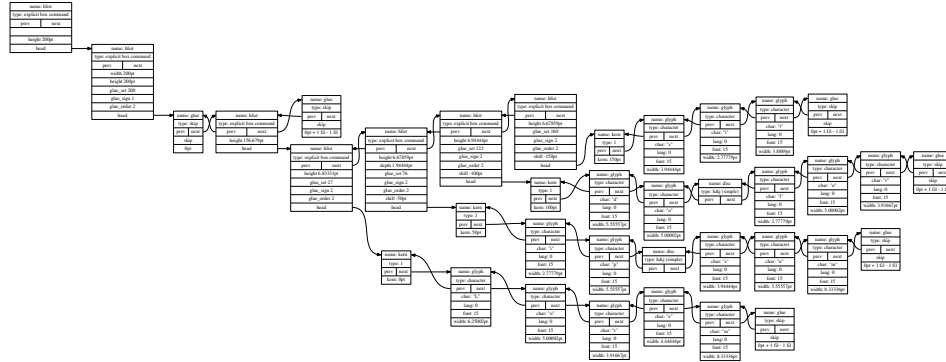
3.2.4.1 Paragraph



3.2.4.2 Tabular environment



3.2.4.3 Picture environment



basic

The corresponding L^AT_EX command to this lua function is `\cloze` ([→ 2.1.1](#)). The argument `head` is the head node of a node list.

```

255 function cloze.basic(head)
256   local n = {} -- node
257   local b = {} -- boolean
258   local l = {} -- length
259   local t = {} -- temp
260   for hlist in node.traverse_id(node.id('hlist'), head) do
261     hlist = nodex.strut_to_hlist(hlist)
262     if b.line_end then
263       b.init_cloze = true
264     end
265     n.current = hlist.head
266     while n.current do
267       if
268         registry.check_marker(n.current, 'basic', 'start')
269       or
270         b.init_cloze
271       then
272         b.init_cloze = false
273         n.start = n.current
274         while n.current do
275           b.line_end = true
276           n.stop = n.current
277           if registry.check_marker(n.stop, 'basic', 'stop') then
278             b.line_end = false
279             break
280           end
281           n.current = n.current.next
282         end
283         cloze.basic_make(head, hlist, n.start, n.stop)
284         n.current = n.stop
285       else
286         n.current = n.current.next

```

```

287     end
288   end
289 end
290 return head
291 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_defined	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_defined	index

Hide text:

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

The argument `head` is the head node of a node list. The argument `start` is the node, where the gap begins. The argument `stop` is the node, where the gap ends.

```

292 function cloze.fix_make(head, start, stop)
293   local l = {} -- length
294   l.width = tex.sp(registry.get_value('width'))
295   local n = {} -- node
296   n.start = start
297   n.stop = stop
298   l.text_width = node.dimensions(n.start, n.stop)
299   local align = registry.get_value('align')
300   if align == 'right' then
301     l.kern_start = -l.text_width
302     l.kern_stop = 0
303   elseif align == 'center' then
304     l.half = (l.width - l.text_width) / 2
305     l.kern_start = -l.half - l.text_width
306     l.kern_stop = l.half
307   else
308     l.kern_start = -l.width
309     l.kern_stop = l.width - l.text_width
310   end
311   head, n.line = nodex.insert_line(head, n.start, l.width)
312   if registry.get_value_show() then
313     n.kern_start = nodex.create_kern(l.kern_start)
314     head, n.kern_start = node.insert_after(head, n.line, n.kern_start)
315     n.color_text = nodex.create_color('text')

```

```

316     node.insert_after(head, n.kern_start, n.color_text)
317     n.color_reset = nodex.create_color('reset')
318     node.insert_before(head, n.stop, n.color_reset)
319     n.kern_stop = nodex.create_kern(l.kern_stop)
320     node.insert_before(head, n.stop, n.kern_stop)
321   else
322     n.line.next = n.stop.next
323   end
324 end

```

fix

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

```

325 function cloze.fix(head)
326   local n = {} -- node
327   n.start, n.stop = false
328   for current in node.traverse_id(node.id('whatsit'), head) do
329     if not n.start then
330       n.start = registry.get_marker(current, 'fix', 'start')
331     end
332     if not n.stop then
333       n.stop = registry.get_marker(current, 'fix', 'stop')
334     end
335     if n.start and n.stop then
336       cloze.fix_make(head, n.start, n.stop)
337       n.start, n.stop = false
338     end
339   end
340   return head
341 end

```

par

The corresponding L^AT_EX environment to this lua function is `clozepar` (→ 2.1.5).

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.

```

342 function cloze.par(head)
343   local l = {} -- length

```

```

344 local n = {} -- node
345 for hlist in node.traverse_id(node.id('hlist'), head) do
346   for whatsit in node.traverse_id(node.id('whatsit'), hlist.head) do
347     registry.get_marker(whatsit, 'par', 'start')
348   end
349   l.width = hlist.width
350   hlist, n.strut, n.head = nodex.strut_to_hlist(hlist)
351   head, n.line = nodex.insert_line(head, n.strut, l.width)
352   if registry.get_value_show() then
353     n.kern = nodex.create_kern(-l.width)
354     head, n.kern = node.insert_after(head, n.line, n.kern)
355     n.color_text = nodex.create_color('text')
356     node.insert_after(head, n.kern, n.color_text)
357     n.tail = node.tail(n.head)
358     n.color_reset = nodex.create_color('reset')
359     node.insert_after(n.head, n.tail, n.color_reset)
360   else
361     n.line.next = nil
362   end
363 end
364 return head
365 end

```

3.2.5 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`.

```

366 function base.register(mode)
367   if mode == 'par' then
368     luatexbase.add_to_callback(
369       'post_linebreak_filter',
370       cloze.par,
371       mode
372     )
373     return true
374   end
375   if not base.is_registered[mode] then
376     if mode == 'basic' then
377       luatexbase.add_to_callback(
378         'post_linebreak_filter',
379         cloze.basic,
380         mode
381       )
382     elseif mode == 'fix' then
383       luatexbase.add_to_callback(

```

```

384         'pre_linebreak_filter',
385         cloze.fix,
386         mode
387     )
388 else
389     return false
390 end
391 base.is_registered[mode] = true
392 end
393 end

```

unregister

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

```

394 function base.unregister(mode)
395   if mode == 'basic' then
396     luatexbase.remove_from_callback('post_linebreak_filter', mode)
397   elseif mode == 'fix' then
398     luatexbase.remove_from_callback('pre_linebreak_filter', mode)
399   else
400     luatexbase.remove_from_callback('post_linebreak_filter', mode)
401   end
402 end

```

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

```

403 base.linefil = nodex.write_linefil
404 base.line = nodex.write_line
405 base.margin = nodex.write_margin
406 base.set_option = registry.set_option
407 base.set_is_global = registry.set_is_global
408 base.unset_local_options = registry.unset_local_options
409 base.reset = registry.unset_global_options
410 base.get_defaults = registry.get_defaults
411 base.marker = registry.write_marker

412 return base

```

Change History

v0.1		v1.1
General: Converted to DTX file . . .	10	
v1.0		General: Make cloze compatible to
General: Inital release	10	LuaTeX version 0.95 10

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.

Symbols	<code>\clozefix</code> <u>123</u>	E
<code>\\</code> 174, 213	<code>\clozefont</code> <u>108</u> , 110, 118, 127, 137	<code>\endcsname</code> 42
C	<code>\clozehide</code> <u>105</u>	environments: <code>clozepar</code> <u>132</u>
<code>\cloze</code> <u>114</u> , 145	<code>\clozeline</code> <u>147</u>	I
<code>\cloze@color</code> <u>42</u> , 78, 88	<code>\clozelinefil</code> . 145, <u>151</u>	<code>\itshape</code> 108
<code>\cloze@margin</code> <u>58</u> , 119, 128	<code>clozepar</code> (environ- ment) <u>132</u>	K
<code>\cloze@set@local@options</code> <u>43</u> , 115, 124, 135, 148, 152	<code>\clozereset</code> <u>99</u>	<code>\kvsetkeys</code> 45, 97
<code>\cloze@set@option</code> <u>39</u> , 69, 71, 73, 74, 78, 81, 83, 84, 88, 91, 93	<code>\clozeset</code> .. <u>95</u> , 103, 106	P
<code>\cloze@set@to@global</code> <u>30</u> , 63, 96	<code>\clozesetfont</code> <u>109</u>	<code>\par</code> 134, 141
<code>\cloze@set@to@local</code> <u>33</u> , 44	<code>\clozeshow</code> <u>102</u>	<code>\ProcessKeyvalOptions</code> 94
<code>\cloze@start@marker</code> . <u>47</u> , 116, 125, 136	<code>\color@</code> 42	R
<code>\cloze@stop@marker</code> . <u>53</u> , 121, 130, 140	<code>\csname</code> 42	<code>\relax</code> 118, 127, 137
<code>\clozefil</code> <u>144</u>	D	<code>\renewcommand</code> 110
	<code>\DeclareStringOption</code> 68, 70, 76, 80, 86, 90, 92	<code>\RequirePackage</code> 26
	<code>\DeclareVoidOption</code> 72, 82	S
	<code>\define@key</code> . 69, 71, 77, 81, 87, 91, 93	<code>\SetupKeyvalOptions</code> 64
		<code>\string</code> 42
		<code>\strut</code> .. 48, 54, 153, 155