

# place<sub>A</sub><sub>T</sub>

v0.1e

Arno L. Trautmann

[arno.trautmann@gmx.de](mailto:arno.trautmann@gmx.de)

## Abstract

The package `placeat` offers the command `\placeat(2,5){}` which places arbitrary content freely on any page. It is mainly thought for use with the `beamer` class but may also be used with any other  $\TeX$  class. This package requires `Lua $\TeX$` ; don't try it with any other  $\TeX$  flavour, it just won't work.

**ATTENTION** This package is in a very preliminary version and released for testing only. Any feature might be subject to changes without notice.

## Introduction

This is the documentation of the package `placeat`. When you load the package, a grid is drawn on every page of your document to aid you at placing stuff where you want it to be. This mainly makes sense in presentations, but might be used in any document. The main macro of this package `\placeat...{}{}{}` offers several ways to use it:

```
\placeat<D4>{some content}
\placeat(3,4){some content}
\placeat{3}{4}{some content}
```

To deactivate the grid, use the setup command `\placeatsetup{nogrid}`. There are also some other commands that allow you to draw simple sketches which might be useful in presentations, too, like arrows, circles etc., but no fancy stuff.

**Attention:** This package is under development and everything presented here might and will be subject to incompatible changes.

If you have any suggestions or comments, just drop me a mail, I'll be happy to get any response! The latest source code is hosted on github – Feel free to comment or report bugs there, to fork,

This package is copyright © 2017 Arno L. Trautmann. It may be distributed and/or modified under the conditions of the LaTeX Project Public License, either version 1.3c of this license or (at your option) any later version. This work has the LPPL maintenance status 'maintained'. Whoever notes the face in the title gets a cookie when we meet.

pull, etc.: <https://github.com/alt/placeat>

# Contents

<b>I</b>	<b>User Documentation</b>	<b>5</b>
<b>1</b>	<b>How do I use it?</b>	<b>5</b>
1.1	Placing – the Main Commands . . . . .	5
1.2	Relative Placing . . . . .	6
1.3	Placing of figures, floats etc. . . . .	6
1.4	User Options . . . . .	6
1.5	The Grid . . . . .	7
1.6	Offsetting . . . . .	7
<b>2</b>	<b>Drawing simple forms</b>	<b>7</b>
2.1	Colored forms . . . . .	8
<b>3</b>	<b>Example</b>	<b>9</b>
3.1	Example use with beamer . . . . .	9
3.2	Example use inside this document . . . . .	9
<b>4</b>	<b>How is it done?</b>	<b>11</b>
<b>5</b>	<b>To Do</b>	<b>11</b>
<b>6</b>	<b>How can I help?</b>	<b>11</b>
<b>7</b>	<b>Thanks</b>	<b>12</b>
<b>II</b>	<b>Implementation</b>	<b>14</b>
<b>8</b>	<b>The <math>\LaTeX</math> package: placeat.sty</b>	<b>14</b>
8.1	Loading Files . . . . .	14
8.2	User Commands . . . . .	14
8.3	Relative Placement . . . . .	15
8.4	Placing of floats etc. . . . .	16
8.5	Helper Macros . . . . .	16
<b>9</b>	<b>The Grid</b>	<b>17</b>
<b>10</b>	<b>Drawing Stuff</b>	<b>18</b>
<b>11</b>	<b>Key-Value Interface</b>	<b>20</b>



## Part I

# User Documentation

## 1 How do I use it?

### 1.1 Placing – the Main Commands

The command `\placeat` takes several arguments, the last of which is the content you want to place:

```
\placeat(4,5){content}
```

This may range from single letters to graphic objects or (mostly) any valid  $\LaTeX$  code. Take note that the content will be placed *above* and *right of*<sup>1</sup> the specified coordinates.<sup>2</sup> Exceptions are floating environments – you have to pack them into a minipage or similar construct, see below.

If you want to place something *left of* the specified coordinates, there is an additional optional argument to `\placeat`:

```
\placeat{4}{5}[left]{right}
```

This allows you to center your content (by hand) around the given place. Do not forget to enter an empty `{}` if you use only the optional content.

Verbatim material does definitely *not* work and makes troubles as always in moving arguments (like footnotes etc.). So far I have no idea how to handle that correctly. Please tell me any further problems, I'll happily tackle them or sadly note them here if I cannot fix it ...

You may use `\placeat` in one of the following variants (feel free to mix them in one document):

```
\placeat<D5>{content-right}  
\placeat(4,5){content-right}  
\placeat{4}{5}{content}
```

The result will be the same in all three cases, so it's just a matter of taste which one you choose. They all will place the `<content>` at a position that is specified by the grid which is drawn on your document. While the grid is drawn using letters and numbers, you might prefer using two numbers as you then also can use decimals for fine tuning which is not possible with a letter-number combination:

```
\placeat{4.3}{5.2}{content}  
\placeat(4.3,5.2){content}
```

---

<sup>1</sup>See below for placing to the left via an optional argument.

<sup>2</sup>To be more precise, the ground line of the first line of the content is placed at the specified vertical coordinate. This may result in strange placement of anything that is not pure text.

Finally, there is one more argument you can give as second-to-last argument:

```
\placeat{4.3}{5.2}[content-left]{content}  
\placeat(4.3,5.2)[content-left]{content}
```

This content will be placed to the left of the specified coordinates as opposed to the normal content expanding to the right.

## 1.2 Relative Placing

It is also possible to place a second element relative to another one. For this, you have to give the first one a name and refer to this name in the second one. Then you can repeat and refer a third one to the second one (or the first one, however you like to).

```
\placeat(4,5){content}[first]  
\placereftto[first](2,2){content2}[second]  
\placereftto[second]{2}{2}{content3}[third]
```

Although it does not make any sense, you still can use the chess-pattern notation for `\placereftto`. But that's just for raising the obscurity level of this package.

## 1.3 Placing of figures, floats etc.

Placing figures might be a bit tricky because the placing actually places the *groundline* of any object. You may make your life easier when inserting figures if you use the `[t]` argument:

```
\placeat{4}{5}{\includegraphics[t]{bose-gas}}
```

This way it is easier to fit graphics at the same height. However, you might have to test where it lands in the end.

For floating environments, even if they don't float (that would be stupid, wouldn't it?), you need to packg them into e.g. a minipage. You can do this by hand or just use the command `\placeminipageat`. This command only has one kind of interface, the one with two braces:

```
\placeminipageat{4}{5}[4cm]{content}
```

Here, the third, argument is optional and specifies the width of the minipage. If not given, it will default to 10cm, which should be wide enough to contain anything you ever want to set using `placeat`.

## 1.4 User Options

Some of this package's features can be adjusted at any time in the document with the command `\placeatsetup{}`

Some of the options only make sense when used in the preamble, others only have a result when used in the text. However, none should result in an error, so feel free to do whatever nonsense you want to.

## 1.5 The Grid

If the number of grid lines does not suit you (there are ten horizontally and vertically), you can increase or decrease the number by

```
\placeatsetup{gridnumber = 12}
```

You may change the gridnumber during your document, but don't expect everything to work fine.

The horizontal and vertical gridnumbers can be adjusted independently:

```
\placeatsetup{
  gridnumberx = 12,
  gridnumbery = 8,
}
```

The grid can be deactivated by the document options `final` or `nogrid` and re-activated by the option `drawgrid` in the setup macro:

```
\placeatsetup{nogrid}
\placeatsetup{drawgrid}
```

## 1.6 Offsetting

You can choose the zero point of the grid by setting the options

```
\placeatsetup{
  offsetx = 2
  offsety = -1
}
```

The grid and placement are adapted correspondingly. If you are a C-head thinking that everything should start with 0 instead of 1, you can call

```
\placeatsetup{
  startzero
}
```

which corresponds to `offsetx = 1, offsety = 1` so that the upper right corner has coordinates (0,0) instead of (1,1).

## 2 Drawing simple forms

This package also allows to draw simple forms like arrows and circles, to support the user e. g. when creating presentations. A single line is drawn by calling

```
\placelineat(2.5,1.5)(1.5,2.5)
```

where the first coordinate pair specifies the start of the line and the second one the end. As you typically need fine tuning to place the line exactly where you want it, it is not possible to use another interface, i. e. the <D4> style.

By now, the following commands and respective forms are possible:

<code>\placelineat(x1,y1)(x2,y2)</code>	Draws a single line pointing from (x1,y1) to (x2,y2)
<code>\placearrowat(x1,y1)(x2,y2)</code>	As the line, but with an arrowhead at the end.
<code>\placecircleat(x,y){r}</code>	Draws a circle at position (x,y) with diameter r. If omitted, r will default to 3. The diameter is not scaled to the same scale as the coordinates, and most likely you have to test what size fits. Start with 5, it's a nice number. Right now, the circle is not really a circle, but slightly deformed as we only have cubic splines. May change to something better.
<code>\placesquareat(x,y){r}</code>	Draws a square with center at (x,y) and side length r. If omitted, r will default to 3.
<code>\placerectangleat(x1,y1)(x2,y2)</code>	draws a rectangle from the (upper left) corner (x1,y1) to the (lower right) corner (x2,y2).
<code>\placefilledrectangleat(x1,y1)(x2,y2)</code>	draws a filled rectangle.

You can change the linewidth and therefore the thickness of lines with the simple call

```
\placeatsetup{linewidth=5}
```

Default is 1, I have no idea in which unit, but it is a very nice thickness, I think. You can change the thickness any time and as often as you want.

Missing are elliptical shapes, maybe rounded corners for the rectangles and maybe some funny stuff.<sup>3</sup> The arrowheads need a lot of work, too, of course.

## 2.1 Colored forms

You need to load the xcolor package to use colors.<sup>4</sup> Every command of the ones listed above takes an optional argument that allows the specification of a color. This is based on the xcolor, so

<sup>3</sup>Yes, I *will* add a penis-shape macro, but that will not be documented explicitly.

<sup>4</sup>Why is it not required in the PLACEAT package? Because you might want to specify package options and that may collide with the loading here. However, every sane document working with color requires the package by default.



all colors known by that package are possible:

```
\placecircleat[blue](5,5)
\placearrowat[green!50!yellow](6,5)(8,5)
\placerectangleat[red!25!black](8,4)(9,6)
\placefilledrectangleat[blue!25!red](8.5,4.5)(8.75,5.6)
```

By now, it is not possible to specify an rgb code or similar. If you want a very special color that is not defined in the `xcolor` package, just define it by yourself. However, as shown above, it is possible to mix colors using the `red!50!green` syntax, which is very flexible and should cover normal every day use.

### 3 Example

Now, here are two examples on how to use the package. The first one is a code example only, while the second one shows the effect directly on the page.

#### 3.1 Example use with beamer

As this package makes most sense in combination with `beamer`, here is a small example about how to use it.

```
\documentclass[ngerman]{beamer}
\usepackage{babel,blindtext}
\usepackage{fontspec}
\usepackage{placeat}
\begin{document}
\begin{frame}{Test frame}
Test
\placeat<D5>{Test}
\placeminipageat{4}{5}[3cm]{\includegraphics{fermi_gas_1}}
\end{frame}
\end{document}
```

#### 3.2 Example use inside this document

The following page is typeset using the features of this package and shows the corresponding code.

However, this very page is using the `drawgrid` option, with an increased grid number of 15.  
There are several elements placed with the given code, respectively.<sup>5</sup>

`\placeat{2.3}{4.1}`

`\placerectangleat[red](8,4)(9,6)`

`\placeat<F5>`

`\placeat(4.5,7.2)`

`\placearrowat[green](6,9)(8.5,5)`

`\placecircleat[blue](6,9)`

---

<sup>5</sup>Don't let me fool you, the code is not printed using `\verb`, but only with a `\texttt`, as verbatim is not possible with `\placeat`.

## 4 How is it done?

The short answer is: Look at the source code. While the coding is quite simple in principle, it might be very confusing when reading it, and I am still surprised it works at all ...

Mainly, everything is based on the  $\text{\TeX}$  command `\put () {}`. You could of course just use this, but then it's hard to get an absolute positioning as `\put` only allows relative positions. You could then put your code into, say, a header line, and that is nearly the idea of this package. However, this would require a header and would not let the user freely decide what to put there. Also, users might do strange stuff to that and that could destroy the placing.

Instead, we use the ability of Heiko Oberdiek's `atbegshi` package which adds content to the to-be-shipped-out-page. I still do not understand how it works, but it is absolutely robust and does just what we need here: It allows to put stuff on the page relative to, say, the upper right corner. Also, it can be put in front of every other thing, so we are sure nothing gets lost.

The next step is collecting and saving the material you specify to be placed somewhere. Collection is done using the `xparse` package which allows for a very flexible macro definition which makes it possible to enter the different positioning options. Finally, everything is glued together with some Lua magic ...

We save the content to be placed in  $\text{\TeX}$  macros that are numbered using a Lua counter; the final coordinates are also calculated by Lua. The  $\text{\TeX}$ -Lua interface is heavily used here which is possible due to the `luacode` package. The macros are then executed in the call of `\AtBeginShipout`, again inside a Lua loop, where also the grid is drawn.

## 5 To Do

A list of things I would like to have solved by some time:

- allow the wave color model as it is very very cool
- placing stuff at every page or reuse stuff at all
- allow course placing (put at upper left corner, put at left side etc.) for presentations
- verbatim in placeat
- drawing maybe based on metapost instead of pdf drawing

## 6 How can I help?

There are several ways how you can help. First, and most important:

Testing. Try to use this code and tell me what you think about it.

Bug reporting. Tell me especially what is buggy. I'd like to keep the package rather small and simple, so some bugs might be called features, but we'll see.

Suggestions. I'm open to extend the functionality. Just tell me what you want and I'll try to implement it as soon as possible. Which might be never. But also maybe the next day. Well, try it! 😊

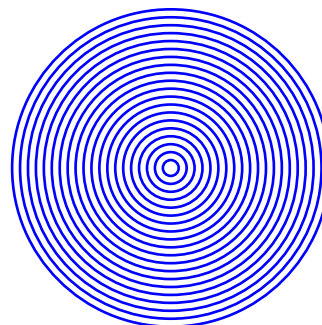
## **7 Thanks**

Of course I have to thank Paul Isambert again for the code for drawing to the pdf file. Also I want to thank Rembrandt Wolpert who was the first one to report bugs and feature requests.

That's it for the documentation, have fun, and



Happy TEXing!



## Part II

# Implementation

## 8 The $\TeX$ package: placeat.sty

Everything to get stuff working from the  $\TeX$  side. Here, only a .sty file is provided and plain/Con $\TeX$ t users have to find their way. I'll happily support them, though!

### 8.1 Loading Files

The Lua file is not found by using a simple `dofile("placeat.lua")` call, but we have to use `kpse's find_file`.

```
1 \ProvidesPackage{placeat}%
2 [2020/05/02 v0.1e absolute content positioning]
3 \RequirePackage{luatexbase}
4 \RequirePackage{luacode}
5 \RequirePackage{atbegshi}
6 \RequirePackage{xparse}
7 \directlua{dofile(kpse.find_file("placeat.lua",'lua'))}
```

### 8.2 User Commands

The main command `\placeat`. There are several ways to use it, so we define a wrapper macro that is only for the user interface. Nice separation of interface and code. But actually, both are quite hard interwoven and it's not really clear at any time what happens. However, it works most of the time.

The macro arguments of `placeat` at the moment are: `g{}g{}`: two braced arguments for coordinates

`d()`: one argument for picture-like coordinate pairs

`d<>`: one argument for alpha-numeric coordinates

`O{}`: content to be typeset on the left of the point

`m`: main content to be typeset on the right. `o`: optional label for relative placement. This might now be the point to change the internal structure and go to a node mode.

```
8 \NewDocumentCommand\placeat{ggd()d<>O{ }mo}{
9   \IfValueT{#1}{                                %% two coordinates in { }{ } pair.
10     \IfValueT{#2}{                                %% if second argument is not given, everything breaks. no
11       \def\cox{#1}
12       \def\coy{#2}
13     }
14   }
15   \IfValueT{#3}{                                %% one argument as ( , ) coordinate pair.
16     \def\cox{\firstof#3X}
```

```

17   \def\coy{\secondof#3X}
18 }
19 \IfValueT{#4}{
20   \luadirect{
21     y = string.byte('#4',1)-64
22     x = string.byte('#4',2)-48
23     x2 = string.byte('#4',3)
24     if x2 then x = x*10 + x2-48 end -- FIXME: what exactly happens here? ...
25   }
26   \def\cox{\luadirect{tex.print(x)}}
27   \def\coy{\luadirect{tex.print(y)}}
28 }
29 \placeatthreenumbers{\cox}{\coy}{\llap{#5}#6}
30
31 \IfValueT{#7}{
32   \expandafter\gdef\csname #7x\endcsname{\firstof#3X}
33   \expandafter\gdef\csname #7y\endcsname{\secondof#3X}
34 }
35 }

```

### 8.3 Relative Placement

The first stage of this works just the same as normal `\placeat`. However, there is an additional first optional argument that actually is *not* optional! This is the node that is taken as base. So the `\placeatthreenumbers` is just called with the given coordinates added to the base coordinates.

```

36 \NewDocumentCommand\placereyto{oggd()d<>0{ }mo}{
37   \IfValueT{#2}{                                     %% two coordinates in { }{ } pair.
38     \IfValueT{#3}{                                     %% if second argument is not given, everything breaks. not
39       \def\cox{#2}
40       \def\coy{#3}
41     }
42   }
43   \IfValueT{#4}{                                     %% one argument as ( , ) coordinate pair.
44     \def\cox{\firstof#4X}
45     \def\coy{\secondof#4X}
46   }
47   \IfValueT{#5}{
48     \luaexec{
49       y = string.byte('#5',1)-64
50       x = string.byte('#5',2)-48
51       x2 = string.byte('#5',3)
52       if x2 then x = x*10 + x2-48 end -- FIXME: what exactly happens here? ...
53       tex.print("\def\cox{"..(x).."}\def\coy{"..(y).."}")
54     }
55   }

```

```

56 \placeatthreenumbers
57   {\cox + \csname #1x\endcsname}
58   {\coy + \csname #1y\endcsname}
59   {\llap{#6}#7}
60 \IfValueT{#8}{
61   \expandafter\xdef\csname #8x\endcsname{\cox + \csname #1x\endcsname}
62   \expandafter\xdef\csname #8y\endcsname{\coy + \csname #1y\endcsname}
63 }
64 }

```

## 8.4 Placing of floats etc.

For floats and similar stuff, it might be necessary or useful to pack everything into a minipage. You can do this by yourself, but I thought it might be nice to specify a corresponding user interface. Using `\placeminipageat` is the same as using `\placeat{}{}{content}` where content is packed into a minipage. The first two argument of `\placeminipageat` must be given in braces `{4}{5}` and determine the position of the content. The third argument is optional and specifies the width of the minipage; if not give, it is assumed to be 10cm, wide enough for mostly anything you ever will place at.

```

65 \NewDocumentCommand\placeminipageat{r()0{10cm}m}{
66   \gdef\widthofplaceat{#2}
67   \placeat{#1}
68   {\begin{minipage}{\widthofplaceat}{#3}\end{minipage}}
69 }

```

## 8.5 Helper Macros

The real stuff is done in the macro `\placeatthreenumbers` which takes exactly three arguments defining the position of the content. The content is stored in a macro that is defined using Lua code, and the position is also calculated by Lua code. Everything is put together into a Lua- $\TeX$ -bastard and surprisingly works stable as far as I can tell.

This place is also where the offset and scaling happens.

```

70 \def\placeatthreenumbers#1#2#3{
71   \luaexec{
72     nr = nr+1
73     dacoordtmp = ((#1-1+offsetx)*tex.pagewidth/65536/gridnrx*1.005)..", "..(-(#2-1+offsety)*tex.pagewidth/65536/gridnry*1.005)
74     dacoord[nr] = "\\put("..dacoordtmp..")"
75     tex.print("\\expandafter\\gdef\\csname command"..(nr).."\\endcsname")}% begin of command definition
76   {#3} %% this is what \command[nr] will contain
77 }

```

Two tiny helpers that might be substituted by some standard commands:

```

78 \def\firstof #1,#2X{#1}
79 \def\secondof #1,#2X{#2}

```



Setup of variables and macros we need later.

```
80 \gdef\drawgridnum{1}
81 \luaexec{
82   arrowheadlength = 5
83   drawgrid = false
84   nr          = 0
85   dacoord     = {}
86   gridnr      = 10
87   gridnrx     = 10
88   gridnry     = 10
89   gridlinewidth = 0.01
90   offsetx     = 0
91   offsety     = 0
92 }
```

Now the code that does the actual work here. We use Heiko Oberdiek's package `atbegshi` with the very useful macros `\AtBeginShipout` and `\AtBeginShipoutUpperLeftForeground`. Using these, we are free from any context of where the code is written, it is always executed at the shipout and therefore absolute positioning is possible.

I have to use a quite weird way of checking whether to draw the grid or not, using a number instead of defining a `\ifdrawgrid`. That one was working at some time, but now it is not anymore. No idea why, some handling of the input parsing in the arguments must have changed. Anyways, this is working and not too ugly, so we'll stick with that one for now.

```
93 \AtBeginDocument{
94   \AtBeginShipout{%
95     \AtBeginShipoutUpperLeftForeground{%
96       \ifnum\drawgridnum = 1 \drawgrid\fi
97       \luaexec{%
98         for i = 1,nr do
99           tex.print(dacoord[i].."{\csname command"..(i).."\\endcsname}")
100         end
101         nr=0
102       }
103     }
104   }
105 }
```

## 9 The Grid

The grid is made by drawing directly into the pdf as suggested by Paul Isambert in his TUGboat article "*Drawing tables: Graphic fun with Lua $\TeX$* ". Labeling is done by simple `\put` commands, controlled via Lua code.

```
106 \def\drawgrid{
107   \luatexlualua{
```

```

108 pdf_print("q")
109 linewidth(gridlinewidth)
110 local factorh = tex.pageheight/gridnry/65536
111 local factorw = tex.pagewidth/gridnrx/65536
112 for i = 1,math.max(gridnrx,gridnry) do
113     h = i*factorh
114     w = i*factorw
115     move(0,-h) line(tex.pagewidth,-h) stroke()
116     move(w,0) line(w,-tex.pageheight) stroke()
117 end
118 pdf_print("Q")
119 }
120 { %% extra grouping to keep font size change local. Going to normalfont seems to make sense. An
121 %% would also be nice to maybe adapt the fontsize to the grid size
122     \normalfont\fontsize{8}{10}\selectfont
123     \luaexec{
124         for i=1,math.max(gridnrx+offsetx,gridnry+offsety) do
125             hfac = tex.pageheight/gridnry/65536
126             wfac = tex.pagewidth/gridnrx/65536*1.005 %% another empirical factor
127             h = (i-1)*hfac
128             w = (i-1)*wfac
129             tex.print("\put("..(w)..",-7){\rlap{"..(i-offsetx).."}}")
130             if alphanumgrid then
131                 tex.print("\put(0,"..(-h-0.05*hfac).."){\\char00".."(64+i-offsety).."}") %%-- for alpha
132             else
133                 tex.print("\put(0,"..(-h-0.05*hfac).."){.."..(i-offsety).."}")
134             end
135         end
136     }
137 }
138 }

```

## 10 Drawing Stuff

Drawing is done in the same way as the grid. While the grid has no interface, the rest of the drawing stuff needs a  $\TeX$  interface, which is defined here. Every command calls a Lua function that does the actual work, as always.

I try to provide a basic set of stuff that might be useful. The  $\TeX$  interface implementation might change, but for now it is done with `xparse` instead of a much more saner simple `\def`. We will see where this will head to. First, there is an arrow, whose head looks very bad. I don't know how to fix this yet. Then there are circle, square and rectangle.

```

139 \NewDocumentCommand\placelineat{or()r()}{
140     \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}} % only to fix the color!
141     \luatexlua\placelineat(#2,#3)}

```

```

142 }
143 }
144 \NewDocumentCommand\placearrowat{or()r()}{
145   \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
146     \luatexlatalua{placearrowat(#2,#3)}
147 }
148 }
149 \NewDocumentCommand\placecircleat{or()D(){.3}}{
150   \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
151     \luatexlatalua{placecircleat(#3,1)}
152 }
153 }
154 \NewDocumentCommand\placefilledcircleat{or()D(){.3}}{
155   \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
156     \luatexlatalua{placecircleat(#3,1,true)}
157 }
158 }
159 \NewDocumentCommand\placesquareat{or()G{3}}{
160   \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
161     \luatexlatalua{placesquareat(#3)}
162 }
163 }
164 \NewDocumentCommand\placecurveat{or()r()r()r()}{
165   \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
166     \luatexlatalua{placecurveat(#2,#3,#4,#5)}
167 }
168 }
169 \NewDocumentCommand\placerectangleat{0{black}r()d()}{
170   \placeat(#2){\ignorespaces\color{#1}%
171     \luatexlatalua{placerectangleat(#2,#3)}
172 }
173 }
174 \NewDocumentCommand\placefilledrectangleat{0{black}r()r()}{
175   \placeat(#2){\ignorespaces\color{#1}%
176     \luatexlatalua{placerectangleat(#2,#3,true)}
177 }
178 }
179 \NewDocumentCommand\placeroundedat{s0{black}r()D(){0.1}D<>{1.5}}{
180   \placeat(#3){\ignorespaces\color{#2}%
181     \IfBooleanTF{#1}{\luatexlatalua{placecircleat(#4,#5,true)}}%
182       {\luatexlatalua{placecircleat(#4,#5)}}
183 }
184 }

```

## 11 Key-Value Interface

It's a modern package, so we make use of  $\LaTeX$ 3 once more. Let's see how stable this is. So far, no options can be used as package option, but only inside the `\placeatsetup{}` macro. I'm not much into  $\LaTeX$ 3 syntax and stuff anymore, so feel free to correct any non-nice coding here!

Especially one thing will be annoying, the space-gobbling. Nice feature on one hand, but annoying inside the `\directlua` on the other hand. Therefore, we need the `~` to separate `gridnr` and `gridnry` below.

```
185 \ExplSyntaxOn
186 \gdef\drawgridnum{1}
187 \keys_define:nn{placeat}{
188   alphanumgrid.code:n    = \directlua{alphanumgrid = true},
189   arrowheadlength.code:n = \directlua{arrowheadlength=#1},
190   final.code:n           = \luaexec{placeat_final = true} \gdef\drawgridnum{0},
191   drawgrid.code:n        = \gdef\drawgridnum{1},
192   gridnumber.code:n       = \directlua{gridnr = #1 gridnrx = gridnr~gridnry = gridnr},
193   gridnumberx.code:n      = \directlua{gridnrx = #1},
194   gridnumbery.code:n      = \directlua{gridnry = #1},
195   gridlinewidth.code:n   = \directlua{gridlinewidth = #1},
196   linewidth.code:n       = {\placeat(1,1){\luatexlatalua{linewidth(#1)}}}, %% FIXME: this is a v
197   nogrid.code:n          = \gdef\drawgridnum{0},
198   numnumgrid.code:n      = \directlua{alphanumgrid = false},
199   offsetx.code:n         = \directlua{offsetx = #1},
200   offsety.code:n         = \directlua{offsety = #1},
201   startzero.code:n       = \directlua{offsetx = 1 offsety = 1}
202 }
203 \DeclareDocumentCommand\placeatsetup{m}{
204   \keys_set:nn{placeat}{#1}
205 }
206 \ExplSyntaxOff
```

## 12 Lua Module

So far, the only usage of the Lua module is for graphics, based on the article by Paul Isambert about drawing directly to the pdf using Lua. We exploit this here and make use of the basic drawing functions he provided. Maybe this will be outsourced once there is a Lua-to-pdf-based graphics bundle.

```
207 function pdf_print (...)
208   for _, str in ipairs({...}) do
209     pdf.print(str .. " ")
210   end
211   pdf.print("\n")
212 end
213
```

```

214 function move (p1,p2)
215   if (p2) then
216     pdf_print(p1,p2,"m")
217   else
218     pdf_print(p1[1],p1[2],"m")
219   end
220 end
221
222 function line(p1,p2)
223   if (p2) then
224     pdf_print(p1,p2,"l")
225   else
226     pdf_print(p1[1],p1[2],"l")
227   end
228 end
229
230 function curve(p11,p12,p21,p22,p31,p32)
231   if (p22) then
232     p1,p2,p3 = {p11,p12},{p21,p22},{p31,p32}
233   else
234     p1,p2,p3 = p11,p12,p21
235   end
236   pdf_print(p1[1], p1[2],
237             p2[1], p2[2],
238             p3[1], p3[2], "c")
239 end
240
241 function linewidth(w)
242   pdf_print(w,"w")
243 end
244
245 function fill()
246   pdf_print("f")
247 end
248
249 function stroke()
250   pdf_print("S")
251 end
252
253 -- welp, let's have some fun!
254 -- with the function radd, a random coordinate change is added if used
255 -- randfact will adjust the amount of randomization
256 -- everything is relative in the grid size
257 -- BUT: In fact, do we really want to have wiggly lines? ...
258 local randfact = 100
259 local radd = function()

```

```

260 return (math.random()-0.5)*randfact
261 end
262
263 function placelineat(x1,y1,x2,y2)
264   xfac = tex.pagewidth/gridnrx/65536 -- factors to convert given number to absolute coordinates
265   yfac = tex.pageheight/gridnry/65536 -- should both be global!
266   xar = (x2-x1)*xfac -- end point of the arrow
267   yar = (y1-y2)*yfac --
268   move(0,0) -- start
269   line(xar,yar) -- draw main line
270   stroke()
271 end
272
273 function placearrowat(x1,y1,x2,y2)
274   xfac = tex.pagewidth/gridnrx/65536 -- factors to convert given number to absolute coordinates
275   yfac = tex.pageheight/gridnry/65536 -- should both be global!
276   xar = (x2-x1)*xfac -- end point of the arrow
277   yar = (y1-y2)*yfac --
278   parx = xar/math.sqrt(xar^2+yar^2) -- direction of the arrow
279   pary = yar/math.sqrt(xar^2+yar^2) --
280   perpx = -pary -- perp of the arrow direction
281   perpy = parx --
282   move(0,0) -- start
283   line(xar,yar) -- draw main line
284   move(xar,yar)
285   line(xar-arrowheadlength*parx+arrowheadlength*perpx,yar-arrowheadlength*pary+arrowheadlength*perpy)
286   move(xar,yar)
287   line(xar-arrowheadlength*parx-arrowheadlength*perpx,yar-arrowheadlength*pary-arrowheadlength*perpy)
288   stroke()
289 end
290
291 -- better circle-approximation by using quarter circles, according to wikipedia article about Bézier
292 -- k = 1 gives a circle, everything else something else ...
293 function placecircleat(r,k,filled)
294   local P0,P1,P2,P3
295   r = r * 59.5 -- next arbitrary scale factor; the circle has radius "1" in x-units
296   local rk = 0.55228*r*k
297
298   P0 = {r,0}
299   move (P0[1],P0[2])
300
301   P1 = {r,rk} P2 = {rk,r} P3 = {0,r}
302   curve (P1,P2,P3)
303
304   P1 = {-rk,r} P2 = {-r,rk} P3 = {-r,0}
305   curve (P1,P2,P3)

```

```

306
307 P1 = {-r,-rk} P2 = {-rk,-r} P3 = {0,-r}
308 curve (P1,P2,P3)
309
310 P1 = {rk,-r} P2 = {r,-rk} P3 = {r,0}
311 curve (P1,P2,P3)
312
313 if filled then
314     fill()
315 end
316 stroke()
317 end
318
319 function placesquareat(length)
320     move (-length,-length)
321     line ( length,-length)
322     line ( length, length)
323     line (-length, length)
324     line (-length,-length)
325     stroke()
326 end
327
328 function placecurveat(x1,y1,x2,y2,x3,y3,x4,y4) -- start point and three numbers. Start is only of.
329     xfac = tex.pagewidth/gridnrx/65536 -- factors to convert given number to absolute coordinates
330     yfac = tex.pageheight/gridnry/65536 -- should both be global!
331     x2 = (x2-x1)*xfac
332     y2 = (y2-y1)*yfac
333     x3 = (x3-x1)*xfac
334     y3 = (y3-y1)*yfac
335     x4 = (x4-x1)*xfac
336     y4 = (y4-y1)*yfac
337     move(0,0) -- start
338     curve(x2,-y2,x3,-y3,x4,-y4) -- coordinates for Bezier curve
339     stroke()
340 end
341
342 function placerectangleat(x1,y1,x2,y2,filled)
343     xfac = tex.pagewidth/gridnrx/65536
344     yfac = tex.pageheight/gridnry/65536
345     x2 = (x2-x1)*xfac
346     y2 = (y1-y2)*yfac
347     move(0,0)
348     line(x2,0)
349     line(x2,y2)
350     line(0,y2)
351     line(0,0)

```

```
352  if filled then
353    fill()
354  end
355  stroke()
356 end
```