placeat

v0.1

Arno Trautmann

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 LaTeX class. This package requires LuaLaTeX; 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.

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

Contents

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*[1] the specified coordinates.[2] 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}
```

---

[1]See below for placing to the left via an optional argument.

[2]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]
\placerelto[first](2,2){content2}[second]
\placerelto[second]{2}{2}{content3}[third]
```

Although it does not make any sense, you still can use the chess-pattern notation for \placerelto. 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. For this, you can either pass the options to the package at loading time:

```
\usepackage[final]{placeat}
```

Or you use, at any time in the document, the command

```
\placeatsetup{}
```

which takes all of the package options and some more that make no sense at package loading time. ATTENTION: Actually, so far the package option interface does not work, but `\placeatsetup` is fine.

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:

`\placelineat(x1,y1)(x2,y2)`	Draws a single line pointing from `(x1,y1)` to `(x2,y2)`
`\placearrowat(x1,y1)(x2,y2)`	As the line, but with an arrowhead at the end.
`\placecircleat(x,y){r}`	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.
`\placesquareat(x,y){r}`	Draws a square with center at `(x,y)` and side lingth `r`. If omitted, `r` will default to 3.
`\placerectangleat(x1,y1)(x2,y2)`	draws a rectangle from the (upper left) corner `(x1,y1)` to the (lower right) corner `(x2,y2)`.
`\placefilledrectangleat(x1,y1)(x2,y2)`	draws a filled rectangle.

Missing are eliptical shapes, maybe rounded corners for the rectangles and maybe some funny stuff.[3]

2.1 Colored forms

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 all colors known by that package are possible:

---

[3]Yes, I *will* add a penis-shape macro, but that will not be documented explicitely.

```
\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.[4]

`\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)`

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 LaTeX 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 TeX macros that are numbered using a Lua counter; the final coordinates are also calculated by Lua. The 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! ☺

That's it for the documentation, have fun, and
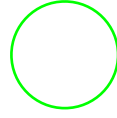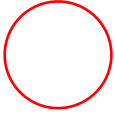
Happy TeXing!

Part II

Implementation

7 The LaTeX package: placeat.sty

Everything to get stuff working from the TeX side. Here, only a .sty file is provided and plain/ConTeXt users have to find their way. I'll happily support them, though!

7.1 Loading Files

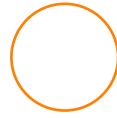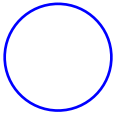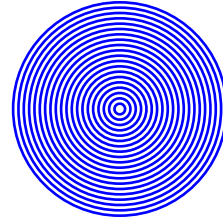The Lua file is not found by using a simple `dofile("placeat.lua")` call, but we have to use kpse's `find_file`.

```
1 \ProvidesPackage{placeat}%
2   [2014/04/15 v0.1 absolute content positioning]
3 \RequirePackage{luatexbase}
4 \RequirePackage{luacode}
5 \RequirePackage{atbegshi}
6 \RequirePackage{xparse}
7 \directlua{dofile(kpse.find_file("placeat.lua"))}
```

7.2 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 }
```

7.3   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\placerelto{oggd()d<>O{}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 }
```

7.4   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{d()O{10cm}m}{
66   \gdef\widthofplaceat{#2}
67   \placeat(#1)
68   {\begin{minipage}{\widthofplaceat}{#3}\end{minipage}}
69 }
```

7.5   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.pa
74     dacoord[nr] = "\\put("..dacoordtmp..")"
75     tex.print("\\expandafter\\gdef\\csname command"..(nr).."\\endcsname")}% begin of command defi
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 \let\ifdrawgrid\iftrue
81 \luaexec{
82   drawgrid = false
83   nr        = 0
84   dacoord  = {}
85   gridnr   = 10
86   gridnrx  = 10
87   gridnry  = 10
88   gridlinewidth = 0.01
89   offsetx  = 0
90   offsety  = 0
91 }
```

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.

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

8   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 LuaTEX*". Labeling is done by simple \put commands, controlled via Lua code.

```
105 \def\drawgrid{
106   \luatexlatelua{
107     pdf_print("q")
108     linewidth(gridlinewidth)
109     for i = 1,math.max(gridnrx,gridnry) do
110       h = i*tex.pageheight/gridnry/65536
111       w = i*tex.pagewidth/gridnrx/65536
```

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

9 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.

```
136 \NewDocumentCommand\placelineat{ou{(}u{,}u{)}u{(}u{,}u{)}}{
137   \placeat{#3}{#4}{\ignorespaces\IfValueT{#1}{\color{#1}}%
138     \luatexlatelua{placelineat(#3,-#4,#5,-#6)}
139   }
140 }
141 \NewDocumentCommand\placearrowat{ou{(}u{,}u{)}u{(}u{,}u{)}}{
142   \placeat{#3}{#4}{\ignorespaces\IfValueT{#1}{\color{#1}}%
143     \luatexlatelua{placearrowat(#3,-#4,#5,-#6)}
144   }
145 }
```

```
146 \NewDocumentCommand\placecircleat{ou{(}u{,}u{)}G{3}}{
147   \placeat{#3}{#4}{\ignorespaces\IfValueT{#1}{\color{#1}}%
148     \luatexlatelua{placecircleat(#5)}
149   }
150 }
151 \NewDocumentCommand\placesquareat{ou{(}u{,}u{)}G{3}}{
152   \placeat{#3}{#4}{\ignorespaces\IfValueT{#1}{\color{#1}}%
153     \luatexlatelua{placesquareat(#5)}
154   }
155 }
156 \NewDocumentCommand\placerectangleat{ou{(}u{,}u{)(}u{,}u{)}}{
157   \placeat{#3}{#4}{\ignorespaces\IfValueT{#1}{\color{#1}}%
158     \luatexlatelua{placerectangleat(#3,-#4,#5,-#6)}
159   }
160 }
161 \NewDocumentCommand\placefilledrectangleat{ou{(}u{,}u{)(}u{,}u{)}}{
162   \placeat{#3}{#4}{\ignorespaces\IfValueT{#1}{\color{#1}}%
163     \luatexlatelua{placefilledrectangleat(#3,-#4,#5,-#6)}
164   }
165 }
```

10   Key-Value Interface

It's a modern package, so we make use of LaTeX3 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 LaTeX3 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.

```
166 \ExplSyntaxOn
167 \keys_define:nn{placeat}{
168   alphanumgrid.code:n    = \directlua{alphanumgrid = true},
169   final.code:n           = \luaexec{placeat_final = true} \let\ifdrawgrid\iffalse,
170   drawgrid.code:n        = \global\let\ifdrawgrid\iftrue,
171   gridnumber.code:n      = \directlua{gridnr = #1 gridnrx = gridnr~gridnry = gridnr},
172   gridnumberx.code:n     = \directlua{gridnrx = #1},
173   gridnumbery.code:n     = \directlua{gridnry = #1},
174   gridlinewidth.code:n   = \directlua{gridlinewidth = #1},
175   nogrid.code:n          = \global\let\ifdrawgrid\iffalse,
176   numnumgrid.code:n      = \directlua{alphanumgrid = false},
177   offsetx.code:n         = \directlua{offsetx = #1},
178   offsety.code:n         = \directlua{offsety = #1},
179   startzero.code:n       = \directlua{offsetx = 1 offsety = 1}
180 }
```

17

```
181 \DeclareDocumentCommand\placeatsetup{m}{
182   \keys_set:nn{placeat}{#1}
183 }
184 \ExplSyntaxOff
```

11   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 outsorced once there is a Lua-to-pdf-based graphics
bundle.

```
185 function pdf_print (...)
186   for _, str in ipairs({...}) do
187     pdf.print(str .. " ")
188   end
189   pdf.print("\n")
190 end
191
192 function move (p1,p2)
193   if (p2) then
194     pdf_print(p1,p2,"m")
195   else
196     pdf_print(p1[1],p1[2],"m")
197   end
198 end
199
200 function line (p1,p2)
201   pdf_print(p1,p2,"l")
202 end
203
204 function curve(p11,p12,p21,p22,p31,p32)
205   if (p22) then
206     p1,p2,p3 = {p11,p12},{p21,p22},{p31,p32}
207   else
208     p1,p2,p3 = p11,p12,p21
209   end
210   pdf_print(p1[1], p1[2],
211             p2[1], p2[2],
212             p3[1], p3[2], "c")
213 end
214
215 function linewidth (w)
216   pdf_print(w,"w")
217 end
218
```

```lua
219 function stroke ()
220   pdf_print("S")
221 end
222
223 -- welp, let's have some fun!
224 -- with the function radd, a random coordinate change is added if used
225 -- randfact will adjust the amount of randomization
226 -- everything is relative in the grid size
227 -- BUT: In fact, do we really want to have wiggly lines? …
228 local randfact = 100
229 local radd = function()
230   return (math.random()-0.5)*randfact
231 end
232
233 function placelineat(x1,y1,x2,y2)
234   xfac = tex.pagewidth/gridnrx/65536  -- factors to convert given number to absolute coordinates
235   yfac = tex.pageheight/gridnry/65536 -- should both be global!
236   xar = (x2-x1)*xfac                  -- end point of the arrow
237   yar = (y2-y1)*yfac                  --
238   move(0,0)                           -- start
239   line(xar,yar)                       -- draw main line
240   stroke()
241 end
242
243 function placearrowat(x1,y1,x2,y2)
244   xfac = tex.pagewidth/gridnrx/65536  -- factors to convert given number to absolute coordinates
245   yfac = tex.pageheight/gridnry/65536 -- should both be global!
246   xar = (x2-x1)*xfac                  -- end point of the arrow
247   yar = (y2-y1)*yfac                  --
248   parx = xar/math.sqrt(xar^2+yar^2)   -- direction of the arrow
249   pary = yar/math.sqrt(xar^2+yar^2)   --
250   perpx = -pary                       -- perp of the arrow direction
251   perpy =  parx                       --
252   move(0,0)                           -- start
253   line(xar,yar)                       -- draw main line
254   move(xar,yar)
255   line(xar-5*parx+5*perpx,yar-5*pary+5*perpy)  -- draw arrowhead
256   move(xar,yar)
257   line(xar-5*parx-5*perpx,yar-5*pary-5*perpy)
258   stroke()
259 end
260
261 -- better circle-approximation by using quarter circles, according to wikipedia article about Béz:
262 function placecircleat(radius)
263   local k = 0.55228
264   local P0,P1,P2,P3
```

```
265
266  P0 = {radius,0}          P1 = {radius,radius*k}
267  P2 = {radius*k,radius}   P3 = {0,radius}
268
269  move  (P0[1],P0[2]) curve (P1,P2,P3)
270
271  P0 = {-radius,0}         P1 = {-radius,radius*k}
272  P2 = {-radius*k,radius}  P3 = {0,radius}
273
274  move  (P0[1],P0[2]) curve (P1,P2,P3)
275
276  P0 = {-radius,0}         P1 = {-radius,-radius*k}
277  P2 = {-radius*k,-radius} P3 = {0,-radius}
278
279  move  (P0[1],P0[2]) curve (P1,P2,P3)
280
281  P0 = {radius,0}          P1 = {radius,-radius*k}
282  P2 = {radius*k,-radius}  P3 = {0,-radius}
283
284  move  (P0[1],P0[2]) curve (P1,P2,P3)
285  stroke()
286 end
287
288 function placesquareat(length)
289  move (-length,-length)
290  line ( length,-length)
291  line ( length, length)
292  line (-length, length)
293  line (-length,-length)
294  stroke()
295 end
296
297 function placerectangleat(x1,y1,x2,y2)
298  xfac = tex.pagewidth/gridnrx/65536
299  yfac = tex.pageheight/gridnry/65536
300  x2 = (x2-x1)*xfac
301  y2 = (y2-y1)*yfac
302  move(0,0)
303  line(x2,0)
304  line(x2,y2)
305  line(0,y2)
306  line(0,0)
307  stroke()
308 end
309
310 function placefilledrectangleat(x1,y1,x2,y2)
```

```
311   xfac = tex.pagewidth/gridnrx/65536
312   yfac = tex.pageheight/gridnry/65536/1.0035  -- well, yes. Another random factor. lalala
313   x2 = (x2-x1)*xfac
314   y2 = (y2-y1)*yfac
315   linewidth(y2)
316   move(0,y2/2)
317   line(x2,y2/2)
318   stroke()
319   linewidth(1)
320 end
```