placeat

v0.1a

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 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 nonesens 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:

`\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.

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 eliptical shapes, maybe rounded corners for the rectangles and maybe some funny stuff.[3] The arrowheads need a lot of work, too, of course.

2.1 Colored forms

You need to load the XCOLOR package to use colors.[4] 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

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

[4] 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

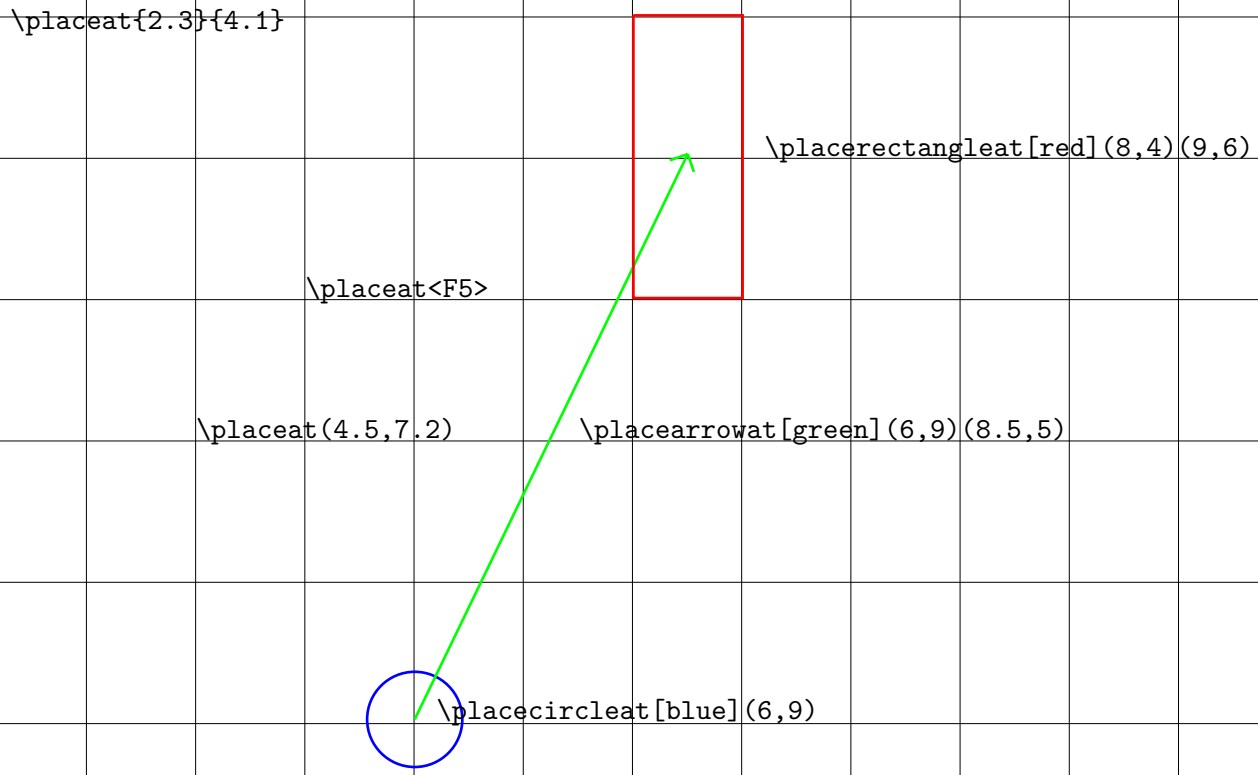
As this package makes most sense in combination with beamer, here is a small example about how to use it.

```
\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.[5]



[5] 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 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?

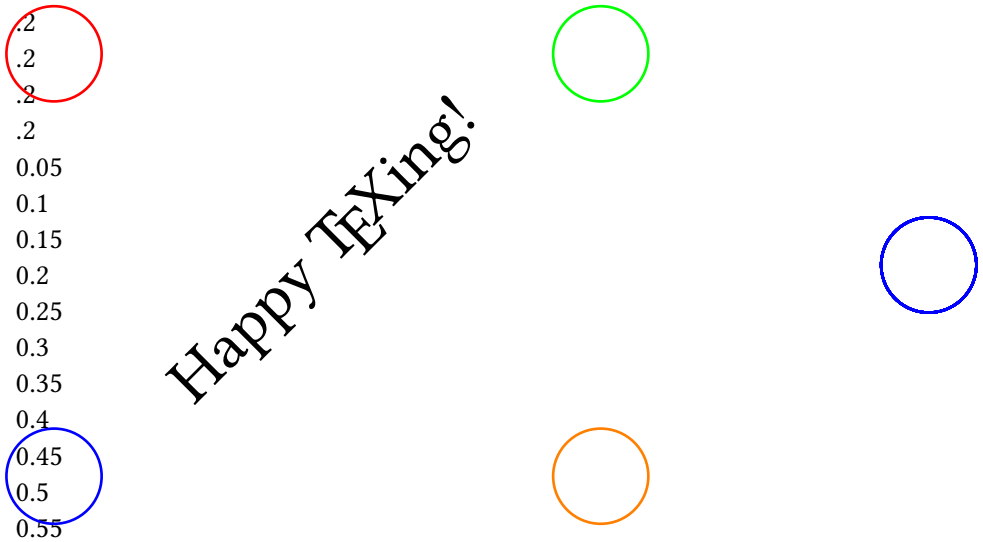
There are several ways how you can help. First, and most important:

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



.2
.2
.2
.2
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1

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`.

```
\ProvidesPackage{placeat}%
  [2014/04/18 v0.1a absolute content positioning]
\RequirePackage{luatexbase}
\RequirePackage{luacode}
\RequirePackage{atbegshi}
\RequirePackage{xparse}
\directlua{dofile(kpse.find_file("placeat.lua",'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.

```
\NewDocumentCommand\placeat{ggd()d<>O{}mo}{
  \IfValueT{#1}{                        %% two coordinates in { }{ } pair.
    \IfValueT{#2}{                      %% if second argument is not given, everything breaks. not
      \def\cox{#1}
      \def\coy{#2}
    }
  }
  \IfValueT{#3}{                        %% one argument as ( , ) coordinate pair.
    \def\cox{\firstof#3X}
```

```
    \def\coy{\secondof#3X}
  }
  \IfValueT{#4}{
    \luadirect{
      y   = string.byte('#4',1)-64
      x   = string.byte('#4',2)-48
      x2  = string.byte('#4',3)
      if x2 then x = x*10 + x2-48 end  -- FIXME: what exactly happens here? …
    }
    \def\cox{\luadirect{tex.print(x)}}
    \def\coy{\luadirect{tex.print(y)}}
  }
  \placeatthreenumbers{\cox}{\coy}{\llap{#5}#6}

  \IfValueT{#7}{
    \expandafter\gdef\csname #7x\endcsname{\firstof#3X}
    \expandafter\gdef\csname #7y\endcsname{\secondof#3X}
  }
}
```

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.

```
\NewDocumentCommand\placerelto{oggd()d<>O{}mo}{
  \IfValueT{#2}{                        %% two coordinates in { }{ } pair.
    \IfValueT{#3}{                      %% if second argument is not given, everything breaks. not
      \def\cox{#2}
      \def\coy{#3}
    }
  }
  \IfValueT{#4}{                        %% one argument as ( , ) coordinate pair.
    \def\cox{\firstof#4X}
    \def\coy{\secondof#4X}
  }
  \IfValueT{#5}{
    \luaexec{
      y   = string.byte('#5',1)-64
      x   = string.byte('#5',2)-48
      x2  = string.byte('#5',3)
      if x2 then x = x*10 + x2-48 end  -- FIXME: what exactly happens here? …
      tex.print("\\def\\cox{"..(x).."}\\def\\coy{"..(y).."}")
    }
  }
```

```
  \placeatthreenumbers
    {\cox + \csname #1x\endcsname}
    {\coy + \csname #1y\endcsname}
    {\llap{#6}#7}
  \IfValueT{#8}{
    \expandafter\xdef\csname #8x\endcsname{\cox + \csname #1x\endcsname}
    \expandafter\xdef\csname #8y\endcsname{\coy + \csname #1y\endcsname}
  }
}
```

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.

```
\NewDocumentCommand\placeminipageat{d()O{10cm}m}{
  \gdef\widthofplaceat{#2}
  \placeat(#1)
  {\begin{minipage}{\widthofplaceat}{#3}\end{minipage}}
}
```

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.

```
\def\placeatthreenumbers#1#2#3{
  \luaexec{
    nr = nr+1
    dacoordtmp = ((#1-1+offsetx)*tex.pagewidth/65536/gridnrx*1.005)..","..(-(#2-1+offsety)*tex.pag
    dacoord[nr] = "\\put("..dacoordtmp..")"
    tex.print("\\expandafter\\gdef\\csname command"..(nr).."\\endcsname")}% begin of command defi
  {#3}  %% this is what \command[nr] will contain
}
```

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

```
\def\firstof #1,#2X{#1}
\def\secondof #1,#2X{#2}
```

Setup of variables and macros we need later.

```
\let\ifdrawgrid\iftrue
\luaexec{
  drawgrid = false
  nr       = 0
  dacoord  = {}
  gridnr   = 10
  gridnrx  = 10
  gridnry  = 10
  gridlinewidth = 0.01
  offsetx  = 0
  offsety  = 0
}
```

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.

```
\AtBeginDocument{
  \AtBeginShipout{%
    \AtBeginShipoutUpperLeftForeground{%
      \ifdrawgrid\drawgrid\fi
      \luaexec{%
        for i = 1,nr do
          tex.print(dacoord[i].."{\\csname command"..(i).."\\endcsname}")
        end
        nr=0
      }
    }
  }
}
```

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.

```
\def\drawgrid{
  \luatexlatelua{
    pdf_print("q")
    linewidth(gridlinewidth)
    local factorh = tex.pageheight/gridnry/65536
    local factorw = tex.pagewidth/gridnrx/65536
    for i = 1,math.max(gridnrx,gridnry) do
```

```
      h = i*factorh
      w = i*factorw
      move(0,-h) line(tex.pagewidth,-h) stroke()
      move(w,0) line(w,-tex.pageheight) stroke()
    end
    pdf_print("Q")
  }
  { %% extra grouping to keep font size change local. Going to normalfont seems to make sense. An
%% would also be nice to maybe adapt the fontsize to the grid size
    \normalfont\fontsize{8}{10}\selectfont
    \luaexec{
      for i=1,math.max(gridnrx+offsetx,gridnry+offsety) do
        hfac = tex.pageheight/gridnry/65536
        wfac = tex.pagewidth/gridnrx/65536*1.005  %% another empirical factor
        h = (i-1)*hfac
        w = (i-1)*wfac
        tex.print("\\put("..(w)..",-7){\\rlap{"..(i-offsetx)..."}}")
        if alphanumgrid then
          tex.print("\\put(0,"..(-h-0.05*hfac)..."){\\char00"..(64+i-offsety)..."}")  %%-- for alpha
        else
          tex.print("\\put(0,"..(-h-0.05*hfac)..."){"..(i-offsety)..."}")
        end
      end
    }
  }
}
```

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.

```
\NewDocumentCommand\placelineat{od()d()}{
  \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
    \luatexlatelua{placelineat(#2,#3)}
  }
}
\NewDocumentCommand\placearrowat{od()d()}{
  \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
    \luatexlatelua{placearrowat(#2,#3)}
```

```
  }
}
\NewDocumentCommand\placecircleat{od()D(){.3}}{
  \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
    \luatexlatelua{placecircleat(#3,1)}
  }
}
\NewDocumentCommand\placefilledcircleat{od()D(){.3}}{
  \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
    \luatexlatelua{placecircleat(#3,1,true)}
  }
}
\NewDocumentCommand\placesquareat{od()G{3}}{
  \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
    \luatexlatelua{placesquareat(#3)}
  }
}
\NewDocumentCommand\placecurveat{od()d()d()d()}{
 \placeat(#2){\ignorespaces\IfValueT{#1}{\color{#1}}%
    \luatexlatelua{placecurveat(#2,#3,#4,#5)}
  }
}
\NewDocumentCommand\placerectangleat{O{black}d()d()}{
  \placeat(#2){\ignorespaces\color{#1}%
    \luatexlatelua{placerectangleat(#2,#3)}
  }
}
\NewDocumentCommand\placefilledrectangleat{O{black}d()d()}{
  \placeat(#2){\ignorespaces\color{#1}%
    \luatexlatelua{placerectangleat(#2,#3,true)}
  }
}
\NewDocumentCommand\placeroundedat{sO{black}d()D(){0.1}D<>{1.5}}{
  \placeat(#3){\ignorespaces\color{#2}%
    \IfBooleanTF{#1}{\luatexlatelua{placecircleat(#4,#5,true)}}%
                    {\luatexlatelua{placecircleat(#4,#5)}}
  }
}
```

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.

```
\ExplSyntaxOn
\keys_define:nn{placeat}{
  alphanumgrid.code:n    = \directlua{alphanumgrid = true},
  final.code:n           = \luaexec{placeat_final = true} \let\ifdrawgrid\iffalse,
  drawgrid.code:n        = \global\let\ifdrawgrid\iftrue,
  gridnumber.code:n      = \directlua{gridnr = #1 gridnrx = gridnr~gridnry = gridnr},
  gridnumberx.code:n     = \directlua{gridnrx = #1},
  gridnumbery.code:n     = \directlua{gridnry = #1},
  gridlinewidth.code:n   = \directlua{gridlinewidth = #1},
  linewidth.code:n       = {\placeat(1,1){\luatexlatelua{linewidth(#1)}}},  %% FIXME: this is a ve
  nogrid.code:n          = \global\let\ifdrawgrid\iffalse,
  numnumgrid.code:n      = \directlua{alphanumgrid = false},
  offsetx.code:n         = \directlua{offsetx = #1},
  offsety.code:n         = \directlua{offsety = #1},
  startzero.code:n       = \directlua{offsetx = 1 offsety = 1}
}
\DeclareDocumentCommand\placeatsetup{m}{
  \keys_set:nn{placeat}{#1}
}
\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.

```
function pdf_print (...)
  for _, str in ipairs({...}) do
    pdf.print(str .. " ")
  end
  pdf.print("\n")
end

function move (p1,p2)
  if (p2) then
    pdf_print(p1,p2,"m")
  else
    pdf_print(p1[1],p1[2],"m")
  end
end

```

```
function line(p1,p2)
  pdf_print(p1,p2,"l")
end

function curve(p11,p12,p21,p22,p31,p32)
  if (p22) then
    p1,p2,p3 = {p11,p12},{p21,p22},{p31,p32}
  else
    p1,p2,p3 = p11,p12,p21
  end
  pdf_print(p1[1], p1[2],
              p2[1], p2[2],
              p3[1], p3[2], "c")
end

function linewidth(w)
  pdf_print(w,"w")
end

function fill()
  pdf_print("f")
end

function stroke()
  pdf_print("S")
end

-- welp, let's have some fun!
-- with the function radd, a random coordinate change is added if used
-- randfact will adjust the amount of randomization
-- everything is relative in the grid size
-- BUT: In fact, do we really want to have wiggly lines? …
local randfact = 100
local radd = function()
  return (math.random()-0.5)*randfact
end

function placelineat(x1,y1,x2,y2)
  xfac = tex.pagewidth/gridnrx/65536  -- factors to convert given number to absolute coordinates
  yfac = tex.pageheight/gridnry/65536 -- should both be global!
  xar = (x2-x1)*xfac                  -- end point of the arrow
  yar = (y1-y2)*yfac                  --
  move(0,0)                           -- start
  line(xar,yar)                       -- draw main line
  stroke()
end
```

```
265
266 function placearrowat(x1,y1,x2,y2)
267   xfac = tex.pagewidth/gridnrx/65536  -- factors to convert given number to absolute coordinates
268   yfac = tex.pageheight/gridnry/65536 -- should both be global!
269   xar = (x2-x1)*xfac                  -- end point of the arrow
270   yar = (y1-y2)*yfac                  --
271   parx = xar/math.sqrt(xar^2+yar^2)   -- direction of the arrow
272   pary = yar/math.sqrt(xar^2+yar^2)   --
273   perpx = -pary                       -- perp of the arrow direction
274   perpy =  parx                       --
275   move(0,0)                           -- start
276   line(xar,yar)                       -- draw main line
277   move(xar,yar)
278   line(xar-5*parx+5*perpx,yar-5*pary+5*perpy)  -- draw arrowhead
279   move(xar,yar)
280   line(xar-5*parx-5*perpx,yar-5*pary-5*perpy)
281   stroke()
282 end
283
284 -- better circle-approximation by using quarter circles, according to wikipedia article about Bézi
285 -- k = 1 gives a circle, everything else something else …
286 function placecircleat(r,k,filled)
287   local P0,P1,P2,P3
288   r = r * 59.5 -- next arbitrary scale factor; the circle has radius "1" in x-units
289   local rk = 0.55228*r*k
290
291   P0 = {r,0}
292   move  (P0[1],P0[2])
293
294   P1 = {r,rk}   P2 = {rk,r}   P3 = {0,r}
295   curve (P1,P2,P3)
296
297   P1 = {-rk,r}  P2 = {-r,rk}  P3 = {-r,0}
298   curve (P1,P2,P3)
299
300   P1 = {-r,-rk} P2 = {-rk,-r} P3 = {0,-r}
301   curve (P1,P2,P3)
302
303   P1 = {rk,-r}  P2 = {r,-rk}  P3 = {r,0}
304   curve (P1,P2,P3)
305
306   if filled then
307     fill()
308   end
309   stroke()
310 end
```

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