placeat

v0.0a

Arno Trautmann
arno.trautmann@gmx.de

Abstract

The package placeat offers the command `\placeat<D4>{}` 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.

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 package option `nogrid` or use the 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

ě

Contents

ě

Part I
User Documentation

1 How do I use it?

1.1 User Commands

The command `\placeat` takes several arguments, the last of which is the content you want to place. This may range from single letters to graphic objects or (mostly) any valid LaTeX code. Exceptions are floating environments – you have to pack them into a minipage or similar construct. You can do this by hand or use the command `\placeatminipage`. This command only has one kind of interface, the one with two braces:

```
\placeatminipage{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`.

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

Actually, this should also be possible with the `( , )` notation, but it is not so far. This is a bug, but I don't know yet how to solve it. In the end, if you need a finer grid to place your stuff, just increase the gridnumber.

1.2 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

```
a
\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.

ẽ

1.3 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 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}
```

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.

3 Example

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}
\placeatminipage{4}{5}[3cm]{\includegraphics{fermi_gas_1}}
\end{frame}
\end{document}
```

č

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.[1]

`\placeat{2.3}{4.1}`

`\placeat<E5>`

`\placeat(4.5,7.2)`

[1] Don't let me fool you, the code is not printed using `\verb`, but only with a `\texttt`.

ě

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 How can I help?

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

Testing. As I have no typographical experience with the letterspacing in (complex) everyday documents, you can greatly improve the default parameters and overall settings and features by telling my how everything turns out on your document.

Bug reporting: There are surely many situations were the simple approaches of this package do not work. Help me identifying and – in the best case – solve them!

That's it for the documentation, have fun trying this out, and

ě

Happy TEXing!

ě

Part II
Implementation

6 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!

6.1 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}%
  [2013/04/08 v0.0a placeat package]
\input{luatexbase.sty}
\RequirePackage{xparse}
\RequirePackage{luacode}
\directlua{dofile(kpse.find_file("placeat.lua"))}
```

6.2 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.

```
\NewDocumentCommand\placeat{ggd()d<>m}{
  \IfValueT{#1}{                          %% two coordinates in { }{ } pair.
    \IfValueT{#2}{                        %% if second argument is not given, everything breaks. not
      \placeatthreenumbers{#1}{#2}{#5}
    }
  }
  \IfValueT{#3}{                          %% one argument as ( , ) coordinate pair.
    \placeatthreenumbers{\firstof#3}{\secondof#3}{#5}
  }
  \IfValueT{#4}{
    \luaexec{
      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
      tex.print("\\placeatthreenumbers{"..(x).."}{"..(y+1).."}")}{#5}
  }
}
```

ě

6.3 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 `\placeatminipage` is the same as using `\placeat{}{}{content}` where content is packed into a minipage. The first two argument of `\placeatminipage` 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\placeatminipage{mmom}{
  \IfValueTF{#3}{\gdef\widthofplaceat{#3}}{\gdef\widthofplaceat{10cm}}
  \placeat{#1}{#2}{\begin{minipage}{\widthofplaceat}{#4}\end{minipage}}
}
```

6.4 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.

```
\def\placeatthreenumbers#1#2#3{
  \luaexec{
    nr = nr+1
    dacoordtmp = ((#1-1)*tex.pagewidth/65536/gridnr*1.005)..","..(-(#2-1)*tex.pageheight/65536/gri
    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,#2{#1}
\def\secondof #1,#2{#2}
```

Setup of variables and macros we need later.

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

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.

ě

I'm trying to understand Heiko's code by adapting it step by step to maybe improve it to my special needs here, but so far I have troubles understanding everything. This means that the following definitions may change from time to time, but the functionality should always be the same. However, if I recognize that any of my changes will decrease the flexibility for some special cases, I will also offer a version that is based on Heiko's code without changes.

```
\def\placeatAddToBoxForeground#1{%
  \AtBeginShipoutBox
  \edef\placeatrestore{%
    \vfuzz=\the\vfuzz\relax
    \vbadness=\the\vbadness\relax
    \dimen\ltx@zero=\the\dimen\ltx@zero\relax
  }%
  \edef\placeatrestorebox{%
    \ht\AtBeginShipoutBox=\the\ht\AtBeginShipoutBox\relax
    \dp\AtBeginShipoutBox=\the\dp\AtBeginShipoutBox\relax
  }%
  \dimen\ltx@zero=\ht\AtBeginShipoutBox
  \advance\dimen\ltx@zero by \dp\AtBeginShipoutBox
  \setbox\AtBeginShipoutBox=\vbox to \dimen\ltx@zero{%
    \setbox\ltx@zero=\hbox{%
      \begingroup
        \placeatrestore
        #1%
      \endgroup
    }%
    \wd\ltx@zero=0pt\relax
    \ht\ltx@zero=0pt\relax
    \dp\ltx@zero=0pt\relax
    \baselineskip=0pt\relax
    \lineskip=0pt\relax
    \lineskiplimit=0pt\relax
    \unvbox\AtBeginShipoutBox
    \kern-\dimen\ltx@zero
    \copy\ltx@zero
    \kern\dimen\ltx@zero
  }%
  \placeatrestore
  \placeatrestorebox
}

\def\placeatUpperLeftForeground#1{%
  \placeatAddToBoxForeground{%
    \kern-\pdfhorigin\relax
    \vbox to 0pt{%
      \kern-\pdfvorigin\relax
```

```
ẽ
      \begingroup
      \picture(0,0)\relax
      \ignorespaces
      #1%
      \endpicture
      \endgroup
      \vss
    }%
  }%
}

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

7 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)
    for i = 1,gridnr do
      h = i*tex.pageheight/gridnr/65536
      w = i*tex.pagewidth/gridnr/65536
      move(0,-h) line(tex.pagewidth,-h) stroke()
      move(w,0) line(w,-tex.pageheight) stroke()
    end
    pdf_print("Q")
  }
  \fontsize{8}{10}\selectfont
  \luaexec{
```

```
  ẽ
    for i=1,gridnr do
      h = i*tex.pageheight/gridnr/65536
      w = i*tex.pagewidth/gridnr/65536
      tex.print("\\put("..(w)..",-7){\\llap{"..i.."}}")
      tex.print("\\put("..(0)..","..(-h)..."){\\char00"..(64+i).."}")
    end
  }
}
```

8 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!

```
\ExplSyntaxOn
\keys_define:nn{placeat}{
  final.code:n           = \directlua{placeat_final = true drawgrid = false},
  nogrid.code:n          = \let\ifdrawgrid\iffalse,
  drawgrid.code:n          = \let\ifdrawgrid\iftrue,
  gridnumber.code:n      = \directlua{gridnr = #1},
  gridlinewidth.code:n   = \directlua{gridlinewidth = #1}
}
\DeclareDocumentCommand\placeatsetup{m}{
  \keys_set:nn{placeat}{#1}
}
\ExplSyntaxOff
```

9 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("\string\n")
end

function move (p1,p2)
  pdf_print(p1,p2,"m")
end

function line (p1,p2)
```

```
  ĕ
157   pdf_print(p1,p2,"l")
158 end
159
160 function linewidth (w)
161   pdf_print(w,"w")
162 end
163
164 function stroke ()
165   pdf_print("S")
166 end
```