**1. Baseline grid.** This package modifies TEX's page building routine so that all the main baselines are on an equally spaced (by `\baselineskip`) grid. This not only mimicks traditional typesetting, but also results in a more pleasing document.

- In multiple colum documents, where the problem is most visible, it is not uncommon for baselines not to align accross colums, resulting in an unbalanced layout.
- In single page documents the problem is less obvious, but it shows up both in facing pages – where each page can be considered to be one column in a double-column document –, and on the same sheet of paper, if the baselines of the pages on that sheet do not align. This last problem is more visible on more "transparent" paper, but one can readily see it, for example, by looking at page 51 of the TEXbook against light: although the baselines of page 51 and 52 start aligned, towards the end of the page they are completely out of sync.

**2.** The solution for this problem seems to have eluded all trials so far, and this is not surprising: one needs to change the page building process to make it work. In particular, at every point when TEX is considering the addition of vertical material to the output box, we must filter it to make sure said material would end up aligned to the baseline grid. This is very hard to do (impossible?) in TEX alone, but LuaTEX has enough hooks to make this possible.

**3.** I will describe the algorithm in more detail later, but the idea is as follows: Firstly, we will keep track of the total height of what was already added to the output box. Let us call that value $b_h$. Then, we process each element in the TEX recent vertical contribution list.

- If the element is a glue with natural dimension $g$, we remove all stretch from it, and update $b_h \leftarrow b_h + g$.
- If the element is a horizontal list with height $h_h$ and depth $h_d$, we insert enough glue $g$ before it to make $(b_h+g+h_h) \bmod x = 0$ (in other words, to align its baseline with the grid), and update $b_h \leftarrow b_h+g+h_d+h_h$.

There are more things we need to do, but the basic idea should be clear: we want to only feed TEX's output box with things that are already aligned to the baseline grid, and give TEX no opportunity to change that by not giving it any room to stretch. One could say we want to make the output box look like an airplane's "economy class" section.

**4. Using this package.** This package is one part of my TEX macro set, and has to be used inside it. This means you must be willing to use Eplain instead of LATEX, for example. Making this package portable should not be too difficult if you wish to do so – if you do, please publish your work so others can use it.

If you still want to use this, clone my `texmf` tree from GitHub, at `https://github.com/ccrusius/texmf`, make sure TEX can find it, and add the following to your document's preamble. The call to `\ccgridsetup` is to make sure everything is properly arranged: it is already called once when you include the package, but if your preamble changes things after that `ccgrid` may have to recompute some values.

⟨ Enabling the baseline grid in your document 4 ⟩ ≡

```
\input ccgrid
% ... rest of preamble ...
\ccgridsetup
```

## 5. The package files.

**6.** ⟨ *xxxccgrid.lua 6 ⟩ ≡
⟨ Lua global variables 11 ⟩
⟨ Lua functions 14 ⟩
```
return { ⟨ Lua module exports 15 ⟩ }
```

**7.** ⟨ *ccgrid.tex 7 ⟩ ≡
⟨ TEX package preamble 8 ⟩
⟨ TEX global variables 12 ⟩
⟨ TEX macros 13 ⟩
⟨ Set up grid parameters 21 ⟩
⟨ TEX package postamble 9 ⟩

**8.** ⟨ TEX package preamble 8 ⟩ ≡
```
\input ccbase
\pragmaonce{ccgrid}
\input ccshowbox
\directlua{ccgrid = dofile(kpse.find_file("ccgrid.lua"))}
\makeatletter
```

**9.** ⟨ TEX package postamble 9 ⟩ ≡
```
\makeatother
\endinput
```

**10. Debugging.** Things did go wrong quite often with this, as baseline gridding is not something TEX was designed to do, so I had to set up a decent enough debugging infrastructure. You will probably not be debugging this package, but the code needs to be here anyway. In order to control what debugging messages are printed, you have to set `ccgrid`'s log level to a suitable value, as follows:

| Level | Information |
|---|---|
| 0 | Nothing. |
| 1 | Adds a baseline grid to every page. |
| 2 | Prints the contents of `\box255` every page. |
| 3 | Trace page building process. |

**11.** In Lua, the log level is stored in the global `loglevel` variable. We will keep a copy of in in TEX's `\ccgridloglevel` counter. To keep both in sync, you should always change the log level by calling the `\setccgridloglevel` TEX macro.

⟨ Lua global variables 11 ⟩ ≡
```
local loglevel = 0
```

**12.** ⟨ TEX global variables 12 ⟩ ≡
```
\newcount\ccgridloglevel\ccgridloglevel=0
```

**13.** ⟨ TEX macros 13 ⟩ ≡
```
\def\setccgridloglevel#1{%
  \directlua{ccgrid.setloglevel(#1)}%
  \global\ccgridloglevel=#1}
```

**14.** ⟨ Lua functions 14 ⟩ ≡
```
local function setloglevel(x) loglevel = x end
```

**15.** ⟨ Lua module exports 15 ⟩ ≡
```
setloglevel = setloglevel,
```

**16.** Most of the debugging messages are printed by the Lua module via a call to the `typeout` function below:

⟨ Lua functions 14 ⟩ +≡
```
local function typeout(lvl,str)
  if loglevel >= lvl then texio.write_nl(str) end
end
```

**17. Displaying a baseline grid.** To enable the display of a baseline grid at every page, use the macro below. Doing this before a final build is a good idea, as it will quickly tell you whether things are working or not. The macro works by setting the appropriate log level, as described previously.

⟨ TEX macros 13 ⟩ +≡
```
\def\ccgriddraft{\setccgridloglevel{1}}
```

**18.** To print a baseline grid at every page, we redefine the `\output` routine.

⟨ TEX macros 13 ⟩ +≡
```
\newtoks\ccgrid@prevoutput
\ccgrid@prevoutput=\expandafter{\the\output}
\output={%
  \ifnum\ccgridloglevel>0%
    \setbox0=⟨ Baseline grid box 19 ⟩
    \setbox255=\vbox to\vsize{\vtop to0pt{\box0\vss}\hrule height 0pt\box255}
  \fi
  \the\ccgrid@prevoutput}
```

**19.** ⟨ Baseline grid box 19 ⟩ ≡
```
\vbox to\vsize{
  ⟨ Baseline grid rule 20 ⟩
  \vskip\topskip
  \cleaders\vbox to\baselineskip{
    ⟨ Baseline grid rule 20 ⟩
    \vfil%
    ⟨ Baseline grid rule 20 ⟩}
  \vfill}
```

**20.** ⟨ Baseline grid rule 20 ⟩ ≡
```
\kern-0.2pt\hrule height0.2pt depth0.2pt width\hsize\kern -0.2pt
```

**21. Parameter initialization.** A gridded TEX run must have some parameters properly initialized. This includes removing stretch from all known skips and sizing things such as `\vsize` properly. All of this is done in the `\ccgridsetup` macro, which is called automatically when `ccgrid.tex` is read.

⟨ Set up grid parameters 21 ⟩ ≡
```
\def\ccgridsetup{
  ⟨ Remove glue from \baselineskip and set the Lua grid 28 ⟩
  ⟨ Remove glue from other TEX skips 29 ⟩
  ⟨ Set \vsize to a multiple of \baselineskip 30 ⟩
  ⟨ Set \lineskip and \lineskiplimit 31 ⟩
  \raggedbottom% We do our own thing, but let's tell others this is the intent
}
\ccgridsetup
```

**22.** The spacing for our grid should be essentially `\baselineskip`, but we need to fix this value at the beginning of the document in a separate variable, since TEX may change it mid-course. In Lua, we keep the value in the `baselineskip` variable, and the user can only change it by first setting `\baselineskip` accordingly, and then calling the TEX`\ccgridseput` macro.

⟨ Lua global variables 11 ⟩ +≡
```
local baselineskip = 0
```

**23.** ⟨ Lua functions 14 ⟩ +≡
```
local function setgrid(x) baselineskip=x end
```

**24.** ⟨ Lua module exports 15 ⟩ +≡
```
setgrid = setgrid,
```

**25.** Some basic math has to be performed when setting these parameters, and that is done on the Lua side of things.

⟨ Lua functions 14 ⟩ +≡
```
local function snapdown(x)
  return baselineskip*math.floor(x/baselineskip)
end
local function freeze(x)
  return (ccbase.spstr(ccbase.tosp(x)))..." plus 0pt minus 0pt"
end
```

**26.** ⟨ Lua module exports 15 ⟩ +≡
```
snapdown = snapdown,
freeze = freeze,
```

**27.** ⟨ TEX macros 13 ⟩ +≡
```
\def\ccgrid@freeze#1{\directlua{tex.print(ccgrid.freeze("#1"))}}
```

**28.** ⟨ Remove glue from `\baselineskip` and set the Lua grid 28 ⟩ ≡

```
\global\baselineskip=\ccgrid@freeze{\the\baselineskip}
\directlua{ccgrid.setgrid(ccbase.tosp("\the\baselineskip"))}
\typeout{ccgrid: baselineskip=\the\baselineskip}
```

**29.** ⟨ Remove glue from other TEX skips 29 ⟩ ≡

```
\global\topskip=\ccgrid@freeze{\the\topskip}
\global\parskip=\ccgrid@freeze{\the\parskip}
\global\abovedisplayskip=\ccgrid@freeze{\the\abovedisplayskip}
\global\belowdisplayskip=\ccgrid@freeze{\the\belowdisplayskip}
\global\abovedisplayshortskip=\ccgrid@freeze{\the\abovedisplayshortskip}
\global\belowdisplayshortskip=\ccgrid@freeze{\the\belowdisplayshortskip}
```

**30.** ⟨ Set `\vsize` to a multiple of `\baselineskip` 30 ⟩ ≡

```
\global\advance\vsize by-\topskip
\global\vsize=\directlua{tex.print(%
  ccbase.spstr(ccgrid.snapdown(ccbase.tosp("\the\vsize"))))}
\global\advance\vsize by\topskip
\typeout{ccgrid: vsize=\the\vsize}
```

**31.** ⟨ Set `\lineskip` and `\lineskiplimit` 31 ⟩ ≡

```
\global\lineskip=0pt
\global\lineskiplimit=-0.5\baselineskip
```