The LuaTEX way: \framed

Hans Hagen
Pragma ADE
http://pragma-ade.com

Abstract

ConTEXt's \framed macro has many flexible options for typesetting a paragraph. This short note discusses its reimplementation in Lua for ConTEXt MkIV.

1 \framed and its width

One of the more powerful commands in ConTEXt is \framed. You can pass quite a few parameters that control the spacing, alignment, backgrounds and more. This command is used all over the place (although often hidden from the user) which means that it also has to be quite stable.

Unfortunately, there is one nasty bit of code that is hard to get right. Calculating the height of a box is not that complex: the height that TEX reports is indeed the height. However, the width of box is determined by the value of \hsize at the time of typesetting. The actual content can be smaller. In the \framed macro by default the width is calculated automatically.

```
\framed
  [align=middle,width=fit]
  {Out beyond the ethernet the spectrum spreads
   \unknown}
```

this shows up as (taken from 'Casino Nation' by Jackson Browne):

Out beyond the ethernet
the spectrum spreads . . .

Or take this quote (from 'A World Without Us' by Alan Weisman):

```
\hsize=.6\hsize
\framed [align=middle,width=fit]
  {\input weisman }
```

This gives a multi-line paragraph:

Since the mid-1990s, humans have taken an unprecedented step in Earthly annals by introducing not just exotic flora or fauna from one ecosystem into another, but actually inserting exotic genes into the operating systems of individual plants and animals, where they're intended to do exactly the same thing: copy themselves, over and over.

Here the outer \hsize was made a bit smaller. As you can see the frame is determined by the widest line. Because it was one of the first features we needed, the code in ConTEXt that is involved in determining the maximum natural width is pretty old. It boils down to unboxing a \vbox and stepwise grabbing the last box, penalty, kern and skip. That is, we unwind the box backwards.

However, one cannot grab everything; or, in TEX speak: there is only a limited number of \lastsomething commands. Special nodes, such as whatsits, cannot be grabbed and make the analyzer abort its analysis. There is no way that we can solve this in traditional TEX and in ConTEXt MkII.

2 \framed with LuaTEX

So how about LuaTEX and ConTEXt MkIV? The macro used in the \framed command is:

```
\doreshapeframedbox{do something
                    with \box\framebox}
```

In LuaTEX we can manipulate box content at the Lua level. Instead of providing a truckload of extra primitives (which would also introduce new data types at the TEX end) we delegate the job to Lua.

```
\def\doreshapeframedbox
  {\ctxlua{commands.doreshapeframedbox
            (\number\framebox)}}
```

Here \ctxlua is our reserved instance for ConTEXt, and commands provides the namespace for commands that we delegate to Lua (so, there are more of them). The amount of Lua code is far smaller than the TEX code (which we will not show here; it's in supp-box.tex if you want to see it).

```
function commands.doreshapeframedbox(n)
if tex.wd[n] ~= 0 then
  local hpack = node.hpack
  local free = node.free
  local copy = node.copy_list
  local noflines, lastlinelength, width = 0,0,0
  local list = tex.box[n].list
  local done = false
  for h in node.traverse_id('hlist',list) do
   done = true
```

```
   local p = hpack(copy(h.list))
   lastlinelength = p.width
   if lastlinelength > width then
     width = lastlinelength
   end
   p.list = nil
   free(p)
  end
  if done then
   if width ~= 0 then
    for h in node.traverse_id('hlist',list) do
     if h.width ~= width then
       h.list = hpack(h.list,width,'exactly')
       h.width = width
     end
    end
   end
   tex.wd[n] = width
  end
  -- we can also work with lastlinelength
 end
end
```

In the first loop we inspect all lines (nodes with type `hlist`) and repack them to their natural width with `node.hpack`. In the process we keep track of the maximum natural width. In the second loop we repack the content again, this time permanently. Now we use the maximum encountered width which is forced by the keyword `exactly`. Because all glue is still present we automatically get the desired alignment. We create local shortcuts to some node functions which makes it run faster; keep in mind that this is a core function called many times in a regular ConTEXt job.

In looking at ConTEXt MkIV you will find quite a lot of Lua code and often it looks rather complex, especially if you have no clue why it's needed. Think of OpenType font handling which involves locating fonts, loading and caching them, storing features and later on applying them to node lists, etc.

However, once we are beyond the stage of developing all the code that is needed to support the basics, we will start doing the things that relate more to the typesetting process itself, such as the previous code. One of the candidates for a similar Lua-based solution is for instance column balancing. From the previous example code you can deduce that manipulating the node lists from Lua can make that easier. Of course we'll be a few more years down the road by then.