low level TEX

debugging

Contents

1 Introduction

Below there will be some examples of how you can see what TEX is doing. We start with some verbose logging but then move on to the more visual features. We occasionally point to some features present in the LuaMetaTEX engine. More details about what is possible can be found in documents in the ConTEXt distribution, for instance the 'lowlevel' manuals.

Typesetting involves par building, page building, inserts (footnotes, floats), vertical adjusters (stuff before and after the current line), marks (used for running headers and footers), alignments (to build tables), math, local boxes (left and right of lines), hyphenation, font handling, and more and each has its own specific ways of tracing, either provided by the engine, or by ConTEXt itself. You can run `context --trackers` to get a list of what ConTEXt can do, as it lists most of them. But we start with the language, where tokens play an important role.

2 Token lists

There are two main types of linked lists in TEX: token lists and node lists. Token lists relate to the language and node lists collect (to be) typeset content and are used for several stack based structures. Both are efficiently memory managed by the engine. Token lists have only forward links, but node lists link in both directions, at least in LuaTEX and LuaMetaTEX.

When you define a macro, like the following, you get a token list:

```
\def\test#1{\bgroup\bf#1\egroup}
```

Internally the `\test` macro has carry the argument part and the body, and each is encoded as a number plus a pointer to the next token.

control sequence: test				
586923	19	49	match	argument 1
97602	20	0	end match	
591826	1	123	left brace	bgroup
478610	143	0	protected call	bf
586987	21	1	parameter reference	
377383	2	125	right brace	egroup

Here the first (large) number is a memory location that holds two 4 byte integers per token: the so called info part codes the command and sub command, the two smaller numbers in the table, and a link part that points to the next memory location, here the nest row. The last columns provide details. A character like 'a' is one token, but a control sequence like `\foo` is also one token because every control sequence gets a number. So, both take eight bytes of memory which is why a format file can become large and memory consumption grows the more macros you use.

In the body of the above `\test` macro we used `\bf` so let's see how that looks:

permanent protected control sequence: bf				
628	137	24	if test	ifmmode
629	131	0	expand after	expandafter
630	143	0	protected call	mathbf
631	137	3	if test	else
632	131	0	expand after	expandafter
633	143	0	protected call	normalbf
634	137	2	if test	fi

Here the numbers are much lower which is an indication that they are likely in the format. They are also ordered, which is a side effect of LuaMetaTEX making sure that the token lists stored in the format file keep their tokens close together in memory which could potentially be a bit faster. But, when we are in a production run, the tokens come from the pool of freed or additionally allocated tokens:

```
\tolerant\permanent\protected\def\test[#1]#:#2%
```

```
{{\iftok{#1}{sl}\bs\else\bf\fi#2}}
```

Gives us:

permanent tolerant protected control sequence: test						
71064	12	91	other char	[	U+0005B	
249906	19	49	match			argument 1
611915	12	93	other char	]	U+0005D	
332190	19	58	match			argument :
611889	19	50	match			argument 2
332233	20	0	end match			
332180	1	123	left brace			
611794	137	29	if test			iftok
478572	1	123	left brace			
97569	21	1	parameter reference			
478594	2	125	right brace			
332217	1	123	left brace			
71011	11	115	letter	s	U+00073	
611847	11	108	letter	l	U+0006C	
332221	2	125	right brace			
611829	143	0	protected call			bs
332218	137	3	if test			else
611787	143	0	protected call			bf
332216	137	2	if test			fi
332226	21	2	parameter reference			
611739	2	125	right brace			

If you are familiar with TEX and spend some time looking at this you will start recognizing entries. For instance `11 115` translates to `letter s` because `11` is the so called command code of letters (also its `\catcode`) and the `s` has utf8 value `115`. The LuaMetaTEX specific `\iftok` conditional has command code `135` and sub code `29`. Internally these are called `cmd` and `chr` codes because in many cases it's characters that are the sub commands.

There is more to tell about these commands and the way macros are defined, for instance `tolerant` here means that we can omit the the first argument (between brackets) in which case we pick up after the `#:`. With `protected` we indicate that the macro will not expand in for instance an `\edef` and `permanent` marks the macro as one that a user cannot redefine (assuming that overload protection is enabled). The extended macro argument parsing features and macro overload protection are something specific to LuaMetaTEX.

These introspective tables can be generated with:

```
\luatokentable\test
```

after loading the module `system-tokens`. The reason for having a module and not a built-in tracer is that users seldom want to do this. Instead they might use `\showluatokens\test` that just reports something similar to the console and/or log file.

There is much more to tell but most users have no need to look into these details unless they are curious about what TEX does. In that case using `tracingall` and inspecting the log file can be revealing too, but be prepared for huge files. In LuaMetaTEX we have tried to improve these traces a bit but that's of course subjective and even then logs can become huge. But even if one doesn't understand all that is shown, it gives an impression about how much work TEX is actually doing.

3 Node lists

A node list is what you get from input that is (to be) typeset. There are several ways to see what node lists are produced but these are all very verbose. Take for instance:

```
\setbox\scratchbox\hbox{test \bf test}

\showboxhere\scratchbox
```

This gives us:

```
\hlist[box][color=1,colormodel=1,mathintervals=1], width 47.8457pt, height 7.48193pt, depth
                                                                0.15576pt, direction l2r, state 1
.\list
..\glyph[unset][color=1,colormodel=1], protected, wd 4.42041pt, ht 7.48193pt, dp 0.15576pt, language
    (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <1: DejaVuSerif @ 11.0pt>, glyph U+0074
..\glyph[unset][color=1,colormodel=1], protected, wd 6.50977pt, ht 5.86523pt, dp 0.15576pt, language
    (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <1: DejaVuSerif @ 11.0pt>, glyph U+0065
..\glyph[unset][color=1,colormodel=1], protected, wd 5.64502pt, ht 5.86523pt, dp 0.15576pt, language
    (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <1: DejaVuSerif @ 11.0pt>, glyph U+0073
..\glyph[unset][color=1,colormodel=1], protected, wd 4.42041pt, ht 7.48193pt, dp 0.15576pt, language
    (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <1: DejaVuSerif @ 11.0pt>, glyph U+0074
..\glue[spaceskip][color=1,colormodel=1] 3.49658pt plus 1.74829pt minus 1.16553pt, font 1
..\glyph[unset][color=1,colormodel=1], protected, wd 5.08105pt, ht 7.48193pt, dp 0.15576pt, language
      (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <10: DejaVuSerif-Bold @ 11.0pt>, glyph
                                                                                          U+0074
..\glyph[unset][color=1,colormodel=1], protected, wd 6.99854pt, ht 5.86523pt, dp 0.15576pt, language
      (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <10: DejaVuSerif-Bold @ 11.0pt>, glyph
                                                                                          U+0065
```

```
..\glyph[unset][color=1,colormodel=1], protected, wd 6.19287pt, ht 5.86523pt, dp 0.15576pt, language
      (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <10: DejaVuSerif-Bold @ 11.0pt>, glyph
                                                                                              U+0073
..\glyph[unset][color=1,colormodel=1], protected, wd 5.08105pt, ht 7.48193pt, dp 0.15576pt, language
      (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <10: DejaVuSerif-Bold @ 11.0pt>, glyph
                                                                                              U+0074
```

The periods indicate the nesting level and the slash in front of the initial field is mostly a historic curiosity because there are no `\hlist` and `\glue` primitives, but actually there is in LuaMetaTEX a `\glyph` primitive but that one definitely doesn't want the shown arguments.

That said, here we have a horizontal list where the list field points to a glyph that itself points to a next one. The space became a glue node. In LuaTEX and even more in LuaMetaTEX all nodes have or get a subtype assigned that indicates what we're dealing with. This is shown between the first pair of brackets. Then there are attributes, between the second pair of brackets, which actually is a also a (sparse) linked list. Here we have two attributes set, the color, where the number points to some stored color specification, and the (here somewhat redundant) color space. The names of these attributes are macro package dependent because attributes are just a combination of a number and value. The engine itself doesn't do anything with them; it is the Lua code you plug in that can do something useful based on the values.

It will be clear that watching a complete page, with many nested boxes, rules, glyphs, discretionaries, glues, kerns, penalties, boundaries etc quickly becomes a challenge which is why we have other means to see what we get so let's move on to that now.

4 Visual debugging

In the early days of ConTEXt, in the mid 90's of the previous century, one of the first presentations at an ntg meeting was about visual debugging. This feature was achieved by overloading the primitives that make boxes, add glue, inject penalties and kerns, etc. It actually worked quite well, although in some cases, for instance where boxes have to be unboxed, one has to disable it. I remember some puzzlement among the audience about the fact that indeed these primitives could be overloaded without too many side effects. It will be no surprise that this feature has been carried on to later versions, and in ConTEXt MkIV it was implemented in a different (less intrusive) way and it got gradually extended.

```
\showmakeup \hbox{test \bf test}
```

This gives us a framed horizontal box, with some text and a space glue:

test test

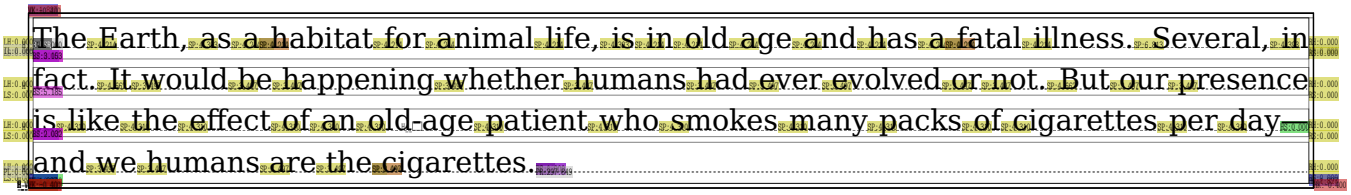
Of course not all information is well visible simply because it can be overlayed by what follows, but one gets the idea. Also, when you have a layer capable pdf viewer you can turn on and off categories, so you can decide to only show glue. You can also do that immediately, with `\showmakeup[glue]`.

There is a lot of granularity: `hbox`, `vbox`, `vtop`, `kern`, `glue`, `penalty`, `fontkern`, `strut`, `whatsit`, `glyph`, `simple`, `simplehbox`, `simplevbox`, `simplevtop`, `user`, `math`, `italic`, `origin`, `discretionary`, `expansion`, `line`, `space`, `depth`, `marginkern`, `mathkern`, `dir`, `par`, `mathglue`, `mark`, `insert`, `boundary`, the more selective `vkern`, `hkern`, `vglue`, `hglue`, `vpenalty` and `hpenalty`, as well as some presets like `boxes`, `makeup` and `all`.

When we have:

```
\showmakeup \framed[align=normal]{\samplefile{ward}}
```

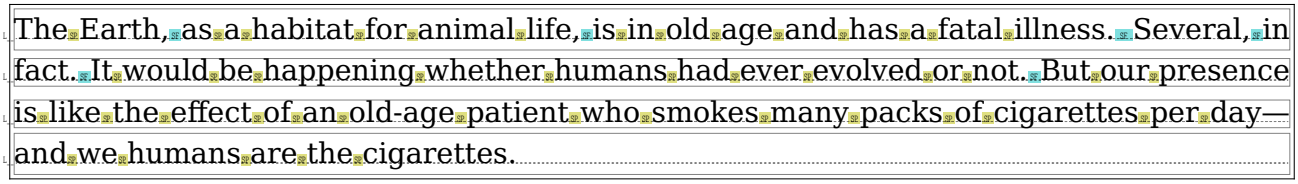
we get:



And that is why exploring this with a layers enabled pdf viewer can be of help. Alternatively a more selective use of `\showmakup` makes sense, like

```
\showmakeup[line,space] \framed[align=normal]{\samplefile{ward}}
```

Here we only see lines, regular spaces and spaces that are determined by the space factor that is driven by punctuation.

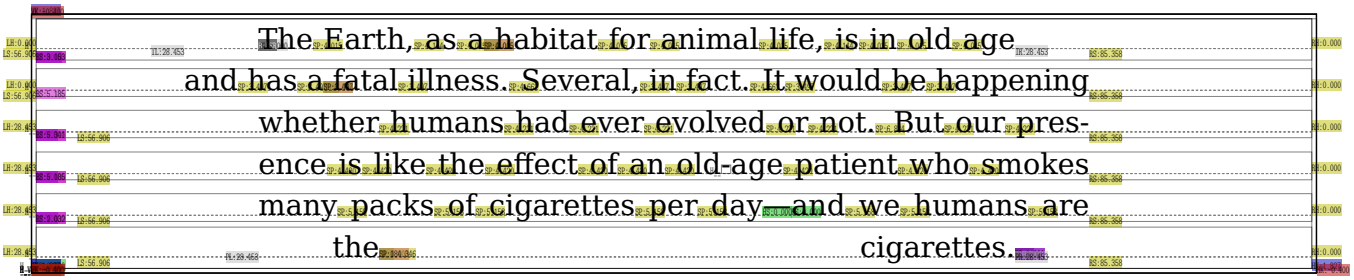


We can typeset the previous example with these settings:

```
\leftskip       2cm
\rightskip      3cm
\hangindent     1cm
```

```
\hangafter        2
\parfillrightskip 1cm
\parfillleftskip  1cm % new
\parinitrightskip 1cm % new
\parinitleftskip  1cm % new
\parindent        2cm % different
```

This time we get:



Looking at this kind of output only makes sense on screen where you can zoom in but what we want to demonstrate here is that in LuaMetaTEX we have not only a bit more control over the paragraph (indicated by comments) but also that we always have the related glue present. The reason is that we then have a more predictable packaged line when we look at one from the Lua end. Where TEX normally moves the final line content left or right via either glue or the shifts property of a box, here we always use the glue. We call this normalization. Keep in mind that TEX was not designed (implemented) with exposing its internals in mind, but for LuaTEX and LuaMetaTEX we have to take care of that.

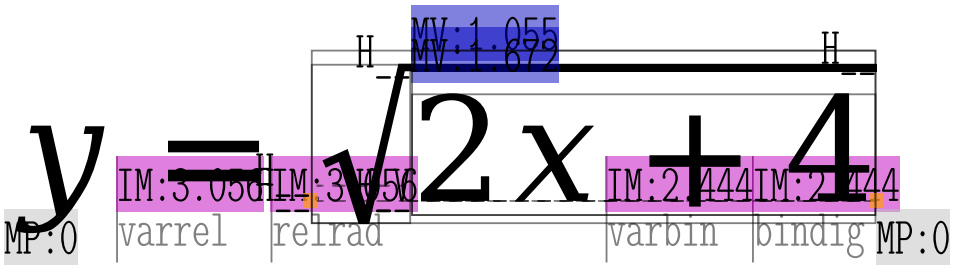
Another characteristic is that the paragraph stores these (and many more) properties in the so called initial par node so that they work well in situations where grouping would interfere with our objectives. As with all extensions, these are things that can be configured in detail but they are enabled in ConTEXt by default.

5 Math

Math is a good example where this kind of tracing helps development. Here is an example:

```
\im { \showmakeup y = \sqrt {2x + 4} }
```

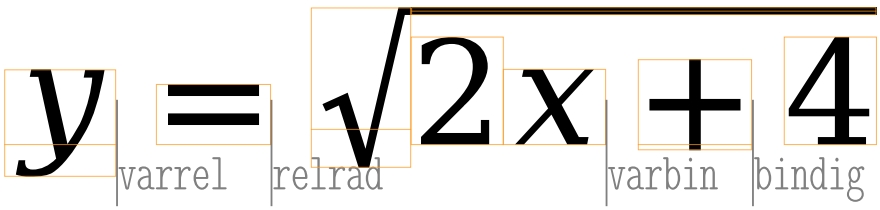
Scaled up we get:



Instead of showing everything we can again be more selective:

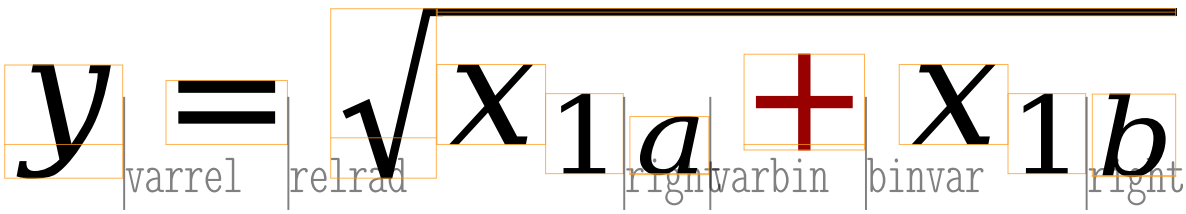
```
\im {
    \showmakeup[mathglue,glyph]
    y = \sqrt {2x + 4}
}
```

Here we not only limit ourselves to math glue, but also enable showing the bounding boxes of glyphs.



This example also shows that in LuaMetaTEX we have more classes than in a traditional TEX engine. For instance, radicals have their own class as do digits. The radical class is an engine one, the digit class is a user defined class. You can set up the spacing between each class depending on the style. For the record: this is just one of the many extensions to the math engine and when extensions are being developed it helps to have this kind of tracing. Take for instance the next example, where we have multiple indexes (indicated by __) on a nucleus, that get separated by a little so called continuation spacing.

```
\im {
    \showmakeup[mathglue,glyph]
    y = \sqrt {x__1__a {\darkred +} x__1__b}
}
```



Here the variable class is used for alphabetic characters and some more, contrary to the more traditional (often engine assigned) ordinary class that is now used for the left-overs.

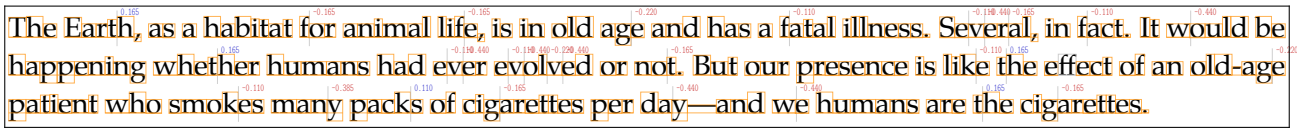
6 Fonts

Some of the mentioned tracing has shortcuts, for instance `\showglyphs`. Here we show the same sample paragraph as before:

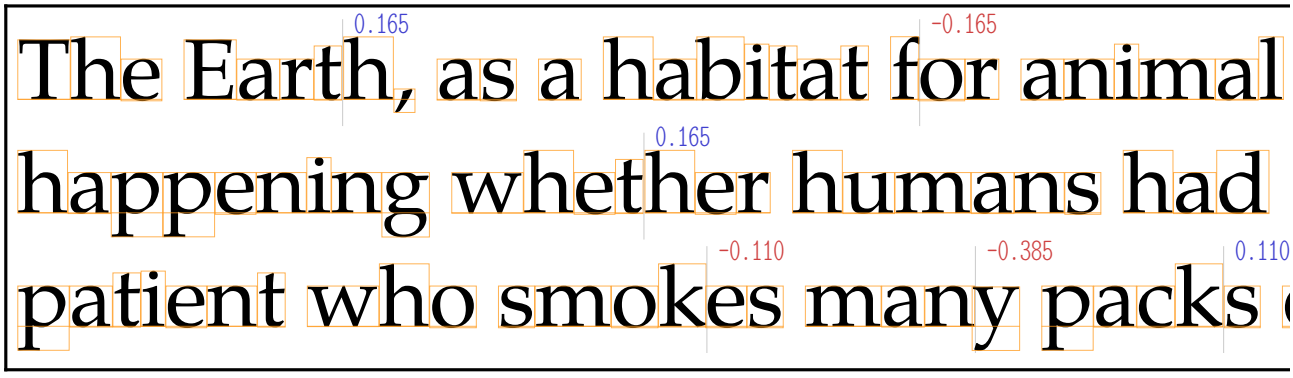
```
\showglyphs
\showfontkerns
\framed[align=normal]{\samplefile{ward}}
```

Here is the upper left corner of the result:

What font kerns we get depends on the font, here we use pagella:

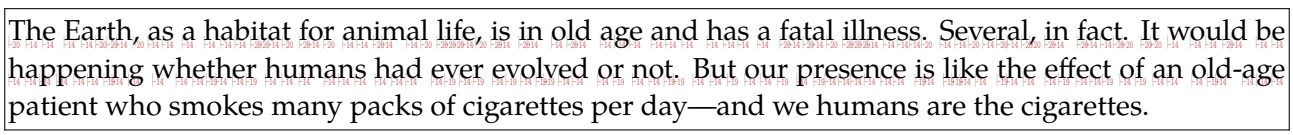


If we zoom in the kerns are more visible:

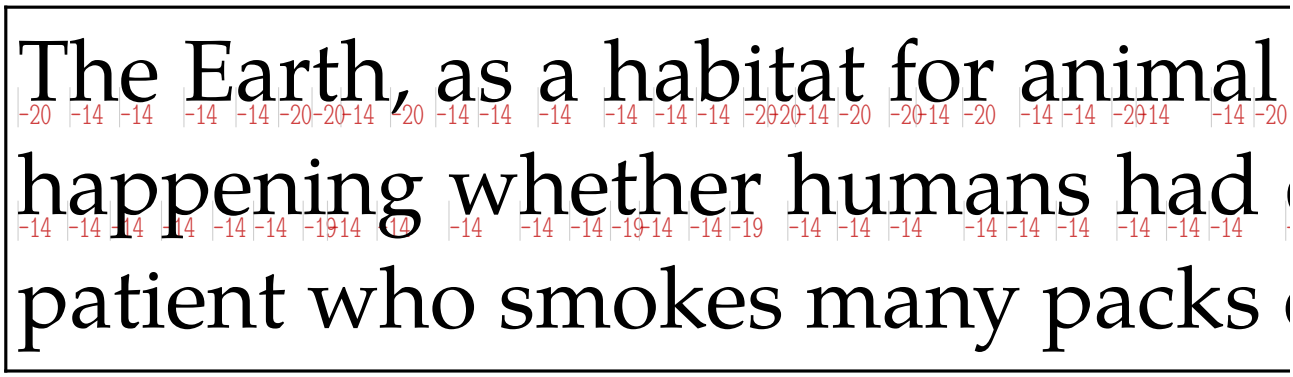


And here is another one:

```
\showfontexpansion
\framed[align={normal,hz}]{\samplefile{ward}}
```



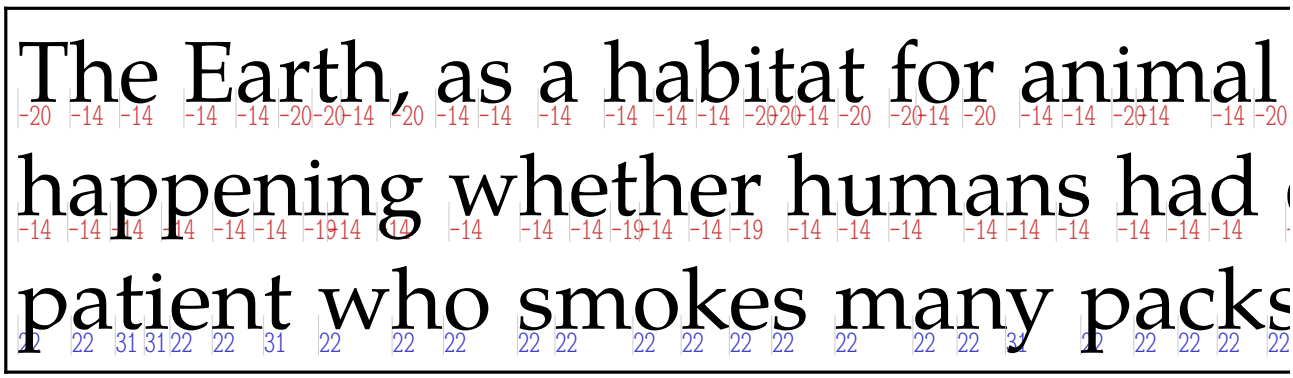
or blown up:



The last line (normally) doesn't need expansion, unless we want it to compatible with preceding lines, space-wise. So when we do this:

```
\showfontexpansion
\framed[align={normal,hz,fit}]{\samplefile{ward}}
```

the `fit` directives results in somewhat different results:



As with other visual tracers you can get some insight in how TEX turns your input into a typeset result.

7 Overflow

By default the engine is a bit stressed to make paragraphs fit well. This means that we can get overflowing lines. Because there is a threshold only visible overflow is reported. If you want a visual clue, you can do this:

```
\enabletrackers[builders.hpack.overflow]
```

With:

```
\ruledvbox{\hsize 3cm test test test test test test test test}
```

We get:

test test test test
test test test test

The red bar indicates a potential problem. We can also get an underflow, as demonstrated here:

```
\ruledvbox {
    \setupalign[verytolerant,stretch]
    \hsize 3cm test test test test test test test test
}
```

Now we get a blue bar that indicates that we have a bit more stretch than is considered optimal:

test test test
test test test
test test

Especially in automated flows it makes sense to increase the tolerance and permit stretch. Only when the strict attempt fails that will kick in.

8 Side floats

Some mechanisms are way more complex than a user might expect from the result. An example is the placement of float and especially side floats.

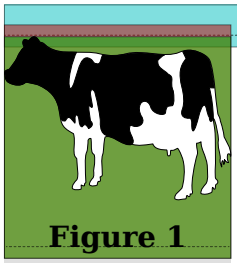


Figure 1

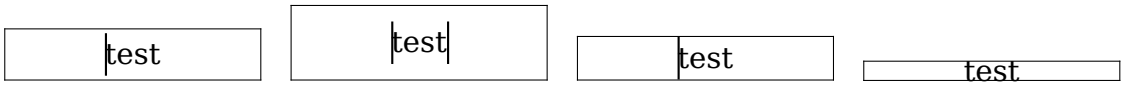
Not only do we have to make sure that the spacing before such a float is as good and consistent as possible, we also need the progression to work out well, that is: the number of lines that we need to indent.

For that we need to estimate the space needed, look at the amount of space before and after the float, check if it will fit and move to the next page if needed. That all involves dealing with interline spacing, interparagraph spacing, spacing at the top of a page, permitted slack at the bottom of page, the depth of the preceding lines, and so on. The tracer shows some of the corrections involved but leave it to the user to imagine what it relates to; the previous sentence gives some clues. This tracker is enables with:

```
\enabletrackers[floats.anchoring]
```

9 Struts

We now come to one of the most important trackers, `\showstruts`, and a few examples shows why:



`width=.2tw` `height=1cm` `offset=0pt` `offset=overlay`

Here in all cases we've set the width to 20 percent of the text width (`tw` is an example of a plugged in dimension). In many places ConTEXt adds struts in order to enforce proper spacing so when spacing is not what you expect, enabling this tracker can help you figure out why.

10 Features

Compared to the time when TEX showed up the current fonts are more complicated, especially because features go beyond only ligaturing and kerning. But even ligaturing can be different, because some fonts use kerning and replacement instead of a new character. Pagella uses a multiple to single replacement:

font 17: texgyrepagella-regular.otf @ 10.0pt

features **[basic:** kern=yes, liga=yes, mark=yes, mkmk=yes, script=dflt**]**
[extra: analyze=yes, autolanguage=position, autoscript=position, checkautoeffect=yes, checkmarks=yes, checkmissing=yes, compoundhyphen=yes, curs=yes, devanagari=yes, dummies=yes, expansion=quality, extensions=yes, extrafeatures=yes, extraprivates=yes, fixdot=yes, indic=auto, itlc=yes, mathrules=yes, mode=node, spacekern=yes, textcontrol=collapsehyphens,replaceapostrophe, visualspace=yes, wipemath=yes**]**

step 1 effe fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f U+65:e [glue] U+66:f U+69:i U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

```
feature 'liga', type 'gsub_ligature', lookup 's_s_9', replacing
  U+66 (f) upto U+66 (f) by ligature U+FB00 (f_f) case 2
```

step 2 effe fietsen U+65:e [pre: U+66:f U+2D:- post: U+66:f replace: U+FB00:ff] U+65:e [glue] U+66:f U+69:i U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

```
feature 'liga', type 'gsub_ligature', lookup 's_s_10',
  replacing U+66 (f) upto U+69 (i) by ligature U+FB01 (f_i)
  case 2
```

step 3 effe fietsen U+65:e [pre: U+66:f U+2D:- post: U+66:f replace: U+FB00:ff] U+65:e [glue] U+FB01:fi U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

```
feature 'kern', type 'gpos_pair', lookup 'p_s_0', inserting
  move -0.14992pt between U+66 (f) and U+65 (e) as
  postinjections
```

result effe fietsen U+65:e [pre: U+66:f U+2D:- post: U+66:f [kern] replace: U+FB00:ff] U+65:e [glue] U+FB01:fi U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

Not all features listed here are provided by the font (only the four character ones) because we're using TEX which, it being TEX, means that we have plenty more ways to mess around with additional features: it's all about detailed control. But what you see here are the steps taken: the font handler loops over the list of glyphs and here we see the intermediate results when something has changed. There can be way more loops that in this simple case.

With Cambria we get a single replacement combined with kerning:

font 19: cambria.ttc @ 10.0pt

features **[basic:** kern=yes, liga=yes, mark=yes, mkmk=yes, script=latn**]** **[extra:** analyze=yes, autolanguage=position, autoscript=position, checkautoeffect=yes, checkmarks=yes, checkmissing=yes, compoundhyphen=yes, curs=yes, devanagari=yes, dummies=yes, expansion=quality, extensions=yes, extrafeatures=yes, extraprivates=yes, fixdot=yes, indic=auto, itlc=yes, mathrules=yes, mode=node, spacekern=yes, textcontrol=collapsehyphens,replaceapostrophe, visualspace=yes, wipemath=yes**]**

step 1 effe fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f U+65:e [glue] U+66:f U+69:i U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

```
feature 'liga', type 'gsub_contextchain', chain lookup
  's_s_38', replacing single U+66 by U+F016C
```

step 2 effe fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f U+65:e [glue] U+F016C:f U+69:i U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

```
feature 'kern', type 'gpos_pair', merged lookup 'p_s_0',
  inserting move -0.12207pt between U+66 and U+65 as injections
```

result effe fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f [kern] U+65:e [glue]
U+F016C:f U+69:i U+65:e U+74:t [pre: U+2D:-] U+73:s U+65:e U+6E:n

One complication is that hyphenation kicks in which means that whatever we do has to take the pre, post and replacement bits into account combined which what comes before and after. Especially for complex scripts this tracker can be illustrative but even then only for those who like to see what fonts do and/or when they add additional features runtime.

11 Profiling

There are some features in ConTEXt that are nice but only useful in some situations. An example is profiling. It is something you will not turn on by default, if only because of the overhead it brings. The next two paragraphs (using Pagella) show the effect.

The command \binom is the standard notation for binomial coefficients and is preferred over \choose, which is an older macro that has limited compatibility with newer packages and font encodings: $|A| = \binom{N}{k}^2$. Additionally, \binom uses proper spacing and size for the binomial symbol. In conclusion, it is recommended to use \binom instead of \choose in TEX for typesetting binomial coefficients for better compatibility and uniform appearance.

The previous paragraph is what comes out by default, while the next one used these settings plus an additional `\enabletrackers[profiling.lines.show]`.

The command \binom is the standard notation for binomial coefficients and is preferred over \choose, which is an older macro that has limited compatibility with newer packages and font encodings: $|A| = \binom{N}{k}^2$. Additionally, \binom uses proper spacing and size for the binomial symbol. In conclusion, it is recommended to use \binom instead of \choose in TEX for typesetting binomial coefficients for better compatibility and uniform appearance.

This feature will bring lines together when there is no clash and is mostly of use when a lot of inline math is used. However, when this variant of profiling (we have an older one too) is enabled on a 300 page math book with thousands of formulas, only in a few places it demonstrated effect; it was hardly needed anyway. So, sometimes tracing shows what makes sense or not.

12 Par builder

Here is is a sample paragraph from Knuths “Digital Typography”:

15. (This procedure maintains four integers (A, B, C, D) with the invariant meaning that "our remaining job is to output the continued fraction for $(Ay + B)/(Cy + D)$, where y is the input yet to come.") Initially set $j \leftarrow k \leftarrow 0$, $(A, B, C, D) \leftarrow (a, b, c, d)$; then input x_j and set $(A, B, C, D) \leftarrow (Ax_j + B, A, Cx_j + D, C)$, $j \leftarrow j + 1$, one or more times until $C + D$ has the same sign as C. (When $j > 1$ and the input has not terminated, we know that $1 \le y < \infty$; and when $C + D$ has the same sign as C we know therefore that $(Ay + B)/(Cy + D)$ lies between $(A + B)/(C + D)$ and A/C.) Now comes the general step: If no integer lies strictly between $(A + B)/(C + D)$ and A/C, output $X_k \leftarrow \lfloor A/C \rfloor$ and set $(A, B, C, D) \leftarrow (C, D, A - X_k C, B - X_k D)$, $k \leftarrow k + 1$; otherwise input x_j and set $(A, B, C, D) \leftarrow (Ax_j + B, A, Cx_j + D, C)$, $j \leftarrow j + 1$. The general step is repeated ad infinitum. However, if at any time the *final* x_j is input, the algorithm immediately switches gears: It outputs the continued fraction for $(Ax_j + B)/(Cx_j + D)$, using Euclid's algorithm, and terminates.

There are indicators with tiny numbers that indicate the possible breakpoints and we can see what the verdict is:

1	*2*	1	0	10	400	decent	glue
	3	2	0	70	6400	tight	glue
2	*2*	3	1	69	6641	loose	glue
		4	2	5	29125	decent	glue
		5	1	5	28084	tight	glue
	3	6	2	0	6500	decent	math
	1	7	2	41	9001	tight	glue
3	*2*	8	3	2	6785	decent	glue
	1	9	6	67	12429	loose	glue
		10	3	67	18490	tight	glue
		11	6	4	29100	decent	glue
	1	12	7	0	9101	decent	math
	2	13	6	0	7589	tight	math
4	*1*	14	8	1	6906	decent	glue
	1	15	9	90	12529	decent	glue
	1	16	8	90	12410	tight	glue
	2	17	12	22	10125	loose	glue
		18	13	0	497689	decent	penalty
	2	19	13	10	7989	decent	math
5		20	15	22	13553	loose	glue
	2	21	14	22	7747	tight	glue
		22	17	64	15601	loose	glue
	1	23	16	64	12510	decent	glue
	1	24	17	20	10225	decent	glue
	1	25	19	1	8110	decent	glue
	1	26	19	28	259433	tight	penalty
6	*2*	27	21	73	7943	decent	glue
		28	23	0	12610	decent	glue
	2	29	21	0	8647	tight	glue
		30	25	39	1113011	loose	penalty
	2	31	24	39	1112825	decent	penalty
	1	32	26	3	1362102	decent	penalty
7	*1*	33	27	95	18968	loose	glue
	1	34	29	3	8816	decent	glue
		35	27	3	10647	tight	glue
	2	36	29	15	18251	tight	glue
		37	31	0	1112925	decent	glue
		38	31	32	1114589	tight	glue
	1	39	32	0	1362202	decent	glue
8	*2*	40	33	1	269089	decent	penalty
	2	41	34	0	8916	decent	glue
		42	36	41	18395	decent	glue
		43	36	25	18980	tight	glue
	1	44	39	92	1622606	tight	penalty
9	*1*	45	40	8	291913	decent	glue
		46	40	0	269189	decent	math
	2	47	41	10	9316	decent	glue
	1	48	41	20	9016	decent	glue
	4	49	44	0	2725206	decent	penalty
10		50	45	41	294514	loose	glue
	1	51	47	99	9640	decent	glue
		52	48	13	9545	loose	glue
		53	47	13	9416	decent	glue
		54	49	59	2729967	loose	glue
	2	55	49	12	2725690	decent	glue
	2	56	49	0	2725306	decent	glue
		57	49	66	2730982	tight	glue
11	*1*	58	51	8	9964	decent	glue
		59	56	55	2729531	loose	glue
		60	55	55	2725811	decent	glue
		61	56	71	2725406	decent	glue
		62	55	71	2733971	tight	glue
12		63	58	61	15005	tight	glue

59 56 49 44 39 32 26 19 13 6 2
60 55 49 44 39 32 26 19 13 6 2
61 56 49 44 39 32 26 19 13 6 2
62 55 49 44 39 32 26 19 13 6 2
63 58 51 47 41 34 29 21 14 8 3 1

pass : 1 demerits : 15105
subpass : P looseness : 0
subpasses : 0

The last lines in the last column show the route that the result takes. Without going into details, here is what we did:

```
\startshowbreakpoints
    \samplefile{math-knuth-dt}
\stopshowbreakpoints

\showbreakpoints
```

This kind of tracing is part of a mechanism that makes it possible to influence the choice by choosing a specific preferred breakpoint but that is something the average user is unlikely to do. The main reason why we have this kind of trackers is that when developing the new multi-step par builder feature we wanted to see what exactly it did influence. That mechanism uses an LuaMetaTEX feature where we can plug in additional passes using the `\parpasses` primitive that can add different strategies that are tried until criteria for over- and underfull thresholds and/or badness are met. Each step can set the relevant parameters differently, including expansion, which actually makes for more efficient output and better runtime when that features is not needed to get better results.

13 More

There are many more visual trackers, for instance `layout.vz` for when you enabled vertical expansion, `typesetters.suspects` for identifying possible issues in the input like invisible spaces. Trackers like `nodes.destinations` and `nodes.references` will show the areas used by these mechanisms. There are also trackers for positions, (cjk and other), script handling, rubies, tagging, italic correction, breakpoints and so on. The examples in the previous sections illustrate what to expect and when to use a specific mechanism knowing this might trigger you to check if a tracker exists. Often the test suite has examples of usage.

13 Colofon

Author	Hans Hagen
ConTEXt	2025.02.12 08:44
LuaMetaTEX	2.11.06 | 20250210
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl