low level TeX

boxes

Contents

1 Preamble

1.1 Introduction

An average ConTEXt user will not use the low level box primitives but a basic understanding of how TEX works doesn't hurt. In fact, occasionally using a box command might bring a solution not easily achieved otherwise, simply because a more high level interface can also be in the way.

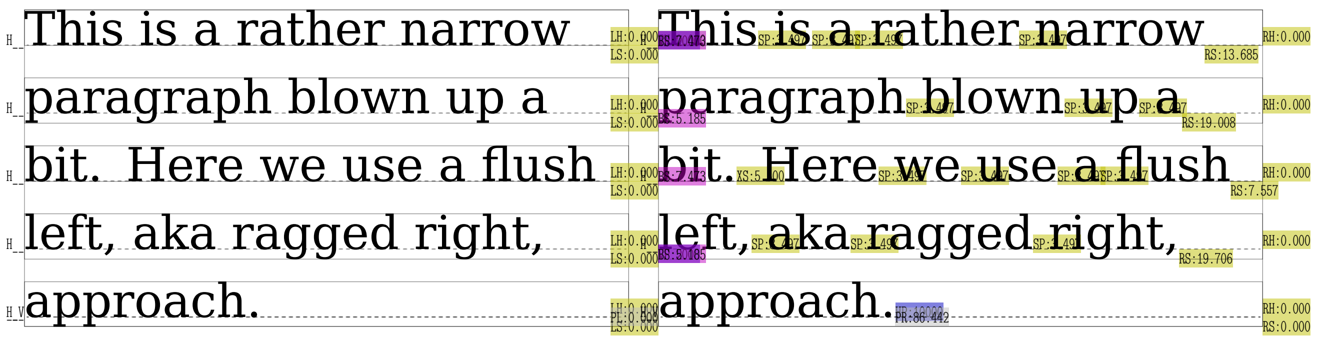
The best reference is of course The TEXbook so if you're really interested in the details you should get a copy of that book. Below I will not go into details about all kind of glues, kerns and penalties, just boxes it is.

This explanation will be extended when I feel the need (or users have questions that can be answered here).

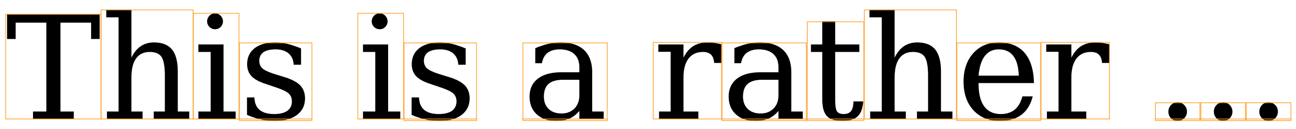
1.2 Boxes

This paragraph of text is made from lines that contain words that themselves contain symbolic representations of characters. Each line is wrapped in a so called horizontal box and eventually those lines themselves get wrapped in what we call a vertical box.

When we expose some details of a paragraph it looks like this:



The left only shows the boxes, the variant at the right shows (font) kerns and glue too. Because we flush left, there is rather strong right skip glue at the right boundary of the box. If font kerns show up depends on the font, not all fonts have them (or have only a few). The glyphs themselves are also kind of boxed, as their dimensions determine the area that they occupy:



But, internally they are not really boxed, as they already are a single quantity. The same is true for rules: they are just blobs with dimensions. A box on the other hand wraps a linked list of so called nodes: glyphs, kerns, glue, penalties, rules, boxes, etc. It is a container with properties like width, height, depth and shift.

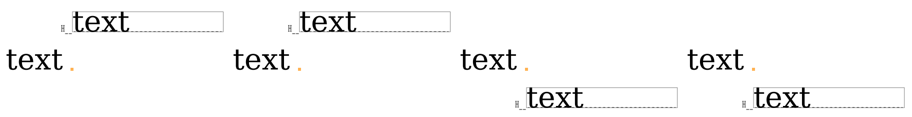
2 TEX primitives

The box model is reflected in TEX's user interface but not by that many commands, most noticeably \hbox, \vbox and \vtop. Here is an example of the first one:

```
\hbox width 10cm{text}
\hbox width 10cm height 1cm depth 5mm{text}
text \raise5mm\hbox{text} text
```

The \raise and \lower commands behave the same but in opposite directions. One could as well have been defined in terms of the other.

```
text \raise  5mm \hbox to 2cm {text}
text \lower -5mm \hbox to 2cm {text}
text \raise -5mm \hbox to 2cm {text}
text \lower  5mm \hbox to 2cm {text}
```



A box can be moved to the left or right but, believe it or not, in ConTEXt we never use that feature, probably because the consequences for the width are such that we can as well use kerns. Here are some examples:

```
text \vbox{\moveleft  5mm \hbox {left}}text !
text \vbox{\moveright 5mm \hbox{right}}text !
```

text lefttext ! text righttext !

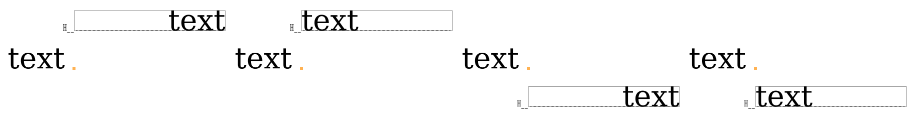
```
text \vbox{\moveleft  25mm \hbox {left}}text !
text \vbox{\moveright 25mm \hbox{right}}text !
```

left text text ! text righttext !

Code like this will produce a complaint about an underfull box but we can easily get around that:

```
text \raise  5mm \hbox to 2cm {\hss text}
text \lower -5mm \hbox to 2cm {text\hss}
text \raise -5mm \hbox to 2cm {\hss text}
text \lower  5mm \hbox to 2cm {text\hss}
```

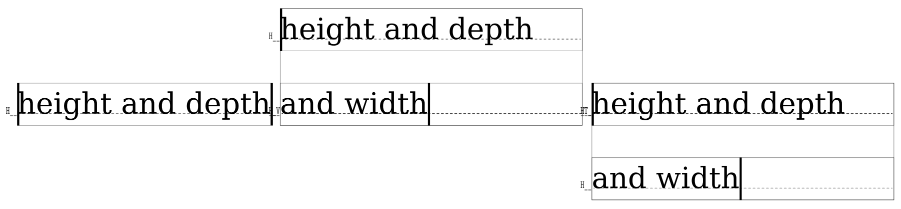
The `\hss` primitive injects a glue that when needed will fill up the available space. So, here we force the text to the right or left.



We have three kind of boxes: `\hbox`, `\vbox` and `\vtop`:

```
\hbox{\strut height and depth\strut}
\vbox{\hsize 4cm \strut height and depth\par and width\strut}
\vtop{\hsize 4cm \strut height and depth\par and width\strut}
```

A `\vbox` aligns at the bottom and a `\vtop` at the top. I have added some so called struts to enforce a consistent height and depth. A strut is an invisible quantity (consider it a black box) that enforces consistent line dimensions: height and depth.



You can store a box in a register but you need to be careful not to use a predefined one. If you need a lot of boxes you can reserve some for your own:

```
\newbox\MySpecialBox
```

but normally you can do with one of the scratch registers, like 0, 2, 4, 6 or 8, for local boxes, and 1, 3, 5, 7 and 9 for global ones. Registers are used like:

```
       \setbox0\hbox{here}
\global\setbox1\hbox{there}
```

In ConTEXt you can also use

```
\setbox\scratchbox   \hbox{here}
\setbox\scratchboxone\hbox{here}
\setbox\scratchboxtwo\hbox{here}
```

and some more. In fact, there are quite some predefined scratch registers (boxes, dimensions, counters, etc). Feel free to investigate further.

When a box is stored, you can consult its dimensions with `\wd`, `\ht` and `\dp`. You can of course store them for later use.

```
\scratchwidth \wd\scratchbox
\scratchheight\ht\scratchbox
\scratchdepth \dp\scratchbox
\scratchtotal \dimexpr\ht\scratchbox+\dp\scratchbox\relax
\scratchtotal \htdp\scratchbox
```

The last line is ConTEXt specific. You can also set the dimensions

```
\wd\scratchbox 10cm
\ht\scratchbox 10mm
\dp\scratchbox  5mm
```

So you can cheat! A box is placed with `\copy`, which keeps the original intact or `\box` which just inserts the box and then wipes the register. In practice you seldom need a copy, which is more expensive in runtime anyway. Here we use copy because it serves the examples.

```
\copy\scratchbox
\box \scratchbox
```

3 ε-TEX primitives

The ε-TEX extensions don't add something relevant for boxes, apart from that you can use the expressions mechanism to mess around with their dimensions. There is a mechanism for typesetting r2l within a paragraph but that has limited capabilities and doesn't change much as it’s mostly a way to trick the backend into outputting a stretch of text in the other direction. This feature is not available in LuaTEX because it has an alternative direction mechanism.

4 LuaTEX primitives

The concept of boxes is the same in LuaTEX as in its predecessors but there are some aspects to keep in mind. When a box is typeset this happens in LuaTEX:

1. A list of nodes is constructed. In LuaTEX this is a double linked list (so that it can easily be manipulated in Lua) but TEX itself only uses the forward links.
2. That list is hyphenated, that is: so called discretionary nodes are injected. This depends on the language properties of the glyph (character) nodes.
3. Then ligatures are constructed, if the font has such combinations. When this built-in mechanism is used, in ConTEXt we speak of base mode.
4. After that inter-character kerns are applied, if the font provides them. Again this is a base mode action.
5. Finally the box gets packaged:
   - In the case of a horizontal box, the list is packaged in a hlist node, basically one liner, and its dimensions are calculated and set.
   - In the case of a vertical box, the paragraph is broken into one or more lines, without hyphenation, with optimal hyphenation or in the worst case with so called emergency stretch applied, and the result becomes a vlist node with its dimensions set.

In traditional TEX the first four steps are interwoven but in LuaTEX we need them split because the step 5 can be overloaded by a callback. In that case steps 3 and 4 (and maybe 2) are probably also overloaded, especially when you bring handling of fonts under Lua control.

New in LuaTEX are three packers: `\hpack`, `\vpack` and `\tpack`, which are companions to `\hbox`, `\vbox` and `\vtop` but without the callbacks applied. Using them is a bit tricky as you never know if a callback should be applied, which, because users can often add their own Lua code, is not something predictable.

Another box related extension is direction. There are four possible directions but because in LuaMetaTEX there are only two. Because this model has been upgraded, it will be discusses in the next section. A ConTEXt user is supposed to use the official ConTEXt interfaces in order to be downward compatible.

5 LuaMetaTEX primitives

There are two possible directions: left to right (the default) and right to left for Hebrew and Arabic. Here is an example that shows how it'd done with low level directives:

```
\hbox direction 0 {from left to right}
\hbox direction 1 {from right to left}
```

from left to right
tfel ot thgir morf

A low level direction switch is done with:

```
\hbox direction 0
    {from left to right \textdirection 1 from right to left}
\hbox direction 1
    {from right to left \textdirection 1 from left to right}
```

from left to right tfel ot thgir morf
thgir ot tfel morf tfel ot thgir morf

but actually this is kind of *not done* in ConTEXt, because there you are supposed to use the proper direction switches:

```
\naturalhbox {from left to right}
\reversehbox {from right to left}
\naturalhbox {from left to right \righttoleft from right to left}
\reversehbox {from right to left \lefttoright from left to right}
```

from left to right
tfel ot thgir morf
from left to right tfel ot thgir morf
from left to right tfel ot thgir morf

Often more is needed to properly support right to left typesetting so using the ConTEXt commands is more robust.

In LuaMetaTEX the box model has been extended a bit, this as a consequence of dropping the vertical directional typesetting, which never worked well. In previous sections we discussed the properties width, height and depth and the shift resulting from a `\raise`, `\lower`, `\moveleft` and `\moveright`. Actually, the shift is also used in for instance positioning math elements.

The way shifting influences dimensions can be somewhat puzzling. Internally, when TEX packages content in a box there are two cases:

- When a horizontal box is made, and `height - shift` is larger than the maximum height so far, that delta is taken. When `depth + shift` is larger than the current depth, then that depth is adapted. So, a shift up influences the height and a shift down influences the depth.
- In the case of vertical packaging, when `width + shift` is larger than the maximum box (line) width so far, that maximum gets bumped. So, a shift to the right can contribute, but a shift to the left cannot result in a negative width. This is also why vertical typesetting, where height and depth are swapped with width, goes wrong: we somehow need to map two properties onto one and conceptually TEX is really set up for horizontal typesetting. (And it's why I decided to just remove it from the engine.)

This is one of these cases where TEX behaves as expected but it also means that there is some limitation to what can be manipulated. Setting the shift using one of the four commands has a direct consequence when a box gets packaged which happens immediately because the box is an argument to the foursome.

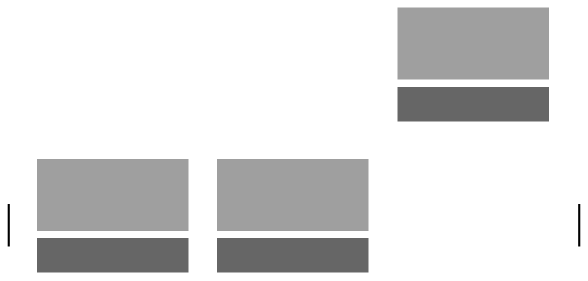
There is in traditional TEX, probably for good reason, no way to set the shift of a box, if only because the effect would normally be none. But in LuaTEX we can cheat, and therefore, for educational purposed ConTEXt has implements some cheats.

We use this sample box:

```
\setbox\scratchbox\hbox\bgroup
    \middlegray\vrule width 20mm depth  -.5mm height 10mm
    \hskip-20mm
    \darkgray  \vrule width 20mm height -.5mm depth   5mm
\egroup
```

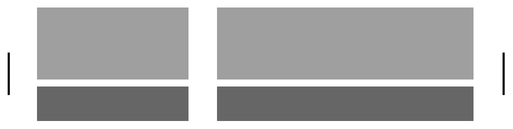
When we mess with the shift using the ConTEXt `\shiftbox` helper, we see no immediate effect. We only get the shift applied when we use another helper, `\hpackbox`.

```
\hbox\bgroup
    \showstruts \strut
    \quad                                   \copy\scratchbox
    \quad \shiftbox\scratchbox -20mm \copy\scratchbox
    \quad \hpackbox\scratchbox       \box \scratchbox
    \quad \strut
\egroup
```



When instead we use `\vpackbox` we get a different result. This time we move left.

```
\hbox\bgroup
    \showstruts \strut
    \quad                            \copy\scratchbox
    \quad \shiftbox\scratchbox -10mm \copy\scratchbox
    \quad \vpackbox\scratchbox       \copy\scratchbox
    \quad \strut
\egroup
```



The shift is set via Lua and the repackaging is also done in Lua, using the low level `hpack` and `vpack` helpers and these just happen to look at the shift when doing their job. At the TEX end this never happens.

This long exploration of shifting serves a purpose: it demonstrates that there is not that much direct control over boxes apart from their three dimensions. However this was never a real problem as one can just wrap a box in another one and use kerns to move the embedded box around. But nevertheless I decided to see if the engine can be a bit more helpful, if only because all that extra wrapping gives some overhead and complications when we want to manipulate boxes. And of course it is also a nice playground.

We start with changing the direction. Changing this property doesn't require repackaging because directions are not really dealt with in the frontend. When a box is converted to (for instance pdf) the reversion happens.

```
\setbox\scratchbox\hbox{whatever}
\the\boxdirection\scratchbox: \copy\scratchbox \crlf
\boxdirection\scratchbox 1
\the\boxdirection\scratchbox: \copy\scratchbox
```

0: whatever
1: revetahw

Another property that can be queried and set is an attribute. In order to get a private attribute we define one.

```
\newattribute\MyAt
\setbox\scratchbox\hbox attr \MyAt 123 {whatever}
[\the\boxattribute\scratchbox\MyAt]
\boxattribute\scratchbox\MyAt 456
[\the\boxattribute\scratchbox\MyAt]
[\ifnum\boxattribute\scratchbox\MyAt>400 okay\fi]
```

[123] [456] [okay]

The sum of the height and depth is available too. Because for practical reasons setting that property is also needed then, the choice was made to distribute the value equally over height and depth.

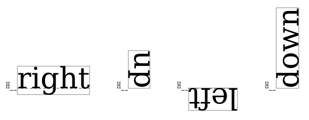
```
\setbox\scratchbox\hbox {height and depth}
[\the\ht\scratchbox]
[\the\dp\scratchbox]
[\the\boxtotal\scratchbox]
\boxtotal\scratchbox=20pt
[\the\ht\scratchbox]
[\the\dp\scratchbox]
[\the\boxtotal\scratchbox]
```

[8.35742pt] [2.44385pt] [10.80127pt] [10.0pt] [10.0pt] [20.0pt]

We've now arrived to a set of properties that relate to each other. They are a bit complex and given the number of possibilities one might need to revert to some trial and error: orientations and offsets. As with the dimensions, directions and attributes, they are passed as box specification. We start with the orientation.

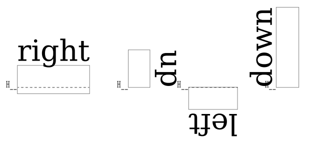
```
\hbox \bgroup \showboxes
          \hbox orientation 0 {right}
```

```
    \quad \hbox orientation 1 {up}
    \quad \hbox orientation 2 {left}
    \quad \hbox orientation 3 {down}
\egroup
```



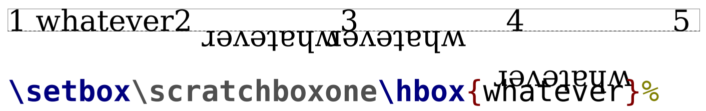
When the orientation is set, you can also set an offset. Where shifting around a box can have consequences for the dimensions, an offset is virtual. It gets effective in the backend, when the contents is converted to some output format.

```
\hbox \bgroup \showboxes
          \hbox orientation 0 yoffset  10pt {right}
    \quad \hbox orientation 1 xoffset  10pt {up}
    \quad \hbox orientation 2 yoffset -10pt {left}
    \quad \hbox orientation 3 xoffset -10pt {down}
\egroup
```



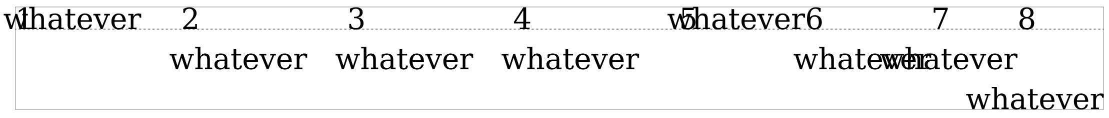
The reason that offsets are related to orientation is that we need to know in what direction the offsets have to be applied and this binding forces the user to think about it. You can also set the offsets using commands.

```
\setbox\scratchbox\hbox{whatever}%
1                                  \copy\scratchbox
2 \boxorientation\scratchbox 2     \copy\scratchbox
3 \boxxoffset    \scratchbox -15pt \copy\scratchbox
4 \boxyoffset    \scratchbox -15pt \copy\scratchbox
5
```



```
\setbox\scratchboxone\hbox{whatever}%
\setbox\scratchboxtwo\hbox{whatever}%
1 \boxxoffset \scratchboxone -15pt \copy\scratchboxone
2 \boxyoffset \scratchboxone -15pt \copy\scratchboxone
3 \boxxoffset \scratchboxone -15pt \copy\scratchboxone
4 \boxyoffset \scratchboxone -15pt \copy\scratchboxone
```

```
5 \boxxmove   \scratchboxtwo -15pt \copy\scratchboxtwo
6 \boxymove   \scratchboxtwo -15pt \copy\scratchboxtwo
7 \boxxmove   \scratchboxtwo -15pt \copy\scratchboxtwo
8 \boxymove   \scratchboxtwo -15pt \copy\scratchboxtwo
```



The move commands are provides as convenience and contrary to the offsets they do adapt the dimensions. Internally, with the box, we register the orientation and the offsets and when you apply these commands multiple times the current values get overwritten. But … because an orientation can be more complex you might not get the effects you expect when the options we discuss next are used. The reason is that we store the original dimensions too and these come into play when these other options are used: anchoring. So, normally you will apply an orientation and offsets once only.

The orientation specifier is actually a three byte number that best can be seen hexadecimal (although we stay within the decimal domain). There are three components: x-anchoring, y-anchoring and orientation:

```
0x<X><Y><O>
```

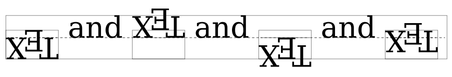
or in TEX speak:

```
"<X><Y><O>
```

The landscape and seascape variants both sit on top of the baseline while the flipped variant has its depth swapped with the height. Although this would be enough a bit more control is possible.

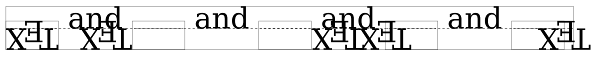
The vertical options of the horizontal variants anchor on the baseline, lower corner, upper corner or center.

```
\ruledhbox orientation "002 {\TEX} and
\ruledhbox orientation "012 {\TEX} and
\ruledhbox orientation "022 {\TEX} and
\ruledhbox orientation "032 {\TEX}
```



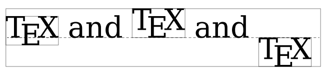
The horizontal options of the horizontal variants anchor in the center, left, right, halfway left and halfway right.

```
\ruledhbox orientation "002 {\TEX} and
\ruledhbox orientation "102 {\TEX} and
\ruledhbox orientation "202 {\TEX} and
\ruledhbox orientation "302 {\TEX} and
\ruledhbox orientation "402 {\TEX}
```



The orientation has consequences for the dimensions so they are dealt with in the expected way in constructing lines, paragraphs and pages, but the anchoring is virtual, like the offsets. There are two extra variants for orientation zero: on top of baseline or below, with dimensions taken into account.

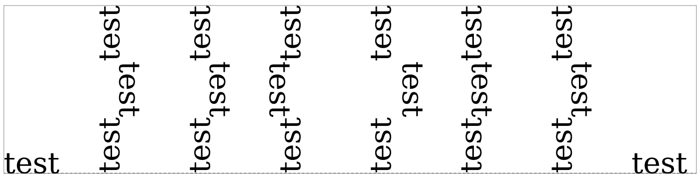
```
\ruledhbox orientation "000 {\TEX} and
\ruledhbox orientation "004 {\TEX} and
\ruledhbox orientation "005 {\TEX}
```



The anchoring can look somewhat confusing but you need to keep in mind that it is normally only used in very controlled circumstances and not in running text. Wrapped in macros users don't see the details. We're talking boxes here, so for instance:

```
test\quad
\hbox orientation 3 \bgroup
    \strut test\hbox orientation "002 \bgroup\strut test\egroup test%
\egroup \quad
\hbox orientation 3 \bgroup
    \strut test\hbox orientation "002 \bgroup\strut test\egroup test%
\egroup \quad
\hbox orientation 3 \bgroup
    \strut test\hbox orientation "012 \bgroup\strut test\egroup test%
\egroup \quad
\hbox orientation 3 \bgroup
    \strut test\hbox orientation "022 \bgroup\strut test\egroup test%
\egroup \quad
\hbox orientation 3 \bgroup
    \strut test\hbox orientation "032 \bgroup\strut test\egroup test%
\egroup \quad
\hbox orientation 3 \bgroup
    \strut test\hbox orientation "042 \bgroup\strut test\egroup test%
```

```
\egroup
\quad test
```


test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test
test