Grouping in hybrid environments

Keywords
ConTeXt mkiv, luatex, grouping, underbar, overbar, overstrike, text backgrounds

Variants

After using TEX for a while you get accustomed to one of its interesting concepts: grouping. Programming languages like Pascal and Modula have keywords begin and end. So, one can say:

```
if test then begin
    print_bold("test 1")
    print_bold("test 2")
end
```

Other languages provide a syntax like:

```
if test {
    print_bold("test 1")
    print_bold("test 2")
}
```

So, in those languages the begin and end and/or the curly braces define a 'group' of statements. In TEX on the other hand we have:

```
test \begingroup \bf test \endgroup test
```

Here the second test comes out in a bold font and the switch to bold (basically a different font is selected) is reverted after the group is closed. So, in TEX grouping deals with scope and not with grouping things together.

It depends on the language whether locally defined variables are visible afterwards. In languages like Lua we have constructs like:

```
for i=1,100 do
    local j = i + 20
    ...
end
```

Here j is visible after the loop ends unless prefixed by local. Yet another example is MetaPost:

```
begingroup ;
    save n ; numeric n ; n := 10 ;
    ...
endgroup ;
```

Here all variables are global unless they are explicitly saved inside a group. This makes perfect sense as the resulting graphic also has a global (accumulated) property. In practice one will rarely need grouping, contrary to TEX where one really wants to keep changes local, if only because document content is so unpredictable that one never knows when some change in state happens.

But in TEX variables are local unless a \global prefix (or one of the shortcuts) is used.

In principle it is possible to carry over information across a group boundary. Consider this somewhat unrealistic example:

```
\begingroup
    \leftskip 10pt
    \begingroup
        ....
        \advance\leftskip 10pt
        ....
    \endgroup
\endgroup
```

How to carry the increased \leftskip over the group boundary without using a global assignment which could have more drastic side effects? Here is the trick:

```
\begingroup
    \leftskip 10pt
    \begingroup
        ....
        \advance\leftskip 10pt
        ....
        \expandafter
    \endgroup
    \expandafter \leftskip \the\leftskip
\endgroup
```

This is a typical example of the kind of code that gives new users the creeps but normally they never have to do that kind of coding. Also, this kind of trick assumes that one knows how many groups are involved.

Implication

What does this all have to do with LuaTEX and MkIV? The user interface of ConTEXt provide lots of commands like:

```
\setupthis[style=bold]
\setupthat[color=green]
```

Most of them obey grouping. However, consider a situation where we use Lua code to deal with some aspect of typesetting, for instance numbering lines or adding ornamental elements to the text. In ConTEXt we flag such actions with attributes and often the real action takes place a bit later, for instance when a paragraph or page becomes available.

A comparable pure TEX example is the following:

```
{test test \bf test \leftskip10pt test}
```

Here the switch to bold happens as expected but no `\leftskip` of 10pt is applied. This is because the set value is already forgotten when the paragraph is actually typeset. So in fact we would need:

```
{test test \bf test \leftskip10pt test \par}
```

Now, say that we have:

```
{test test test \setupflag[option=1]
                \flagnexttext test}
```

We flag some text (using an attribute) and expect it to get a treatment where option 1 is used. However, the real action might take place when TEX deals with the paragraph or page and by that time the specific option is already forgotten or it might have received another value. So, the rather natural TEX grouping does not work out that well in a hybrid situation.

As the user interface assumes a consistent behaviour we cannot simply make these settings global even if this makes much sense in practice. One solution is to carry the information with the flagged text i.e. associate it somehow with the attribute's value. Of course, as we never know in advance when this information is used, this might result in quite some states being stored persistently.

A side effect of this 'problem' is that new commands might get suboptimal user interfaces (especially inheritance or cloning of constructs) that are somewhat driven by these 'limitations'. Of course we may wonder if the end user will notice this.

To summarize this far, we have three sorts of grouping to deal with:

- TEX's normal grouping model limits its scope to the local situation and normally has only direct and local consequences. We cannot carry information over groups.
- Some of TEX's properties are applied later, for instance when a paragraph or page is typeset and in order to make 'local' changes effective, the user needs to add explicit paragraph ending commands (like `\par` or `\page`).
- Features dealt with asynchronously by Lua are at that time unaware of grouping and variables set that were active at the time the feature was triggered so there we need to make sure that our settings travel with the feature. There is not much that a user can do about it as this kind of management has to be done by the feature itself.

It is the third case that I will give an example of in the next section. I will leave it up to the user whether it gets noticed in the user interface.

An example

A group of commands that has been reimplemented using a hybrid solution is underlining or more generically: bars. Just take a look at the following examples and try to get an idea of how to deal with grouping. Keep in mind that:

- Colors are attributes and are resolved in the back-end, so way after the paragraph has been typesetting.
- Overstrike is also handled by an attribute and gets applied in the back-end as well, before colours are applied.
- Nested overstrikes might have different settings.
- An overstrike rule either inherits from the text or has its own colour setting.

First an example where we inherit colour from the text:

```
\definecolor[myblue][b=.75]
\definebar[myoverstrike][overstrike][color=]

Test \myoverstrike{%
    Test \myoverstrike{\myblue
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Because colour is also implemented using attributes and processed later we can access that information when we deal with the bar.

The following example has its own colour setting:

```
\definecolor[myblue][b=.75]
\definecolor[myred] [r=.75]
\definebar[myoverstrike][overstrike][color=myred]

Test \myoverstrike{%
    Test \myoverstrike{\myblue
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

See how can we colour the levels differently:

```
\definecolor[myblue] [b=.75]
\definecolor[myred]  [r=.75]
\definecolor[mygreen][g=.75]

\definebar[myoverstrike:1]
          [overstrike][color=myblue]
\definebar[myoverstrike:2]
          [overstrike][color=myred]
\definebar[myoverstrike:3]
          [overstrike][color=mygreen]

Test \myoverstrike{%
    Test \myoverstrike{%
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Watch this:

```
\definecolor[myblue] [b=.75]
\definecolor[myred]  [r=.75]
\definecolor[mygreen][g=.75]

\definebar[myoverstrike]
          [overstrike][max=1,dy=0,offset=.5]
\definebar[myoverstrike:1]
          [myoverstrike][color=myblue]
\definebar[myoverstrike:2]
          [myoverstrike][color=myred]
\definebar[myoverstrike:3]
          [myoverstrike][color=mygreen]

Test \myoverstrike{%
    Test \myoverstrike{%
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Is this the perfect user interface? Probably not, but at least it keeps the implementation quite simple.

The behaviour of the MkIV implementation is roughly the same as in MkII, although now we specify the dimensions and placement in terms of the ratio of the x-height of the current font.

```
Test \overstrike{Test \overstrike{Test
  \overstrike{Test} Test} Test} Test \blank
Test \underbar  {Test \underbar  {Test
  \underbar  {Test} Test} Test} Test \blank
Test \overbar   {Test \overbar   {Test
  \overbar   {Test} Test} Test} Test \blank
Test \underbar  {Test \overbar   {Test
  \overstrike{Test} Test} Test} Test \blank
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Test Test Test Test Test Test Test

Test Test Test Test Test Test Test

Test Test Test ~~Test~~ Test Test Test

As a bonus this mechanism can also provide simple backgrounds. The normal background mechanism uses MetaPost and the advantage is that we can use arbitrary shapes but it also carries some limitations. When the development of LuaTEX is a bit further along the road I will add the possibility to use MetaPost shapes in this mechanism.

Before we come to backgrounds, first take a look at these examples:

```
\startbar[underbar]  \input zapf \stopbar \blank
\startbar[underbars] \input zapf \stopbar \blank
```

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typog-

raphy from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

First notice that it is no problem to span multiple lines and that hyphenation is not influenced at all. Second you can see that continuous rules are also possible. From such a continuous rule to a background is a small step:

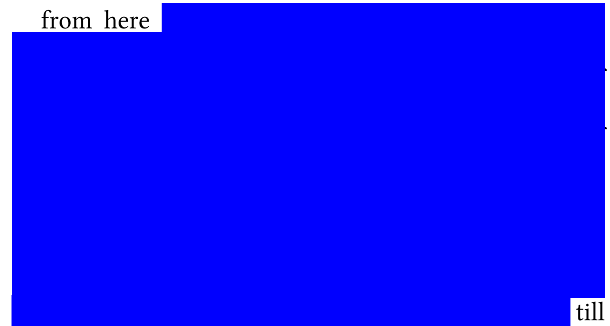
```
\definebar
  [backbar]
  [offset=1.5,rulethickness=2.8,color=blue,
   continue=yes,order=background]

\definebar
  [forebar]
  [offset=1.5,rulethickness=2.8,color=blue,
   continue=yes,order=foreground]
```

The following example code looks messy but this has to do with the fact that we want properly spaced sample injection.

```
from here
    \startcolor[white]%
        \startbar[backbar]%
            \input zapf
            \removeunwantedspaces
        \stopbar
    \stopcolor
\space till here
\blank
from here
    \startbar[forebar]%
        \input zapf
        \removeunwantedspaces
    \stopbar
\space till here
```

from here Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on. till here

from here



till here

Watch how we can use the order to hide content. By default rules are drawn on top of the text.

Nice effects can be accomplished with transparencies:

```
\definecolor [tblue] [b=.5,t=.25,a=1]
\setupbars [backbar] [color=tblue]
\setupbars [forebar] [color=tblue]
```

We use as example:

```
from here {\white \backbar{test test}
    \backbar {nested nested} \backbar{also also}}
till here
from here {\white \backbar{test test
    \backbar {nested nested}          also also}}
till here
from here {\white \backbar{test test
    \backbar {nested nested}          also also}}
till here
```

from here test test nested nested also also till here from here test test nested nested also also till here from here test test nested nested also also till here

The darker nested variant is just the result of two transparent bars on top of each other. We can limit stacking, for instance:

```
\setupbars[backbar][max=1]
\setupbars[forebar][max=1]
```

This gives

from here test test nested nested also also till here from here test test nested nested also also till here from here test test nested nested also also till here

There are currently some limitations, mostly due to the fact that MkIV uses only one attribute for this feature and a change in the value therefore triggers different handling. So, there is no real nesting here.

The default commands are defined as follows:

```
\definebar[overstrike]
```

```
          [method=0,dy= 0.4,offset= 0.5]
\definebar[underbar]
          [method=1,dy=-0.4,offset=-0.3]
\definebar[overbar]
          [method=1,dy= 0.4,offset= 1.8]

\definebar[overstrikes]
          [overstrike] [continue=yes]
\definebar[underbars]
          [underbar]   [continue=yes]
\definebar[overbars]
          [overbar]    [continue=yes]
```

As the implementation is rather non-intrusive you can use bars almost everywhere. You can underbar a whole document but you can stick to fooling around with for instance formulas equally well.

```
\definecolor [tred]   [r=.5,t=.25,a=1]
\definecolor [tgreen] [g=.5,t=.25,a=1]
\definecolor [tblue]  [b=.5,t=.25,a=1]

\definebar [mathred]   [backbar] [color=tred]
\definebar [mathgreen] [backbar] [color=tgreen]
\definebar [mathblue]  [backbar] [color=tblue]

\startformula
\mathred{e} =
  \mathgreen{\white mc} ^ {\mathblue{\white e}}
\stopformula
```

We get:

$$e = mc^{e}$$

We started this chapter with some words on grouping. In the examples you see no difference between adding bars and for instance applying colour. However you need to keep in mind that this is only because behind the screens we keep the current settings along with the attribute. In practice this is only noticeable when you do lots of (local) changes to the settings. Take:

```
{test test test
  \setupbars[color=red] \underbar{test} test}
```

This results in a local change in settings, which in turn will associate a new attribute to `\underbar`. So, in fact the following underbar becomes a different one from the previous underbars. When the page is prepared, the unique attribute value will relate to those settings. Of course there are more mechanisms where such associations take place.

More to come

Is this all there is? No, as usual the underlying mechanisms can be used for other purposes as well. Take for instance in-line notes:

```
According to Wikipedia this is the longest
English word:
pneumonoultramicroscopicsilicovolcanoconiosis~%
\shiftup {other long
words are pseudopseudohypoparathyroidism and
flocci-nauci-nihili-pili-fication}. Of course
in languages like Dutch and German we can make
arbitrary long words by pasting words together.
```

This will produce:

According to Wikipedia this is the longest English word: pneumonoultramicroscopicsilicovolcanoconiosis other long words are pseudopseudohypoparathyroidism and floccinaucinihilipilification. Of course in languages like Dutch and German we can make arbitrary long words by pasting words together.

I wonder when users really start using such features.

Summary

Although under the hood the MkIV bar commands are quite different from their MkII counterparts users probably won't notice much difference at first sight. However, the new implementation does not interfere with the par builder and other mechanisms. Plus, it is configurable and it offers more functionality. However, as it is processed in delayed fashion, side effects might occur that are not foreseen.

So, if you ever notice such unexpected side effects, you know where it might result from: what you asked for is processed much later and by then the circumstances might have changed. If you suspect that it relates to grouping there is a simple remedy: define a new bar command in the document preamble instead of changing properties mid-document. After all, you are supposed to separate rendering and content in the first place.

Hans Hagen