low level TEX

lowlevel

Contents

Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

2 Conditionals

low level TEX

conditionals

Contents

2.1 Preamble

2.1.1 Introduction

You seldom need the low level conditionals because there are quite some so called support macros available in ConTEXt. For instance, when you want to compare two values (or more accurate: sequences of tokens), you can do this:

```
\doifelse {foo} {bar} {
    the same
} {
    different
}
```

But if you look in the ConTEXt code, you will see that often we use primitives that start with `\if` in low level macros. There are good reasons for this. First of all, it looks familiar when you also code in other languages. Another reason is performance but that is only true in cases where the snippet of code is expanded very often, because TEX is already pretty fast. Using low level TEX can also be more verbose, which is not always nice in a document source. But, the most important reason (for me) is the layout of the code. I often let the look and feel of code determine the kind of coding. This also relates to the syntax highlighting that I am using, which is consistent for TEX, MetaPost, Lua, etc. and evolved over decades. If code looks bad, it probably is bad. Of course this doesn't mean all my code looks good; you're warned. In general we can say that I often use `\if...` when coding core macros, and `\doifelse...` macros in (document) styles and modules.

In the sections below I will discuss the low level conditions in TEX. For the often more convenient ConTEXt wrappers you can consult the source of the system and support modules, the wiki and/or manuals.

Some of the primitives shown here are only available in LuaTEX, and some only in LuaMetaTEX. We could do without them for decades but they were added to these engines because of convenience and, more important, because then made for nicer code. Of course there's also the fun aspect. This manual is not an invitation to use these very low level primitives in your document source. The ones that probably make most sense are `\ifnum`, `\ifdim` and `\ifcase`. The others are often wrapped into support macros that are more convenient.

2.1.2 Number and dimensions

Numbers and dimensions are basic data types in TEX. When you enter one, a number is just that but a dimension gets a unit. Compare:

```
1234
1234pt
```

If you also use MetaPost, you need to be aware of the fact that in that language there are not really dimensions. The `post` part of the name implies that eventually a number becomes a PostScript unit which represents a base point (`bp`) in TEX. When in MetaPost you entry `1234pt` you actually multiply `1234` by the variable `pt`. In TEX on the other hand, a unit like `pt` is one of the keywords that gets parsed. Internally dimensions are also numbers and the unit (keyword) tells the scanner what multiplier to use. When that multiplier is one, we're talking of scaled points, with the unit `sp`.

```
\the\dimexpr 12.34pt \relax
\the\dimexpr 12.34sp \relax
\the\dimexpr 12.99sp \relax
\the\dimexpr 1234sp  \relax
\the\numexpr 1234    \relax
```

```
12.34pt
0.00018pt
0.00018pt
0.01883pt
1234
```

When we serialize a dimension it always shows the dimension in points, unless we serialize it as number.

```
\scratchdimen1234sp
\number\scratchdimen
\the\scratchdimen
```

1234
0.01883pt

When a number is scanned, the first thing that is taken care of is the sign. In many cases, when TEX scans for something specific it will ignore spaces. It will happily accept multiple signs:

```
\number +123
\number +++123
\number + + + 123
\number +-+-+123
\number --123
\number ---123
```

123
123
123
123
123
-123

Watch how the negation accumulates. The scanner can handle decimal, hexadecimal and octal numbers:

```
\number -123
\number -"123
\number -'123
```

-123
-291
-83

A dimension is scanned like a number but this time the scanner checks for upto three parts: an either or not signed number, a period and a fraction. Here no number means zero, so the next is valid:

```
\the\dimexpr  . pt \relax
\the\dimexpr 1. pt \relax
\the\dimexpr  .1pt \relax
\the\dimexpr 1.1pt \relax
```

0.0pt
1.0pt

0.1pt
1.1pt

Again we can use hexadecimal and octal numbers but when these are entered, there can be no fractional part.

```
\the\dimexpr  16 pt \relax
\the\dimexpr "10 pt \relax
\the\dimexpr '20 pt \relax
```

16.0pt
16.0pt
16.0pt

The reason for discussing numbers and dimensions here is that there are cases where when TEX expects a number it will also accept a dimension. It is good to know that for instance a macro defined with `\chardef` or `\mathchardef` also is treated as a number. Even normal characters can be numbers, when prefixed by a ` (backtick).

The maximum number in TEX is 2147483647 so we can do this:

```
\scratchcounter2147483647
```

but not this

```
\scratchcounter2147483648
```

as it will trigger an error. A dimension can be positive and negative so there we can do at most:

```
\scratchdimen  1073741823sp
```

```
\scratchdimen1073741823sp
\number\scratchdimen
\the\scratchdimen
\scratchdimen16383.99998pt
\number\scratchdimen
\the\scratchdimen
```

1073741823
16383.99998pt
1073741823
16383.99998pt

We can also do this:

```
\scratchdimen16383.99999pt
\number\scratchdimen
\the\scratchdimen
```

1073741823
16383.99998pt

but the next one will fail:

```
\scratchdimen16383.9999999pt
```

Just keep in mind that TEX scans both parts as number so the error comes from checking if those numbers combine well.

```
\ifdim 16383.99999  pt = 16383.99998  pt the same \else different \fi
\ifdim 16383.999979 pt = 16383.999980 pt the same \else different \fi
\ifdim 16383.999987 pt = 16383.999991 pt the same \else different \fi
```

Watch the difference in dividing, the / rounds, while the : truncates.

the same
the same
the same

You need to be aware of border cases, although in practice they never really are a problem:

```
\ifdim \dimexpr16383.99997 pt/2\relax = \dimexpr 16383.99998 pt/2\relax
    the same \else different
\fi
\ifdim \dimexpr16383.99997 pt:2\relax = \dimexpr 16383.99998 pt:2\relax
    the same \else different
\fi
```

different
the same

```
\ifdim \dimexpr1.99997 pt/2\relax = \dimexpr 1.99998 pt/2\relax
    the same \else different
\fi
\ifdim \dimexpr1.99997 pt:2\relax = \dimexpr 1.99998 pt:2\relax
    the same \else different
```

```
\fi
```

different
the same

```
\ifdim \dimexpr1.999999 pt/2\relax = \dimexpr 1.9999995 pt/2\relax
    the same \else different
\fi
\ifdim \dimexpr1.999999 pt:2\relax = \dimexpr 1.9999995 pt:2\relax
    the same \else different
\fi
```

the same
the same

This last case demonstrates that at some point the digits get dropped (still assuming that the fraction is within the maximum permitted) so these numbers then are the same. Anyway, this is not different in other programming languages and just something you need to be aware of.

2.2 TEX primitives

2.2.1 \if

I seldom use this one. Internally TEX stores (and thinks) in terms of tokens. If you see for instance `\def` or `\dimen` or `\hbox` these all become tokens. But characters like `A` or `@` also become tokens. In this test primitive all non-characters are considered to be the same. In the next examples this is demonstrated.

```
[\if AB yes\else nop\fi]
[\if AA yes\else nop\fi]
[\if CDyes\else nop\fi]
[\if CCyes\else nop\fi]
[\if\dimen\font yes\else nop\fi]
[\if\dimen\font yes\else nop\fi]
```

Watch how spaces after the two characters are kept: [nop] [yes] [nop] [yes] [yes] [yes]. This primitive looks at the next two tokens but when doing so it expands. Just look at the following:

```
\def\AA{AA}%
```

```
\def\AB{AB}%
[\if\AA yes\else nop\fi]
[\if\AB yes\else nop\fi]
```

We get: [yes] [nop].

2.2.2 \ifcat

In TEX characters (in the input) get interpreted according to their so called catcodes. The most common are letters (alphabetic) and and other (symbols) but for instance the backslash has the property that it starts a command, the dollar signs trigger math mode, while the curly braced deal with grouping. If for instance either or not the ampersand is special (for instance as column separator in tables) depends on the macro package.

```
[\ifcat AB yes\else nop\fi]
[\ifcat AA yes\else nop\fi]
[\ifcat CDyes\else nop\fi]
[\ifcat CCyes\else nop\fi]
[\ifcat C1yes\else nop\fi]
[\ifcat\dimen\font yes\else nop\fi]
[\ifcat\dimen\font yes\else nop\fi]
```

This time we also compare a letter with a number: [yes] [yes] [yes] [yes] [nop] [yes] [yes]. In that case the category codes differ (letter vs other) but in this test comparing the letters result in a match. This is a test that is used only once in ConTEXt and even that occasion is dubious and will go away.

You can use `\noexpand` to prevent expansion:

```
\def\A{A}%
\let\B B%
\def\C{D}%
\let\D D%
[\ifcat\noexpand\A Ayes\else nop\fi]
[\ifcat\noexpand\B Byes\else nop\fi]
[\ifcat\noexpand\C Cyes\else nop\fi]
[\ifcat\noexpand\C Dyes\else nop\fi]
[\ifcat\noexpand\D Dyes\else nop\fi]
```

We get: [nop] [yes] [nop] [nop] [yes], so who still thinks that TEX is easy to understand for a novice user?

2.2.3 \ifnum

This condition compares its argument with another one, separated by an <, = or > character.

```
\ifnum\scratchcounter<0
    less than
\else\ifnum\scratchcounter>0
    more than
\else
    equal to
\fi zero
```

This is one of these situations where a dimension can be used instead. In that case the dimension is in scaled points.

```
\ifnum\scratchdimen<0
    less than
\else\ifnum\scratchdimen>0
    more than
\else
    equal to
\fi zero
```

Of course this equal treatment of a dimension and number is only true when the dimension is a register or box property.

2.2.4 \ifdim

This condition compares one dimension with another one, separated by an <, = or > sign.

```
\ifdim\scratchdimen<0pt
    less than
\else\ifdim\scratchdimen>0pt
    more than
\else
    equal to
\fi zero
```

While when comparing numbers a dimension is a valid quantity but here you cannot mix them: something with a unit is expected.

2.2.5 \ifodd

This one can come in handy, although in ConTEXt it is only used in checking for an odd of even page number.

```
\scratchdimen  3sp
\scratchcounter4

\ifodd\scratchdimen   very \else not so \fi odd
\ifodd\scratchcounter very \else not so \fi odd
```

As with the previously discussed \ifnum you can use a dimension variable too, which is then interpreted as representing scaled points. Here we get:

very odd
not so odd

2.2.6 \ifvmode

This is a rather trivial check. It takes no arguments and just is true when we're in vertical mode. Here is an example:

```
\hbox{\ifvmode\else\par\fi\ifvmode v\else h\fi mode}
```

We're always in horizontal mode and issuing a \par inside a horizontal box doesn't change that, so we get: hmode.

2.2.7 \ifhmode

As with \ifvmode this one has no argument and just tells if we're in vertical mode.

```
\vbox {
    \noindent \ifhmode h\else v\fi mode
    \par
    \ifhmode h\else \noindent v\fi mode
}
```

You can use it for instance to trigger injection of code, or prevent that some content (or command) is done more than once:

hmode

vmode

2.2.8 \ifmmode

Math is something very TEX so naturally you can check if you're in math mode. here is an example of using this test:

```
\def\enforcemath#1{\ifmmode#1\else$ #1 $\fi}
```

Of course in reality macros that do such things are more advanced than this one.

2.2.9 \ifinner

```
\def\ShowMode
  {\ifhmode      \ifinner inner \fi hmode
   \else\ifvmode \ifinner inner \fi vmode
   \else\ifmmode \ifinner inner \fi mmode
   \else         \ifinner inner \fi unset
   \fi\fi\fi}

\ShowMode \ShowMode

\vbox{\ShowMode}

\hbox{\ShowMode}

$\ShowMode$

$$\ShowMode$$
```

The first line has two tests, where the first one changes the mode to horizontal simply because a text has been typeset. Watch how display math is not inner.

vmode hmode
inner vmode
inner hmode
innermmode
innermmode

By the way, moving the `\ifinner` test outside the branches (to the top of the macro) won't work because once the word `inner` is typeset we're no longer in vertical mode, if we were at all.

2.2.10 \ifvoid

A box is one of the basic concepts in TEX. In order to understand this primitive we present four cases:

```
\setbox0\hbox{}          \ifvoid0 void \else content \fi
\setbox0\hbox{123}       \ifvoid0 void \else content \fi
\setbox0\hbox{} \box0    \ifvoid0 void \else content \fi
\setbox0\hbox to 10pt{} \ifvoid0 void \else content \fi
```

In the first case, we have a box which is empty but it's not void. It helps to know that internally an hbox is actually an object with a pointer to a linked list of nodes. So, the first two can be seen as:

```
hlist -> [nothing]
hlist -> 1 -> 2 -> 3 -> [nothing]
```

but in any case there is a hlist. The third case puts something in a hlist but then flushes it. Now we have not even the hlist any more; the box register has become void. The last case is a variant on the first. It is an empty box with a given width. The outcome of the four lines (with a box flushed in between) is:

content
content

void
content

So, when you want to test if a box is really empty, you need to test also its dimensions, which can be up to three tests, depending on your needs.

```
\setbox0\emptybox                \ifvoid0 void\else content\fi
\setbox0\emptybox       \wd0=10pt \ifvoid0 void\else content\fi
\setbox0\hbox to 10pt {}          \ifvoid0 void\else content\fi
\setbox0\hbox         {} \wd0=10pt \ifvoid0 void\else content\fi
```

Setting a dimension of a void (empty) box doesn't make it less void:

void
void
content
content

2.2.11 \ifhbox

This test takes a box number and gives true when it is an hbox.

2.2.12 \ifvbox

This test takes a box number and gives true when it is an vbox. Both a \vbox and \vtop are vboxes, the difference is in the height and depth and the baseline. In a \vbox the last line determines the baseline

vbox or vtop

vtop or vbox

And in a \vtop the first line takes control:

vbox or vtop

vtop or vbox

but, once wrapped, both internally are just vlists.

2.2.13 \ifx

This test is actually used a lot in ConTEXt: it compares two token(list)s:

```
                  \ifx a b  Y\else N\fi
                  \ifx ab   Y\else N\fi
\def\A {a}\def\B{b}\ifx \A\B Y\else N\fi
\def\A{aa}\def\B{a}\ifx \A\B Y\else N\fi
\def\A {a}\def\B{a}\ifx \A\B Y\else N\fi
```

Here the result is: “NNNNY”. It does not expand the content, if you want that you need to use an \edef to create two (temporary) macros that get compared, like in:

```
\edef\TempA{...}\edef\TempB{...}\ifx\TempA\TempB ...\else ...\fi
```

2.2.14 \ifeof

This test checks if a the pointer in a given input channel has reached its end. It is also true when the file is not present. The argument is a number which relates to the \openin primitive that is used to open files for reading.

2.2.15 \iftrue

It does what it says: always true.

2.2.16 \iffalse

It does what it says: always false.

2.2.17 \ifcase

The general layout of an \ifcase tests is as follows:

```
\ifcase<number>
    when zero
\or
    when one
\or
    when two
\or
    ...
\else
    when something else
\fi
```

As in other places a number is a sequence of signs followed by one of more digits

2.3 ε-TEX primitives

2.3.1 \ifdefined

This primitive was introduced for checking the existence of a macro (or primitive) and with good reason. Say that you want to know if \MyMacro is defined? One way to do that is:

```
\ifx\MyMacro\undefined
    {\bf undefined indeed}
\fi
```

This results in: **undefined indeed**, but is this macro really undefined? When TEX scans your source and sees a the escape character (the forward slash) it will grab the next

characters and construct a control sequence from it. Then it finds out that there is nothing with that name and it will create a hash entry for a macro with that name but with no meaning. Because `\undefined` is also not defined, these two macros have the same meaning and therefore the `\ifx` is true. Imagine that you do this many times, with different macro names, then your hash can fill up. Also, when a user defined `\undefined` you're suddenly get a different outcome.

In order to catch the last problem there is the option to test directly:

```
\ifdefined\MyOtherMacro \else
    {\bf also undefined}
\fi
```

This (or course) results in: **also undefined**, but the macro is still sort of defined (with no meaning). The next section shows how to get around this.

2.3.2 \ifcsname

A macro is often defined using a ready made name, as in:

```
\def\OhYes{yes}
```

The name is made from characters with catcode letter which means that you cannot use for instance digits or underscores unless you also give these characters that catcode, which is not that handy in a document. You can however use `\csname` to define a control sequence with any character in the name, like:

```
\expandafter\def\csname Oh Yes : 1\endcsname{yes}
```

Later on you can get this one with `\csname`:

```
\csname Oh Yes : 1\endcsname
```

However, if you say:

```
\csname Oh Yes : 2\endcsname
```

you won't get some result, nor a message about an undefined control sequence, but the name triggers a define anyway, this time not with no meaning (undefined) but as equivalent to `\relax`, which is why

```
\expandafter\ifx\csname Oh Yes : 2\endcsname\relax
    {\bf relaxed indeed}
```

```
\fi
```

is the way to test its existence. As with the test in the previous section, this can deplete the hash when you do lots of such tests. The way out of this is:

```
\ifcsname Oh Yes : 2\endcsname \else
    {\bf unknown indeed}
\fi
```

This time there is no hash entry created and therefore there is not even an undefined control sequence.

In LuaTEX there is an option to return false in case of a messy expansion during this test, and in LuaMetaTEX that is default. This means that tests can be made quite robust as it is pretty safe to assume that names that make sense are constructed from regular characters and not boxes, font switches, etc.

2.3.3 \iffontchar

This test was also part of the ε-TEX extensions and it can be used to see if a font has a character.

```
\iffontchar\font`A
    {\em This font has an A!}
\fi
```

And, as expected, the outcome is: *"This font has an A!"*. The test takes two arguments, the first being a font identifier and the second a character number, so the next checks are all valid:

```
\iffontchar\font     `A yes\else nop\fi\par
\iffontchar\nullfont `A yes\else nop\fi\par
\iffontchar\textfont0`A yes\else nop\fi\par
```

In the perspective of LuaMetaTEX I considered also supporting `\fontid` but it got a bit messy due to the fact that this primitive expands in a different way so this extension was rejected.

2.3.4 \unless

You can negate the results of a test by using the `\unless` prefix, so for instance you can replace:

```
\ifdim\scratchdimen=10pt
    \dosomething
\else\ifdim\scratchdimen<10pt
    \dosomething
\fi\fi
```

by:

```
\unless\ifdim\scratchdimen>10pt
    \dosomething
\fi
```

An `\unless` makes little sense when used with `\ifcase` but contrary to the other engines we don't error or it; we just give a warning. Some conditionals internally use a case so there we can actually provide a variant:

```
\ifcase 1 \relax zero \or one \or two \else else \fi = one \par
\ifcase 2 \relax zero \or one \or two \else else \fi = two \par

\unless\ifcase 1 \relax zero \or one \or two \else else \fi % warning
\unless\ifcase 2 \relax zero \or one \or two \else else \fi % warning

\ifchkdim1pt\or yes \else nop \fi = yes \par
\ifchkdim2  \or nop \else yes \fi = yes \par

\unless\ifchkdim1pt\or nop \else yes \fi = yes \par
\unless\ifchkdim2  \or yes \else nop \fi = yes \par
```

The `\ifchkdim`, `\ifchkdimension`, `\ifchknum`, `\ifchknumber` and `\ifparameter` are supported.

one = one
two = two
one two
yes = yes
yes = yes
yes = yes
yes = yes

2.4 LuaTEX primitives

2.4.1 \ifincsname

As it had no real practical usage uit might get dropped in LuaMetaTEX, so it will not be discussed here.

2.4.2 \ifprimitive

As it had no real practical usage due to limitations, this one is not available in LuaMetaTEX so it will not be discussed here. If really needed you can use `\ifflags`.

2.4.3 \ifabsnum

This test is inherited from pdfTEX and behaves like `\ifnum` but first turns a negative number into a positive one.

2.4.4 \ifabsdim

This test is inherited from pdfTEX and behaves like `\ifdim` but first turns a negative dimension into a positive one.

2.4.5 \ifcondition

This is not really a test but in order to unstand that you need to know how TEX internally deals with tests.

```
\ifdimen\scratchdimen>10pt
    \ifdim\scratchdimen<20pt
        result a
    \else
        result b
    \fi
\else
    result c
\fi
```

When we end up in the branch of “result a” we need to skip two `\else` branches after we're done. The `\if..` commands increment a level while the `\fi` decrements a level.

The \else needs to be skipped here. In other cases the true branch needs to be skipped till we end up a the right \else. When doing this skipping, TEX is not interested in what it encounters beyond these tokens and this skipping (therefore) goes real fast but it does see nested conditions and doesn't interpret grouping related tokens.

A side effect of this is that the next is not working as expected:

```
\def\ifmorethan{\ifdim\scratchdimen>}
\def\iflessthan{\ifdim\scratchdimen<}

\ifmorethan10pt
    \iflessthan20pt
        result a
    \else
        result b
    \fi
\else
    result c
\fi
```

The \iflessthan macro is not seen as an \if... so the nesting gets messed up. The solution is to fool the scanner in thinking that it is. Say we have:

```
\scratchdimen=25pt

\def\ifmorethan{\ifdim\scratchdimen>}
\def\iflessthan{\ifdim\scratchdimen<}
```

and:

```
\ifcondition\ifmorethan10pt
    \ifcondition\iflessthan20pt
        result a
    \else
        result b
    \fi
\else
    result c
\fi
```

When we expand this snippet we get: “result b” and no error concerning a failure in locating the right \fi's. So, when scanning the \ifcondition is seen as a valid

`\if...` but when the condition is really expanded it gets ignored and the `\ifmorethan` has better come up with a match or not.

In this perspective it is also worth mentioning that nesting problems can be avoided this way:

```
\def\WhenTrue {something \iftrue  ...}
\def\WhenFalse{something \iffalse ...}

\ifnum\scratchcounter>123
    \let\next\WhenTrue
\else
    \let\next\WhenFalse
\fi
\next
```

This trick is mentioned in The TEXbook and can also be found in the plain TEX format. A variant is this:

```
\ifnum\scratchcounter>123
    \expandafter\WhenTrue
\else
    \expandafter\WhenFalse
\fi
```

but using `\expandafter` can be quite intimidating especially when there are multiple in a row. It can also be confusing. Take this: an `\ifcondition` expects the code that follows to produce a test. So:

```
\def\ifwhatever#1%
  {\ifdim#1>10pt
      \expandafter\iftrue
   \else
      \expandafter\iffalse
   \fi}

\ifcondition\ifwhatever{10pt}
    result a
\else
    result b
\fi
```

This will not work! The reason is in the already mentioned fact that when we end up in the greater than `10pt` case, the scanner will happily push the `\iftrue` after the `\fi`, which is okay, but when skipping over the `\else` it sees a nested condition without matching `\fi`, which makes ity fail. I will spare you a solution with lots of nasty tricks, so here is the clean solution using `\ifcondition`:

```
\def\truecondition {\iftrue}
\def\falsecondition{\iffalse}

\def\ifwhatever#1%
  {\ifdim#1>10pt
      \expandafter\truecondition
   \else
      \expandafter\falsecondition
   \fi}

\ifcondition\ifwhatever{10pt}
    result a
\else
    result b
\fi
```

It will be no surprise that the two macros at the top are predefined in ConTEXt. It might be more of a surprise that at the time of this writing the usage in ConTEXt of this `\ifcondition` primitive is rather minimal. But that might change.

As a further teaser I'll show another simple one,

```
\def\HowOdd#1{\unless\ifnum\numexpr ((#1):2)*2\relax=\numexpr#1\relax}

\ifcondition\HowOdd{1}very \else not so \fi odd
\ifcondition\HowOdd{2}very \else not so \fi odd
\ifcondition\HowOdd{3}very \else not so \fi odd
```

This renders:

very odd
not so odd
very odd

The code demonstrates several tricks. First of all we use `\numexpr` which permits more complex arguments, like:

```
\ifcondition\HowOdd{4+1}very \else not so \fi odd
```

```
\ifcondition\HowOdd{2\scratchcounter+9}very \else not so \fi odd
```

Another trick is that we use an integer division (the `:`) which is an operator supported by LuaMetaTEX.

2.5 LuaMetaTEX primitives

2.5.1 \ifnum and ifdim

These have been extended with a few more operators. For instance, we can use a negation:

```
\ifnum 10  > 5 Y\else N\fi
\ifnum 10 !> 5 Y\else N\fi
```

Results in: YN. A bitwise comparison is possible too:

```
\ifnum "02  & 2 Y\else N\fi
\ifnum "02  & 4 Y\else N\fi
\ifnum "02 !& 8 Y\else N\fi
```

yields: YNY. You can also use the Unicode variants ∈, ∉, ≠, ≤, ≥, ≮, and ≯.

2.5.2 \iffloat

This is a test for a float, much like a test for a dimen without unit.

2.5.3 \ifabsfloat

This is a test for a float, much like a test for a dimen without unit.

2.5.4 \ifintervalnum

This is a test for equality of two numbers within an interval, as in:

```
\ifintervalnum   1   2 1 Y\else N\fi
\ifintervalnum   1   3 1 Y\else N\fi
\ifintervalnum 100 102 1 Y\else N\fi
\ifintervalnum 100 102 3 Y\else N\fi
```

which results in: YNNY.

2.5.5 \ifintervaldim

This is a test for equality of two dimensions within an interval, as in:

```
\ifintervaldim   1pt   2pt 1pt Y\else N\fi
\ifintervaldim   1pt   3pt 1pt Y\else N\fi
\ifintervaldim 100pt 102pt 1pt Y\else N\fi
\ifintervaldim 100pt 102pt 3pt Y\else N\fi
```

We get: YNNY.

2.5.6 \ifintervalfloat

This is a test for a float, much like a test for a dimen without unit.

2.5.7 \ifdimexpression

This is a boolean checker so the comparison is done as part of the expression, as in:

```
\ifdimexpression{10pt > (4pt + 8pt)}Y\else N\fi
```

2.5.8 \ifnumexpression

This is a boolean checker so the comparison is done as part fo the expression, as in:

```
\ifnumexpression{10 > (4 + 8)}Y\else N\fi
```

2.5.9 \ifcmpnum

This one is part of s set of three tests that all are a variant of a `\ifcase` test. A simple example of the first test is this:

```
\ifcmpnum 123 345 less \or equal \else more \fi
```

The test scans for two numbers, which of course can be registers or expressions, and sets the case value to 0, 1 or 2, which means that you then use the normal `\or` and `\else` primitives for follow up on the test.

2.5.10 \ifchknum

This test scans a number and when it's okay sets the case value to 1, and otherwise to 2. So you can do the next:

```
\ifchknum 123\or good \else bad \fi
\ifchknum bad\or good \else bad \fi
```

An error message is suppressed and the first `\or` can be seen as a sort of recovery token, although in fact we just use the fast scanner mode that comes with the `\ifcase`: because the result is 1 or 2, we never see invalid tokens.

In order to avoid another scan the a valid result it is made available in `\lastchknumber`.

2.5.11 \ifchknumber

This one is a more rigorous variant of `\ifchknum` and doesn't like trailing non numeric crap.

2.5.12 \ifchknumexpr

This test goes a bit further and accepts an expression.

```
\ifchknumexpr 123 + 45\or good \else bad \fi
```

As with the other checkers, if there is a valid result it is available in `\lastchknumber`.

2.5.13 \ifnumval

A sort of combination of the previous two is `\ifnumval` which checks a number but also if it's less, equal or more than zero:

```
\ifnumval 123\or less \or equal \or more \else error \fi
\ifnumval bad\or less \or equal \or more \else error \fi
```

You can decide to ignore the bad number or do something that makes more sense. Often the to be checked value will be the content of a macro or an argument like `#1`.

2.5.14 \ifcmpdim

This test is like `\ifcmpnum` but for dimensions.

2.5.15 \ifchkdim

This test is like `\ifchknum` but for dimensions. The last checked value is available as `\lastchknumber`.

2.5.16 \ifchkdimension

This one is a more rigorous variant of \ifchkdim and doesn't like trailing rubish.

2.5.17 \ifchkdimexpr

This test is like \ifchknumexpr but for dimensions. The last checked value is available as \lastchkdimension.

2.5.18 \ifdimval

This test is like \ifnumval but for dimensions. The last checked value is available as \lastchkdim

2.5.19 \iftok

Although this test is still experimental it can be used. What happens is that two to be compared 'things' get scanned for. For each we first gobble spaces and \relax tokens. Then we can have several cases:

1. When we see a left brace, a list of tokens is scanned upto the matching right brace.
2. When a reference to a token register is seen, that register is taken as value.
3. When a reference to an internal token register is seen, that register is taken as value.
4. When a macro is seen, its definition becomes the to be compared value.
5. When a number is seen, the value of the corresponding register is taken

An example of the first case is:

```
\iftok {abc} {def}%
  ...
\else
  ...
\fi
```

The second case goes like this:

```
\iftok\scratchtoksone\scratchtokstwo
  ...
\else
  ...
\fi
```

Case one and four mixed:

```
\iftok{123}\TempX
  ...
\else
  ...
\fi
```

The last case is more a catch: it will issue an error when no number is given. Eventually that might become a bit more clever (depending on our needs.)

2.5.20 \ifzeronum, \ifzerodim, \ifzerofloat

The names of these three tells what they do: checking for a zero value.

```
(\ifzerodim   10pt\norelax A\orelse\ifzerodim   0pt\norelax B\else C\fi)
(\ifzeronum   10  \norelax A\orelse\ifzeronum   0  \norelax B\else C\fi)
(\ifzerofloat 10.0\norelax A\orelse\ifzerofloat 0.0\norelax B\else C\fi)
```

Here we use the \norelax to get rid of trailing spaces: (B) (B) (B).

2.5.21 \ifhaschar, \ifhastok, \ifhastoks,\ifhasxtoks

These checkers can be used to identify a (sequence) of token(s) in a given token list. Their working can best be shown with a few examples:

```
\ifhaschar   c {abcd}Y\else N\fi
\ifhastok    c {abcd}Y\else N\fi
\ifhastoks  {c}{abcd}Y\else N\fi
\ifhasxtoks {c}{abcd}Y\else N\fi

\def\abcd{abcd}

\ifhaschar   c {\abcd}Y\else N\fi
\ifhastok    c {\abcd}Y\else N\fi
\ifhastoks  {c}{\abcd}Y\else N\fi
\ifhasxtoks {c}{\abcd}Y\else N\fi

\ifhaschar   c {a{bc}d}Y\else N\fi
\ifhastok    c {a{bc}d}Y\else N\fi
\ifhastoks  {c}{a{bc}d}Y\else N\fi
\ifhasxtoks {c}{a{bc}d}Y\else N\fi
```

```
\def\abcd{a{bc}d}

\ifhaschar   c {\abcd}Y\else N\fi
\ifhastok    c {\abcd}Y\else N\fi
\ifhastoks  {c}{\abcd}Y\else N\fi
\ifhasxtoks {c}{\abcd}Y\else N\fi
```

YYYY

NNNY

NYYY

NNNY

The `\ifhaschar` test will not descend into a braced sublist. The `x` variants expand the list before comparison.

2.5.22 \ifcstok

There is a subtle difference between this one and `\iftok`: spaces and `\relax` tokens are skipped but nothing gets expanded. So, when we arrive at the to be compared 'things' we look at what is there, as-is.

2.5.23 \iffrozen

This is an experimental test. Commands can be defined with the `\frozen` prefix and this test can be used to check if that has been the case.

2.5.24 \ifprotected

Commands can be defined with the `\protected` prefix (or in ConTEXt, for historic reasons, with `\unexpanded`) and this test can be used to check if that has been the case.

2.5.25 \ifarguments

This conditional can be used to check how many arguments were matched. It only makes sense when used with macros defined with the `\tolerant` prefix and/or when the sentinel `\ignorearguments` after the arguments is used. More details can be found in the lowlevel macros manual.

2.5.26 \ifrelax

The following tests all return the same: YYY; it is a shortcut for `\ifx ... \relax` that looks nicer in code.

```
            \ifrelax\relax                  Y\else N\fi
            \ifrelax\norelax                Y\else N\fi
\expandafter\ifrelax\csname ReLaX\endcsname Y\else N\fi
```

2.5.27 \ifempty

This is again a shortcut, this time for `\ifx ...\empty` assuming that `\empty` is defined as being nothing. Instead of a token you can also pass a list, so here we get YNY.

```
\ifempty{}     Y\else N\fi
\ifempty{!}    Y\else N\fi
\ifempty\empty Y\else N\fi
```

2.5.28 \iflastnamedcs

This test is part of the `\csname` repertoire and uses the last valid result from such a command.

```
\def\Hello{upper}
\def\hello{lower}
\ifcsname Hello\endcsname
    \iflastnamedcs\hello
      world
    \orelse\iflastnamedcs\Hello
      World
    \fi
\fi
```

Here the 'Hello' test result in 'World'. It is an example of a follow up test, most likely used in user interfacing.

2.5.29 \ifboolean

Another new one is the following: it tests a number for being zero or not. As with any primitive that scans for a number, it accepts a braced expression too.

```
(\ifboolean 0 T\else F\fi)
(\ifboolean 1 T\else F\fi)
(\ifboolean {(2 * 4) < 5} T\else F\fi)
(\ifboolean \dimexpression{(1em > 20pt) or  (1ex > 15pt)} T\else F\fi)
(\ifboolean \dimexpression{(1em > 3pt)  and (1ex <  3pt)} T\else F\fi)
```

We get: (F) (T) (F) (F) (F).

2.5.30 \iflist

The \ifvoid test doesn't really test for a box being empty, which is why we have an additional primitive. Compare the following:

```
\setbox0\hbox{}
\setbox2\hbox{!}
\setbox4\emptybox % \box\voidbox
\setbox8\box6

\wd0 10pt \wd2 10pt \wd4 10pt \wd6 10pt

[\ifvoid0 Y\else N\fi \iflist0 Y\else N\fi \the\wd0] % empty  hbox
[\ifvoid2 Y\else N\fi \iflist2 Y\else N\fi \the\wd2] % hbox with content
[\ifvoid4 Y\else N\fi \iflist4 Y\else N\fi \the\wd4] % no box
[\ifvoid6 Y\else N\fi \iflist6 Y\else N\fi \the\wd6] % no box
```

The result demonstrates that we check if there is any content at all, independent of dimensions or the presence of a wrapping list node.

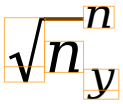
[NN10.0pt] [NY10.0pt] [YN0.0pt] [YN0.0pt]

2.5.31 \ifcramped

This test relates to math and in particular to four of the eight states:

```
\im {
    \sqrt
        {\ifcramped\mathstyle y\else n\fi}
      ^ {\ifcramped\mathstyle y\else n\fi}
      _ {\ifcramped\mathstyle y\else n\fi}
}
```

Because a math formula is first read and then processed in several passes you need to be aware of this state not always being easily predictable because there can be a delay between that read and successive treatments.



2.5.32 \ifmathparameter

The next example demonstrates what this test provides:

```
[\ifmathparameter\Umathextrasubspace   \displaystyle zero\or set\else
  unset\fi]
[\ifmathparameter\Umathaccentbaseheight\displaystyle zero\or set\else
  unset\fi]
[\ifmathparameter\Umathaccentbasedepth \displaystyle zero\or set\else
  unset\fi]
```

There are three possible outcomes; here we get: [zero] [set] [set]. In LuaMetaTEX we have more math parameters than in LuaTEX, and some are set in font specific so called ‘goodie’ files.

2.5.33 \ifmathstyle

Here you need to keep in mind that you test the style that is set when TEX scans for formula. Processing happens afterwards and then styles can change.

```
   {\ifmathstyle D\or D'\or T\or T'\or S\or S'\or SS\or SS'\else ?\fi}
\im{\ifmathstyle D\or D'\or T\or T'\or S\or S'\or SS\or SS'\else ?\fi}
\dm{\ifmathstyle D\or D'\or T\or T'\or S\or S'\or SS\or SS'\else ?\fi}
```

We get: ? T D. The odd values are cramped.

2.5.34 \ifinalignment

This test is an experimental one:

```
\halign \bgroup
    \aligncontent
    \aligntab
    \aligncontent
```

```
    \cr
    one \aligntab \ifinalignment two\else three\fi \cr
    \noalign{\ifinalignment yes\else no\fi}
    one \aligntab \hbox{\ifinalignment two\else three\fi} \cr
\egroup

\hbox{\ifinalignment two\else three\fi}
```

We get:

one two
yes
one two
three

2.5.35 \ifinsert

This primitive checks if an insert box has content. Usage depends on the macro package so for instance in ConTEXt, after `\footnote{A note.}` you can actually check it with:

```
\setupheadertexts[\ifinsert\namedinsertionnumber{footnote} Y\else N\fi]
```

You pass the number of a insert class and in this example the content, set by the page builder, hasn't yet been flushed.

2.5.36 \ifflags

This one related to interfacing. When a macro is defined, one can apply several prefixes to that macro. Some of these prefixes result in a specific kind of macro, for instance a protected, tolerant, tolerant protected, or regular macro. When a macro is defined global, its (internal) level value indicates that. In addition macros, or actually any control sequence, also the built-in ones, can have a set of flags. Some, have consequences in the engine, so for instance an untraced macro will present itself as a primitive, without details that clutter a log. Other flags get meaning when the overload protection mechanisms are enabled.

Testing flags can give some insight but in ConTEXt there is little reason to use this test other than for illustrative purposes. Take this definition

```
\global\protected\def\Foo{Foo}
```

This macro is internally represented as follows; here we used `\meaningasis`:

```
\global \protected \def \Foo {Foo}
```

When we use \meaning we get:

```
protected macro:Foo
```

With \meaningfull we get:

```
global protected macro:Foo
```

Here is how you can test what properties and flags are set.

```
\ifflags\Foo\global    global    \fi
\ifflags\Foo\protected protected \fi
\ifflags\Foo\tolerant  tolerant  \fi
```

We only show a few tests here:

```
global protected
```

Instead of a prefix you can also pass a number:

```
\ifflags\relax\primitiveflagcode primitive \fi
\ifflags\relax\permanentflagcode permanent \fi
```

```
primitive
```

In ConTEXt many macros are defined as permanent which in terms of overload protection has the same impact. Relevant flag values are available in `tex.getflagvalues()` but in ConTEXt we prefer predefined constants:

\aliasedflagcode, \conditionalflagcode, \constantflagcode, \deferredflagcode, \frozenflagcode, \globalflagcode, \immediateflagcode, \immutableflagcode, \inheritedflagcode, \instanceflagcode, \mutableflagcode, \noalignedflagcode, \overloadedflagcode, \permanentflagcode, \primitiveflagcode, \protectedflagcode, \semiprotectedflagcode, \tolerantflagcode, \untracedflagcode, \valueflagcode

2.5.37 \ifparameters

This is an \ifcase where the number is the number of parameters passed to the current macro. Of course, when used in a macro one should be aware of the fact that another macro call will change this number.

2.5.38 \ifparameter

This test checks if a parameter has been set, and it's used as follows:

```
\ifparameter#4\or set\else unset\fi
```

because #4 is actually a reference it refers to the parameter in the current macro and is not influences by nested macro calls which makes if more reliable than a \ifparameters test.

2.5.39 \orelse

This it not really a test primitive but it does act that way. Say that we have this:

```
\ifdim\scratchdimen>10pt
    case 1
\else\ifdim\scratchdimen<20pt
    case 2
\else\ifcount\scratchcounter>10
    case 3
\else\ifcount\scratchcounter<20
    case 4
\fi\fi\fi\fi
```

A bit nicer looks this:

```
\ifdim\scratchdimen>10pt
    case 1
\orelse\ifdim\scratchdimen<20pt
    case 2
\orelse\ifcount\scratchcounter>10
    case 3
\orelse\ifcount\scratchcounter<20
    case 4
\fi
```

We stay at the same level. Sometimes a more flat test tree had advantages but if you think that it gives better performance then you will be disappointed. The fact that we stay at the same level is compensated by a bit more parsing, so unless you have millions such cases (or expansions) it might make a bit of a difference. As mentioned, I'm a bit sensitive for how code looks so that was the main motivation for introducing it. There is a companion \orunless continuation primitive.

A rather neat trick is the definition of \quitcondition:

```
\def\quitcondition{\orelse\iffalse}
```

This permits:

```
\ifdim\scratchdimen>10pt
    case 1a
    \quitcondition
    case 4b
\fi
```

where, of course, the quitting normally is the result of some intermediate extra test. But let me play safe here: beware of side effects.

2.5.40 \orunless

This is the negated variant of \orelse.

2.6 For the brave

2.6.1 Full expansion

If you don't understand the following code, don't worry. There is seldom much reason to go this complex but obscure TEX code attracts some users so ...

When you have a macro that has for instance assignments, and when you expand that macro inside an \edef, these assignments are not actually expanded but tokenized. In LuaMetaTEX there is a way to apply these assignments without side effects and that feature can be used to write a fully expandable user test. For instance:

```
\def\truecondition {\iftrue}
\def\falsecondition{\iffalse}

\def\fontwithidhaschar#1#2%
  {\beginlocalcontrol
   \scratchcounter\numexpr\fontid\font\relax
   \setfontid\numexpr#1\relax
   \endlocalcontrol
   \iffontchar\font\numexpr#2\relax
      \beginlocalcontrol
```

```
      \setfontid\scratchcounter
      \endlocalcontrol
      \expandafter\truecondition
   \else
      \expandafter\falsecondition
   \fi}
```

The \iffontchar test doesn't handle numeric font id, simply because at the time it was added to ε-TEX, there was no access to these id's. Now we can do:

```
\edef\foo{\fontwithidhaschar{1} {75}yes\else nop\fi} \meaning\foo
\edef\foo{\fontwithidhaschar{1}{999}yes\else nop\fi} \meaning\foo

[\ifcondition\fontwithidhaschar{1} {75}yes\else nop\fi]
[\ifcondition\fontwithidhaschar{1}{999}yes\else nop\fi]
```

These result in:

macro:yes
macro:nop

[yes]
[nop]

If you remove the \immediateassignment in the definition above then the typeset results are still the same but the meanings of \foo look different: they contain the assignments and the test for the character is actually done when constructing the content of the \edef, but for the current font. So, basically that test is now useless.

2.6.2 User defined if's

There is a \newif macro that defines three other macros:

```
\newif\ifOnMyOwnTerms
```

After this, not only \ifOnMyOwnTerms is defined, but also:

```
\OnMyOwnTermstrue
\OnMyOwnTermsfalse
```

These two actually are macros that redefine \ifOnMyOwnTerms to be either equivalent to \iftrue and \iffalse. The (often derived from plain TEX) definition of \newif is a

bit if a challenge as it has to deal with removing the `if` in order to create the two extra macros and also make sure that it doesn't get mixed up in a catcode jungle.

In ConTEXt we have a variant:

```
\newconditional\MyConditional
```

that can be used with:

```
\settrue\MyConditional
\setfalse\MyConditional
```

and tested like:

```
\ifconditional\MyConditional
    ...
\else
    ...
\fi
```

This one is cheaper on the hash and doesn't need the two extra macros per test. The price is the use of `\ifconditional`, which is *not* to confused with `\ifcondition` (it has bitten me already a few times).

2.7 Relaxing

When TEX scans for a number or dimension it has to check tokens one by one. On the case of a number, the scanning stops when there is no digit, in the case of a dimension the unit determine the end of scanning. In the case of a number, when a token is not a digit that token gets pushed back. When digits are scanned a trailing space or `\relax` is pushed back. Instead of a number of dimension made from digits, periods and units, the scanner also accepts registers, both the direct accessors like `\count` and `\dimen` and those represented by one token. Take these definitions:

```
\newdimen\MyDimenA \MyDimenA=1pt  \dimen0=\MyDimenA
\newdimen\MyDimenB \MyDimenB=2pt  \dimen2=\MyDimenB
```

I will use these to illustrate the side effects of scanning. Watch the spaces in the result.

First I show what effect we want to avoid. When second argument contains a number (digits) the zero will become part of it so we actually check `\dimen00` here.

```
\def\whatever#1#2%
```

```
  {\ifdim#1=#20\else1\fi}

\whatever{1pt}{2pt}            [macro:1]
\whatever{1pt}{1pt}            [macro:0]
\whatever{\dimen 0}{\dimen 2}  [macro:1]
\whatever{\dimen 0}{\dimen 0}  [macro:]
\whatever\MyDimenA\MyDimenB    [macro:1]
\whatever\MyDimenA\MyDimenB    [macro:1]
```

The solution is to add a space but watch how that one can end up in the result:

```
\def\whatever#1#2%
  {\ifdim#1=#2 0\else1\fi}

\whatever{1pt}{2pt}            [macro:1]
\whatever{1pt}{1pt}            [macro:0]
\whatever{\dimen 0}{\dimen 2}  [macro:1]
\whatever{\dimen 0}{\dimen 0}  [macro:0]
\whatever\MyDimenA\MyDimenB    [macro:1]
\whatever\MyDimenA\MyDimenB    [macro:1]
```

A variant is using `\relax` and this time we get this token retained in the output.

```
\def\whatever#1#2%
  {\ifdim#1=#2\relax0\else1\fi}

\whatever{1pt}{2pt}            [macro:1]
\whatever{1pt}{1pt}            [macro:\relax 0]
\whatever{\dimen 0}{\dimen 2}  [macro:1]
\whatever{\dimen 0}{\dimen 0}  [macro:\relax 0]
\whatever\MyDimenA\MyDimenB    [macro:1]
\whatever\MyDimenA\MyDimenB    [macro:1]
```

A solution that doesn't have side effects of forcing the end of a number (using a space or `\relax` is one where we use expressions. The added overhead of scanning expressions is taken for granted because the effect is what we like:

```
\def\whatever#1#2%
  {\ifdim\dimexpr#1\relax=\dimexpr#2\relax0\else1\fi}

\whatever{1pt}{2pt}            [macro:1]
\whatever{1pt}{1pt}            [macro:0]
\whatever{\dimen 0}{\dimen 2}  [macro:1]
```

```
\whatever{\dimen 0}{\dimen 0}  [macro:0]
\whatever\MyDimenA\MyDimenB    [macro:1]
\whatever\MyDimenA\MyDimenB    [macro:1]
```

Just for completeness we show a more obscure trick: this one hides assignments to temporary variables. Although performance is okay, it is the least efficient one so far.

```
\def\whatever#1#2%
  {\beginlocalcontrol
   \MyDimenA#1\relax
   \MyDimenB#2\relax
   \endlocalcontrol
   \ifdim\MyDimenA=\MyDimenB0\else1\fi}

\whatever{1pt}{2pt}            [macro:1]
\whatever{1pt}{1pt}            [macro:0]
\whatever{\dimen 0}{\dimen 2}  [macro:1]
\whatever{\dimen 0}{\dimen 0}  [macro:0]
\whatever\MyDimenA\MyDimenB    [macro:1]
\whatever\MyDimenA\MyDimenB    [macro:1]
```

It is kind of a game to come up with alternatives but for sure those involve dirty tricks and more tokens (and runtime). The next can be considered a dirty trick too: we use a special variant of `\relax`. When a number is scanned it acts as relax, but otherwise it just is ignored and disappears.

```
\def\whatever#1#2%
  {\ifdim#1=#2\norelax0\else1\fi}

\whatever{1pt}{2pt}            [macro:1]
\whatever{1pt}{1pt}            [macro:0]
\whatever{\dimen 0}{\dimen 2}  [macro:1]
\whatever{\dimen 0}{\dimen 0}  [macro:0]
\whatever\MyDimenA\MyDimenB    [macro:1]
\whatever\MyDimenA\MyDimenB    [macro:1]
```

2.7 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

3 Boxes

low level TEX

boxes

Contents

3.1 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.

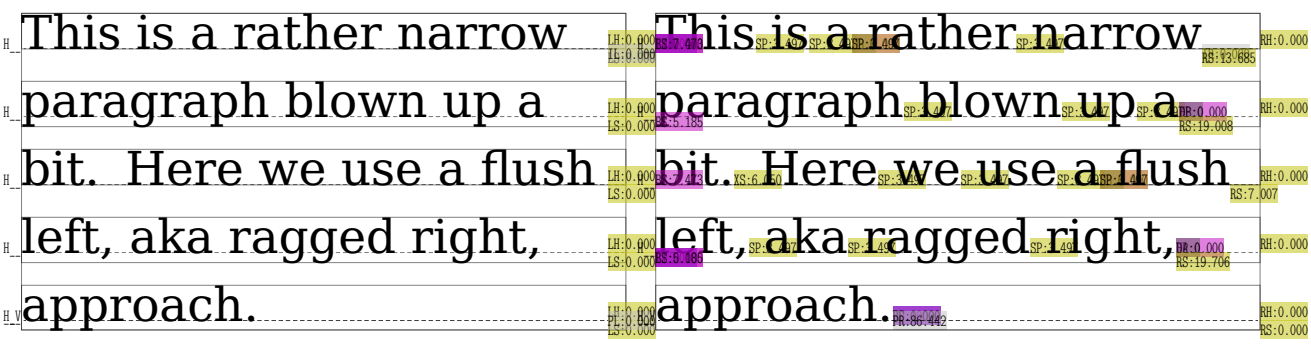
The best reference is of course The TEXbook so if you're really interested in the details you should get a copy of that book. Below I will not go into details about all kind of glues, kerns and penalties, just boxes it is.

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

3.2 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.

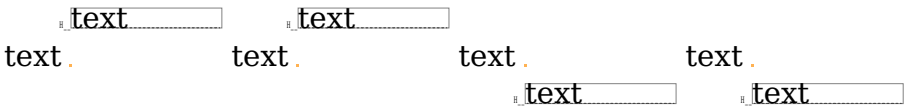
3.3 TEX primitives

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 !
```

texttext ! text righttext !

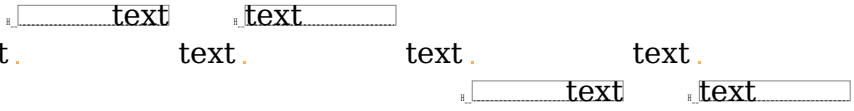
```
text \vbox{\moveleft  25mm \hbox {left}}text !
text \vbox{\moveright 25mm \hbox{right}}text !
```

left text text ! text righttext !

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

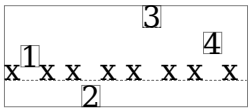
```
text \raise  5mm \hbox to 2cm {\hss text}
text \lower -5mm \hbox to 2cm {text\hss}
text \raise -5mm \hbox to 2cm {\hss text}
text \lower  5mm \hbox to 2cm {text\hss}
```

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.



Instead of \raise you can also provide the shift (up or down) with a keyword:

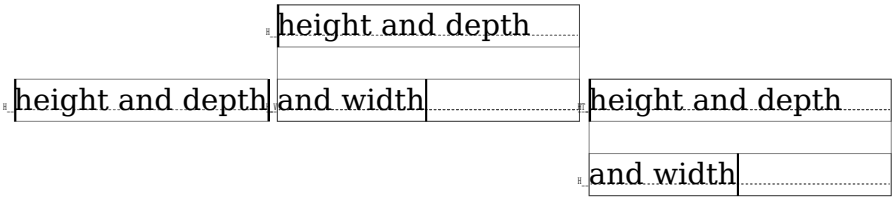
```
\ruledhbox\bgroup
    x\raise  5pt\ruledhbox            {1}x
    x\raise-10pt\ruledhbox            {2}x
    x\raise -5pt\ruledhbox shift -20pt{3}x
    x\ruledhbox             shift -10pt{4}x
\egroup
```



We have three kind of boxes: \hbox, \vbox and \vtop. Actually we have a fourth type \dbox but that is a variant on \vbox to which we come back later.

```
\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
```

3.4 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.

3.5 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.

3.6 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:

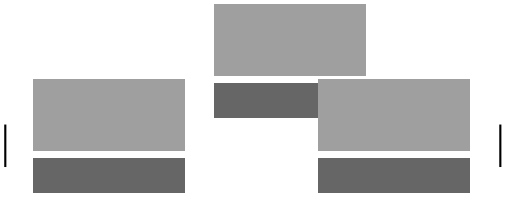
```
\setbox\scratchbox\hbox\bgroup
    \middlegray\vrule width 20mm depth  -.5mm height 10mm
    \hskip-20mm
    \darkgray  \vrule width 20mm height -.5mm depth   5mm
\egroup
```

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.

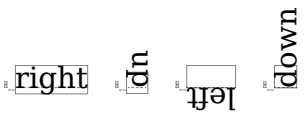
```
\setbox\scratchbox\hbox {height and depth}
[\the\ht\scratchbox]
[\the\dp\scratchbox]
[\the\boxtotal\scratchbox]
\boxtotal\scratchbox=20pt
[\the\ht\scratchbox]
```

```
[\the\dp\scratchbox]
[\the\boxtotal\scratchbox]
```

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

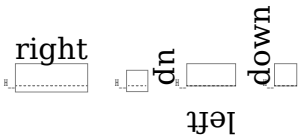
We've now arrived to a set of properties that relate to each other. They are a bit complex and given the number of possibilities one might need to revert to some trial and error: orientations and offsets. As with the dimensions, directions and attributes, they are passed as box specification. We start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

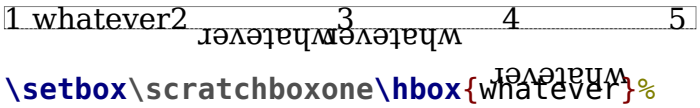
```
\hbox \bgroup \showboxes
          \hbox orientation 0 yoffset  10pt {right}
    \quad \hbox orientation 1 xoffset  10pt {up}
    \quad \hbox orientation 2 yoffset -10pt {left}
    \quad \hbox orientation 3 xoffset -10pt {down}
\egroup
```



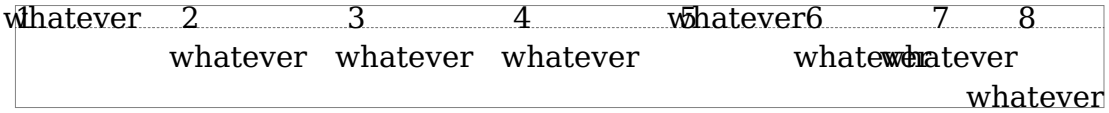
The reason that offsets are related to orientation is that we need to know in what direction the offsets have to be applied and this binding forces the user to think about it. You can also set the offsets using commands.

```
\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>
```

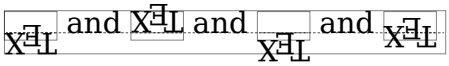
or in TEX speak:

```
"<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.

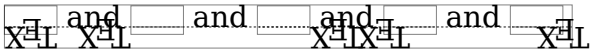
The vertical options of the horizontal variants anchor on the baseline, lower corner, upper corner or center.

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



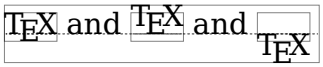
The horizontal options of the horizontal variants anchor in the center, left, right, halfway left and halfway right.

```
\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}
```

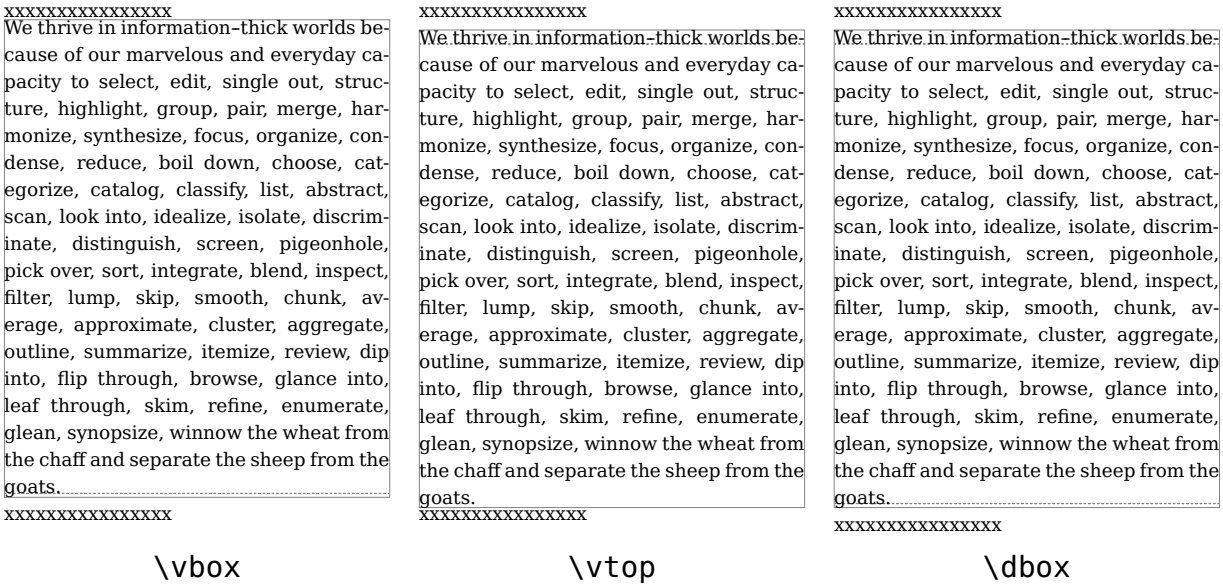


The anchoring can look somewhat confusing but you need to keep in mind that it is normally only used in very controlled circumstances and not in running text. Wrapped in macros users don't see the details. We're talking boxes here, so for instance:

```
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
```

Where a `\vtop` has the baseline at the top, a `\vbox` has it at the bottom. In LuaMetaTEX we also have a `\dbox`, which is a `\vbox` with that behaves like a `\vtop` when it's appended to a vertical list: the height of the first box or rule determines the (base)line correction that gets applied. The following example demonstrates this:



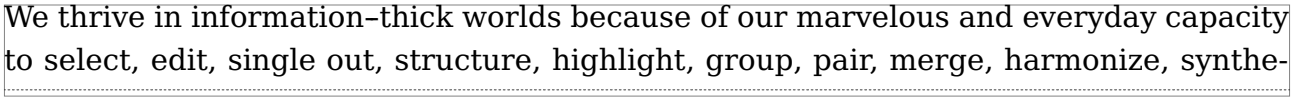
`\vbox` `\vtop` `\dbox`

The d stands for ‘dual’ because we (sort of) have two baselines. The regular height and depth are those of a `\vbox`.

3.7 Splitting

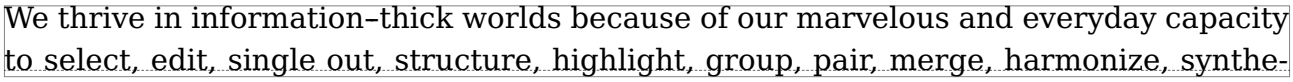
When you feed TEX a paragraph of text it will accumulate the content in a list of nodes. When the paragraphs is finished by `\par` or an empty line it will be fed into the par builder that will try to break the lines as good as possible. Normally that paragraph will be added to the page and at some point there can be breaks between lines in order not to overflow the page. When you collect the paragraph in a box you can use `\vsplit` to emulate this.

```
\setbox\scratchbox\vbox{\samplefile{tufte}}

\startlinecorrection
\ruledhbox{\vsplit\scratchbox to 2\lineheight}
\stoplinecorrection
```



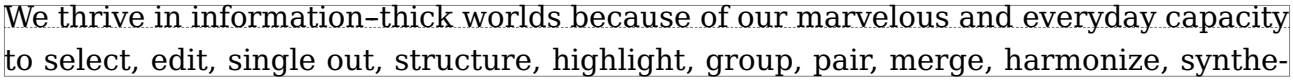
The split off box is given the specified height, but in LuaMetaTEX you can also get the natural dimensions:

```
\setbox\scratchbox\vbox{\samplefile{tufte}}

\startlinecorrection
\ruledhbox{\vsplit\scratchbox upto 2\lineheight}
\stoplinecorrection
```



We can force a resulting box type by using `\vsplit`, `\tsplit` and `\dsplit` (here we use the visualized variants):

```
\setbox\scratchbox\vbox{\samplefile{tufte}}

\startlinecorrection
\ruledtsplit \scratchbox upto 2\lineheight
\stoplinecorrection
```



```
\setbox\scratchbox\vbox{\samplefile{tufte}}
```

```
\startlinecorrection
\ruledvsplit \scratchbox upto 2\lineheight
\stoplinecorrection
```

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthe-

```
\setbox\scratchbox\vbox{\samplefile{tufte}}

\startlinecorrection
\ruleddsplit \scratchbox upto 2\lineheight
\stoplinecorrection
```

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthe-

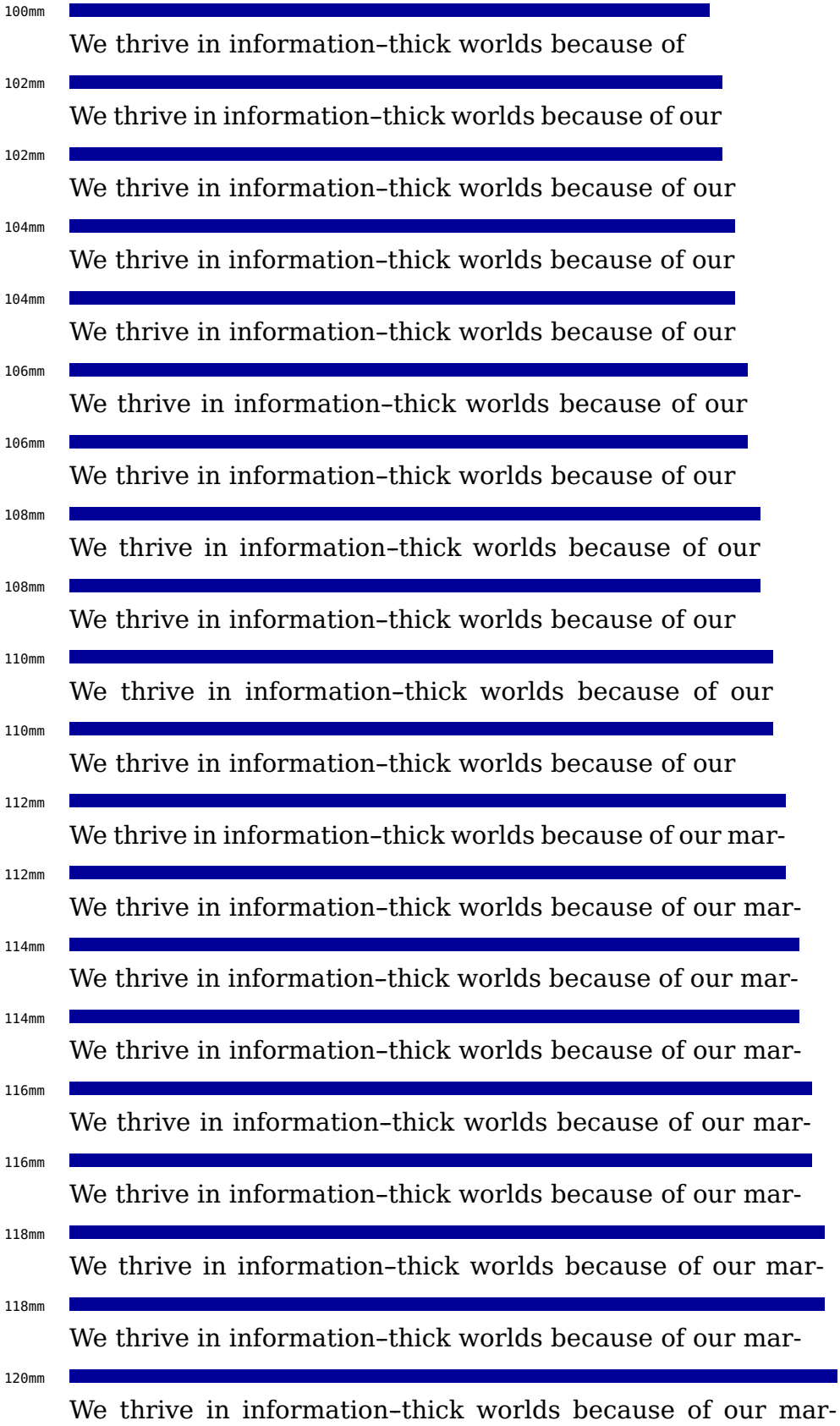
The engine provides vertical splitters but ConTEXt itself also has a horizontal one.[1]

```
\starttexdefinition Test #1#2#3
    \par
    \dontleavehmode
    \strut
    \llap{{\infofont #2}\quad}
    \blackrule[width=#2,color=darkblue]
    \par
    \setbox\scratchbox\hbox{\samplefile{#1}}
    \hsplit\scratchbox
        to              #2
        depth           \strutdp
        height          \strutht
        shrinkcriterium #3 % badness
    \par
\stoptexdefinition

\dostepwiserecurse {100} {120} {2} {
    \Test{tufte}{#1mm}{1000}
    \Test{tufte}{#1mm}{-100}
}
```

100mm

We thrive in information–thick worlds because of

[1] At some point I might turn that one into a native engine primitive.


100mm
We thrive in information–thick worlds because of
102mm
We thrive in information–thick worlds because of our
102mm
We thrive in information–thick worlds because of our
104mm
We thrive in information–thick worlds because of our
104mm
We thrive in information–thick worlds because of our
106mm
We thrive in information–thick worlds because of our
106mm
We thrive in information–thick worlds because of our
108mm
We thrive in information–thick worlds because of our
108mm
We thrive in information–thick worlds because of our
110mm
We thrive in information–thick worlds because of our
110mm
We thrive in information–thick worlds because of our
112mm
We thrive in information–thick worlds because of our mar-
112mm
We thrive in information–thick worlds because of our mar-
114mm
We thrive in information–thick worlds because of our mar-
114mm
We thrive in information–thick worlds because of our mar-
116mm
We thrive in information–thick worlds because of our mar-
116mm
We thrive in information–thick worlds because of our mar-
118mm
We thrive in information–thick worlds because of our mar-
118mm
We thrive in information–thick worlds because of our mar-
120mm
We thrive in information–thick worlds because of our mar-

120mm 

We thrive in information–thick worlds because of our mar-

A split off box gets packed at its natural size and a badness as well as overshoot amount is calculated. When the overshoot is positive and the the badness is larger than the stretch criterium, the box gets repacked to the natural size. The same happens when the overshoot is negative and the badness exceeds the shrink criterium. When the overshoot is zero (basically we have a fit) but the badness still exceeds the stretch or shrink we also repack. Indeed this is a bit fuzzy, but so is badness.

```
\starttexdefinition Test #1#2#3
    \par
    \dontleavehmode
    \strut
    \llap{{\infofont #2}\quad}
    \blackrule[width=#2,color=darkblue]
    \par
    \setbox\scratchbox\hbox{\samplefile{#1}}
    \doloop {
        \ifvoid\scratchbox
            \exitloop
        \else
            \hsplit\scratchbox
                to     #2
                depth  \strutdp
                height \strutht
                #3
            \par
            \allowbreak
        \fi
    }
\stoptexdefinition

\Test{tufte}{100mm}{shrinkcriterium 1000}
\Test{tufte}{100mm}{shrinkcriterium 0}
\Test{tufte}{100mm}{}
```

100mm 

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge,

harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

100mm

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

100mm

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synop-

BBB
size, winnow the wheat from the chaff and separate the sheep from the goats.

Watch how the last line get stretched when we set the criterium to zero. I'm sure that users will find reasons to abuse this effect.

3.7 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

111111
XXX

222222

AAA

4 Expansion

low level TEX

expansion

Contents

4.1 Preamble

This short manual demonstrates a couple of properties of the macro language. It is not an in-depth philosophical expose about macro languages, tokens, expansion and such that some TEXies like. I prefer to stick to the practical aspects. Occasionally it will be technical but you can just skip those paragraphs (or later return to them) when you can't follow the explanation. It's often not that relevant. I won't talk in terms of mouth, stomach and gut the way the TEXbook does and although there is no way to avoid the word 'token' I will do my best to not complicate matters by too much token speak. Examples show best what we mean.

4.2 TEX primitives

The TEX language provides quite some commands and those built in are called primitives. User defined commands are called macros. A macro is a shortcut to a list of primitives and/or macro calls. All can be mixed with characters that are to be typeset somehow.

```
\def\MyMacro{b}

a\MyMacro c
```

When TEX reads this input the `a` gets turned into a glyph node with a reference to the current font set and the character `a`. Then the parser sees a macro call, and it will enter another input level where it expands this macro. In this case it sees just an `b` and it will give this the same treatment as the `a`. The macro ends, the input level decrements and the `c` gets its treatment.

Before we move on to more examples and differences between engines, it is good to stress that `\MyMacro` is not a primitive command: we made our command here. The `b` actually can be seen as a sort of primitive because in this macro it gets stored as so

called token with a primitive property. That primitive property can later on be used to determine what to do. More explicit examples of primitives are `\hbox`, `\advance` and `\relax`. It will be clear that ConTEXt extends the repertoire of primitive commands with a lot of macro commands. When we typeset a source using module `m-scite` the primitives come out dark blue.

The amount of primitives differs per engine. It all starts with TEX as written by Don Knuth. Later ε-TEX added some more primitives and these became official extensions adopted by other variants of TEX. The pdfTEX engine added quite some and as follow up on that LuaTEX added more but didn't add all of pdfTEX. A few new primitives came from Omega (Aleph). The LuaMetaTEX engine drops a set of primitives that comes with LuaTEX and adds plenty new ones. The nature of this engine (no backend and less frontend) makes that we need to implement some primitives as macros. But the basic set is what good old TEX comes with.

Internally these so called primitives are grouped in categories that relate to their nature. They can be directly expanded (a way of saying that they get immediately interpreted) or delayed (maybe stored for later usage). They can involve definitions, calculations, setting properties and values or they can result in some typesetting. This is what makes TEX confusing to new users: it is a macro programming language, an interpreter but at the same time an executor of typesetting instructions.

A group of primitives is internally identified as a command (they have a `cmd` code) and the sub commands are flagged by their `chr` code. This sounds confusing but just thing of the fact that most of what we input are characters and therefore they make up most sub commands. For instance the 'letter `cmd`' is used for characters that are seen as letters that can be used in the name of user commands, can be typeset, are valid for hyphenation etc. The letter related `cmd` can have many `chr` codes (all of Unicode). I'd like to remark that the grouping is to a large extend functional, so sometimes primitives that you expect to be similar in nature are in different groups. This has to do with the fact that TEX needs to be a able to determine efficiently if a primitive is operating (or forbidden) in horizontal, vertical and/or math mode.

There are more than 150 internal `cmd` groups. if we forget about the mentioned character related ones, some, have only a few sub commands (`chr`) and others many more (just consider all the OpenType math spacing related parameters). A handful of these commands deal with what we call macros: user defined combinations of primitives and other macros, consider them little programs. The `\MyMacro` example above is an example. There are differences between engines. In standard TEX there are `\outer` and `\long` commands, and most engines have these. However, in LuaMetaTEX the later to be discussed `\protected` macros have their own specific 'call `cmd`'. Users don't need to bother about this.

So, when from now on we talk about primitives, we mean the built in, hard coded commands, and when we talk about macros we mean user commands. Although internally there are less `cmd` categories than primitives, from the perspective of the user they are all unique. Users won't consult the source anyway but when they do they are warned. Also, when in LuaMetaTEX you use the low level interfacing to TEX you have to figure out these subtle aspects because there this grouping does matter.

Before we continue I want to make clear that expansion (as discussed in this document) can refer to a macro being expanded (read: its meaning gets injected into the input, so the engine kind of sidetracks from what is was doing) but also to direct consequences of running into a primitive. However, users only need to consider expansion in the perspective of macros. If a user has `\advance` in the input it immediately gets done. But when it's part of a macro definition it only is executed when the macro expands. A good check in (traditional) TEX is to compare what happens in `\def` and `\edef` which is why we will use these two in the upcoming examples. You put something in a macro and then check what `\meaning` or `\show` reports.

Now back to user defined macros. A macro can contain references to macros so in practice the input can go several levels up and some applications push back a lot so this is why your TEX input stack can be configured to be huge.

```
\def\MyMacroA{ and }
\def\MyMacroB{1\MyMacroA 2}

a\MyMacroA b
```

When `\MyMacroB` is defined, its body gets three so called tokens: the character token `1` with property 'other', a token that is a reference to the macro `\MyMacroB`, and a character token `2`, also with property 'other' The meaning of `\MyMacroA` is five tokens: a reference to a space token, then three character tokens with property 'letter', and finally a space token.

```
\def \MyMacroA{ and }
\edef\MyMacroB{1\MyMacroA 2}

a\MyMacroA b
```

In the second definition an `\edef` is used, where the `e` indicates expansion. This time the meaning gets expanded immediately. So we get effectively the same as in:

```
\def\MyMacroB{1 and 2}
```

Characters are easy: they just expand to themselves or trigger adding a glyph node, but not all primitives expand to their meaning or effect.

```
\def\MyMacroA{\scratchcounter = 1 }
\def\MyMacroB{\advance\scratchcounter by 1}
\def\MyMacroC{\the\scratchcounter}

\MyMacroA a
\MyMacroB b
\MyMacroB c
\MyMacroB d
\MyMacroC
```

a b c d 4

```
macro:\scratchcounter = 1
macro:\advance \scratchcounter by 1
macro:\the \scratchcounter
```

Let's assume that `\scratchcounter` is zero to start with and use `\edef`'s:

```
\edef\MyMacroA{\scratchcounter = 1 }
\edef\MyMacroB{\advance\scratchcounter by 1}
\edef\MyMacroC{\the\scratchcounter}

\MyMacroA a
\MyMacroB b
\MyMacroB c
\MyMacroB d
\MyMacroC
```

a b c d 0

```
macro:\scratchcounter = 1
macro:\advance \scratchcounter by 1
macro:0
```

So, this time the third macro has its meaning frozen, but we can prevent this by applying a `\noexpand` when we do this:

```
\edef\MyMacroA{\scratchcounter = 1 }
\edef\MyMacroB{\advance\scratchcounter by 1}
\edef\MyMacroC{\noexpand\the\scratchcounter}

\MyMacroA a
\MyMacroB b
```

```
\MyMacroB c
\MyMacroB d
\MyMacroC
```

a b c d 4

```
macro:\scratchcounter = 1
macro:\advance \scratchcounter by 1
macro:\the \scratchcounter
```

Of course this is a rather useless example but it serves its purpose: you'd better be aware what gets expanded immediately in an `\edef`. In most cases you only need to worry about `\the` and embedded macros (and then of course their meanings).

You can also store tokens in a so-called token register. Here we use a predefined scratch register:

```
\def\MyMacroA{ and }
\def\MyMacroB{1\MyMacroA 2}
\scratchtoks {\MyMacroA}
```

The content of `\scratchtoks` is: "`\MyMacroA`", so no expansion has happened here.

```
\def\MyMacroA{ and }
\def\MyMacroB{1\MyMacroA 2}
\scratchtoks \expandafter {\MyMacroA}
```

Now the content of `\scratchtoks` is: " `and` ", so this time expansion has happened.

```
\def\MyMacroA{ and }
\def\MyMacroB{1\MyMacroA 2}
\scratchtoks \expandafter {\MyMacroB}
```

Indeed the macro gets expanded but only one level: "`1\MyMacroA 2`". Compare this with:

```
\def\MyMacroA{ and }
\edef\MyMacroB{1\MyMacroA 2}
\scratchtoks \expandafter {\MyMacroB}
```

The trick is to expand in two steps with an intermediate `\edef`: "`1 and 2`". Later we will see that other engines provide some more expansion tricks. The only way to get some grip on expansion is to just play with it.

The \expandafter primitive expands the token (which can be a macro) standing after the next next one and then injects its meaning into the stream. So:

```
\expandafter \MyMacroA \MyMacroB
```

works okay. In a normal document you will never need this kind of hackery: it only happens in a bit more complex macros. Here is an example:

```
\scratchcounter 1
\bgroup
\advance\scratchcounter 1
\egroup
\the\scratchcounter

\scratchcounter 1
\bgroup
\advance\scratchcounter 1
\expandafter
\egroup
\the\scratchcounter
```

The first one gives 1, while the second gives 2.

4.3 ε-TEX primitives

In this engine a couple of extensions were added and later on pdfTEX added some more. We only discuss a few that relate to expansion. There is however a pitfall here. Before ε-TEX showed up, ConTEXt already had a few mechanism that also related to expansion and it used some names for macros that clash with those in ε-TEX. This is why we will use the \normal prefix here to indicate the primitive.[2].

```
\def\MyMacroA{a}
\def\MyMacroB{b}
\normalprotected\def\MyMacroC{c}
\edef\MyMacroABC{\MyMacroA\MyMacroB\MyMacroC}
```

These macros have the following meanings:

```
macro:a
macro:b
```

[2] In the meantime we no longer have a low level \protected macro so one can use the primitive

```
protected macro:c
macro:ab\MyMacroC
```

In ConTEXt you will use the \unexpanded prefix instead, because that one did something similar in older versions of ConTEXt. As we were early adopters of ε-TEX, this later became a synonym to the ε-TEX primitive.

```
\def\MyMacroA{a}
\def\MyMacroB{b}
\normalprotected\def\MyMacroC{c}
\normalexpanded{\scratchtoks{\MyMacroA\MyMacroB\MyMacroC}}
```

Here the wrapper around the token register assignment will expand the three macros, unless they are protected, so its content becomes "ab\MyMacroC". This saves either a lot of more complex \expandafter usage or the need to use an intermediate \edef. In ConTEXt the \expanded macro does something simpler but it doesn't expand the first token as this is meant as a wrapper around a command, like:

```
\expanded{\chapter{....}} % a ConTeXt command
```

where we do want to expand the title but not the \chapter command (not that this would happen actually because \chapter is a protected command.)

The counterpart of \normalexpanded is \normalunexpanded, as in:

```
\def\MyMacroA{a}
\def\MyMacroB{b}
\normalprotected\def\MyMacroC{c}
\normalexpanded {\scratchtoks
    {\MyMacroA\normalunexpanded {\MyMacroB}\MyMacroC}}
```

The register now holds "a\MyMacroB\MyMacroC": three tokens, one character token and two macro references.

Tokens can represent characters, primitives, macros or be special entities like starting math mode, beginning a group, assigning a dimension to a register, etc. Although you can never really get back to the original input, you can come pretty close, with:

```
\detokenize{this can $ be anything \bgroup}
```

This (when typeset monospaced) is: this can $ be anything \bgroup. The detokenizer is like \string applied to each token in its argument. Compare this to:

```
\normalexpanded {
```

```
    \normaldetokenize{10pt}
}
```

We get four tokens: `10pt`.

```
\normalexpanded {
    \string 1\string 0\string p\string t
}
```

So that was the same operation: `10pt`, but in both cases there is a subtle thing going on: characters have a catcode which distinguishes them. The parser needs to know what makes up a command name and normally that's only letters. The next snippet shows these catcodes:

```
\normalexpanded {
    \noexpand\the\catcode`\string 1 \noexpand\enspace
    \noexpand\the\catcode`\string 0 \noexpand\enspace
    \noexpand\the\catcode`\string p \noexpand\enspace
    \noexpand\the\catcode`\string t \noexpand
}
```

The result is "`12 12 11 11`": two characters are marked as 'letter' and two fall in the 'other' category.

4.4 LuaTEX primitives

This engine adds a little to the expansion repertoire. First of all it offers a way to extend token lists registers:

```
\def\MyMacroA{a}
\def\MyMacroB{b}
\normalprotected\def\MyMacroC{b}
\scratchtoks{\MyMacroA\MyMacroB}
```

The result is: "`\MyMacroA\MyMacroB`".

```
\toksapp\scratchtoks{\MyMacroA\MyMacroB}
```

We're now at: "`\MyMacroA\MyMacroB\MyMacroA\MyMacroB\MyMacroA\MyMacroB`".

```
\etoksapp\scratchtoks{\MyMacroA\space\MyMacroB\space\MyMacroC}
```

The register has this content: “\MyMacroA\MyMacroB\MyMacroA\MyMacroB a b \MyMacroC a b \MyMacroC”, so the additional context got expanded in the process, except of course the protected macro \MyMacroC.

There is a bunch of these combiners: \toksapp and \tokspre for local appending and prepending, with global companions: \gtoksapp and \gtokspre, as well as expanding variant: \etoksapp, \etokspre, \xtoksapp and \xtokspre.

These are not beforehand more efficient that using intermediate expanded macros or token lists, simply because in the process TEX has to create tokens lists too, but sometimes they're just more convenient to use. In ConTEXt we actually do benefit from these.

4.5 LuaMetaTEX primitives

We already saw that macro's can be defined protected which means that

```
          \def\TestA{A}
\protected \def\TestB{B}
          \edef\TestC{\TestA\TestB}
```

gives this:

```
\TestC : A\TestB
```

One way to get \TestB expanded it to prefix it with \expand:

```
          \def\TestA{A}
\protected \def\TestB{B}
          \edef\TestC{\TestA\TestB}
          \edef\TestD{\TestA\expand\TestB}
```

We now get:

```
\TestC : A\TestB
\TestD : AB
```

There are however cases where one wishes this to happen automatically, but that will also make protected macros expand which will create havoc, like switching fonts.

```
              \def\TestA{A}
\protected    \def\TestB{B}
\semiprotected \def\TestC{C}
             \edef\TestD{\TestA\TestB\TestC}
```

```
\edef\TestE{\normalexpanded{\TestA\TestB\TestC}}
\edef\TestF{\semiexpanded  {\TestA\TestB\TestC}}
```

This time `\TestC` looses its protection:

```
\TestA : A
\TestB : B
\TestC : C
\TestD : A\TestB \TestC
\TestE : A\TestB \TestC
\TestF : A\TestB C
```

Actually adding `\fullyexpanded` would be trivial but it makes not much sense to add the overhead (at least not now). This feature is experimental anyway so it might go away when I see no real advantage from it.

When you store something in a macro or token register you always need to keep an eye on category codes. A dollar in the input is normally treated as math shift, a hash indicates a macro parameter or preamble entry. Characters like 'A' are letters but '[' and ']' are tagged as 'other'. The TEX scanner acts according to these codes. If you ever find yourself in a situation that changing catcodes is no option or cumbersome, you can do this:

```
\edef\TestOA{\expandtoken\othercatcode `A}
\edef\TestLA{\expandtoken\lettercatcode`A}
```

In both cases the meaning is `A` but in the first case it's not a letter but a character flagged as 'other'.

A whole new category of commands has to do with so called local control. When TEX scans and interprets the input, a process takes place that is called tokenizing: (sequences of) characters get a symbolic representation and travel through the system as tokens. Often they immediately get interpreted and are then discarded. But when for instance you define a macro they end up as a linked list of tokens in the macro body. We already saw that expansion plays a role. In most cases, unless TEX is collecting tokens, the main action is dealt with in the so-called main loop. Something gets picked up from the input but can also be pushed back, for instance because of some lookahead that didn't result in an action. Quite some time is spent in pushing and popping from the so-called input stack.

When we are in Lua, we can pipe back into the engine but all is collected till we're back in TEX where the collected result is pushed into the input. Because TEX is a mix of programming and action there basically is only that main loop. There is no real way

to start a sub run in Lua and do all kind of things independent of the current one. This makes sense when you consider the mix: it would get too confusing.

However, in LuaTEX and even better in LuaMetaTEX, we can enter a sort of local state and this is called 'local control'. When we are in local control a new main loop is entered and the current state is temporarily forgotten: we can for instance expand where one level up expansion was not done. It sounds complicated an indeed it is complicated so examples have to clarify it.

```
1 \setbox0\hbox to 10pt{2} \count0=3 \the\count0 \multiply\count0 by 4
```

This snippet of code is not that useful but illustrates what we're dealing with:

- The 1 gets typeset. So, characters like that are seen as text.
- The `\setbox` primitive triggers picking up a register number, then goes on scanning for a box specification and that itself will typeset a sequence of whatever until the group ends.
- The `count` primitive triggers scanning for a register number (or reference) and then scans for a number; the equal sign is optional.
- The `the` primitive injects some value into the current input stream and it does so by entering a new input level.
- The `multiply` primitive picks up a register specification and multiplies that by the next scanned number. The `by` is optional.

We now look at this snippet again but with an expansion context:

```
\def \TestA{1 \setbox0\hbox{2} \count0=3 \the\count0}

\edef\TestB{1 \setbox0\hbox{2} \count0=3 \the\count0}
```

These two macros have a slightly different body. Make sure you see the difference before reading on.

control sequence: TestA						
593713	12	49	other char	1	U+00031	
593005	10	32	spacer			
592630	129	0	set box			setbox
299124	12	48	other char	0	U+00030	
593731	31	14	make box			hbox
9501	1	123	left brace			

593669	12	50	other char	2	U+00032	
593002	2	125	right brace			
479029	10	32	spacer			
584387	122	1	register			count
592643	12	48	other char	0	U+00030	
589048	12	61	other char	=	U+0003D	
593391	12	51	other char	3	U+00033	
593617	10	32	spacer			
593175	140	0	the			the
332829	122	1	register			count
593191	12	48	other char	0	U+00030	

control sequence: TestB

594046	12	49	other char	1	U+00031	
593248	10	32	spacer			
593875	129	0	set box			setbox
594048	12	48	other char	0	U+00030	
593026	31	14	make box			hbox
593330	1	123	left brace			
593605	12	50	other char	2	U+00032	
593579	2	125	right brace			
592849	10	32	spacer			
479069	122	1	register			count
592770	12	48	other char	0	U+00030	
479082	12	61	other char	=	U+0003D	
592991	12	51	other char	3	U+00033	
593569	10	32	spacer			
594101	12	49	other char	1	U+00031	

We now introduce a new primitive `\localcontrolled`:

```
\edef\TestB{1 \setbox0\hbox{2} \count0=3 \the\count0}

\edef\TestC{1 \setbox0\hbox{2} \localcontrolled{\count0=3} \the\count0}
```

Again, watch the subtle differences:

control sequence: TestB

593204	12	49	other char	1	U+00031	
592601	10	32	spacer			
479034	129	0	set box			setbox

594071	12	48	other char	0	U+00030	
593014	31	14	make box			hbox
592461	1	123	left brace			
592473	12	50	other char	2	U+00032	
593089	2	125	right brace			
594039	10	32	spacer			
594079	122	1	register			count
594057	12	48	other char	0	U+00030	
593282	12	61	other char	=	U+0003D	
593170	12	51	other char	3	U+00033	
593249	10	32	spacer			
593064	12	49	other char	1	U+00031	

control sequence: TestC

592517	12	49	other char	1	U+00031	
594046	10	32	spacer			
593248	129	0	set box			setbox
593875	12	48	other char	0	U+00030	
594048	31	14	make box			hbox
593026	1	123	left brace			
593330	12	50	other char	2	U+00032	
593605	2	125	right brace			
593579	10	32	spacer			
593569	10	32	spacer			
591676	12	51	other char	3	U+00033	

Another example:

```
\edef\TestB{1 \setbox0\hbox{2} \count0=3 \the\count0}

\edef\TestD{\localcontrolled{1 \setbox0\hbox{2} \count0=3 \the\count0}}
```

1 3 ← Watch how the results end up here!

control sequence: TestB

593226	12	49	other char	1	U+00031	
332844	10	32	spacer			
593272	129	0	set box			setbox
594118	12	48	other char	0	U+00030	
479102	31	14	make box			hbox
578587	1	123	left brace			

594077	12	50	other char	2	U+00032	
594076	2	125	right brace			
592765	10	32	spacer			
592831	122	1	register			count
593598	12	48	other char	0	U+00030	
71331	12	61	other char	=	U+0003D	
593815	12	51	other char	3	U+00033	
593326	10	32	spacer			
593407	12	51	other char	3	U+00033	

control sequence: TestD

<no tokens>

We can use this mechanism to define so called fully expandable macros:

```
\def\WidthOf#1%
  {\beginlocalcontrol
   \setbox0\hbox{#1}%
   \endlocalcontrol
   \wd0 }

\scratchdimen\WidthOf{The Rite Of Spring}

\the\scratchdimen
```

104.72021pt

When you want to add some grouping, it quickly can become less pretty:

```
\def\WidthOf#1%
  {\dimexpr
      \beginlocalcontrol
        \begingroup
          \setbox0\hbox{#1}%
          \expandafter
        \endgroup
      \expandafter
      \endlocalcontrol
      \the\wd0
   \relax}

\scratchdimen\WidthOf{The Rite Of Spring}
```

```
\the\scratchdimen
```

104.72021pt

A single token alternative is available too and its usage is like this:

```
 \def\TestA{\scratchcounter=100 }
\edef\TestB{\localcontrol\TestA \the\scratchcounter}
\edef\TestC{\localcontrolled{\TestA} \the\scratchcounter}
```

The content of `\TestB` is '`100`' and of course the `\TestC` macro gives '` 100`'.

We now move to the Lua end. Right from the start the way to get something into TEX from Lua has been the print functions. But we can also go local (immediate). There are several methods:

- via a set token register
- via a defined macro
- via a string

Among the things to keep in mind are catcodes, scope and expansion (especially in when the result itself ends up in macros). We start with an example where we go via a token register:

```
\toks0={\setbox0\hbox{The Rite Of Spring}}
\toks2={\setbox0\hbox{The Rite Of Spring!}}

\startluacode
tex.runlocal(0) context("[1: %p]",tex.box[0].width)
tex.runlocal(2) context("[2: %p]",tex.box[0].width)
\stopluacode
```

[1: 104.72021pt][2: 109.14062pt]

We can also use a macro:

```
\def\TestA{\setbox0\hbox{The Rite Of Spring}}
\def\TestB{\setbox0\hbox{The Rite Of Spring!}}

\startluacode
tex.runlocal("TestA") context("[3: %p]",tex.box[0].width)
tex.runlocal("TestB") context("[4: %p]",tex.box[0].width)
\stopluacode
```

[3: 104.72021pt][4: 109.14062pt]

A third variant is more direct and uses a (Lua) string:

```
\startluacode
tex.runstring([[\setbox0\hbox{The Rite Of Spring}]])

context("[5: %p]",tex.box[0].width)

tex.runstring([[\setbox0\hbox{The Rite Of Spring!}]])

context("[6: %p]",tex.box[0].width)
\stopluacode
```

[5: 104.72021pt][6: 109.14062pt]

A bit more high level:

```
context.runstring([[\setbox0\hbox{(Here \bf 1.2345)}]])
context.runstring([[\setbox0\hbox{(Here \bf   %.3f)}]],1.2345)
```

Before we had `runstring` this was the way to do it when staying in Lua was needed:

```
\startluacode
token.setmacro("TestX",[[\setbox0\hbox{The Rite Of Spring}]])
tex.runlocal("TestX")
context("[7: %p]",tex.box[0].width)
\stopluacode
```

[7: 104.72021pt]

```
\startluacode
tex.scantoks(0,tex.ctxcatcodes,[[\setbox0\hbox{The Rite Of Spring!}]])
tex.runlocal(0)
context("[8: %p]",tex.box[0].width)
\stopluacode
```

[8: 109.14062pt]

The order of flushing matters because as soon as something is not stored in a token list or macro body, TEX will typeset it. And as said, a lot of this relates to pushing stuff into the input which is stacked. Compare:

```
\startluacode
```

```
context("[HERE 1]")
context("[HERE 2]")
\stopluacode
```

[HERE 1][HERE 2]

with this:

```
\startluacode
tex.pushlocal() context("[HERE 1]") tex.poplocal()
tex.pushlocal() context("[HERE 2]") tex.poplocal()
\stopluacode
```

[HERE 1][HERE 2]

You can expand a macro at the Lua end with `token.expandmacro` which has a peculiar interface. The first argument has to be a string (the name of a macro) or a userdata (a valid macro token). This macro can be fed with parameters by passing more arguments:

string	serialized to tokens
true	wrap the next string in curly braces
table	each entry will become an argument wrapped in braces
token	inject the token directly
number	change control to the given catcode table

There are more scanner related primitives, like the ε-TEX primitive `\detokenize`:

```
[\detokenize {test \relax}]
```

This gives: `[test \relax ]` . In LuaMetaTEX we also have complementary primitive(s):

```
[\tokenized   catcodetable \vrbcatcodes {test {\bf test} test}]
[\tokenized                             {test {\bf test} test}]
[\retokenized              \vrbcatcodes {test {\bf test} test}]
```

The `\tokenized` takes an optional keyword and the examples above give: `[test {\bf test} test] [test test test] [test {\bf test} test]` . The LuaTEX primitive `\scantextokens` which is a variant of ε-TEX's `\scantokens` operates under the current catcode regime (the last one honors `\everyeof`). The difference with `\tokenized` is that this one first serializes the given token list (just like `\detokenize`).[3]

[3] The `\scan *tokens` primitives now share the same helpers as Lua, but they should behave the same as in LuaTEX.

With \retokenized the catcode table index is mandatory (it saves a bit of scanning and is easier on intermixed \expandafter usage. There often are several ways to accomplish the same:

```
\def\MyTitle{test {\bf test} test}
\detokenize                \expandafter{\MyTitle}: 0.46\crlf
\meaningless                            \MyTitle : 0.47\crlf
\retokenized               \notcatcodes{\MyTitle}: 0.87\crlf
\tokenized   catcodetable  \notcatcodes{\MyTitle}: 0.93\crlf
```

test {\bf test} test: 0.46
test {\bf test} test: 0.47
test {\bf test} test: 0.87
test {\bf test} test: 0.93

Here the numbers show the relative performance of these methods. The \detokenize and \meaningless win because they already know that a verbose serialization is needed. The last two first serialize and then reinterpret the resulting token list using the given catcode regime. The last one is slowest because it has to scan the keyword.

There is however a pitfall here:

```
\def\MyText {test}
\def\MyTitle{test \MyText\space test}
\detokenize                \expandafter{\MyTitle}\crlf
\meaningless                            \MyTitle \crlf
\retokenized               \notcatcodes{\MyTitle}\crlf
\tokenized   catcodetable  \notcatcodes{\MyTitle}\crlf
```

The outcome is different now because we have an expandable embedded macro call. The fact that we expand in the last two primitives is also the reason why they are 'slower'.

test \MyText \space test
test \MyText \space test
test test test
test test test

To complete this picture, we show a variant than combines much of what has been introduced in this section:

```
\semiprotected\def\MyTextA {test}
\def\MyTextB {test}
\def\MyTitle{test \MyTextA\space \MyTextB\space test}
\detokenize                \expandafter{\MyTitle}\crlf
\meaningless                            \MyTitle \crlf
\retokenized               \notcatcodes{\MyTitle}\crlf
\retokenized               \notcatcodes{\semiexpanded{\MyTitle}}\crlf
\tokenized   catcodetable \notcatcodes{\MyTitle}\crlf
\tokenized   catcodetable \notcatcodes{\semiexpanded{\MyTitle}}
```

This time compare the last four lines:

test \MyTextA \space \MyTextB \space test
test \MyTextA \space \MyTextB \space test
test \MyTextA test test
test test test test
test \MyTextA test test
test test test test

Of course the question remains to what extend we need this and eventually will apply in ConTEXt. The `\detokenize` is used already. History shows that eventually there is a use for everything and given the way LuaMetaTEX is structured it was not that hard to provide the alternatives without sacrificing performance or bloating the source.

4.6 Dirty tricks

When I was updating this manual Hans vd Meer and I had some discussions about expansion and tokenization related issues when combining of xml processing with TEX macros where he did some manipulations in Lua. In these mixed cases you can run into catcode related problems because in xml you want for instance a # to be a hash mark (other character) and not an parameter identifier. Normally this is handled well in ConTEXt but of course there are complex cases where you need to adapt.

Say that you want to compare two strings (officially we should say token lists) with mixed catcodes. Let's also assume that you want to use the normal `\if` construct (which was part of the discussion). We start with defining a test set. The reason that we present this example here is that we use commands discussed in previous sections:

```
              \def\abc{abc}
\semiprotected \def\xyz{xyz}
              \edef\pqr{\expandtoken\notcatcodes`p%
```

```
                    \expandtoken\notcatcodes`q%
                    \expandtoken\notcatcodes`r}

1: \ifcondition\similartokens{abc} {def}YES\else NOP\fi (NOP) \quad
2: \ifcondition\similartokens{abc}{\abc}YES\else NOP\fi (YES)

3: \ifcondition\similartokens{xyz} {pqr}YES\else NOP\fi (NOP) \quad
4: \ifcondition\similartokens{xyz}{\xyz}YES\else NOP\fi (YES)

5: \ifcondition\similartokens{pqr} {pqr}YES\else NOP\fi (YES) \quad
6: \ifcondition\similartokens{pqr}{\pqr}YES\else NOP\fi (YES)
```

So, we have a mix of expandable and semi expandable macros, and also a mix of catcodes. A naive approach would be:

```
\permanent\protected\def\similartokens#1#2%
  {\iftok{#1}{#2}}
```

but that will fail on some cases:

```
1: NOP(NOP)   2: YES(YES)
3: NOP(NOP)   4: NOP(YES)
5: YES(YES)   6: NOP(YES)
```

So how about:

```
\permanent\protected\def\similartokens#1#2%
  {\iftok{\detokenize{#1}}{\detokenize{#2}}}
```

That one is even worse:

```
1: NOP(NOP)   2: NOP(YES)
3: NOP(NOP)   4: NOP(YES)
5: YES(YES)   6: NOP(YES)
```

We need to expand so we end up with this:

```
\permanent\protected\def\similartokens#1#2%
  {\normalexpanded{\noexpand\iftok
     {\noexpand\detokenize{#1}}
     {\noexpand\detokenize{#2}}}}
```

Better:

```
1: NOP(NOP)   2: YES(YES)
```

```
3: NOP(NOP)   4: NOP(YES)
5: YES(YES)   6: YES(YES)
```

But that will still not deal with the mildly protected macro so in the end we have:

```
\permanent\protected\def\similartokens#1#2%
  {\semiexpanded{\noexpand\iftok
     {\noexpand\detokenize{#1}}
     {\noexpand\detokenize{#2}}}}
```

Now we're good:

```
1: NOP(NOP)   2: YES(YES)
3: NOP(NOP)   4: YES(YES)
5: YES(YES)   6: YES(YES)
```

Finally we wrap this one in the usual `\doifelse...` macro:

```
\permanent\protected\def\doifelsesimilartokens#1#2%
  {\ifcondition\similartokens{#1}{#2}%
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

so that we can do:

```
\doifelsesimilartokens{pqr}{\pqr}{YES}{NOP}
```

A companion macro of this is `\wipetoken` but for that one you need to look into the source.

4.6 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

5 Registers

low level TEX

registers

Contents

5.1 Preamble

Registers are sets of variables that are accessed by index and a such resemble registers in a processing unit. You can store a quantity in a register, retrieve it, and also manipulate it.

There is hardly any need to use them in ConTEXt so we keep it simple.

5.2 TEX primitives

There are several categories:

- Integers (int): in order to be portable (at the time it surfaced) there are only integers and no floats. The only place where TEX uses floats internally is when glue gets effective which happens in the backend.
- Dimensions (dimen): internally these are just integers but when they are entered they are sliced into two parts so that we have a fractional part. The internal representation is called a scaled point.
- Glue (skip): these are dimensions with a few additional properties: stretch and shrink. Being a compound entity they are stored differently and thereby a bit less efficient than numbers and dimensions.
- Math glue (muskip): this is the same as glue but with a unit that adapts to the current math style properties. It's best to think about them as being relative measures.
- Token lists (toks): these contain a list of tokens coming from the input or coming from a place where they already have been converted.

The original TEX engine had 256 entries per set. The first ten of each set are normally reserved for scratch purposes: the even ones for local use, and the odd ones for global

usage. On top of that macro packages can reserve some for its own use. It was quite easy to reach the maximum but there were tricks around that. This limitation is no longer present in the variants in use today.

Let's set a few dimension registers:

```
\dimen 0 = 10 pt
\dimen2=10pt
\dimen4 10pt
\scratchdimen 10pt
```

We can serialize them with:

```
\the    \dimen0
\number \dimen2
\meaning\dimen4
\meaning\scratchdimen
```

The results of these operations are:

```
10.0pt
655360
\dimen4
constant dimension 10.0pt
```

The last two is not really useful but it is what you see when tracing options are set. Here `\scratchdimen` is a shortcut for a register. This is *not* a macro but a defined register. The low level `\dimendef` is used for this but in a macro package you should not use that one but the higher level `\newdimen` macro that uses it.

```
\newdimen\MyDimenA
\def     \MyDimenB{\dimen999}
\dimendef\MyDimenC998

\meaning\MyDimenA
\meaning\MyDimenB
\meaning\MyDimenC
```

Watch the difference:

```
\dimen269
macro:\dimen 999
\dimen998
```

The first definition uses a yet free register so you won't get a clash. The second one is just a shortcut using a macro and the third one too but again direct shortcut. Try to imagine how the second line gets interpreted:

```
\MyDimenA10pt \MyDimenA10.5pt
\MyDimenB10pt \MyDimenB10.5pt
\MyDimenC10pt \MyDimenC10.5pt
```

Also try to imagine what messing around with `\MyDimenC` will do when we also have defined a few hundred extra dimensions with `\newdimen`.

In the case of dimensions the `\number` primitive will make the register serialize as scaled points without unit `sp`.

Next we see some of the other registers being assigned:

```
\count  0 = 100
\skip   0 = 10pt plus 3pt minus 2pt
\skip   0 = 10pt plus 1fill
\muskip 0 = 10mu plus 3mu minus 2mu
\muskip 0 = 10mu minus 1 fil
\toks   0 = {hundred}
```

When a number is expected, you can use for instance this:

```
\scratchcounter\scratchcounterone
```

Here we use a few predefined scratch registers. You can also do this:

```
\scratchcounter\numexpr\scratchcounterone+\scratchcountertwo\relax
```

There are some quantities that also qualify as number:

```
\chardef\MyChar=123 % refers to character 123 (if present)
\scratchcounter\MyChar
```

In the past using `\chardef` was a way to get around the limited number of registers, but it still had (in traditional TEX) a limitation: you could not go beyond 255. The `\mathchardef` could fo higher as it also encodes a family number and class. This limitation has been lifted in LuaTEX.

A character itself can also be interpreted as number, in which case it has to be prefixed with a reverse quote: `, so:

```
\scratchcounter\numexpr`0+5\relax
\char\scratchcounter
```

produces “5” because the `0 expands into the (ascii and utf8) slot 48 which represents the character zero. In this case the next makes more sense:

```
\char\numexpr`0+5\relax
```

If you want to know more about all these quantities, “TEX By Topic” provides a good summary of what TEX has to offer, and there is no need to repeat it here.

5.3 ε-TEX primitives

Apart from the ability to use expressions, the contribution to registers that ε-TEX brought was that suddenly we could use upto 65K of them, which is more than enough. The extra registers were not as efficient as the first 256 because they were stored in the hash table, but that was not really a problem. In Omega and later LuaTEX regular arrays were used, at the cost of more memory which in the meantime has become cheap. As ConTEXt moved to ε-TEX rather early its users never had to worry about it.

5.4 LuaTEX primitives

The LuaTEX engine introduced attributes. These are numeric properties that are bound to the nodes that are the result of typesetting operations. They are basically like integer registers but when set their values get bound and when unset they are kind of invisible.

- Attribute (attribute): a numeric property that when set becomes part of the current attribute list that gets assigned to nodes.

Attributes can be used to communicate properties to Lua callbacks. There are several functions available for setting them and querying them.

```
\attribute999 = 123
```

Using attributes this way is dangerous (of course I can only speak for ConTEXt) because an attribute value might trigger some action in a callback that gives unwanted side effects. For convenience ConTEXt provides:

```
\newattribute\MyAttribute
```

Which currently defines `\MyAttribute` as `constant integer 1026` and is meant to be used as:[4]

```
\attribute\MyAttribute = 123
```

Just be aware that defining attributes can have an impact on performance. As you cannot access them at the TEX end you seldom need them. If you do you can better use the proper more high level definers (not discussed here).

5.5 LuaMetaTEX primitives

The fact that scanning stops at a non-number or `\relax` can be sort of unpredictable which is why in LuaMetaTEX we also support the following variant:

```
\scratchdimen\dimexpr  10pt + 3pt \relax
\scratchdimen\dimexpr {10pt + 3pt}
```

At the cost of one more token braces can be used as boundaries instead of the single `\relax` boundary.

An important property of registers is that they can be accessed by a number. This has big consequences for the implementation: they are part of the big memory store and consume dedicated ranges. If we had only named access TEX's memory layout could be a bit leaner. In principle we could make the number of registers smaller because any limit on the amount at some point can be an obstacle. It is for that reason that we also have name-only variants:

```
\dimensiondef \MyDimenA   12pt
\integerdef   \MyIntegerA 12
\gluespecdef  \MyGlueA    12pt + 3pt minus 4pt
\mugluespecdef\MyMuA      12mu + 3mu minus 4mu
```

These are as efficient but not accessible by number but they behave like registers which means that you (can) use `\the`, `\advance`, `\multiply` and `\divide` with them.[5] In case you wonder why there is no alternative for `\toksdef`, there actually are multiple: they are called macros.

todo: expressions

[4] The low level `\attributedef` command is rather useless in the perspective of ConTEXt.

[5] There are also the slightly more efficient `\advanceby`, `\multiplyby` and `\divideby` that don't check for the `by` keyword.

5.6 Units

The LuaMetaTEX engine supports the following units. The first batch is constant with hard coded fine tuned values. The second set is related to the current font. The last group is kind of special, the `es` is a replacement for the `in` and has a little sister in `ts`. The `dk` is dedicated to the master and makes a nice offset for so called TEX pages that we use for demos.

pt	1.0	point
bp	1.00374	big point (aka postscript point)
in	72.26999	inch
cm	28.45274	centimeter
mm	2.84526	milimeter
dd	1.07	didot
cc	12.8401	cicero
pc	12.0	pica
sp	0.00002	scaled points
px	0.00002	pixel
ex	5.70947	ex height
em	11.0	em width
mu	1.0	math unit
ts	7.11317	tove
es	71.13177	edith
eu	71.13177	european unit
dk	6.43985	knuth

The `fi[lll]` unit is not really a unit but a multiplier for infinite stretch and shrink; original TEX doesn't have the simple `fi`.

In addition to these we can have many more. In principle a user can define additional ones but there's always a danger of clashing. For users we reserve the units starting with an `u`. Here is how you define your own, we show three variants:

```
\newdimension \FooA   \FooA 1.23pt
\newdimen     \FooB   \FooB 12.3pt
\protected\def\FooC   {\the\dimexpr\FooA +\FooB\relax}

\pushoverloadmode % just in case
    \newuserunit\FooA ua
    \newuserunit\FooB ub
    \newuserunit\FooC uc
```

```
\popoverloadmode
```

And this is how they show up:

2.45999pt 24.6pt 27.06pt

with

```
\the\dimexpr 2 ua \relax\quad
\the\dimexpr 2 ub \relax\quad
\the\dimexpr 2 uc \relax
```

The following additional units are predefined (reserved). The values are in points and some depend on the current layout and document font.

```
pi    3.14159  π for Mikael
ft  867.23999  foot for Alan
fs       11.0  (global body) font size
tw  483.69687  (layout) text width
th  645.88272  (layout) text height
hs  483.69687  (current) hsize
vs  645.88272  (current) vsize
cd        0.0  (when set) column distance
cw  483.69687  (when set) column width
cx  236.34843  combination cell width
uu   28.45274  user unit (MetaFun)
fw        0.0  framed width
fh        0.0  framed height
fo        0.0  framed offset
lw        0.4  line width
sh   11.51031  strut height
sd    4.47621  strut depth
st   15.98653  strut total
ch    6.99854  width of zero (css)
fa    8.35742  font ascender
fd    1.71338  font descender
fc    8.01904  font cap height
```

Here is an example of usage:

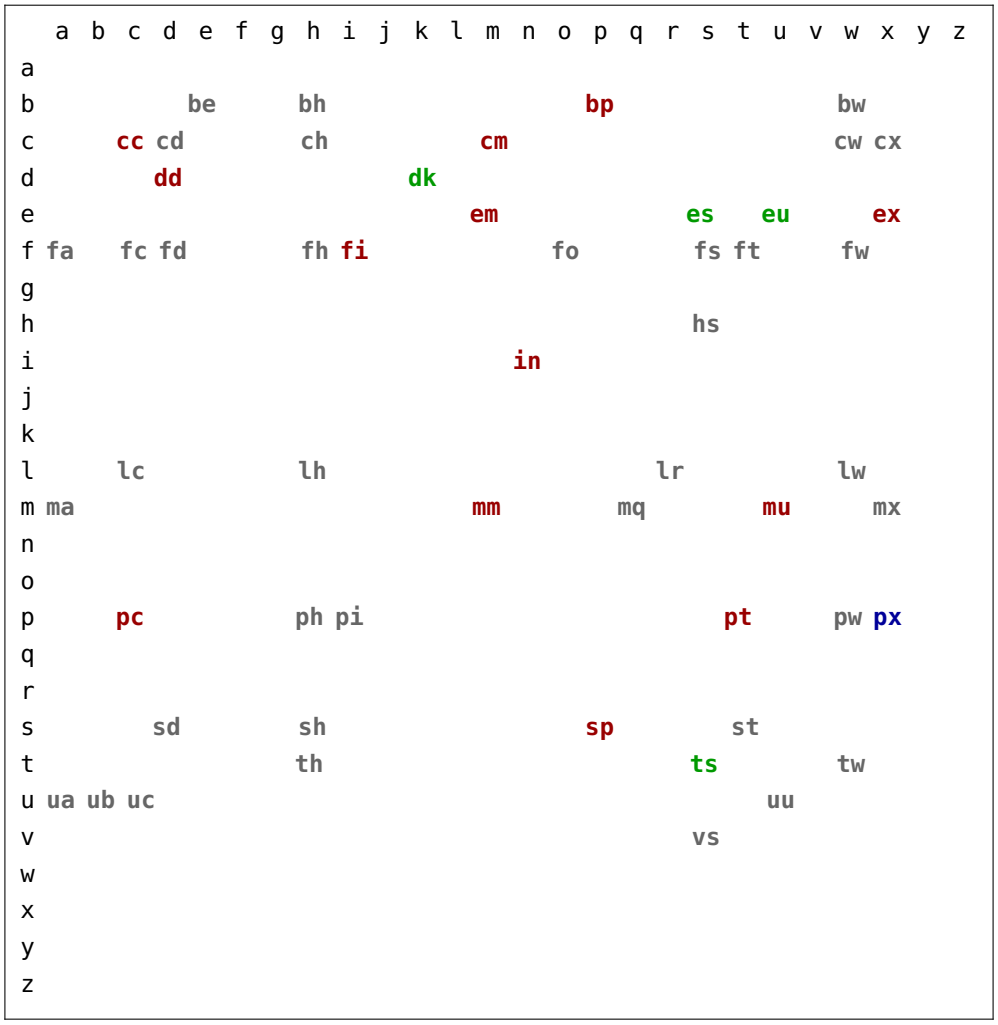
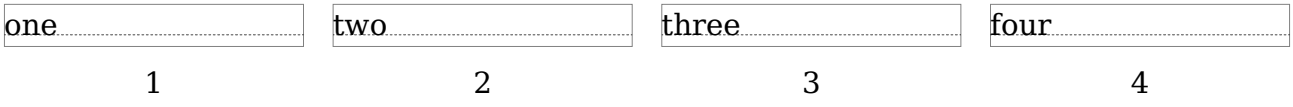


Figure 5.1 A map of available units

```
\startcombination[nx=4,ny=1]
    {\ruledhbox to 1cx{\strut one}}   {1}
    {\ruledhbox to 1cx{\strut two}}   {2}
    {\ruledhbox to 1cx{\strut three}} {3}
    {\ruledhbox to 1cx{\strut four}}  {4}
\stopcombination
```



The uu can be set by users using the `\uunit` dimension variable. The default valu sis 1cm. Its current value is also known at the MetaPost end, as demonstrated in figure 5.2.

```
\startcombination[nx=2,ny=1]
```

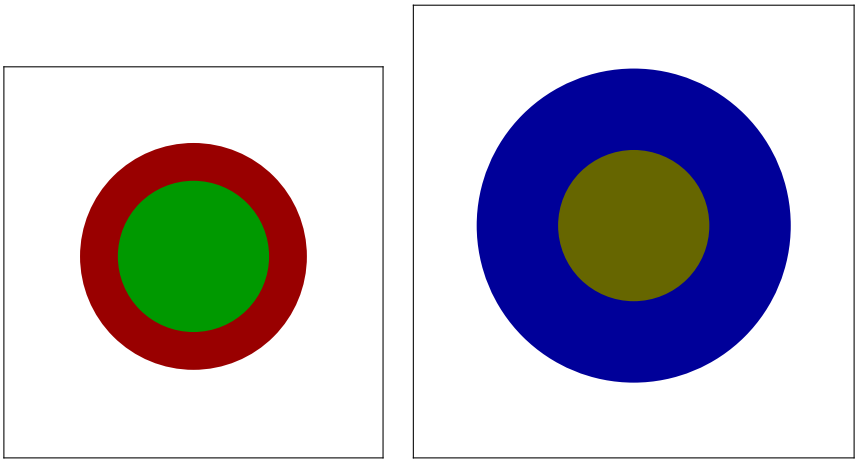
```
    \startcontent
        \uunit=1cm
        \framed[offset=1uu]
            \bgroup
                \startMPcode
                    fill fullcircle scaled 3uu withcolor "darkred"   ;
                    fill fullcircle scaled 2cm withcolor "darkgreen" ;
                \stopMPcode
            \egroup
    \stopcontent
    \startcaption
        \type {\uunit = 1cm}
    \stopcaption
    \startcontent
        \uunit=1cx
        \framed[offset=.1uu]
            \bgroup
                \startMPcode
                    fill fullcircle scaled .5uu withcolor "darkblue"   ;
                    fill fullcircle scaled  2cm withcolor "darkyellow" ;
                \stopMPcode
            \egroup
    \stopcontent
    \startcaption
        \type {\uunit = 1cx}
    \stopcaption
\stopcombination
```

There is one catch here. If you use your own uu as numeric, you might need this:

```
save uu ; numeric uu ; uu := 1cm ;
```

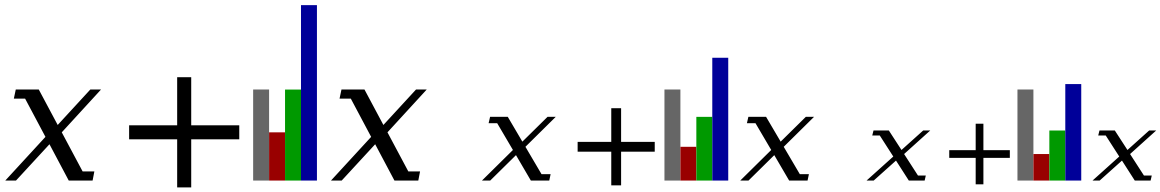
That is: make sure the meaning is restored afterwards and explicitly declare the variable. But this is good practice anyway when you generate multiple graphics using the same MetaPost instance.

There a few units not mentioned yet and those concern math, where we need to adapt to the current style.



\uunit = 1cm \uunit = 1cx

Figure 5.2 Shared user units in TEX and MetaFun.



text style script style script script style

The bars show `1ex`, `1ma` (axis), `1mx` (ex-height) and `1mq` (em-width or quad). The last three adapt themselves to the style. Often the `mx` makes more sense than `ex`.

5.6 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

6 Macros

low level TEX

macros

Contents

6.1 Preamble

This chapter overlaps with other chapters but brings together some extensions to the macro definition and expansion parts. As these mechanisms were stepwise extended, the other chapters describe intermediate steps in the development.

Now, in spite of the extensions discussed here the main ides is still that we have TEX act like before. We keep the charm of the macro language but these additions make for easier definitions, but (at least initially) none that could not be done before using more code.

6.2 Definitions

A macro definition normally looks like like this:[6]

```
\def\macro#1#2%
  {\dontleavehmode\hbox to 6em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

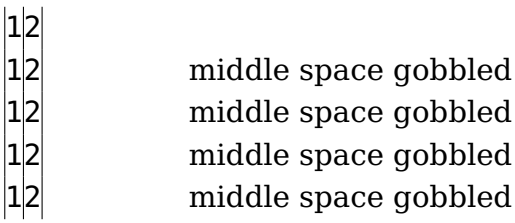
Such a macro can be used as:

```
\macro {1}{2}
\macro {1} {2}  middle space gobbled
\macro 1 {2}    middle space gobbled
\macro {1} 2    middle space gobbled
```

[6] The `\dontleavehmode` command make the examples stay on one line.

```
\macro 1 2      middle space gobbled
```

We show the result with some comments about how spaces are handled:



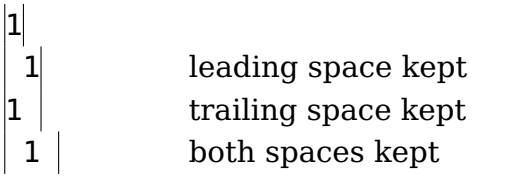
A definition with delimited parameters looks like this:

```
\def\macro[#1]%
  {\dontleavehmode\hbox to 6em{\vl\type{#1}\vl\hss}}
```

When we use this we get:

```
\macro [1]
\macro [ 1]    leading space kept
\macro [1 ]    trailing space kept
\macro [ 1 ]   both spaces kept
```

Again, watch the handling of spaces:



Just for the record we show a combination:

```
\def\macro[#1]#2%
  {\dontleavehmode\hbox to 6em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

With this:

```
\macro [1]{2}
\macro [1] {2}
\macro [1] 2
```

we can again see the spaces go away:

12

A definition with two separately delimited parameters is given next:

```
\def\macro[#1#2]%
  {\dontleavehmode\hbox to 6em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

When used:

```
\macro [12]
\macro [ 12]     leading space gobbled
\macro [12 ]     trailing space kept
\macro [ 12 ]    leading space gobbled, trailing space kept
\macro [1 2]     middle space kept
\macro [ 1 2 ]   leading space gobbled, middle and trailing space kept
```

We get ourselves:

12
12 leading space gobbled
12 trailing space kept
12 leading space gobbled, trailing space kept
1 2 middle space kept
1 2 leading space gobbled, middle and trailing space kept

These examples demonstrate that the engine does some magic with spaces before (and therefore also between multiple) parameters.

We will now go a bit beyond what traditional TEX engines do and enter the domain of LuaMetaTEX specific parameter specifiers. We start with one that deals with this hard coded space behavior:

```
\def\macro[#^#^]%
  {\dontleavehmode\hbox to 6em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

The #^ specifier will count the parameter, so here we expect again two arguments but the space is kept when parsing for them.

```
\macro [12]
\macro [ 12]
\macro [12 ]
\macro [ 12 ]
\macro [1 2]
```

```
\macro [ 1 2 ]
```

Now keep in mind that we could deal well with all kind of parameter handling in ConTEXt for decades, so this is not really something we missed, but it complements the to be discussed other ones and it makes sense to have that level of control. Also, availability triggers usage. Nevertheless, some day the #^ specifier will come in handy.

```
12
 12
12
 12
1 2
 1 2
```

We now come back to an earlier example:

```
\def\macro[#1]%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\hss}}
```

When we use this we see that the braces in the second call are removed:

```
\macro [1]
\macro [{1}]
```

```
1  1
```

This can be prohibited by the #+ specifier, as in:

```
\def\macro[#+]%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\hss}}
```

As we see, the braces are kept:

```
\macro [1]
\macro [{1}]
```

Again, we could easily get around that (for sure intended) side effect but it just makes nicer code when we have a feature like this.

```
1  {1}
```

Sometimes you want to grab an argument but are not interested in the results. For this we have two specifiers: one that just ignores the argument, and another one that keeps counting but discards it, i.e. the related parameter is empty.

```
\def\macro[#1][#0][#3][#-][#4]%
  {\dontleavehmode\hbox spread 1em
     {\vl\type{#1}\vl\type{#2}\vl\type{#3}\vl\type{#4}\vl\hss}}
```

The second argument is empty and the fourth argument is simply ignored which is why we need #4 for the fifth entry.

```
\macro [1][2][3][4][5]
```

Here is proof that it works:

|1|3|5|

The reasoning behind dropping arguments is that for some cases we get around the nine argument limitation, but more important is that we don't construct token lists that are not used, which is more memory (and maybe even cpu cache) friendly.

Spaces are always kind of special in TEX, so it will be no surprise that we have another specifier that relates to spaces.

```
\def\macro[#1]#*[#2]%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

This permits usage like the following:

```
\macro [1][2]
\macro [1] [2]
```

|1|2| |1|2|

Without the optional ‘grab spaces’ specifier the second line would possibly throw an error. This because TEX then tries to match `][` so the `] [` in the input is simply added to the first argument and the next occurrence of `][` will be used. That one can be someplace further in your source and if not TEX complains about a premature end of file. But, with the #* option it works out okay (unless of course you don't have that second argument `[2]`.

Now, you might wonder if there is a way to deal with that second delimited argument being optional and of course that can be programmed quite well in traditional macro code. In fact, ConTEXt does that a lot because it is set up as a parameter driven system with optional arguments. That subsystem has been optimized to the max over years and it works quite well and performance wise there is very little to gain. However, as soon as you enable tracing you end up in an avalanche of expansions and that is no fun.

This time the solution is not in some special specifier but in the way a macro gets defined.

```
\tolerant\def\macro[#1]#*[#2]%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

The magic `\tolerant` prefix with delimited arguments and just quits when there is no match. So, this is acceptable:

```
\macro [1][2]
\macro [1] [2]
\macro [1]
\macro
```

|1|2| |1|2| |1| |

We can check how many arguments have been processed with a dedicated conditional:

```
\tolerant\def\macro[#1]#*[#2]%
  {\ifarguments 0\or 1\or 2\or ?\fi: \vl\type{#1}\vl\type{#2}\vl}
```

We use this test:

```
\macro [1][2] \macro [1] [2] \macro [1] \macro
```

The result is: 2: |1|2| 2: |1|2| 1: |1|0: | which is what we expect because we flush inline and there is no change of mode. When the following definition is used in display mode, the leading `n=` can for instance start a new paragraph and when code in `\everypar` you can loose the right number when macros get expanded before the `n` gets injected.

```
\tolerant\def\macro[#1]#*[#2]%
  {n=\ifarguments 0\or 1\or 2\or ?\fi: \vl\type{#1}\vl\type{#2}\vl}
```

In addition to the `\ifarguments` test primitive there is also a related internal counter `\lastarguments` set that you can consult, so the `\ifarguments` is actually just a shortcut for `\ifcase\lastarguments`.

We now continue with the argument specifiers and the next two relate to this optional grabbing. Consider the next definition:

```
\tolerant\def\macro#1#*#2%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

With this test:

```
\macro {1} {2}
\macro {1}
\macro
```

We get:

```
|1|2|   |1|\macro|
```

This is okay because the last `\macro` is a valid (single token) argument. But, we can make the braces mandate:

```
\tolerant\def\macro#=#*#=%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

Here the #= forces a check for braces, so:

```
\macro {1} {2}
\macro {1}
\macro
```

gives this:

```
|1|2|   |1|   |
```

However, we do loose these braces and sometimes you don't want that. Of course when you pass the results downstream to another macro you can always add them, but it was cheap to add a related specifier:

```
\tolerant\def\macro#_#*#_%
  {\dontleavehmode\hbox spread 1em{\vl\type{#1}\vl\type{#2}\vl\hss}}
```

Again, the magic `\tolerant` prefix works will quit scanning when there is no match. So:

```
\macro {1} {2}
\macro {1}
\macro
```

leads to:

```
|{1}|{2}|   |{1}|   |
```

When you're tolerant it can be that you still want to pick up some argument later on. This is why we have a continuation option.

```
\tolerant\def\foo      [#1]#*[#2]#:#3{!#1!#2!#3!}
\tolerant\def\oof[#1]#*[#2]#:(#3)#:#4{!#1!#2!#3!#4!}
\tolerant\def\ofo      [#1]#:(#2)#:#3{!#1!#2!#3!}
```

Hopefully the next example demonstrates how it works:

```
\foo{3} \foo[1]{3} \foo[1][2]{3}
\oof{4} \oof[1]{4} \oof[1][2]{4}
\oof[1][2](3){4} \oof[1](3){4} \oof(3){4}
\ofo{3} \ofo[1]{3}
\ofo[1](2){3} \ofo(2){3}
```

As you can see we can have multiple continuations using the `#:` directive:

```
!!!3! !1!!3! !1!2!3!
!!!!4! !1!!!4! !1!2!!4!
!1!2!3!4! !1!!3!4! !!!3!4!
!!!3! !1!!3!
!1!2!3! !!2!3!
```

The last specifier doesn't work well with the `\ifarguments` state because we no longer know what arguments were skipped. This is why we have another test for arguments. A zero value means that the next token is not a parameter reference, a value of one means that a parameter has been set and a value of two signals an empty parameter. So, it reports the state of the given parameter as a kind if `\ifcase`.

```
\def\foo#1#2{ [\ifparameter#1\or(ONE)\fi\ifparameter#2\or(TWO)\fi] }
```

Of course the test has to be followed by a valid parameter specifier:

```
\foo{1}{2} \foo{1}{} \foo{}{2} \foo{}{}
```

The previous code gives this:

[(ONE)(TWO)] [(ONE)] [(TWO)] []

A combination check `\ifparameters`, again a case, matches the first parameter that has a value set.

We could add plenty of specifiers but we need to keep in ind that we're not talking of an expression scanner. We need to keep performance in mind, so nesting and backtracking are no option. We also have a limited set of useable single characters, but here's one that uses a symbol that we had left:

```
\def\startfoo[#/]#/\stopfoo{ [#1](#2) }
```

The slash directive removes leading and trailing so called spacers as well as tokens that represent a paragraph end:

```
\startfoo [x ] x \stopfoo
\startfoo [ x ] x \stopfoo
\startfoo [ x] x \stopfoo
\startfoo [ x] \par x \par \par \stopfoo
```

So we get this:

[x](x) [x](x) [x](x) [x](x)

The next directive, the quitter `#;`, is demonstrated with an example. When no match has occurred, scanning picks up after this signal, otherwise we just quit.

```
\tolerant\def\foo[#1]#;(#2){/#1/#2/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo(1)\quad\foo(2)\quad\foo(3)\par

\tolerant\def\foo[#1]#;#={/#1/#2/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo{1}\quad\foo{2}\quad\foo{3}\par

\tolerant\def\foo[#1]#;#2{/#1/#2/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo{1}\quad\foo{2}\quad\foo{3}\par

\tolerant\def\foo[#1]#;(#2)#;#={/#1/#2/#3/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo(1)\quad\foo(2)\quad\foo(3)\par
\foo{1}\quad\foo{2}\quad\foo{3}\par
```

/1// /2// /3//
//1/ //2/ //3/
/1// /2// /3//
//1/ //2/ //3/
/1// /2// /3//
//1/ //2/ //3/

/1/// /2/// /3///
//1// //2// //3//
///1/ ///2/ ///3/

I have to admit that I don't really need it but it made some macros that I was redefining behave better, so there is some self-interest here. Anyway, I considered some other features, like picking up a detokenized argument but I don't expect that to be of much use. In the meantime we ran out of reasonable characters, but some day #? and #! might show up, or maybe I find a use for #< and #>. A summary of all this is given here:

+	keep the braces
-	discard and don't count the argument
/	remove leading an trailing spaces and pars
=	braces are mandate
_	braces are mandate and kept
^	keep leading spaces
1-9	an argument
0	discard but count the argument
*	ignore spaces
:	pick up scanning here
;	quit scanning
.	ignore pars and spaces
,	push back space when quit

The last two have not been discussed and were added later. The period directive gobbles space and par tokens and discards them in the process. The comma directive is like * but it pushes back a space when the matching quits.

```
\tolerant\def\foo[#1]#;(#2){/#1/#2/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo(1)\quad\foo(2)\quad\foo(3)\par

\tolerant\def\foo[#1]#;#={/#1/#2/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo{1}\quad\foo{2}\quad\foo{3}\par

\tolerant\def\foo[#1]#;#2{/#1/#2/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo{1}\quad\foo{2}\quad\foo{3}\par
```

```
\tolerant\def\foo[#1]#;(#2)#;#={/#1/#2/#3/}

\foo[1]\quad\foo[2]\quad\foo[3]\par
\foo(1)\quad\foo(2)\quad\foo(3)\par
\foo{1}\quad\foo{2}\quad\foo{3}\par
```

```
/1//   /2//   /3//
//1/   //2/   //3/
/1//   /2//   /3//
//1/   //2/   //3/
/1//   /2//   /3//
//1/   //2/   //3/
/1///   /2///   /3///
//1//   //2//   //3//
///1/   ///2/   ///3/
```

Gobbling spaces versus pushing back is an interface design decision because it has to do with consistency.

6.3 Runaway arguments

There is a particular troublesome case left: a runaway argument. The solution is not pretty but it's the only way: we need to tell the parser that it can quit.

```
\tolerant\def\foo[#1=#2]%
  {\ifarguments 0\or 1\or 2\or 3\or 4\fi:\vl\type{#1}\vl\type{#2}\vl}
```

The outcome demonstrates that one still has to do some additional checking for sane results and there are alternative way to (ab)use this mechanism. It all boils down to a clever combination of delimiters and `\ignorearguments`.

```
\dontleavehmode \foo[a=1]
\dontleavehmode \foo[b=]
\dontleavehmode \foo[=]
\dontleavehmode \foo[x]\ignorearguments
```

All calls are accepted:

```
2:|a|1|
2:|b|
2:|
1:|x]|
```

Just in case you wonder about performance: don't expect miracles here. On the one hand there is some extra overhead in the engine (when defining macros as well as when collecting arguments during a macro call) and maybe using these new features can sort of compensate that. As mentioned: the gain is mostly in cleaner macro code and less clutter in tracing. And I just want the ConTEXt code to look nice: that way users can look in the source to see what happens and not drown in all these show-off tricks, special characters like underscores, at signs, question marks and exclamation marks.

For the record: I normally run tests to see if there are performance side effects and as long as processing the test suite that has thousands of files of all kind doesn't take more time it's okay. Actually, there is a little gain in ConTEXt but that is to be expected, but I bet users won't notice it, because it's easily offset by some inefficient styling. Of course another gain of loosing some indirectness is that error messages point to the macro that the user called for and not to some follow up.

6.4 Introspection

A macro has a meaning. You can serialize that meaning as follows:

```
\tolerant\protected\def\foo#1[#2]#*[#3]%
  {(1=#1) (2=#3) (3=#3)}

\meaning\foo
```

The meaning of `\foo` comes out as:

```
tolerant protected macro:#1[#2]#*[#3]->(1=#1) (2=#3) (3=#3)
```

When you load the module `system-tokens` you can also say:

```
\luatokentable\foo
```

This produces a table of tokens specifications:

tolerant protected macro:#1[#2]#*[#3]->(1=#1) (2=#3) (3=#3)

tolerant protected control sequence: foo						
596683	19	49	match			argument 1
593223	12	91	other char	[	U+0005B	
592616	19	50	match			argument 2
592626	12	93	other char	]	U+0005D	
596803	19	42	match			argument *

593689	12	91	other char	[	U+0005B	
596518	19	51	match			argument 3
597141	12	93	other char	]	U+0005D	
593495	20	0	end match			
324593	12	40	other char	(	U+00028	
595597	12	49	other char	1	U+00031	
594657	12	61	other char	=	U+0003D	
595551	21	1	parameter reference			
147481	12	41	other char	)	U+00029	
592804	10	32	spacer			
594201	12	40	other char	(	U+00028	
593410	12	50	other char	2	U+00032	
593623	12	61	other char	=	U+0003D	
596874	21	3	parameter reference			
596733	12	41	other char	)	U+00029	
596982	10	32	spacer			
592388	12	40	other char	(	U+00028	
596871	12	51	other char	3	U+00033	
593925	12	61	other char	=	U+0003D	
594804	21	3	parameter reference			
595636	12	41	other char	)	U+00029	

A token list is a linked list of tokens. The magic numbers in the first column are the token memory pointers. and because macros (and token lists) get recycled at some point the available tokens get scattered, which is reflected in the order of these numbers. Normally macros defined in the macro package are more sequential because they stay around from the start. The second and third row show the so called command code and the specifier. The command code groups primitives in categories, the specifier is an indicator of what specific action will follow, a register number a reference, etc. Users don't need to know these details. This macro is a special version of the online variant:

```
\showluatokens\foo
```

That one is always available and shows a similar list on the console. Again, users normally don't want to know such details.

6.5 nesting

You can nest macros, as in:

```
\def\foo#1#2{\def\oof##1{<#1>##1<#2>}}
```

At first sight the duplication of # looks strange but this is what happens. When TEX scans the definition of `\foo` it sees two arguments. Their specification ends up in the preamble that defines the matching. When the body is scanned, the `#1` and `#2` are turned into a parameter reference. In order to make nested macros with arguments possible a # followed by another # becomes just one #. Keep in mind that the definition of `\oof` is delayed till the macro `\foo` gets expanded. That definition is just stored and the only thing that get's replaced are the two references to a macro parameter

control sequence: foo						
596823	19	49	match			argument 1
596778	19	50	match			argument 2
595609	20	0	end match			
596821	128	1	def			def
479047	146	0	tolerant call			oof
595510	6	35	parameter			
596995	12	49	other char	1	U+00031	
594324	1	123	left brace			
594261	12	60	other char	<	U+0003C	
596686	21	1	parameter reference			
592403	12	62	other char	>	U+0003E	
596845	6	35	parameter			
596761	12	49	other char	1	U+00031	
597021	12	60	other char	<	U+0003C	
596980	21	2	parameter reference			
595551	12	62	other char	>	U+0003E	
596814	2	125	right brace			

Now, when we look at these details, it might become clear why for instance we have 'variable' names like `#4` and not `#whatever` (with or without hash). Macros are essentially token lists and token lists can be seen as a sequence of numbers. This is not that different from other programming environments. When you run into buzzwords like 'bytecode' and 'virtual machines' there is actually nothing special about it: some high level programming (using whatever concept, and in the case of TEX it's macros) eventually ends up as a sequence of instructions, say bytecodes. Then you need some machinery to run over that and act upon those numbers. It's something you arrive at naturally when you play with interpreting languages.[7]

[7] I actually did when I wrote an interpreter for some computer assisted learning system, think of a kind of interpreted Pascal, but later realized that it was a a bytecode plus virtual machine thing. I'd just applied what I learned when playing with eight bit processors that took bytes, and interpreted opcodes and such.

So, internally a #4 is just one token, a operator-operand combination where the operator is “grab a parameter” and the operand tells “where to store” it. Using names is of course an option but then one has to do more parsing and turn the name into a number[8], add additional checking in the macro body, figure out some way to retain the name for the purpose of reporting (which then uses more token memory or strings). It is simply not worth the trouble, let alone the fact that we loose performance, and when TEX showed up those things really mattered.

It is also important to realize that a # becomes either a preamble token (grab an argument) or a reference token (inject the passed tokens into a new input level). Therefore the duplication of hash tokens ## that you see in macro nested bodies also makes sense: it makes it possible for the parser to distinguish between levels. Take:

```
\def\foo#1{\def\oof##1{#1##1#1}}
```

Of course one can think of this:

```
\def\foo#fence{\def\oof#text{#fence#text#fence}}
```

But such names really have to be unique then! Actually ConTEXt does have an input method that supports such names, but discussing it here is a bit out of scope. Now, imagine that in the above case we use this:

```
\def\foo[#1][#2]{\def\oof##1{#1##1#2}}
```

If you're a bit familiar with the fact that TEX has a model of category codes you can imagine that a predictable “hash followed by a number” is way more robust than enforcing the user to ensure that catcodes of ‘names’ are in the right category (read: is a bracket part of the name or not). So, say that we go completely arbitrary names, we then suddenly needs some escaping, like:

```
\def\foo[#{left}][#{right}]{\def\oof#{text}{#{left}#{text}#{right}}}
```

And, if you ever looked into macro packages, you will notice that they differ in the way they assign category codes. Asking users to take that into account when defining macros makes not that much sense.

So, before one complains about TEX being obscure (the hash thing), think twice. Your demand for simplicity for your coding demand will make coding more cumbersome for

There's nothing spectacular about all this and I only realized decades later that the buzzwords describes old natural concepts.

[8] This is kind of what MetaPost does with parameters to macros. The side effect is that in reporting you get `text0`, `expr2` and such reported which doesn't make things more clear.

the complex cases that macro packages have to deal with. It's comparable using TEX for input or using (say) mark down. For simple documents the later is fine, but when things become complex, you end up with similar complexity (or even worse because you lost the enforced detailed structure). So, just accept the unavoidable: any language has its peculiar properties (and for sure I do know why I dislike some languages for it). The TEX system is not the only one where dollars, percent signs, ampersands and hashes have special meaning.

6.6 Prefixes

Traditional TEX has three prefixes that can be used with macros: `\global`, `\outer` and `\long`. The last two are no-op's in LuaMetaTEX and if you want to know what they do (did) you can look it up in the TEXbook. The ε-TEX extension gave us `\protected`.

In LuaMetaTEX we have `\global`, `\protected`, `\tolerant` and overload related prefixes like `\frozen`. A protected macro is one that doesn't expand in an expandable context, so for instance inside an `\edef`. You can force expansion by using the `\expand` primitive in front which is also something LuaMetaTEX.

Frozen macros cannot be redefined without some effort. This feature can to some extent be used to prevent a user from overloading, but it also makes it harder for the macro package itself to redefine on the fly. You can remove the lock with `\unletfrozen` and add a lock with `\letfrozen` so in the end users still have all the freedoms that TEX normally provides.

```
                \def\foo{foo} 1: \meaning\foo
          \frozen\def\foo{foo} 2: \meaning\foo
     \unletfrozen    \foo      3: \meaning\foo
\protected\frozen\def\foo{foo} 4: \meaning\foo
     \unletfrozen    \foo      5: \meaning\foo
```

1: macro:foo
2: macro:foo
3: macro:foo
4: protected macro:foo
5: protected macro:foo

This actually only works when you have set `\overloadmode` to a value that permits redefining a frozen macro, so for the purpose of this example we set it to zero.

A `\tolerant` macro is one that will quit scanning arguments when a delimiter cannot be matched. We saw examples of that in a previous section.

These prefixes can be chained (in arbitrary order):

```
\frozen\tolerant\protected\global\def\foo[#1]#*[#2]{...}
```

There is actually an additional prefix, `\immediate` but that one is there as signal for a macro that is defined in and handled by Lua. This prefix can then perform the same function as the one in traditional TEX, where it is used for backend related tasks like `\write`.

Now, the question is of course, to what extent will ConTEXt use these new features. One important argument in favor of using `\tolerant` is that it gives (hopefully) better error messages. It also needs less code due to lack of indirectness. Using `\frozen` adds some safeguards although in some places where ConTEXt itself overloads commands, we need to defrost. Adapting the code is a tedious process and it can introduce errors due to mistypings, although these can easily be fixed. So, it will be used but it will take a while to adapt the code base.

One problem with frozen macros is that they don't play nice with for instance `\futurelet`. Also, there are places in ConTEXt where we actually do redefine some core macro that we also want to protect from redefinition by a user. One can of course `\unletfrozen` such a command first but as a bonus we have a prefix `\overloaded` that can be used as prefix. So, one can easily redefine a frozen macro but it takes a little effort. After all, this feature is mainly meant to protect a user for side effects of definitions, and not as final blocker.[9]

A frozen macro can still be overloaded, so what if we want to prevent that? For this we have the `\permanent` prefix. Internally we also create primitives but we don't have a prefix for that. But we do have one for a very special case which we demonstrate with an example:

```
\def\FOO % trickery needed to pick up an optional argument
  {\noalign{\vskip10pt}}

\noaligned\protected\tolerant\def\OOF[#1]%
  {\noalign{\vskip\iftok{#1}\emptytoks10pt\else#1\fi}}

\starttabulate[|l|l|]
    \NC test \NC test \NC \NR
    \NC test \NC test \NC \NR
```

[9] As usual adding features like this takes some experimenting and we're now at the third variant of the implementation, so we're getting there. The fact that we can apply such features in large macro package like ConTEXt helps figuring out the needs and best approaches.

```
    \FOO
    \NC test \NC test \NC \NR
    \OOF[30pt]
    \NC test \NC test \NC \NR
    \OOF
    \NC test \NC test \NC \NR
\stoptabulate
```

When TEX scans input (from a file or token list) and starts an alignment, it will pick up rows. When a row is finished it will look ahead for a `\noalign` and it expands the next token. However, when that token is protected, the scanner will not see a `\noalign` in that macro so it will likely start complaining when that next macro does get expanded and produces a `\noalign` when a cell is built. The `\noaligned` prefix flags a macro as being one that will do some `\noalign` as part of its expansion. This trick permits clean macros that pick up arguments. Of course it can be done with traditional means but this whole exercise is about making the code look nice.

The table comes out as:

test test
test test

test test

test test

test test

One can check the flags with `\ifflags` which takes a control sequence and a number, where valid numbers are:

1	frozen	2	permanent	4	immutable	8	primitive
16	mutable	32	noaligned	64	instance		

The level of checking is controlled with the `\overloadmode` but I'm still not sure about how many levels we need there. A zero value disables checking, the values 1 and 3 give warnings and the values 2 and 4 trigger an error.

6.7 Arguments

The number of arguments that a macro takes is traditionally limited to nine (or ten if one takes the trailing # into account). That this is enough for most cases is demonstrated

by the fact that ConTEXt has only a handful of macros that use #9. The reason for this limitation is in part a side effect of the way the macro preamble and arguments are parsed. However, because in LuaMetaTEX we use a different implementation, it was not that hard to permit a few more arguments, which is why we support upto 15 arguments, as in:

```
\def\foo#1#2#3#4#5#6#7#8#9#A#B#C#D#E#F{...}
```

We can support the whole alphabet without much trouble but somehow sticking to the hexadecimal numbers makes sense. It is unlikely that the core of ConTEXt will use this option but sometimes at the user level it can be handy. The penalty in terms of performance can be neglected.

```
\tolerant\def\foo#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=%
  {(#1)(#2)(#3)(#4)(#5)(#6)(#7)(#8)(#9)(#A)(#B)(#C)(#D)(#E)(#F)}

\foo{1}{2}
```

In the previous example we have 15 optional arguments where braces are mandate (otherwise we the scanner happily scoops up what follows which for sure gives some error).

6.8 Constants

The LuaMetaTEX engine has lots of efficiency tricks in the macro parsing and expansion code that makes it not only fast but also let is use less memory. However, every time that the body of a macro is to be injected the expansion machinery kicks in. This often means that a copy is made (pushed in the input and used afterwards). There are however cases where the body is just a list of character tokens (with category letter or other) and no expansion run over the list is needed.

It is tempting to introduce a string data type that just stores strings and although that might happen at some point it has the disadvantage that one need to tokenize that string in order to be able to use it, which then defeats the gain. An alternative has been found in constant macros, that is: a macro without parameters and a body that is considered to be expanded and never freed by redefinition. There are two variants:

```
\cdef      \foo          {whatever}
\cdefcsname foo\endcsname{whatever}
```

These are actually just equivalents to

```
\edef      \foo          {whatever}
```

```
\edefcsname foo\endcsname{whatever}
```

just to make sure that the body gets expanded at definition time but they are also marked as being constant which in some cases might give some gain, for instance when used in csname construction. The gain is less then one expects although there are a few cases in ConTEXt where extreme usage of parameters benefits from it. Users are unlikely to use these two primitives.

Another example of a constant usage is this:

```
\lettonothing\foo
```

which gives `\foo` an empty body. That one is used in the core, if only because it gives a bit smaller code. Performance is no that different from

```
\let\foo\empty
```

but it saves one token (8 bytes) when used in a macro. The assignment itself is not that different because `\foo` is made an alias to `\empty` which in turn only needs incrementing a reference counter.

6.9 Passing parameters

When you define a macro, the #1 and more parameters are embedded as a reference to a parameter that is passed. When we have four parameters, the parameter stack has four entries and when an entry is eventually accessed a new input level is pushed and tokens are fetched from that list. This has some side effects when we check a parameter. This can happen multiple times, depending on how often we access a parameter. Take the following:

```
\def\oof#1{#1}

\tolerant\def\foo[#1]#*[#2]%
  {1:\ifparameter#1\or Y\else N\fi\quad
   2:\ifparameter#2\or Y\else N\fi\quad
   \oof{3:\ifparameter #1\or Y\else N\fi\quad
        4:\ifparameter #2\or Y\else N\fi\quad}%
   \par}

\foo \foo[] \foo[][] \foo[A] \foo[A][B]
```

This gives:

```
1:N  2:N  3:N  4:N
1:N  2:N  3:N  4:N
1:N  2:N  3:N  4:N
1:Y  2:N  3:Y  4:N
1:Y  2:Y  3:Y  4:Y
```

as you probably expect. However the first two checks are different from the embedded checks because they can check against the parameter reference. When we expand `\oof` its argument gets passed to the macro as a list and when the scanner collects the next token it will then push the parameter content on the input stack. So, then, instead of a reference we get the referenced parameter list. Internally that means that in 3 and 4 we check for a token and not for the length of the list (as in case 1 & 2). This means that

```
\iftok{#1}\emptytoks Y\else N\fi
\ifparameter#1\or    Y\else N\fi
```

are different. In the first case we have a proper token list and nested conditionals in that list are okay. In the second case we just look ahead to see if there is an `\or`, `\else` or other condition related command and if so we decide that there is no parameter. So, if `\ifparameter` is a suitable check for empty depends on the need for expansion.

When you define macros that themselves call macros that should operate on the arguments of its parent you can easily pass these:

```
\def\foo#1#2%
  {\oof{#1}{#2}{P}%
   \oof{#1}{#2}{Q}%
   \oof{#1}{#2}{R}}

\def\oof#1#2#3%
  {[#1][#1]%
   #3%
   [#2][#2]}
```

Here the nested call to `\oof` involved three passed parameters. You can avoid that as follows:

```
\def\foo#1#2%
  {\def\MyIndexOne{#1}%
   \def\MyIndexTwo{#2}%
   \oof{P}\oof{Q}\oof{R}}
```

```
\def\oof#1%
  {(\MyIndexOne)(\MyIndexOne)%
   #1%
   (\MyIndexTwo)(\MyIndexTwo)}
```

You can also do this:

```
\def\foo#1#2%
  {\def\oof##1%
     {/#1/#2/%
     ##1%
     /#1//#2/}%
   \oof{P}\oof{Q}\oof{R}}
```

These parameters indicated by # in the macro body are in fact references. When we call for instance `\foo{1}{2}` the two parameters get pushed on a parameter stack and the embodied references point to these stack entries. By the time that body gets expanded TEX bumps the input level and pushes the parameter list onto the input stack. It then continues expansion. The parameter is not copied, because it can't be changed anyway. The only penalty in terms of performance and memory usage is the pushing and popping of the input. So how does that work out for these three cases?

When in the first case the `\oof{#1}{#2}{P}` is seen, TEX starts expanding the `\oof` macro. That one expects three arguments. The #1 reference is seen and in this case a copy of that parameter is passed. The same is true for the other two. Then, inside `\oof` expansion happens on the parameters on the stack and no copies have to be made there.

The second case defines two macros so again two copies are made that make the bodies of these macros. This comes at the cost of some runtime and memory. However, this time with `\oof{P}` only one argument gets passed and instead expansion of the macros happen in there.

Normally macro arguments are not that large but there can be situations where we really want to avoid useless copying. This not only saves memory but also can give a bit better performance. In the examples above the second variant is some 10%faster than the first one. We can gain another 10%with the following trick:

```
\def\foo#1#2%
  {\parameterdef\MyIndexOne\plusone % 1
   \parameterdef\MyIndexTwo\plustwo % 2
   \oof{P}\oof{Q}\oof{R}\norelax}
```

```
\def\oof#1%
  {<\MyIndexOne><\MyIndexOne>%
   #1%
   <\MyIndexTwo><\MyIndexTwo>}
```

Here we define an explicit parameter reference that we access later on. There is the overhead of a definition but it can be neglected. We use that reference (abstraction) in `\oof`. Actually you can use that reference in any call down the chain.

When applied to `\foo{1}{2}` the four variants above give us:

```
[1][1]P[2][2][1][1]Q[2][2][1][1]R[2][2]
(1)(1)P(2)(2)(1)(1)Q(2)(2)(1)(1)R(2)(2)
/1/2/P/1//2//1/2/Q/1//2//1/2/R/1//2/
<1><1>P<2><2><1><1>Q<2><2><1><1>R<2><2>
```

Before we had `parameterdef` we had this:

```
\def\foo#1#2%
  {\integerdef\MyIndexOne\parameterindex\plusone % 1
   \integerdef\MyIndexTwo\parameterindex\plustwo % 2
   \oof{P}\oof{Q}\oof{R}\norelax}

\def\oof#1%
  {<\expandparameter\MyIndexOne><\expandparameter\MyIndexOne>%
   #1%
   <\expandparameter\MyIndexTwo><\expandparameter\MyIndexTwo>}
```

It involves more tokens, is a bit less abstract, but as it is a cheap extension we kept it. It actually demonstrates that one can access parameters in the stack by index, but it one then needs to keep track of where access takes place. In principle one can debug the call chain this way.

To come back to performance and memory usage, when the arguments become larger the fourth variant with the `\parameterdef` quickly gains over the others. But it only shows in exceptional usage. This mechanism is more about abstraction: it permits us to efficiently turn arguments into local variables without the overhead involved in creating macros. You can test if a parameter is set

```
\tolerant\protected\def\MyMacro[#1]#:#2%
  {\parameterdef\MyArgumentOne\plusone
   \parameterdef\MyArgumentTwo\plustwo
   \ifparameter\MyArgumentOne\or
```

```
      (\MyArgumentOne)
   \fi
   /\MyArgumentTwo/}

\MyMacro[one]{two}
\MyMacro{two}
```

Indeed we get:

(one) /two/ /two/

Of course `\ifparameter#1\or...` is more efficient but once you use named parameters like this it's probably not something you're worry too much about,

6.10 Nesting

We also have a few preamble features that relate to nesting. Although we can do without (as shown for years in LMTX) they do have some benefits. They are discussed as group here and because they are only useful for low level programming we stick to simple examples. The `#L` and `#R` use the following token as delimiters. Here we use `[` and `]` but they can be a `\cs` as well. Nested delimiters are handled well.

The `#S` grabs the argument till the next final square bracket `]` but in the process will grab nested with it sees a `[`. The `#P` does the same for parentheses and `#X` for angle brackets. In the next examples the `#*` just gobbles optional spaces but we've seen that one already.

The `#G` argument just registers the next token as delimiter but it will grab multiple of them. The `#M` gobbles more: in addition to the delimiter spaces are gobbled.

```
\tolerant\def\fooA               [#1]{(#1)}
\tolerant\def\fooB          [#L[#R]#1{(#1)}
\tolerant\def\fooC               #S#1{(#1)}
\tolerant\def\fooE              #S#1,{(#1)}
\tolerant\def\fooF         #S#1#*#S#2{(#1/#2)}
\tolerant\def\fooG [#1]#S[#2]#*#S[#3]{(#1/#2/#3)}
\tolerant\def\fooH [#1][#S#2]#*[#S#3]{(#1/#2/#3)}
\tolerant\def\fooI           #1=#2#G,{(#1=#2)}
\tolerant\def\fooJ           #1=#2#M,{(#1=#2)}

\fooA[x]           (x)           (x)
\fooB[x]           (x)           (x)
```

```
\fooC[1[2]3[4]5]    ([1[2]3[4]5])  (1[2]3[4]5)
\fooE X[,]X,        (X[,]X)        (X[,]X)
\fooF[A] [B]        ([A]/[B])      (A/B)
\fooF[] []          ([]/[])        (/)
\fooG[a][b][c]      (a/b/c)        (a/b/c)
\fooG[a][b]         (a/b/)         (a/b/)
\fooG[a]            (a//)          (a//)
\fooG[a][x[x]x][c]  (a/x[x]x/c)    (a/x[x]x/c)
\fooH[a][x[x]x][c]  (a/x[x]x/c)    (a/x[x]x/c)
\fooI X=X,,,        (X=X)          (X=X)
\fooJ X=X, , ,      (X=X)          (X=X)
```

These features make it possible to support nested setups more efficiently and also makes it possible to accept values that contain balanced brackets in setup commands without additional overhead. Although it has never been an issue to let users specify:

```
\defineoverlay[whatever][{some \command[withparameters] here}]

\setupfoo[before={\blank[big]}]
```

it might be less confusing to permit:

```
\defineoverlay[whatever][some \command[withparameters] here]

\setupfoo[before=\blank[big]]
```

as well, if only because occasionally users get hit by this.

6.11 Duplicate hashes

In TEX every character has a so called category code. Most characters are classified as ‘letter’ (they make up words) or as ‘other’. In Unicode we distinguish symbols, punctuation, and more, but in TEX these are all of category ‘other’. In math however we can classify them differently but in this perspective we ignore that. The backslash has category ‘escape’ and it starts a control sequence. The curly braces are (internally) of category ‘left brace’ and ‘right brace’ aka ‘begin group’ and ‘end group’ but, no matter what they are called, they begin and end something: a group, argument, token list, box, etc. Any character can have those categories. Although it would look strange to a TEX user, this can be made valid:

```
!protected !gdef !weird¶1
B
```

```
    something: ¶1
E
!weird BhereE
```

In such a setup spaces can be of category ‘invisible’. The paragraph symbol takes the place of the hash as parameter identifier. The next code shows how this is done. Here we wrap all in a macro so that we don't get catcode interference in the document source.

```
\def\NotSoTeX
  {\begingroup
   \catcode `B \begingroupcatcode
   \catcode `E \endgroupcatcode
   \catcode `¶ \parametercatcode
   \catcode `! \escapecatcode
   \catcode 32 \ignorecatcode
   \catcode 13 \ignorecatcode
   % this buffer has a definition:
   \getbuffer
   % which is now known globally
   \endgroup}
\NotSoTeX
\weird{there}
```

This results in:

something:here
something:there

In the first line the `!`, `B` and `E` are used as escape and argument delimiters, in the second one we use the normal characters. When we show the `\meaningasis` we get:

```
\global \protected \def \weird #1{something:#1}
```

or in more detail:

```
protected control sequence: weird
596712  19   49  match                          argument 1
593074  20    0  end match
593658  11  115  letter              s  U+00073
596717  11  111  letter              o  U+0006F
597542  11  109  letter              m  U+0006D
595543  11  101  letter              e  U+00065
```

Duplicate hashes

```
594036  11  116  letter              t  U+00074
596976  11  104  letter              h  U+00068
595406  11  105  letter              i  U+00069
597132  11  110  letter              n  U+0006E
593638  11  103  letter              g  U+00067
596805  12   58  other char          :  U+0003A
596683  21    1  parameter reference
```

So, no matter how we set up the system, in the end we get some generic representation. When we see #1 in 'print' it can be either two tokens, # (catcode parameter) followed by 1 with catcode other, or one token referring to parameter 1 where the character 1 is the opcode of an internal 'reference command'. In order to distinguish a reference from the two token case, parameter hash tokens get shown as doubles.

```
\def\test #1{x#1x##1x####1x}
\def\tset ¶1{x¶1x¶¶1x¶¶¶¶1x}
```

And with `\meaning` we get, consistent with the input:

```
macro:#1->x#1x##1x####1x
macro:#1->x#1x¶¶1x¶¶¶¶1x
```

These are equivalent, apart from the parameter character in the body of the definition:

control sequence: test

```
592919  19   49  match                          argument 1
595485  20    0  end match
597550  11  120  letter              x  U+00078
596687  21    1  parameter reference
594643  11  120  letter              x  U+00078
596223   6   35  parameter
598001  12   49  other char          1  U+00031
597143  11  120  letter              x  U+00078
593930   6   35  parameter
593228   6   35  parameter
597623  12   49  other char          1  U+00031
596007  11  120  letter              x  U+00078
```

control sequence: tset

```
597752  19   49  match                          argument 1
```

```
596833  20    0  end match
-------------------------------------------------------
597020  11  120  letter               x  U+00078
593248  21    1  parameter reference
595969  11  120  letter               x  U+00078
593294   6  182  parameter
597753  12   49  other char           1  U+00031
597022  11  120  letter               x  U+00078
597588   6  182  parameter
595585   6  182  parameter
597329  12   49  other char           1  U+00031
597883  11  120  letter               x  U+00078
-------------------------------------------------------
```

Watch how every ‘parameter’ is just a character with the Unicode index of the used input character as property. Let us summarize the process. When a single parameter character is seen in the input, the next characer determines how it will be interpreted. If there is a digit then it becomes a reference to a parameter in the preamble, and when followed by another parameter character it will be appended to the body of the macro and that second one is dropped. So, two parameter characters become one, and four become two. One parameter character becomes a reference and from that you can guess what three in a row become. However, when TEX is showing the macro definition (using `meaning`) the hashes get duplicated in order to distinguish parameter references from parameter characters that were kept (e.g. for nested definitions). One can make an argument for `\parameterchar` as we also have `\escapechar` but by now this convention is settled and it doesn't look that bad anyway.

We now come to the more tricky part with respect to the doubling of hashes. When TEX was written its application landscape looked a bit different. For instance, fonts were limited and therefore it was natural to access special characters by name. Using `\#` to get a hash in the text was not that problematic, if one needed that character at all. The same can be said for the braces, backslash and even the dollar (after all TEX is free software).

But what if we have more visualization and/or serialization than meanings and tracing? When we opened op the internals in LuaTEX and even more in LuaMetaTEX the duplicating of hashes became a bit of a problem. There we don't need to distinguish between a parameter reference and a parameter character because by that time these references are resolved. All hashes that we encounter are just that: hashes. And this is why in LuaMetaTEX we disable the duplication for those cases where it serves no purpose.

When the engine scans a macro definition it starts with picking up the name of the macro. Then it starts scanning the preamble up to the left brace. In the preamble of a

macro the scanner converts hashes followed by another token into single match token. Then when the macro body is scanned single hashes followed by a number become a reference, while double hashes become one hash and get interpreted at expansion time (possibly triggering an error when not followed by a valid specifier like a number). In traditional TEX we basically had this:

```
\def\test#1{#1}
\def\test#1{##}
\def\test#1{#X}
\def\test#1{##1}
```

There can be a trailing # in the preamble for special purposes but we forget about that now. The first definition is valid, the second definition is invalid when the macro is expanded and the third definition triggers an error at definition time. The last definition will again trigger an error at expansion time.

However, in LuaMetaTEX we have an extended preamble where the following preamble parameters are handled (some only in tolerant mode):

`#n`	parameter	index `1` upto `E`
`#0`	throw away parameter	increment index
`#-`	ignore parameter	keep index
`#*`	gobble white space	
`#+`	keep (honor) the braces	
`#.`	ignore pars and spaces	
`#,`	push back space when no match	
`#/`	remove leading and trailing spaces and pars	
`#=`	braces are mandate	
`#^`	keep leading spaces	
`#_`	braces are mandate and kept (obey)	
`#@`	par delimiter	only for internal usage
`#:`	pick up scanning here	
`#;`	quit scanning	
`#L`	left delimiter token	followed by token
`#R`	right delimiter token	followed by token
`#G`	gobble token	followed by token
`#M`	gobble token and spaces	followed by token

#S	nest square brackets	only inner pairs
#X	nest angle brackets	only inner pairs
#P	nest parentheses	only inner pairs

As mentioned these will become so called match tokens and only when we show the meaning the hash will show up again.

```
\def\test[#1]#*[*S#2]{.#1.#2.}
```

control sequence: test

```
593215  12  91  other char          [  U+0005B
23064   19  49  match                          argument 1
596871  12  93  other char          ]  U+0005D
592815  19  42  match                          argument *
597323  12  91  other char          [  U+0005B
596950  12  42  other char          *  U+0002A
595964  11  83  letter              S  U+00053
595615  19  50  match                          argument 2
593328  12  93  other char          ]  U+0005D
597077  20   0  end match
-----------------------------------------------------------
594458  12  46  other char          .  U+0002E
595999  21   1  parameter reference
597636  12  46  other char          .  U+0002E
596908  21   2  parameter reference
598031  12  46  other char          .  U+0002E
```

This means that in the body of a macro you will not see #* show up. It is just a directive that tells the macro parser that spaces are to be skipped. The #S directive makes the parser for the second parameter handle nested square bracket. The only hash that we can see end up in the body is the one that we entered as double hash (then turned single) followed by (in traditional terms) a number that when all gets parsed with then become a reference: the sequence ##1 internally is #1 and becomes ‘reference to parameter 1’ assuming that we define a macro in that body. If no number is there, an error is issued. This opens up the possibility to add more variants because it will only break compatibility with respect to what is seen as error. As with the preamble extensions, old documents that have them would have crashed before they became available.

So, this means that in the body, and actually anywhere in the document apart from preambles, we now support the following general parameter specifiers. Keep in mind that they expand in an expansion context which can be tricky when they overlap with

preamble entries, like for instance #R in such an expansion. Future extensions can add more so *any* hashed shortcut is sensitive for that.

#I	current iterator	\currentloopiterator
#P	parent iterator	\previousloopiterator 1
#G	grandparent iterator	\previousloopiterator 2
#H	hash escape	#
#S	space escape	
#T	tab escape	\t
#L	newline escape	\n
#R	return escape	\r
#X	backslash escape	\
#N	nbsp	U+00A0 (under consideration)
#Z	zws	U+200B (under consideration)

Some will now argue that we already have ^^ escapes in TEX and ^^^^ and ^^^^^^ in LuaTEX and that is true. However, these can be disabled, and in ConTEXt they are, where we instead enable the prescript, postscript, and index features in mathmode and there type ^ and _ are used. Even more: in ConTEXt we just let ^, _ and & be what they are. Occasionally I consider $ to be just that but as I don't have dollars I will happily leave that for inline math. When users are not defining macros or are using the alternative definitions we can consider making the # a hash. An excellent discussion of how TEX reads it's input and changes state accordingly can be found in Victor Eijkhouts “TEX By Topic”, section 2.6: when ^^ is followed by a character with $\nu < 128$ the interpreter will inject a character with code $\nu - 64$. When followed by two (!) lowercase hexadecimal characters, the corresponding character will be injected. Anyway, it not only looks kind of ugly, it also is somewhat weird because what follows is interpreted mixed way. The substitution happens early on (which is okay). But, how about the output? Traditional TEX serializes special characters with a similar syntax but that has become optional when eight bit mode was added to the engines, it is configurable in LuaTEX and has been dropped in LuaMetaTEX: we operate in a utf universum.

6.11 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

7 Grouping

low level TEX

grouping

Contents

7.1 Introduction

This is a rather short explanation. I decided to write it after presenting the other topics at the 2019 ConTEXt meeting where there was a question about grouping.

7.1.1 Pascal

In a language like Pascal, the language that TEX has been written in, or Modula, its successor, there is no concept of grouping like in TEX. But we can find keywords that suggests this:

```
for i := 1 to 10 do begin ... end
```

This language probably inspired some of the syntax of TEX and MetaPost. For instance an assignment in MetaPost uses `:=` too. However, the `begin` and `end` don't really group but define a block of statements. You can have local variables in a procedure or function but the block is just a way to pack a sequence of statements.

7.1.2 TEX

In TEX macros (or source code) the following can occur:

```
\begingroup
    ...
\endgroup
```

as well as:

```
\bgroup
    ...
\egroup
```

Here we really group in the sense that assignments to variables inside a group are forgotten afterwards. All assignments are local to the group unless they are explicitly done global:

```
\scratchcounter=1
\def\foo{foo}
\begingroup
    \scratchcounter=2
    \global\globalscratchcounter=2
    \gdef\foo{FOO}
\endgroup
```

Here `\scratchcounter` is still one after the group is left but its global counterpart is now two. The `\foo` macro is also changed globally.

Although you can use both sets of commands to group, you cannot mix them, so this will trigger an error:

```
\bgroup
\endgroup
```

The bottomline is: if you want a value to persist after the group, you need to explicitly change its value globally. This makes a lot of sense in the perspective of TEX.

7.1.3 MetaPost

The MetaPost language also has a concept of grouping but in this case it's more like a programming language.

```
begingroup ;
    n := 123 ;
endgroup ;
```

In this case the value of n is 123 after the group is left, unless you do this (for numerics there is actually no need to declare them):

```
begingroup ;
    save n ; numeric n ; n := 123 ;
endgroup ;
```

Given the use of MetaPost (read: MetaFont) this makes a lot of sense: often you use macros to simplify code and you do want variables to change. Grouping in this language

serves other purposes, like hiding what is between these commands and let the last expression become the result. In a `vardef` grouping is implicit.

So, in MetaPost all assignments are global, unless a variable is explicitly saved inside a group.

7.1.4 Lua

In Lua all assignments are global unless a variable is defines local:

```
local x = 1
local y = 1
for i = 1, 10 do
    local x = 2
    y = 2
end
```

Here the value of `x` after the loop is still one but `y` is now two. As in LuaTEX we mix TEX, MetaPost and Lua you can mix up these concepts. Another mixup is using `:=`, `endfor`, `fi` in Lua after done some MetaPost coding or using `end` instead of `endfor` in MetaPost which can make the library wait for more without triggering an error. Proper syntax highlighting in an editor clearly helps.

7.1.5 C

The Lua language is a mix between Pascal (which is one reason why I like it) and C.

```
int x = 1 ;
int y = 1 ;
for (i=1; i<=10;i++) {
    int x = 2 ;
    y = 2 ;
}
```

The semicolon is also used in Pascal but there it is a separator and not a statement end, while in MetaPost it does end a statement (expression).

7.2 Kinds of grouping

Explicit grouping is accomplished by the two grouping primitives:

```
\begingroup
    \sl render slanted here
\endgroup
```

However, often you will find this being used:

```
{\sl render slanted here}
```

This is not only more compact but also avoids the `\endgroup` gobbling following spaces when used inline. The next code is equivalent but also suffers from the gobbling:

```
\bgroup
    \sl render slanted here
\egroup
```

The `\bgroup` and `\egroup` commands are not primitives but aliases (made by `\let`) to the left and right curly brace. These two characters have so called category codes that signal that they can be used for grouping. The *can be* here suggest that there are other purposes and indeed there are, for instance in:

```
\toks 0 = {abs}
\hbox {def}
```

In the case of a token list assignment the curly braces fence the assignment, so scanning stops when a matching right brace is found. The following are all valid:

```
\toks 0 = {a{b}s}
\toks 0 = \bgroup a{b}s}
\toks 0 = {a{\bgroup b}s}
\toks 0 = {a{\egroup b}s}
\toks 0 = \bgroup a{\bgroup b}s}
\toks 0 = \bgroup a{\egroup b}s}
```

They have in common that the final fence has to be a right brace. That the first one can be a an alias is due to the fact that the scanner searches for a brace equivalent when it looks for the value. Because the equal is optional, there is some lookahead involved which involves expansion and possibly push back while once scanning for the content starts just tokens are collected, with a fast check for nested and final braces.

In the case of the box, all these specifications are valid:

```
\hbox {def}
\hbox \bgroup def\egroup
```

```
\hbox \bgroup def}
\hbox \bgroup d{e\egroup f}
\hbox {def\egroup
```

This is because now the braces and equivalent act as grouping symbols so as long as they match we're fine. There is a pitfall here: you cannot mix and match different grouping, so the next issues an error:

```
\bgroup xxx\endgroup   % error
\begingroup xxx\egroup % error
```

This can make it somewhat hard to write generic grouping macros without trickery that is not always obvious to the user. Fortunately it can be hidden in macros like the helper `\groupedcommand`. In LuaMetaTEX we have a clean way out of this dilemma:

```
\beginsimplegroup xxx\endsimplegroup
\beginsimplegroup xxx\endgroup
\beginsimplegroup xxx\egroup
```

When you start a group with `\beginsimplegroup` you can end it in the three ways shows above. This means that the user (or calling macro) doesn't take into account what kind of grouping was used to start with.

When we are in math mode things are different. First of all, grouping with `\begingroup` and `\endgroup` in some cases works as expected, but because the math input is converted in a list that gets processed later some settings can become persistent, like changes in style or family. You can bet better use `\beginmathgroup` and `\endmathgroup` as they restore some properties. We also just mention the `\frozen` prefix that can be used to freeze assignments to some math specific parameters inside a group.

7.3 Hooks

In addition to the original `\aftergroup` primitive we have some more hooks. They can best be demonstrated with an example:

```
\begingroup \bf
    %
    \aftergroup   A \aftergroup   1
    \atendofgroup B \atendofgroup 1
    %
    \aftergrouped   {A2}
    \atendofgrouped {B2}
```

```
    %
    test
\endgroup
```

These collectors are accumulative. Watch how the bold is applied to what we inject before the group ends.

test B1B2A1A2

7.4 Local versus global

When TEX enters a group and an assignment is made the current value is stored on the save stack, and at the end of the group the original value is restored. In LuaMetaTEX this mechanism is made a bit more efficient by avoiding redundant stack entries. This is also why the next feature can give unexpected results when not used wisely.

Now consider the following example:

```
\newdimension\MyDimension

\starttabulate[|||||]
    \NC         \MyDimension10pt \the\MyDimension
    \NC \advance\MyDimension10pt \the\MyDimension
    \NC \advance\MyDimension10pt \the\MyDimension \NC \NR
    \NC         \MyDimension10pt \the\MyDimension
    \NC \advance\MyDimension10pt \the\MyDimension
    \NC \advance\MyDimension10pt \the\MyDimension \NC \NR
\stoptabulate
```

10.0pt	10.0pt	10.0pt
10.0pt	10.0pt	10.0pt

The reason why we get the same values is that cells are a group and therefore the value gets restored as we move on. We can use the `\global` prefix to get around this

```
\starttabulate[|||||]
    \NC \global        \MyDimension10pt \the\MyDimension
    \NC \global\advance\MyDimension10pt \the\MyDimension
    \NC \global\advance\MyDimension10pt \the\MyDimension \NC \NR
    \NC \global        \MyDimension10pt \the\MyDimension
    \NC \global\advance\MyDimension10pt \the\MyDimension
    \NC \global\advance\MyDimension10pt \the\MyDimension \NC \NR
```

```
\stoptabulate
```

10.0pt 20.0pt 30.0pt
10.0pt 20.0pt 30.0pt

Instead of using a global assignment and increment we can also use the following

```
\constrained\MyDimension\zeropoint
\starttabulate[|||||]
    \NC \retained        \MyDimension10pt \the\MyDimension
    \NC \retained\advance\MyDimension10pt \the\MyDimension
    \NC \retained\advance\MyDimension10pt \the\MyDimension \NC \NR
    \NC \retained        \MyDimension10pt \the\MyDimension
    \NC \retained\advance\MyDimension10pt \the\MyDimension
    \NC \retained\advance\MyDimension10pt \the\MyDimension \NC \NR
\stoptabulate
```

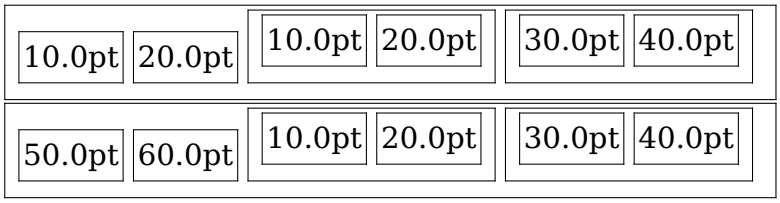
10.0pt 20.0pt 30.0pt
10.0pt 20.0pt 30.0pt

So what is the difference with the global approach? Say we have these two buffers:

```
\startbuffer[one]
    \global\MyDimension\zeropoint
    \framed {
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
    }
    \framed {
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
    }
\stopbuffer

\startbuffer[two]
    \global\MyDimension\zeropoint
    \framed {
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
        \getbuffer[one]
    }
    \framed {
```

```
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
        \framed {\global\advance\MyDimension10pt \the\MyDimension}
        \getbuffer[one]
    }
\stopbuffer
```

Typesetting the second buffer gives us:

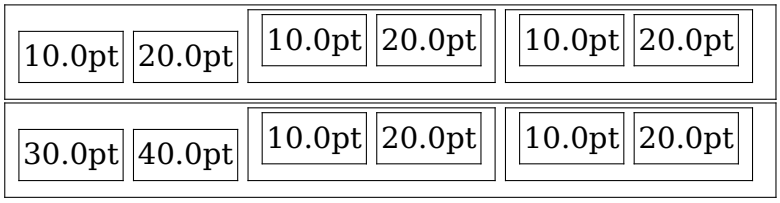


When we want to have these entities independent and not use different variables, the global settings bleeding from one into the other entity is messy. Therefore we can use this:

```
\startbuffer[one]
    \constrained\MyDimension\zeropoint
    \framed {
        \framed {\retained        \MyDimension10pt \the\MyDimension}
        \framed {\retained\advance\MyDimension10pt \the\MyDimension}
    }
    \framed {
        \framed {\retained        \MyDimension10pt \the\MyDimension}
        \framed {\retained\advance\MyDimension10pt \the\MyDimension}
    }
\stopbuffer

\startbuffer[two]
    \constrained\MyDimension\zeropoint
    \framed {
        \framed {\retained\advance\MyDimension10pt \the\MyDimension}
        \framed {\retained\advance\MyDimension10pt \the\MyDimension}
        \getbuffer[one]
    }
    \framed {
        \framed {\retained\advance\MyDimension10pt \the\MyDimension}
        \framed {\retained\advance\MyDimension10pt \the\MyDimension}
        \getbuffer[one]
    }
```

```
\stopbuffer
```

Now we get this:



The `\constrained` prefix makes sure that we have a stack entry, without being clever with respect to the current value. Then the `\retained` prefix can do its work reliably and avoid pushing the old value on the stack. Without the constrain it gets a bit unpredictable because then it all depends on where further up the chain the value was put on the stack. Of course one can argue that we should not have the "save stack redundant entries optimization" but that's not going to be removed.

7.5 Files

Although it doesn't really fit in this chapter, here are some hooks into processing files:

```
Hello World!\atendoffiled         {\writestatus{FILE}{ATEOF B #1}}\par
Hello World!\atendoffiled         {\writestatus{FILE}{ATEOF A #1}}\par
Hello World!\atendoffiled reverse {\writestatus{FILE}{ATEOF C #1}}\par
Hello World!\begingroup\sl \atendoffiled {\endgroup}\par
```

Inside a file you can register tokens that will be expanded when the file ends. You can also do that beforehand using a variant of the `\input` primitive:

```
\eofinput {\writestatus{FILE}{DONE}} {thatfile.tex}
```

This feature is mostly there for consistency with the hooks into groups and paragraphs but also because `\everyeof` is kind of useless given that one never knows beforehand if a file loads another file. The hooks mentioned above are bound to the current file.

7.5 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

8 Security

low level TEX security

Contents

8.1 Preamble

Here I will discuss a moderate security subsystem of LuaMetaTEX and therefore ConTEXt LMTX. This is not about security in the sense of the typesetting machinery doing harm to your environment, but more about making sure that a user doesn't change the behavior of the macro package in ways that introduce interference and thereby unwanted side effect. It's all about protecting macros.

This is all very experimental and we need to adapt the ConTEXt source code to this. Actually that will happen a few times because experiments trigger that. It might take a few years before the security model is finalized and all files are updated accordingly. There are lots of files and macros involved. In the process the underlying features in the engine might evolve.

8.2 Flags

Before we go into the security levels we see what flags can be set. The TEX language has a couple of so called prefixes that can be used when setting values and defining macros. Any engine that has traditional TEX with ε-TEX extensions can do this:

```
                \def\foo{foo}
\global         \def\foo{foo}
\global\protected\def\foo{foo}
```

And LuaMetaTEX adds another one:

```
      \tolerant          \def\foo{foo}
\global\tolerant         \def\foo{foo}
\global\tolerant\protected\def\foo{foo}
```

What these prefixes do is discussed elsewhere. For now is is enough to know that the two optional prefixes `\protected` and `\tolerant` make for four distinctive cases of macro calls.

But there are more prefixes:

`frozen`	a macro that has to be redefined in a managed way
`permanent`	a macro that had better not be redefined
`primitive`	a primitive that normally will not be adapted
`immutable`	a macro or quantity that cannot be changed, it is a constant
`mutable`	a macro that can be changed no matter how well protected it is
`instance`	a macro marked as (for instance) be generated by an interface
`noaligned`	the macro becomes acceptable as `\noalign` alias
`overloaded`	when permitted the flags will be adapted
`enforced`	all is permitted (but only in zero mode or ini mode)
`aliased`	the macro gets the same flags as the original

These prefixed set flags to the command at hand which can be a macro but basically any control sequence.

To what extent the engine will complain when a property is changed in a way that violates the above depends on the parameter `\overloadmode`. When this parameter is set to zero no checking takes place. More interesting are values larger than zero. If that is the case, when a control sequence is flagged as mutable, it is always permitted to change. When it is set to immutable one can never change it. The other flags determine the kind of checking done. Currently the following overload values are used:

		immutable	**permanent**	**primitive**	**frozen**	**instance**
1	warning	*	*	*		
2	error	*	*	*		
3	warning	*	*	*	*	
4	error	*	*	*	*	
5	warning	*	*	*	*	*
6	error	*	*	*	*	*

The even values (except zero) will abort the run. In ConTEXt we plug in a callback that deals with the messages. A value of 255 will freeze this parameter. At level five and above the `instance` flag is also checked but no drastic action takes place. We use this to signal to the user that a specific instance is redefined (of course the definition macros can check for that too).

So, how does it work. The following is okay:

```
\def\MacroA{A}
\def\MacroB{B}
```

```
\let\MyMacro\MacroA
\let\MyMacro\MacroB
```

The first two macros are ordinary ones, and the last two lines just create an alias. Such an alias shares the definition, but when for instance `\MacroA` is redefined, its new meaning will not be reflected in the alias.

```
\permanent\protected\def\MacroA{A}
\permanent\protected\def\MacroB{B}
\let\MyMacro\MacroA
\let\MyMacro\MacroB
```

This also works, because the `\let` will create an alias with the protected property but it will not take the `permanent` propery along. For that we need to say:

```
\permanent\protected\def\MacroA{A}
\permanent\protected\def\MacroB{B}
\permanent\let\MyMacro\MacroA
\permanent\let\MyMacro\MacroB
```

or, when we want to copy all properties:

```
\permanent\protected\def\MacroA{A}
\permanent\protected\def\MacroB{B}
\aliased\let\MyMacro\MacroA
\aliased\let\MyMacro\MacroB
```

However, in ConTEXt we have commands that we like to protect against overloading but at the same time have a different meaning depending on the use case. An example is the `\NC` (next column) command that has a different implementation in each of the table mechanisms.

```
\permanent\protected\def\NC_in_table    {...}
\permanent\protected\def\NC_in_tabulate{...}
\aliased\let\NC\NC_in_table
\aliased\let\NC\NC_in_tabulate
```

Here the second aliasing of `\NC` fails (assuming of course that we enabled overload checking). One can argue that grouping can be used but often no grouping takes place when we redefine on the fly. Because `frozen` is less restrictive than `primitive` or `permanent`, and of course `immutable`, the next variant works:

```
\frozen\protected\def\NC_in_table    {...}
```

```
\frozen\protected\def\NC_in_tabulate{...}
\overloaded\let\NC\NC_in_table
\overloaded\let\NC\NC_in_tabulate
```

However, in practice, as we want to keep the overload checking, we have to do:

```
\frozen\protected\def\NC_in_table   {...}
\frozen\protected\def\NC_in_tabulate{...}
\overloaded\frozen\let\NC\NC_in_table
\overloaded\frozen\let\NC\NC_in_tabulate
```

or use `\aliased`, but there might be conflicting permissions. This is not that nice, so there is a kind of dirty trick possible. Consider this:

```
\frozen\protected\def\NC_in_table   {...}
\frozen\protected\def\NC_in_tabulate{...}
\def\setNCintable   {\enforced\let\frozen\let\NC\NC_in_table}
\def\setNCintabulate{\enforced\let\frozen\let\NC\NC_in_tabulate}
```

When we're in so called `initex` mode or when the overload mode is zero, the `\enforced` prefix is internalized in a way that signals that the follow up is not limited by the overload mode and permissions. This definition time binding mechanism makes it possible to use `permanent` macros that users cannot redefine, but existing macros can, unless of course they tweak the mode parameter.

Now keep in mind that users can always cheat but that is intentional. If you really want to avoid that you can set the overload mode to 255 after which it cannot be set any more. However, it can be useful to set the mode to zero (or some warning level) when foreign macro packages are used.

8.3 Complications

One side effect of all this is that all those prefixes can lead to more code. On the other hand we save some due to the extended macro argument handling features. When you take the size of the format file as reference, in the end we get a somewhat smaller file. Every token that you add of remove gives a 8 bytes difference. The extra overhead that got added to the engine is compensated by the fact that some macro implementations can be more efficient. In the end, in spite of these new features and the more extensive testing of flags performance is about the same.[10]

[10] And if you wonder about memory, by compacting the used (often scattered) token memory before dumping I manages to save some 512K on the format file, so often the loss and gain are somewhere else.

8.4 Introspection

In case you want to get some details about the properties of a macro, you can check its meaning. The full variant shows all of them.

```
% a macro with two optional arguments with optional spacing in between:

\permanent\tolerant\protected\def\MyFoo[#1]#*[#2]{(#1)(#2)}

\meaningless\MyFoo\par
\meaning    \MyFoo\par
\meaningfull\MyFoo\par
```

```
[#1]#*[#2]->(#1)(#2)
tolerant protected macro:[#1]#*[#2]->(#1)(#2)
permanent tolerant protected macro:[#1]#*[#2]->(#1)(#2)
```

8.4 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

9 Characters

low level TEX

characters

Contents

9.1 Introduction

This explanation is part of the low level manuals because in practice users will not have to deal with these matters in MkIV and even less in LMTX. You can skip to the last section for commands.

9.2 History

If we travel back in time to when TEX was written we end up in eight bit character universe. In fact, the first versions assumed seven bits, but for comfortable use with languages other than English that was not sufficient. Support for eight bits permits the usage of so called code pages as supported by operating systems. Although ascii input became kind of the standard soon afterwards, the engine can be set up for different encodings. This is not only true for TEX, but for many of its companions, like MetaFont and therefore MetaPost.[11]

Core components of a TEX engine are hyphenation of words, applying inter-character kerns and build ligatures. In traditional TEX engines those processes are interwoven into the par builder but in LuaTEX these are separate stages. The original approach is the reason that there is a relation between the input encoding and the font encoding: the character in the input is the slot used in a reference to a glyph. When producing the final result (e.g. pdf) there can also be a mapping to an index in a font resource.

```
input A [tex ->] font slot A [backend ->] glyph index A
```

The mapping that TEX does is normally one-to-one but an input character can undergo some transformation. For instance a character beyond ascii 126 can be made active and expand to some character number that then becomes the font slot. So, it is the

[11] This remapping to an internal representation (e.g. ebcdic) is not present in LuaTEX where we assume utf8 to be the input encoding. The MetaPost library that comes with LuaTEX still has that code but in LuaMetaTEX it's gone. There one can set up the machinery to be utf8 aware too.

expansion (or meaning) of a character that end up as numeric reference in the glyph node. Virtual fonts can introduce yet another remapping but that's only visible in the backend.

Actually, in LuaTEX the same happens but in practice there is no need to go active because (at least in ConTEXt) we assume a Unicode path so there the font slot is the Unicode got from the utf8 input.

In the eight bit universe macro packages (have to) provide all kind of means to deal with (in the perspective of English) special characters. For instance, `\"a` would put a diaeresis on top of the a or even better, refer to a character in the encoding that the chosen font provides. Because there are some limitations of what can go in an eight bit font, and because in different countries the used TEX fonts evolved kind of independent, we ended up with quite some different variants of fonts. It was only with the Latin Modern project that this became better. Interesting is that when we consider the fact that such a font has often also hardly used symbols (like registered or copyright) coming up with an encoding vector that covers most (latin based) European languages (scripts) is not impossible[12] Special symbols could simply go into a dedicated font, also because these are always accessed via a macro so who cares about the input. It never happened.

Keep in mind that when utf8 is used with eight bit engines, ConTEXt will convert sequences of characters into a slot in a font (depending on the font encoding used which itself depends on the coverage needed). For this every first (possible) byte of a multibyte utf sequence is an active character, which is no big deal because these are outside the ascii range. Normal ascii characters are single byte utf sequences and fall through without treatment.

Anyway, in ConTEXt MkII we dealt with this by supporting mixed encodings, depending on the (local) language, referencing the relevant font. It permits users to enter the text in their preferred input encoding and also get the words properly hyphenated. But we can leave these MkII details behind.

9.3 The heritage

In MkIV we got rid of input and font encodings, although one can still load files in a specific code page.[13] We also kept the means to enter special characters, if only because

[12] And indeed in the Latin Modern project we came up with one but it was already to late for it to become popular.

[13] I'm not sure if users ever depend on an input encoding different from utf8.

text editors seldom support(ed) a wide range of visual editing of those. This is why we still have

```
\"u \^a \v{s} \AE \ij \eacute \oslash
```

and many more. The ones with one character names are rather common in the TEX community but it is definitely a weird mix of symbols. The next two are kind of outdated: in these days you delegate that to the font handler, where turning them into 'single' character references depends on what the font offers, how it is set up with respect to (for instance) ligatures, and even might depend on language or script.

The ones with the long names partly are tradition, but as we have a lot of them, in MkII they actually serve a purpose. These verbose names are used in the so called encoding vectors and are part of the utf expansion vectors. They are also used in labels so that we have a good indication if what goes in there: remember that in those times editors often didn't show characters, unless the font for display had them, or the operating system somehow provided them from another font. These verbose names are used for latin, greek and cyrillic and for some other scripts and symbols. They take up quite a bit of hash space and the format file.[14]

9.4 The LMTX approach

In the process of tagging all (public) macros in LMTX (which happened in 2020-2021) I wondered if we should keep these one character macros, the references to special characters and the verbose ones. When asked on the mailing list it became clear that users still expect the short ones to be present, often just because old bibTEX files are used that might need them. However, in MkIV and LMTX we load bibTEX files in a way that turn these special character references into proper utf8 input so it makes a weak argument. Anyway, although they could go, for now we keep them because users expect them. However, in LMTX the implementation is somewhat different now, a bit more efficient in terms of hash and memory, potentially a bit less efficient in runtime, but no one will notice that.

A new command has been introduced, the very short `\chr`.

```
\chr {a} \chr {a} \chr {a}
\chr {`a} \chr {'a} \chr {"a}
\chr {a acute} \chr {a grave} \chr {a umlaut}
```

[14] In MkII we have an abstract front-end with respect to encodings and also an abstract backend with respect to supported drivers but both approaches no longer make sense today.

```
\chr {aacute}  \chr {agrave}  \chr {aumlaut}
```

In the first line the composed character using two characters, a base and a so called mark. Actually, one doesn't have to use `\chr` in that case because ConTEXt does already collapse characters for you. The second line looks like the shortcuts `\``, `\'` and `\"`. The third and fourth lines could eventually replace the more symbolic long names, if we feel the need. Watch out: in Unicode input the marks come *after*.

à á ä
à á ä
á à a˘mła˘t
á à a˘mła˘t

Currently the repertoire is somewhat limited but it can be easily be extended. It all depends on user needs (doing Greek and Cyrillic for instance). The reason why we actually save code deep down is that the helpers for this have always been there.[15]

The `\"` commands are now just aliases to more verbose and less hackery looking macros:

```
\withgrave         à  \`  à
\withacute         á  \'  á
\withcircumflex    â  \^  â
\withtilde         ã  \~  ã
\withmacron        ā  \=  ā
\withbreve         ĕ  \u  ĕ
\withdotaccent     ċ  \.  .c
\withdiaeresis     ë  \"  ë
\withring          ů  \r  ů
\withhungarumlaut  ű  \H  ű
\withcaron         ě  \v  ě
\withcedilla       ę  \c  ę
\withogonek        ę  \k  ę
```

Not all fonts have these special characters. Most natural is to have them available as precomposed single glyphs, but it can be that they are just two shapes with the marks anchored to the base. It can even be that the font somehow overlays them, assuming (roughly) equal widths. The `compose` font feature in ConTEXt normally can handle most well.

[15] So if needed I can port this approach back to MkIV, but for now we keep it as is because we then have a reference.

An occasional ugly rendering doesn't matter that much: better have something than nothing. But when it's the main language (script) that needs them you'd better look for a font that handles them. When in doubt, in ConTEXt you can enable checking:

command	**equivalent to**
`\checkmissingcharacters`	`\enabletrackers[fonts.missing]`
`\removemissingcharacters`	`\enabletrackers[fonts.missing=remove]`
`\replacemissingcharacters`	`\enabletrackers[fonts.missing=replace]`
`\handlemissingcharacters`	`\enabletrackers[fonts.missing={decompose,replace}]`

The decompose variant will try to turn a composed character into its components so that at least you get something. If that fails it will inject a replacement symbol that stands out so that you can check it. The console also mentions missing glyphs. You don't need to enable this by default[16] but you might occasionally do it when you use a font for the first time.

In LMTX this mechanism has been upgraded so that replacements follow the shape and are actually real characters. The decomposition has not yet been ported back to MkIV.

The full list of commands can be queried when a tracing module is loaded:

```
\usemodule[s][characters-combinations]

\showcharactercombinations
```

We get this list:

acute	U+00301	◌́	`\withacute`
breve	U+00306	◌̆	`\withbreve`
caron	U+0030C	◌̌	`\withcaron`
caron below	U+0032C	◌̬	`\withcaronbelow`
cedilla	U+00327	◌̧	`\withcedilla`
circumflex	U+00302	◌̂	`\withcircumflex`
circumflex below	U+0032D	◌̭	`\withcircumflexbelow`
comma below	U+00326	◌̦	`\withcommabelow`
diaeresis	U+00308	◌̈	`\withdiaeresis`
dieresis	U+00308	◌̈	`\withdieresis`
dot	U+00307	◌̇	`\withdot`
dot below	U+00323	◌̣	`\withdotbelow`
double acute	U+0030B	◌̋	`\withdoubleacute`

[16] There is some overhead involved here.

double grave	U+0030F	̏	\withdoublegrave
double vertical line	U+0030E	̎	\withdoubleverticalline
grave	U+00300	̀	\withgrave
hook	U+00309	̉	\withhook
hook below	U+1FA9D		\withhookbelow
hungarumlaut	U+0030B	̋	\withhungarumlaut
inverted breve	U+00311	̑	\withinvertedbreve
line	U+00304	̄	\withline
line below	U+00331	̲	\withlinebelow
macron	U+00304	̄	\withmacron
macron below	U+00331	̲	\withmacronbelow
middle dot	U+000B7	·	\withmiddledot
ogonek	U+00328	̨	\withogonek
overline	U+00305	̅	
ring	U+0030A	̊	\withring
ring below	U+00325	̥	\withringbelow
slash	U+0002F	/	\withslash
stroke	U+0002F	/	\withstroke
tilde	U+00303	̃	\withtilde
tilde below	U+00330	̰	\withtildebelow
vertical line	U+0030D	̍	\withverticalline

Some combinations are special for ConTEXt because Unicode doesn't specify decomposition for all composed characters.

9.5 spaces

The engine has no real concept of a space. When the input has one it is turned into a glue, likely with some stretch and shrink. When `\nospaces` is set to one, no glue will be inserted. A value of two will inject a zero width glue. When set to three a glyph will be inserted with the character code set by `\spacechar`.

```
\nospaces\plusthree
\spacechar\underscoreasciicode
\hccode\underscoreasciicode\underscoreasciicode
Where are the spaces?
```

The `hccode` tells the machinery that the underscore is a valid word separator (think compound words).

Where_are_the_spaces?_

9.5 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

10 Scope

low level TEX

scope

Contents

10.1 Introduction

When_I_visited_the_file_where_register_allocations_are_implemented_I_wondered_to_what_extend_it_made_sense_to_limit_allocation_to_global_instances_only._This_chapter_deals_with_this_phenomena._

10.2 Registers

In_TEX_definitions_can_be_local_or_global._Most_assignments_are_local_within_a_group._Registe

```
\scratchcounter = 1
here the counter has value 1
\begingroup
    \scratchcounter = 2
    here the counter has value 2
\endgroup
here the counter has value 1
```

with:_

```
\setbox\scratchbox=\hbox{}
here the box has zero width
\begingroup
    \wd\scratchbox=10pt
    here the box is 10pt wide
\endgroup
here the box is 10pt wide
```

It_all_makes_sense_so_a_remark_like_“Assignments_to_box_dimensions_are_always_global”_are_sort_of_confusing._Just_look_at_this:_

```
\setbox\scratchbox=\hbox to 20pt{}
here the box is \the\wd\scratchbox\ wide\par
```

```
\begingroup
    \setbox\scratchbox=\hbox{}
    here the box is \the\wd\scratchbox\ wide\par
    \begingroup
        \wd\scratchbox=15pt
        here the box is \the\wd\scratchbox\ wide\par
    \endgroup
    here the box is \the\wd\scratchbox\ wide\par
\endgroup
here the box is \the\wd\scratchbox\ wide\par
```

here_the_box_is_20.0pt_wide
here_the_box_is_0.0pt_wide
here_the_box_is_15.0pt_wide
here_the_box_is_15.0pt_wide
here_the_box_is_20.0pt_wide

If_you_don't_think_about_it,_what_happens_is_what_you_expect._Now_watch_the_next_variant:_

The_`\global`_is_only_effective_for_the_current_box._It_is_good_to_realize_that_when_we_talk_registers,_the_box_register_behaves_just_like_any_other_register_but_the_manipulations_happen_to_the_current_one._

```
\setbox\scratchbox=\hbox to 20pt{}
here the box is \the\wd\scratchbox\ wide\par
\begingroup
    \setbox\scratchbox=\hbox{}
    here the box is \the\wd\scratchbox\ wide\par
    \begingroup
        \global\wd\scratchbox=15pt
        here the box is \the\wd\scratchbox\ wide\par
    \endgroup
    here the box is \the\wd\scratchbox\ wide\par
\endgroup
here the box is \the\wd\scratchbox\ wide\par
```

here_the_box_is_20.0pt_wide
here_the_box_is_0.0pt_wide
here_the_box_is_15.0pt_wide
here_the_box_is_15.0pt_wide
here_the_box_is_20.0pt_wide

```
\scratchdimen=20pt
here the dimension is \the\scratchdimen\par
\begingroup
    \scratchdimen=0pt
    here the dimension is \the\scratchdimen\par
    \begingroup
        \global\scratchdimen=15pt
        here the dimension is \the\scratchdimen\par
    \endgroup
    here the dimension is \the\scratchdimen\par
\endgroup
here the dimension is \the\scratchdimen\par
```

here_the_dimension_is_20.0pt
here_the_dimension_is_0.0pt
here_the_dimension_is_15.0pt
here_the_dimension_is_15.0pt
here_the_dimension_is_15.0pt

10.3 Allocation

The_plain_TEX_format_has_set_some_standards_and_one_of_them_is_that_registers_are_allocated

```
\newcount\mycounta
\newdimen\mydimena
```

These_commands_take_a_register_from_the_pool_and_relate_the_given_name_to_that_entry._In_ConTEXt_we_have_a_bunch_of_predefined_scratch_registers_for_general_use,_like:_

```
scratchcounter    : \meaningfull\scratchcounter
scratchcounterone : \meaningfull\scratchcounterone
scratchcountertwo : \meaningfull\scratchcountertwo
scratchdimen      : \meaningfull\scratchdimen
scratchdimenone   : \meaningfull\scratchdimenone
scratchdimentwo   : \meaningfull\scratchdimentwo
```

The_meaning_reveals_what_these_are:_

```
scratchcounter_:_global_constant_integer_1026
scratchcounterone_:_global_constant_integer_0
scratchcountertwo_:_global_constant_integer_0
scratchdimen_:_global_constant_dimension_15.0pt
```

```
scratchdimenone_:_global_constant_dimension_0.0pt
scratchdimentwo_:_global_constant_dimension_0.0pt
```

You can use the numbers directly but that is a bad idea because they can clash! In the original TEX engine there are only 256 registers and some are used by the engine and t TEX upped that to 64K. One could go higher but what makes sense? These registers are tak

As mentioned, commands like `\newcount\foo` create a global control sequence `\foo` referencing a counter. You can locally redefine that control sequence unless in LuaMeta-TEX you have so called overload mode enabled. You can do local or global assignments to th

```
\scratchcounter = 123
\begingroup
    \scratchcounter = 456
    \begingroup
        \global\scratchcounter = 789
    \endgroup
\endgroup
```

And in both cases count register 257 is set. When an assignment is global, all current values to that register get the same value. Normally this is all quite transparent: you get what you ask for. However the drawback is that as a user you cannot know what variables are already defined, which means that this will fail (that is: it will issue a message):

```
\newcount\scratchcounter
```

as will the second line in:

```
\newcount\myscratchcounter
\newcount\myscratchcounter
```

In ConTEXt the scratch registers are visible but there are lots of internally used ones are pr TEXt barking to you about not being able to define it. This is why in LMTX (and maybe som

```
\begingroup
  \newlocaldimen\mydimena     \mydimena1\onepoint
  \newlocaldimen\mydimenb     \mydimenb2\onepoint
  (\the\mydimena,\the\mydimenb)
  \begingroup
    \newlocaldimen\mydimena   \mydimena3\onepoint
    \newlocaldimen\mydimenb   \mydimenb4\onepoint
    \newlocaldimen\mydimenc   \mydimenc5\onepoint
```

```
    (\the\mydimena,\the\mydimenb,\the\mydimenc)
    \begingroup
      \newlocaldimen\mydimena \mydimena6\onepoint
      \newlocaldimen\mydimenb \mydimenb7\onepoint
      (\the\mydimena,\the\mydimenb)
    \endgroup
    \newlocaldimen\mydimend   \mydimend8\onepoint
    (\the\mydimena,\the\mydimenb,\the\mydimenc,\the\mydimend)
  \endgroup
  (\the\mydimena,\the\mydimenb)
\endgroup
```

The_allocated_registers_get_zero_values_but_you_can_of_course_set_them_to_any_value_that_fits_their_nature:_

(1.0pt,2.0pt)
(3.0pt,4.0pt,5.0pt)
(6.0pt,7.0pt)
(3.0pt,4.0pt,5.0pt,8.0pt)
(1.0pt,2.0pt)

You_can_also_use_the_next_variant_where_you_also_pass_the_initial_value:_

```
\begingroup
  \setnewlocaldimen\mydimena     1\onepoint
  \setnewlocaldimen\mydimenb     2\onepoint
  (\the\mydimena,\the\mydimenb)
  \begingroup
    \setnewlocaldimen\mydimena   3\onepoint
    \setnewlocaldimen\mydimenb   4\onepoint
    \setnewlocaldimen\mydimenc   5\onepoint
    (\the\mydimena,\the\mydimenb,\the\mydimenc)
    \begingroup
      \setnewlocaldimen\mydimena 6\onepoint
      \setnewlocaldimen\mydimenb 7\onepoint
      (\the\mydimena,\the\mydimenb)
    \endgroup
    \setnewlocaldimen\mydimend   8\onepoint
    (\the\mydimena,\the\mydimenb,\the\mydimenc,\the\mydimend)
  \endgroup
  (\the\mydimena,\the\mydimenb)
```

```
\endgroup
```

So,_again_we_get:_

(1.0pt,2.0pt)
(3.0pt,4.0pt,5.0pt)
(6.0pt,7.0pt)
(3.0pt,4.0pt,5.0pt,8.0pt)
(1.0pt,2.0pt)

When_used_in_the_body_of_the_macro_there_is_of_course_a_little_overhead_involved_in_the_repetitive_allocation_but_normally_that_can_be_neglected._

10.4 Files

When_adding_these_new_allocators_I_also_wondered_about_the_read_and_write_allocators._We_don't_use_them_in_ConTEXt_but_maybe_users_like_them,_so_let's_give_an_example_and

```
\integerdef\StartHere\numexpr\inputlineno+2\relax
\starthiding
SOME LINE 1
SOME LINE 2
SOME LINE 3
SOME LINE 4
\stophiding
\integerdef\StopHere\numexpr\inputlineno-2\relax

\begingroup
  \newlocalread\myreada
  \immediate\openin\myreada {lowlevel-scope.tex}
  \dostepwiserecurse{\StopHere}{\StartHere}{-1}{
    \readline\myreada line #1 to \scratchstring #1 : \scratchstring \par
  }
  \blank
  \dostepwiserecurse{\StartHere}{\StopHere}{1}{
    \read    \myreada line #1 to \scratchstring #1 : \scratchstring \par
  }
  \immediate\closein\myreada
\endgroup
```

Here,_instead_of_hard_coded_line_numbers_we_used_the_stored_values._The_optional_`line`_keyword_is_a_LMTX_speciality._

281_:_SOME_LINE_4
280_:_SOME_LINE_3
279_:_SOME_LINE_2
278_:_SOME_LINE_1

278_:_SOME_LINE_1_
279_:_SOME_LINE_2_
280_:_SOME_LINE_3_
281_:_SOME_LINE_4_

Actually_an_application_can_be_found_in_a_small_(demonstration)_module:_

```
\usemodule[system-readers]
```

This_provides_the_code_for_doing_this:_

```
\startmarkedlines[test]
SOME LINE 1
SOME LINE 2
SOME LINE 3
\stopmarkedlines

\begingroup
  \newlocalread\myreada
  \immediate\openin\myreada {\markedfilename{test}}
  \dostepwiserecurse{\lastmarkedline{test}}{\firstmarkedline{test}}{-1}{
    \readline\myreada line #1 to \scratchstring #1 : \scratchstring \par
  }
  \immediate\closein\myreada
\endgroup
```

As_you_see_in_these_examples,_we_an_locally_define_a_read_channel_without_getting_a_message_about_it_already_being_defined._

10.4 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

11 Paragraphs

low level TEX

paragraphs

Contents

11.1 Introduction

This_manual_is_mostly_discussing_a_few_low_level_wrappers_around_low_level_TEX_features._It
MetaFun_manuals_where_we_mess_a_bit_with_shapes._It_gave_a_good_reason_to_also_cover_so

Because_paragraphs_and_their_construction_are_rather_central_to_TEX,_you_can_imagine_that_t
TEXt._Intercepting_and_tweaking_paragraph_properties_is_even_more_tricky,_which_is_why_we_

11.2 Paragraphs

Before_we_demonstrate_some_trickery,_let's_see_what_a_paragraph_is._Normally_a_document_source_is_formatted_like_this:_

```
some text (line 1)
some text (line 2)

some more test (line 1)
some more test (line 2)
```

[17] For this we have `\enableexperiments` which one can use in `cont-loc.mkxl` or `cont-exp.mkxl`, files that are loaded runtime when on the system. When you use them, make sure they don't interfere; they are not part of the updates, contrary to `cont-new.mkxl`.

There_are_two_blocks_of_text_here_separated_by_an_empty_line_and_they_become_two_paragraphs._Unless_configured_otherwise_an_empty_line_is_an_indication_that_we_end_a_paragraph._You_can_also_explicitly_do_that:_

```
some text (line 1)
some text (line 2)
\par
some more test (line 1)
some more test (line 2)
```

When_TEX_starts_a_paragraph,_it_actually_also_does_something_think_of:_

```
[\the\everypar]some text      (line 1) some text      (line 2) \par
[\the\everypar]some more test (line 1) some more test (line 2) \par
```

or_more_accurate:_

```
[\the\everypar]some text      some text      \par
[\the\everypar]some more test some more test \par
```

because_the_end-of-line_character_has_become_a_space._As_mentioned,_an_empty_line_is_actually_the_end_of_a_paragraph._But_in_LuaMetaTEX_we_can_cheat_a_bit._If_we_have_this:_

```
line 1

line 2
```

We_can_do_this_(watch_how_we_need_to_permit_overloading_a_primitive_when_we_have_enabled_\overloadmode):_

```
\pushoverloadmode
\def\linepar{\removeunwantedspaces !\ignorespaces}
\popoverloadmode
line 1

line 2
```

This_comes_out_as:_

line_1_

line_2_

I_admit_that_since_it_got_added_(as_part_of_some_cleanup_halfway_the_overhaul_of_the_engine)_I_never_saw_a_reason_to_use_it,_but_it_is_a_cheap_feature._The_`\linepar`_primitive_is_undefined_(`\undefined`)_by_default_so_no_user_sees_it_anyway._Just_don't_use_it_unless_maybe_for_some_pseudo_database_trickery_(I_considered_using_it_for_the_database_module_but_it_is_not_needed)._In_a_similar_fashion,_just_don't_redefine_`\par`:_it's_asking_for_troubles_and_'not_done'_in_ConTEXt_anyway._

Back_to_reality._In_LuaTEX_we_get_a_node_list_that_starts_with_a_so_called_`localpar`_node_and

When_the_paragraph_is_broken_into_lines_hanging_indentation_or_a_so_called_par_shape_can_be_applied,_and_we_will_see_more_of_that_later,_here_we_talk_`\par`_and_show_another_LuaMetaTEX_trick:_

```
\def\foo{{\bf test:} \ignorepars}

\foo

line
```

The_macro_typesets_some_text_and_then_skips_to_the_next_paragraph:_

test:_line_

Think_of_this_primitive_as_being_a_more_powerful_variant_of_`\ignorespaces`._This_leaves_one_aspect:_how_do_we_start_a_paragraph._Technically_we_need_to_force_TEX_into_so_called_hTEXt_we_have_more_high_level_variants,_for_instance_we_have_`\noindentation`._

A_robust_way_to_make_sure_that_you_get_in_horizontal_mode_is_using_`\dontleavehmode`_whichis_a_wink_to_`\leavevmode`,_a_command_that_you_should_never_use_in_ConTEXt,_so_when_you_c

When_TEX_starts_with_a_paragraph_the_`\everypar`_token_list_is_expanded_and_again_this_is_a_

One_of_the_things_that_TEX_does_in_injecting_the_indentation._Even_when_there_is_none,_it_geTEXt_we_enable_the_LuaMetaTEX_feature_that_does_use_a_skip_instead_of_a_box._It's_part_of_t

A_bit_off_topic_is_the_fact_that_in_traditional_TEX_empty_lines_or_`\par`_primitives_can_trigger_aTEXt_MkII_most_macros_that_could_be_sensitive_for_this_were_defined_as_`\long`_so_that_users_TEX_these_error-triggers_could_be_disabled_which_of_course_we_enable_in_ConTEXt_and_in_LuaMetaTEX_these_features_have_been_removed_altogether._I_don't_think_users_will_complain_abo

If_you_want_to_enforce_a_newline_but_not_a_new_paragraph_you_can_use_the_`\crlf`_command._When_used_on_its_own_it_will_produce_an_empty_line._Don't_use_this_to_create_whitespace_between_lines._

If_you_want_to_do_something_after_so_called_par_tokens_are_seen_you_can_do_this:_

```
\def\foo{{\bf >>>> }}
\expandafterpars\foo

this is a new paragraph ...

\expandafterpars\foo
\par\par\par\par
this is a new paragraph ...
```

This_not_to_be_confused_with_`\everypar`_which_is_a_token_list_that_TEX_itself_injects_before_e

>>>>_this_is_a_new_paragraph_..._

>>>>_this_is_a_new_paragraph_..._

This_is_typically_a_primitive_that_will_only_be_used_in_macros._You_can_actually_program_it_using_macros:_pickup_a_token,_check_and_push_it_back_when_it's_not_a_par_equivalent_token._The_primitive_is_is_just_nicer_(and_easier_on_the_log_when_tracing_is_enabled)._

11.3 Properties

A_paragraph_is_just_a_collection_of_lines_that_result_from_one_input_line_that_got_broken._This_process_of_breaking_into_lines_is_influenced_by_quite_some_parameters._In_traditional_TEX_and_also_in_LuaMetaTEX_by_default_the_values_that_are_in_effect_when_the_en

However,_in_LuaMetaTEX_we_can_optionally_store_them_with_the_paragraph._When_that_happe

variable	category	code
`\hsize`	hsize	0x00000001
`\leftskip`	skip	0x00000002
`\rightskip`	skip	0x00000002
`\hangindent`	hang	0x00000004
`\hangafter`	hang	0x00000004
`\parindent`	indent	0x00000008
`\parfillleftskip`	parfill	0x00000010
`\parfillskip`	parfill	0x00000010
`\parinitleftskip`	parfill	0x00000010
`\parinitrightskip`	parfill	0x00000010
`\emergencyleftskip`	emergency	0x00800000

\emergencyrightskip	emergency	0x00800000
\adjustspacing	adjust	0x00000020
\protrudechars	protrude	0x00000040
\pretolerance	tolerance	0x00000080
\tolerance	tolerance	0x00000080
\emergencystretch	stretch	0x00000100
\looseness	looseness	0x00000200
\lastlinefit	lastline	0x00000400
\linepenalty	linepenalty	0x00000800
\interlinepenalty	linepenalty	0x00000800
\clubpenalty	clubpenalty	0x00001000
\widowpenalty	widowpenalty	0x00002000
\displaywidowpenalty	displaypenalty	0x00004000
\lefttwindemerits	twindemerits	0x20000000
\righttwindemerits	twindemerits	0x20000000
\brokenpenalty	brokenpenalty	0x00008000
\adjdemerits	demerits	0x00010000
\doublehyphendemerits	demerits	0x00010000
\finalhyphendemerits	demerits	0x00010000
\parshape	shape	0x00020000
\interlinepenalties	linepenalty	0x00000800
\clubpenalties	clubpenalty	0x00001000
\widowpenalties	widowpenalty	0x00002000
\displaywidowpenalties	displaypenalty	0x00004000
\brokenpenalties	brokenpenalty	0x00008000
\orphanpenalties	orphanpenalty	0x00200000
\toddlerpenalties	toddlerpenalty	0x00400000
\fitnessclasses	fitnessclasses	0x40000000
\adjacentdemerits	demerits	0x00010000
\mathleftclass	orphanpenalty	0x00200000
\baselineskip	line	0x00040000
\lineskip	line	0x00040000
\lineskiplimit	line	0x00040000
\adjustspacingstep	adjust	0x00000020
\adjustspacingshrink	adjust	0x00000020
\adjustspacingstretch	adjust	0x00000020
\hyphenationmode	hyphenation	0x00080000
\shapingpenaltiesmode	shapingpenalty	0x00100000
\shapingpenalty	shapingpenalty	0x00100000
\emergencyextrastretch	emergency	0x00800000

\parpasses	parpasses	0x01000000
\linebreakchecks	linebreakchecks	0x10000000
\singlelinepenalty	singlelinepenalty	0x02000000
\hyphenpenalty	hyphenpenalty	0x04000000
\exhyphenpenalty	exhyphenpenalty	0x08000000

As you can see here, there are more paragraph related parameters than in for instance pdfTEX and LuaTEX and these are (to be) explained in the LuaMetaTEX manual. You can ima

This is pretty low level and there are a bunch of helpers that support this but these are not really user level macros. As with everything TEX you can mess around as much as y TEXt core functionality.

In LMTX taking these snapshots is turned on by default and because it thereby fundamentally influences the par builder, users can run into compatibility issues but in practice there has been no complaints (and this feature has been in use quite a while before this document was written). One reason for users not noticing is that one of the big benefits is probably handled by tricks mentioned on the mailing list. Imagine that you have this:

```
{\bf watch out:} here is some text
```

In this small example the result will be as expected. But what if something magic with the start of a paragraph is done? Like this:

```
\placefigure[left]{A cow!}{\externalfigure[cow.pdf]}

{\bf watch out:} here is some text ... of course much more is needed to
    get a flow around the figure!
```

The figure will hang at the left side of the paragraph but it is put there when the text starts and that happens inside the bold group. It means that the properties we set in order to get the shape around the figure are lost as soon as we're at 'here is some text' and definitely is wrong when the paragraph ends and the par builder has to use them to get the shape right. We get text overlapping the figure. A trick to overcome this is:

```
\dontleavehmode {\bf watch out:} here is some text ... of course much
    more is needed to get a flow around the figure!
```

where the first macro makes sure we already start a paragraph before the group is entered (using a \strut also works). It's not nice and I bet users have been bitten by this and by now know the tricks. But, with snapshots such fuzzy hacks are not needed any more! The same is true with this:

```
{\leftskip 1em some text \par}
```

where we had to explicitly end the paragraph inside the group in order to retain the skip. I suppose that users normally use the high level environments so they never had to worry about this. It's also why users probably won't notice that this new mechanism has been active for a while. Actually, when you now change a parameter inside the paragraph its new value will not be applied (unless you prefix it with `\frozen` or snapshot it) but no one did that anyway.

11.4 Wrapping up

In ConTEXt LMTX we have a mechanism to exercise macros (or content) before a paragraph

Although the high level interface has been around for a while it still needs a bit more testing (read: use cases are needed). In the few cases where we already use it application can be different because again it relates to snapshots. This because in the past we had to use tricks that also influenced the user interface of some macros (which made them less natural as one would expect). So the question is: where do we apply it in old mechanisms and where not.

todo: accumulation, interference, where applied, limitations

11.5 Hanging

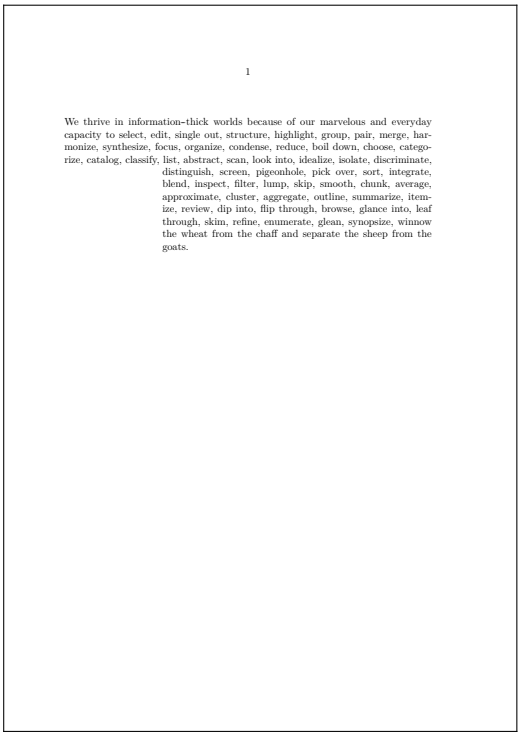
There are two mechanisms for getting a specific paragraph shape: rectangular hanging and arbitrary shapes. Both mechanisms work top-down. The first mechanism uses a combination of `\hangafter` and `\hangindent`, and the second one depends on `\parshape`. this section we discuss the rectangular one.

```
\hangafter  4 \hangindent  4cm \samplefile{tufte} \page
\hangafter -4 \hangindent  4cm \samplefile{tufte} \page
\hangafter  4 \hangindent -4cm \samplefile{tufte} \page
\hangafter -4 \hangindent -4cm \samplefile{tufte} \page
```

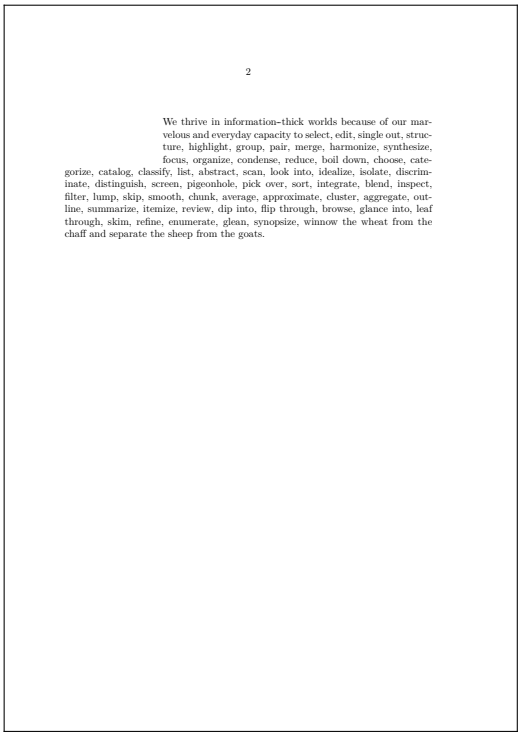
As you can see in figure 11.1, the four cases are driven by the sign of the values. If you wa

11.6 Shapes

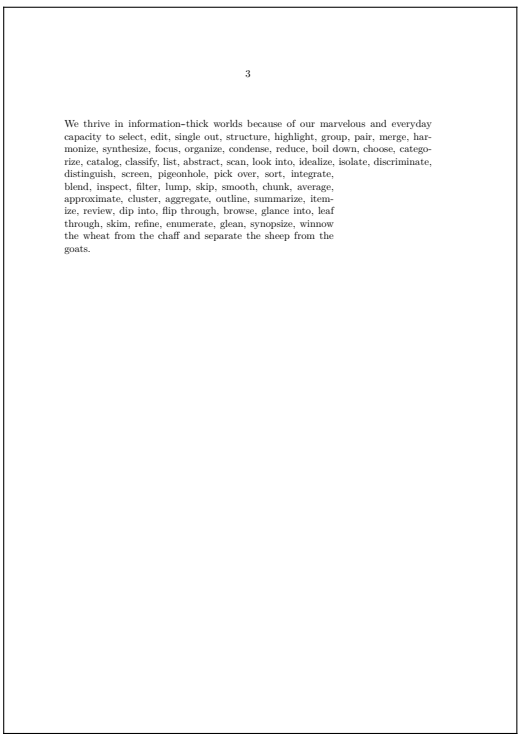
In ConTEXt we don't use `\parshape` a lot. It is used in for instance side floats but even ther MetaFun, and the manual also needed an update, one of the examples in that manual that



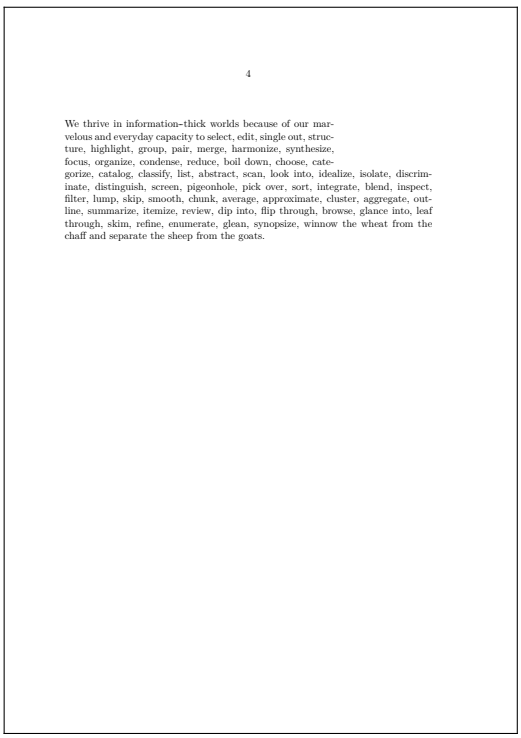

```
\hangafter +4
\hangindent +4cm
```




```
\hangafter -4
\hangindent +4cm
```




```
\hangafter +4
\hangindent -4cm
```




```
\hangafter -4
\hangindent -4cm
```

Figure 11.1 Hanging indentation

Shapes

One_important_property_of_the_\parshape_mechanism_is_that_it_works_per_paragraph._You_define_a_shape_in_terms_of_a_left_margin_and_width_of_a_line._The_shape_has_a_fixed_number_of_such_pairs_and_when_there_is_more_content,_the_last_one_is_used_for_the_rest_of_the_lines._When_the_paragraph_is_finished,_the_shape_is_forgotten._[18]_

The_high_level_interface_is_a_follow_up_on_the_example_in_the_MetaFun_manual_and_uses_shapes_that_carry_over_to_the_next_paragraph._In_addition_we_can_cycle_over_a_shape._In_this_interface_shapes_are_defined_using_keyword._Here_are_some_examples:_

```
\startparagraphshape[test]
    left 1mm right 1mm
    left 5mm right 5mm
\stopparagraphshape
```

This_shape_has_only_two_entries_so_the_first_line_will_have_a_1mm_margin_while_later_lines_will_get_5mm_margins._This_translates_into_a_\parshape_like:_

```
\parshape 2
    1mm \dimexpr\hsize-1mm\relax
    5mm \dimexpr\hsize-5mm\relax
```

Watch_the_number_2:_it_tells_how_many_specification_lines_follow._As_you_see,_we_need_to_calculate_the_width._

```
\startparagraphshape[test]
    left 1mm right 1mm
    left 5mm right 5mm
    repeat
\stopparagraphshape
```

This_variant_will_alternate_between_1mm_and_5mm_margins._The_repeating_feature_is_translated_as_follows._Maybe_at_some_point_I_will_introduce_a_few_more_options._

```
\parshape 2 options 1
    1mm \dimexpr\hsize-1mm\relax
    5mm \dimexpr\hsize-5mm\relax
```

A_shape_can_have_some_repetition,_and_we_can_save_keystrokes_by_copying_the_last_entry._The_resulting_\parshape_becomes_rather_long._

[18] Not discussed here is a variant that might end up in LuaMetaTEX that works with the progression, i.e. takes the height of the content so far into account. This is somewhat tricky because for that to work vertical skips need to be frozen, which is no real big deal but has to be done careful in the code.

```
\startparagraphshape[test]
    left 1mm right 1mm
    left 2mm right 2mm
    left 3mm right 3mm
    copy 8
    left 4mm right 4mm
    left 5mm right 5mm
    left 5mm hsize 10cm
\stopparagraphshape
```

Also_watch_the_hsize_keyword:_we_don't_calculate_the_hsize_from_the_left_and_right_values_but_explicitly_set_it._

```
\startparagraphshape[test]
    left 1mm right 1mm
    right 3mm
    left 5mm right 5mm
    repeat
\stopparagraphshape
```

When_a_right_keywords_comes_first_the_left_is_assumed_to_be_zero._In_the_examples_that_follow_we_will_use_a_couple_of_definitions:_

```
\startparagraphshape[test]
    both 1mm both 2mm both 3mm both 4mm both 5mm both 6mm
    both 7mm both 6mm both 5mm both 4mm both 3mm both 2mm
\stopparagraphshape

\startparagraphshape[test-repeat]
    both 1mm both 2mm both 3mm both 4mm both 5mm both 6mm
    both 7mm both 6mm both 5mm both 4mm both 3mm both 2mm
    repeat
\stopparagraphshape
```

The_last_one_could_also_be_defines_as:_

```
\startparagraphshape[test-repeat]
    \rawparagraphshape{test} repeat
\stopparagraphshape
```

In_the_previous_code_we_already_introduced_the_repeat_option._This_will_make_the_shape_repeat_at_the_engine_level_when_the_shape_runs_out_of_specified_lines._In_the_application_of_a_shape_definition_we_can_specify_a_method_to_be_used_and_that_deter-

mine if the next paragraph will start where we left off and discard afterwards (`shift`) or that we move the discarded lines up front so that we never run out of lines (`cycle`). It sounds complicated but just keep in mind that `repeat` is part of the `\parshape` and act within a paragraph while `shift` and `cycle` are applied when a new paragraph is started.

In figure 11.2 you see the following applied:

```
\startshapedparagraph[list=test]
    \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph

\startshapedparagraph[list=test-repeat]
    \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph
```

In figure 11.3 we use this instead:

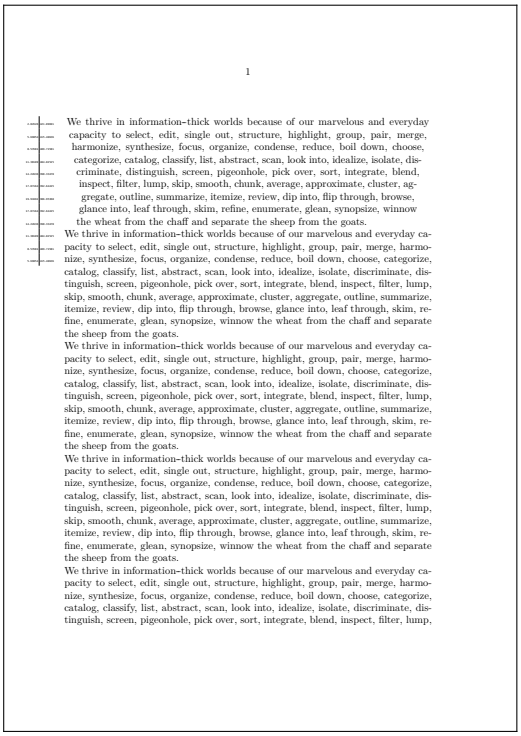
```
\startshapedparagraph[list=test,method=shift]
    \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph
```

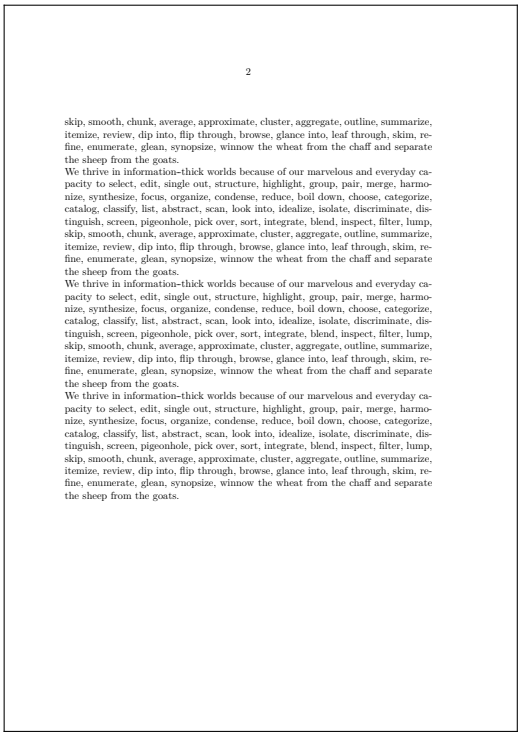
Finally, in figure 11.4 we use:

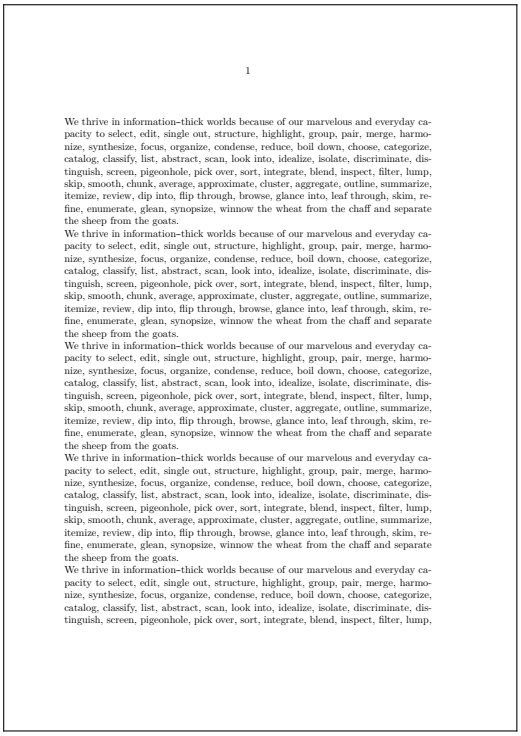
```
\startshapedparagraph[list=test,method=cycle]
    \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph
```

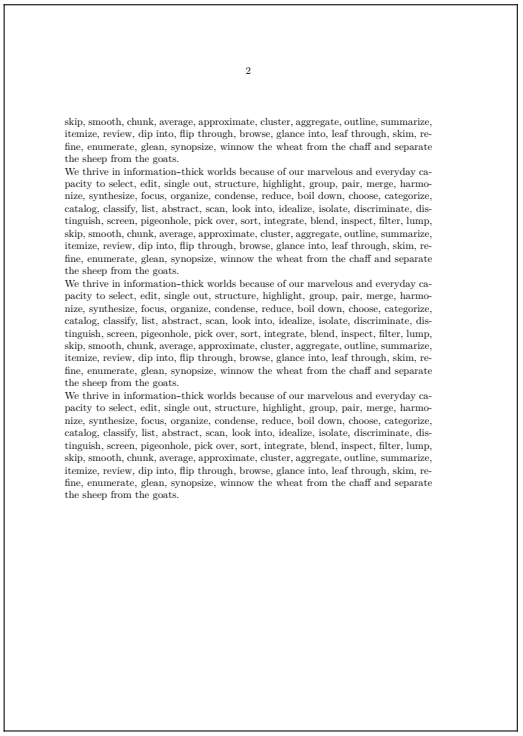
These examples are probably too small to see the details but you can run them yourself or zoom in on the details. In the margin we show the values used. Here is a simple example of (non) poetry. There are other environments that can be used instead but this makes a good example anyway.

```
\startparagraphshape[test]
    left 0em right 0em
    left 1em right 0em
    repeat
\stopparagraphshape

\startshapedparagraph[list=test,method=cycle]
    verse line 1.1\crlf verse line 2.1\crlf
    verse line 3.1\crlf verse line 4.1\par
    verse line 1.2\crlf verse line 2.2\crlf
```









1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,

2

skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

discard, finite shape, page 1 discard, finite shape, page 2

1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,

2

skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

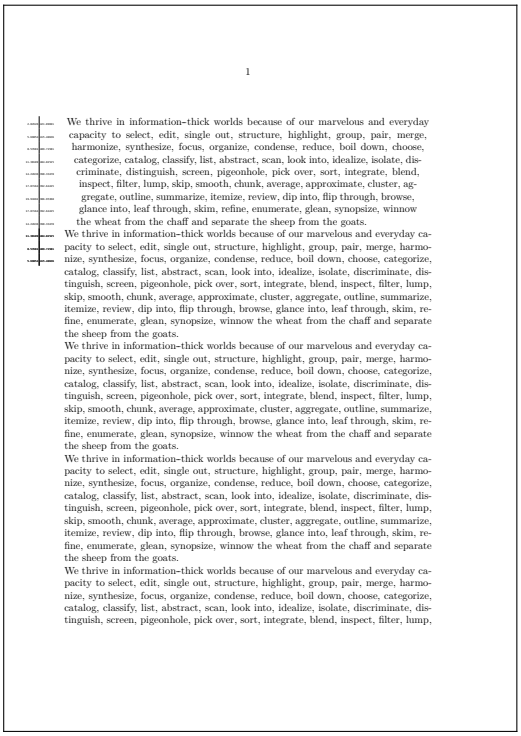
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

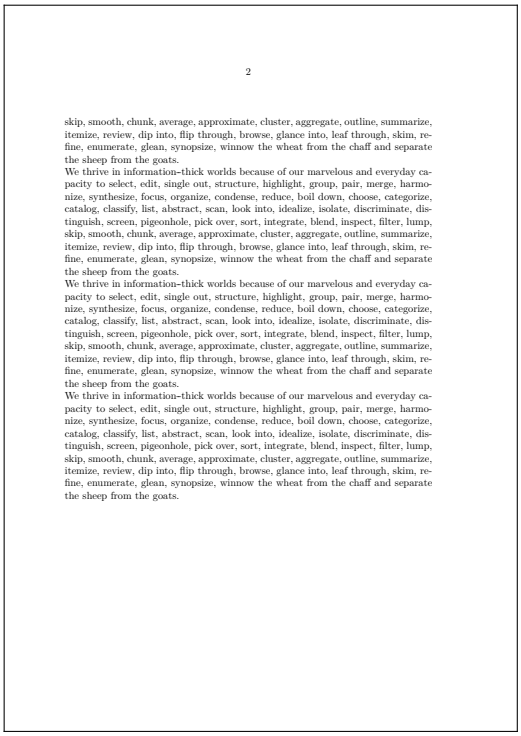
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

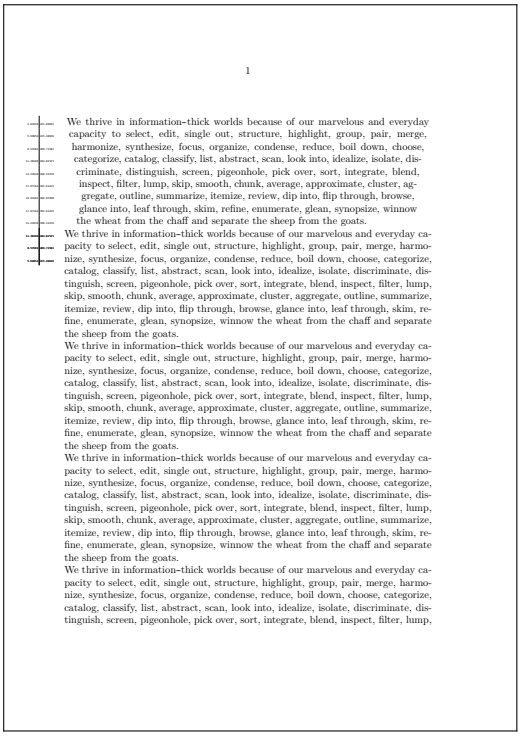
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

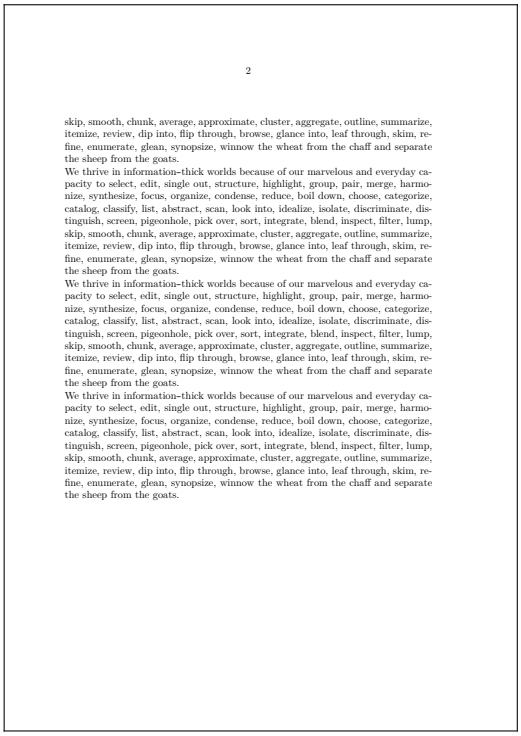
discard, repeat in shape, page 1 discard, repeat in shape, page 2

Figure 11.2 Discarded shaping









shift, finite shape, page 1

1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,

shift, finite shape, page 2

2

skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

shift, repeat in shape, page 1

1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,

shift, repeat in shape, page 2

2

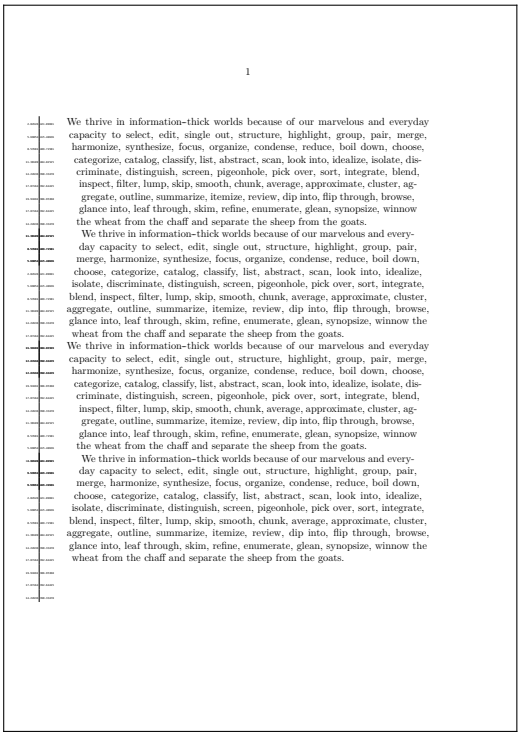
skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

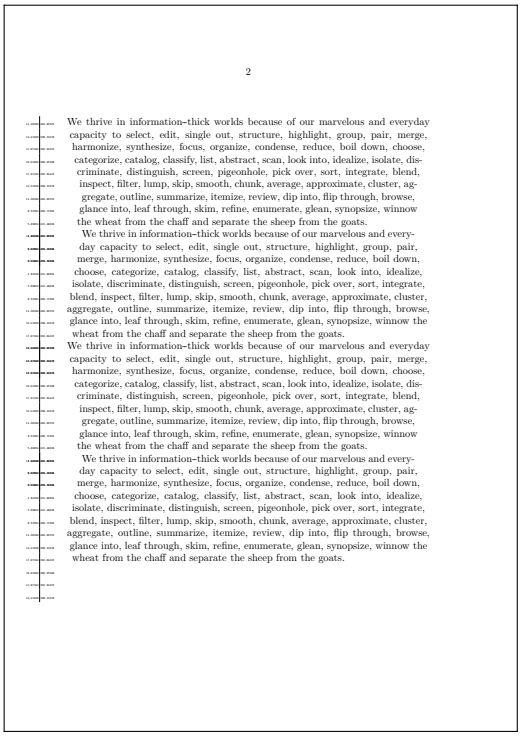
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

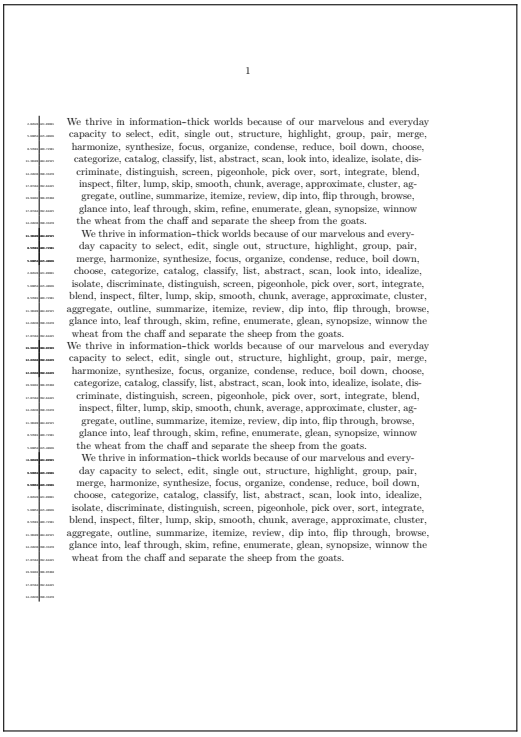
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

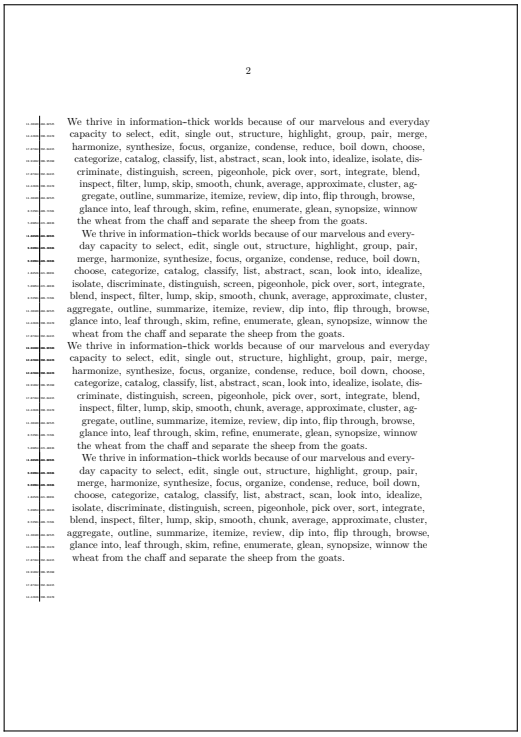
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 11.3 Shifted shaping









1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

cycle, finite shape, page 1

2

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

cycle, finite shape, page 2

1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

cycle, repeat in shape, page 1

2

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

cycle, repeat in shape, page 2

Figure 11.4 Cycled shaping

```
    verse line 3.2\crlf verse line 4.2\crlf
    verse line 5.2\crlf verse line 6.2\par
\stopshapedparagraph
```

verse_line_1.1
  verse_line_2.1
verse_line_3.1
  verse_line_4.1

verse_line_1.2
  verse_line_2.2
verse_line_3.2
  verse_line_4.2
verse_line_5.2
  verse_line_6.2

Because_the_idea_for_this_feature_originates_in_MetaFun,_we_will_now_kick_in_some_MetaPost._The_following_code_creates_a_shape_for_a_circle._We_use_a_2mm_offset_here:_

```
\startuseMPgraphic{circle}
    path p ; p := fullcircle scaled TextWidth ;
    build_parshape(p,
        2mm, 0, 0,
        LineHeight, StrutHeight, StrutDepth, StrutHeight
    ) ;
\stopuseMPgraphic
```

We_plug_this_into_the_already_described_macros:_

```
\startshapedparagraph[mp=circle]%
    \setupalign[verytolerant,stretch,last]%
    \samplefile{tufte}
    \samplefile{tufte}
\stopshapedparagraph
```

And_get_ourself_a_circular_shape._Watch_out,_at_this_moment_the_shape_environment_does_not_add_grouping_so_when_as_in_this_case_you_change_the_alignment_it_can_influence_the_document._

We_thrive_in_information–
thick_worlds_because_of_our_marvelous_
and_everyday_capacity_to_select,_edit,_single_
out,_structure,_highlight,_group,_pair,_merge,_harmo-

nize,_synthesize,_focus,_organize,_condense,_reduce,_boil_
down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_
into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_
over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_av-
erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_re-
view,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_re-
fine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_sep-
arate_the_sheep_from_the_goats.__We_thrive_in_information–thick_worlds_be-
cause_of_our_marvelous_and_everyday_capacity_to_select,_edit,_single_out,_struc-
ture,_highlight,_group,_pair,_merge,_harmonize,_synthesize,_focus,_organize,_con-
dense,_reduce,_boil_down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_
into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_in-
tegrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_clus-
ter,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_
into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_
wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__

Assuming_that_the_shape_definition_above_is_in_a_buffer_we_can_do_this:_

```
\startshapedparagraph[mp=circle]%
    \setupalign[verytolerant,stretch,last]%
    \samplefile{tufte}
    \samplefile{tufte}
\stopshapedparagraph
```

The_result_is_shown_in_figure_11.5._Because_all_action_happens_in_the_framed_environment,_w

```
\startuseMPgraphic{circle}
    path p ; p := fullcircle scaled \the\dimexpr\framedwidth+\framedoffset
      *2\relax ;
    build_parshape(p,
        \framedoffset, 0, 0,
        LineHeight, StrutHeight, StrutDepth, StrutHeight
    ) ;
    draw p ;
\stopuseMPgraphic
```

A_mechanism_like_this_is_often_never_completely_automatic_in_the_sense_that_you_need_
to_keep_an_eye_on_the_results._Depending_on_user_demands_more_features_can_be_
added._With_weird_shapes_you_might_want_to_set_up_the_alignment_to_be_tolerant_and_
have_some_stretch._

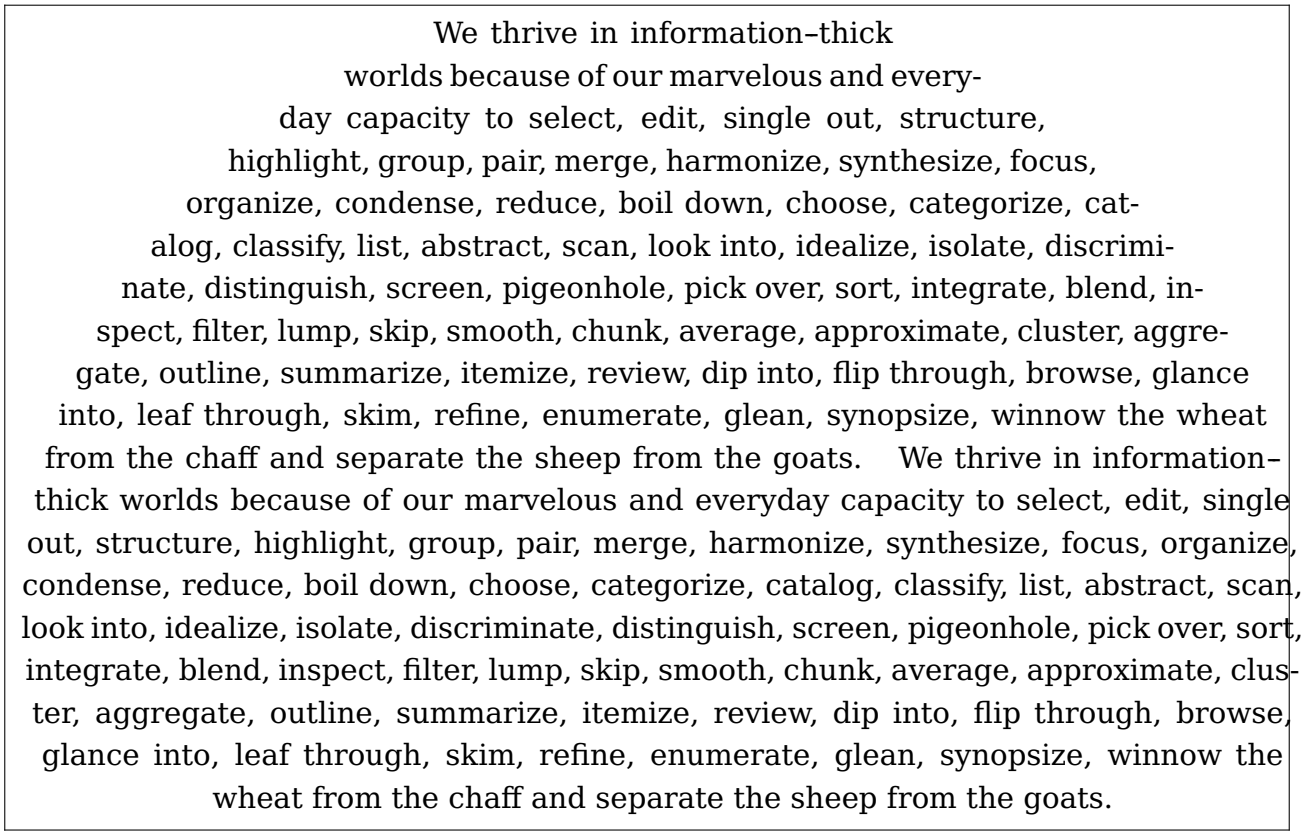


Figure 11.5 A framed circular shape

The_interface_described_in_the_MetaFun_manual_is_pretty_old,_the_time_stamp_of_the_original_code_is_mid_2000,_but_the_principles_didn't_change._The_examples_in_meta-imp-txt.now_be_written_as:_

```
\startshapetext[test 1,test 2,test 3,test 4]
  \setupalign[verytolerant,stretch,normal]%
  \samplefile{douglas} % Douglas R. Hofstadter
\stopshapetext
\startcombination[2*2]
  {\framed[offset=overlay,frame=off,background=test 1]{\getshapetext}}
      {test 1}
  {\framed[offset=overlay,frame=off,background=test 2]{\getshapetext}}
      {test 2}
  {\framed[offset=overlay,frame=off,background=test 3]{\getshapetext}}
      {test 3}
  {\framed[offset=overlay,frame=off,background=test 4]{\getshapetext}}
      {test 4}
\stopcombination
```

In_figure_11.6_we_see_the_result._Watch_how_for_two_shapes_we_have_enabled_tracing._Of_co

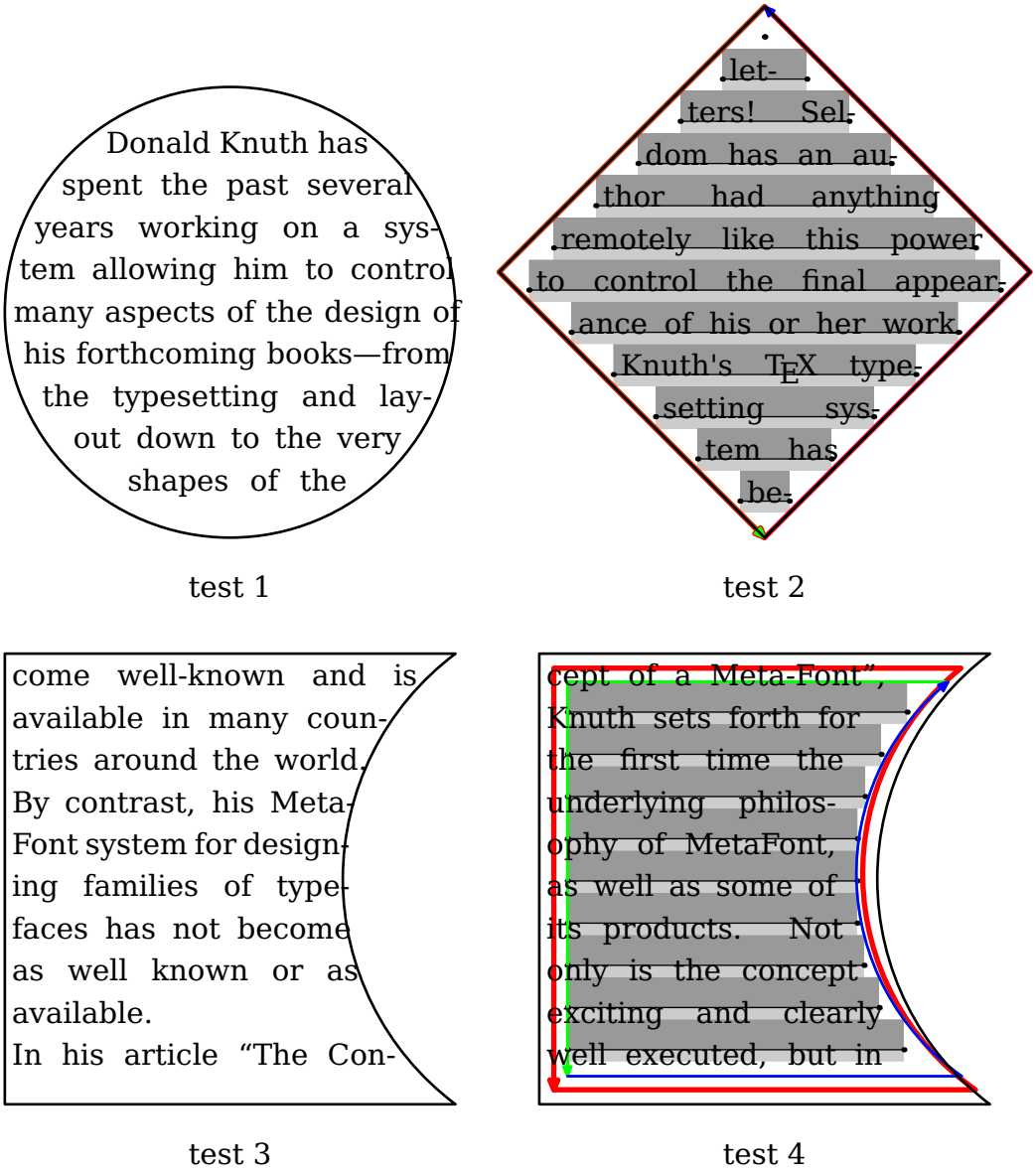


Figure 11.6

Here_is_a_bit_more_extreme_example._Again_we_use_a_circle:_

```
\startuseMPgraphic{circle}
    lmt_parshape [
        path       = fullcircle scaled 136mm,
        offset     = 2mm,
        bottomskip = - 1.5LineHeight,
    ] ;
\stopuseMPgraphic
```

But_we_output_a_longer_text:_

```
\startshapedparagraph[mp=circle,repeat=yes,method=cycle]%
    \setupalign[verytolerant,stretch,last]\dontcomplain
    {\darkred     \samplefile{tufte}}\par
    {\darkgreen   \samplefile{tufte}}\par
    {\darkblue    \samplefile{tufte}}\par
    {\darkcyan    \samplefile{tufte}}\par
    {\darkmagenta \samplefile{tufte}}\par
\stopshapedparagraph
```

We_get_a_multi-page_shape:_

We_thrive_in_information–
thick_worlds_because_of_our_marvelous_
and_everyday_capacity_to_select,_edit,_single_
out,_structure,_highlight,_group,_pair,_merge,_harmo-
nize,_synthesize,_focus,_organize,_condense,_reduce,_boil_
down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_
into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_
over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_av-
erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_re-
view,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_re-
fine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_sepa-
rate_the_sheep_from_the_goats._

We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_
capacity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_har-
monize,_synthesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_cat-
egorize,_catalog,_classify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_dis-
criminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_integrate,_blend,_in-
spect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_cluster,_ag-
gregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_
into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_
wheat_from_the_chaff_and_separate_the_sheep_from_the_goats._

We_thrive_in_information–thick_worlds_because_of_our_marvelous_
and_everyday_capacity_to_select,_edit,_single_out,_structure,_high-
light,_group,_pair,_merge,_harmonize,_synthesize,_focus,_or-
ganize,_condense,_reduce,_boil_down,_choose,_catego-
rize,_catalog,_classify,_list,_abstract,_scan,_look_
into,_idealize,_isolate,_discriminate,_distin-

guish_screen_pigeonhole_pick_
over_sort_in-
tegrate_blend_inspect_fil-
ter_lump_skip_smooth_chunk_av-
erage_approximate_cluster_aggregate_out-
line_summarize_itemize_review_dip_into_flip_through_browse_glance_
into_leaf_through_skim_refine_enumerate_glean_synop-
size_winnow_the_wheat_from_the_chaff_and_separate_the_sheep_
from_the_goats._

We_thrive_in_information-thick_worlds_because_of_our_marvelous_and_
everyday_capacity_to_select_edit_single_out_structure_highlight_group_pair_merge_har-
monize_synthesize_focus_organize_condense_reduce_boil_down_choose_cat-
egorize_catalog_classify_list_abstract_scan_look_into_idealize_isolate_dis-
criminate_distinguish_screen_pigeonhole_pick_over_sort_integrate_blend_in-
spect_filter_lump_skip_smooth_chunk_average_approximate_cluster_aggre-
gate_outline_summarize_itemize_review_dip_into_flip_through_browse_glance_
into_leaf_through_skim_refine_enumerate_glean_synopsize_winnow_the_wheat_
from_the_chaff_and_separate_the_sheep_from_the_goats._

We_thrive_in_information-thick_worlds_because_of_our_marvelous_and_everyday_
capacity_to_select_edit_single_out_structure_highlight_group_pair_merge_har-
monize_synthesize_focus_organize_condense_reduce_boil_down_choose_cat-
egorize_catalog_classify_list_abstract_scan_look_into_idealize_isolate_dis-
criminate_distinguish_screen_pigeonhole_pick_over_sort_integrate_blend_in-
spect_filter_lump_skip_smooth_chunk_average_approximate_clus-
ter_aggregate_outline_summarize_itemize_review_dip_into_flip_
through_browse_glance_into_leaf_through_skim_refine_enu-
merate_glean_synopsize_winnow_the_wheat_from_the_
chaff_and_separate_the_sheep_from_the_goats._

Compare_this_with:_

```
\startshapedparagraph[mp=circle,repeat=yes,method=cycle]%
    \setupalign[verytolerant,stretch,last]\dontcomplain
    {\darkred     \samplefile{tufte}}
    {\darkgreen   \samplefile{tufte}}
    {\darkblue    \samplefile{tufte}}
    {\darkcyan    \samplefile{tufte}}
    {\darkmagenta \samplefile{tufte}}
\stopshapedparagraph
```

Which_gives:_

We_thrive_in_information–
thick_worlds_because_of_our_marvelous_
and_everyday_capacity_to_select,_edit,_single_
out,_structure,_highlight,_group,_pair,_merge,_harmo-
nize,_synthesize,_focus,_organize,_condense,_reduce,_boil_
down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_
into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_
over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_av-
erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_re-
view,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_re-
fine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_sep-
arate_the_sheep_from_the_goats.__We_thrive_in_information–thick_worlds_be-
cause_of_our_marvelous_and_everyday_capacity_to_select,_edit,_single_out,_struc-
ture,_highlight,_group,_pair,_merge,_harmonize,_synthesize,_focus,_organize,_con-
dense,_reduce,_boil_down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_
into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_in-
tegrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_clus-
ter,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_
into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_
wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__We_thrive_
in_information–thick_worlds_because_of_our_marvelous_and_everyday_
capacity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_har-
monize,_synthesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_cat-
egorize,_catalog,_classify,_list,_abstract,_scan,_look_into,_ide-
alize,_isolate,_discriminate,_distinguish,_screen,_pigeon-
hole,_pick_over,_sort,_integrate,_blend,_inspect,_fil-
ter,_lump,_skip,_smooth,_chunk,_average,_ap-
proximate,_cluster,_aggregate,_out-
line,_summa-
rize,_itemize,_review,_dip_
into,_flip_through,_browse,_glance_into,_leaf_
through,_skim,_refine,_enumerate,_glean,_synop-
size,_winnow_the_wheat_from_the_chaff_and_separate_
the_sheep_from_the_goats.__We_thrive_in_information–thick_
worlds_because_of_our_marvelous_and_everyday_capacity_to_se-
lect,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_har-
monize,_synthesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_cat-
egorize,_catalog,_classify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_dis-
criminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_integrate,_blend,_in-

spect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__We_thrive_in_information-thick_worlds_because_of_our_marvelous_and_everyday_capacity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_synthesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__

Here_the_bottomskip_takes_care_of_subtle_rounding_issues_as_well_as_discarding_the_last_line_in_the_shape_so_that_we_get_nicer_continuation._There_is_no_full_automated_solution_for_all_you_can_come_up_with._

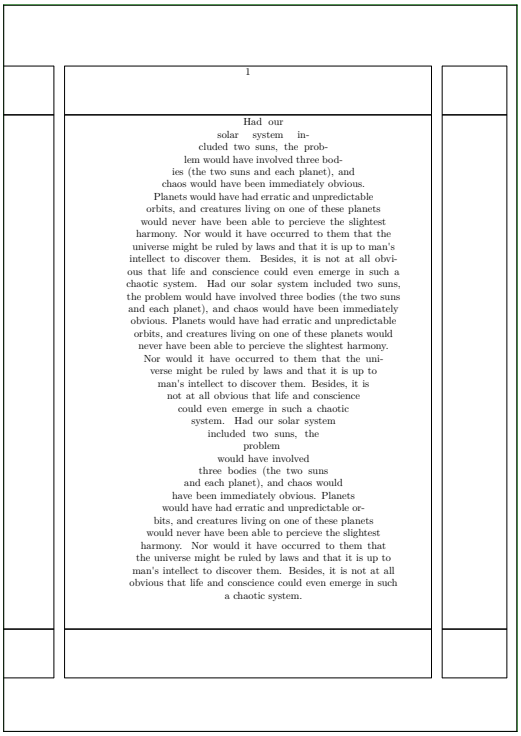
Mixing_a_MetaPost_specification_into_a_regular_one_is_also_possible._The_next_example_demonstrates_this_as_well_as_the_option_to_remove_some_lines_from_a_specification:_

```
\startparagraphshape[test]
    left 0em right 0em
    left 1em right 0em
    metapost {circle}
    delete 3
    metapost {circle,circle,circle}
    delete 7
    metapost {circle}
    repeat
\stopparagraphshape
```

You_can_combine_a_shape_with_narrowing_a_paragraph._Watch_the_absolute_keyword_in_the_next_code._The_result_is_shown_in_figure_11.7._

```
\startuseMPgraphic{circle}
    lmt_parshape [
        path       = fullcircle scaled TextWidth,
        bottomskip = - 1.5LineHeight,
    ] ;
\stopuseMPgraphic
```

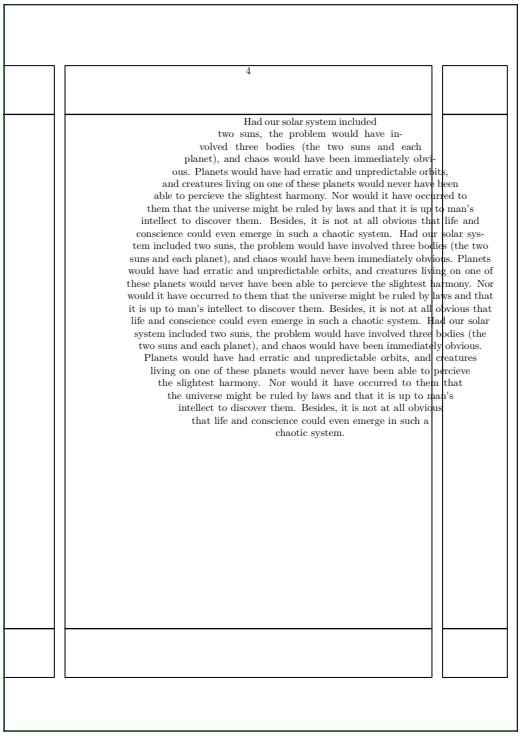
```
\startparagraphshape[test-1]
    metapost {circle} repeat
\stopparagraphshape

\startparagraphshape[test-2]
    absolute left metapost {circle} repeat
\stopparagraphshape

\startparagraphshape[test-3]
    absolute right metapost {circle} repeat
\stopparagraphshape

\startparagraphshape[test-4]
    absolute both metapost {circle} repeat
\stopparagraphshape

\showframe

\startnarrower[4*left,2*right]
    \startshapedparagraph[list=test-1,repeat=yes,method=repeat]%
        \setupalign[verytolerant,stretch,last]\dontcomplain
        \dorecurse{3}{\samplefile{thuan}}
    \stopshapedparagraph
    \page
    \startshapedparagraph[list=test-2,repeat=yes,method=repeat]%
        \setupalign[verytolerant,stretch,last]\dontcomplain
        \dorecurse{3}{\samplefile{thuan}}
    \stopshapedparagraph
    \page
    \startshapedparagraph[list=test-3,repeat=yes,method=repeat]%
        \setupalign[verytolerant,stretch,last]\dontcomplain
        \dorecurse{3}{\samplefile{thuan}}
    \stopshapedparagraph
    \page
    \startshapedparagraph[list=test-4,repeat=yes,method=repeat]%
        \setupalign[verytolerant,stretch,last]\dontcomplain
        \dorecurse{3}{\samplefile{thuan}}
    \stopshapedparagraph
\stopnarrower
```



test 1

test 2, left

test 3, right



test 4, both

Figure 11.7 Skip compensation

The_shape_mechanism_has_a_few_more_tricks_but_these_are_really_meant_for_usage_in_specific_situations,_where_one_knows_what_one_deals_with._The_following_examples_are_visualized_in_figure_11.8._

```
\useMPlibrary[dum]
\usemodule[article-basics]

\startbuffer
    \externalfigure[dummy][width=6cm]
\stopbuffer

\startshapedparagraph[text=\getbuffer]
    \dorecurse{3}{\samplefile{ward}\par}
\stopshapedparagraph

\page

\startshapedparagraph[text=\getbuffer,distance=1em]
    \dorecurse{3}{\samplefile{ward}\par}
\stopshapedparagraph

\page

\startshapedparagraph[text=\getbuffer,distance=1em,
        hoffset=-2em]
    \dorecurse{3}{\samplefile{ward}\par}
\stopshapedparagraph

\page

\startshapedparagraph[text=\getbuffer,distance=1em,
        voffset=-2ex,hoffset=-2em]
    \dorecurse{3}{\samplefile{ward}\par}
\stopshapedparagraph

\page

\startshapedparagraph[text=\getbuffer,distance=1em,
        voffset=-2ex,hoffset=-2em,lines=1]
    \dorecurse{3}{\samplefile{ward}\par}
\stopshapedparagraph
```

```
\page

\startshapedparagraph[width=4cm,lines=4]
    \dorecurse{3}{\samplefile{ward}\par}
\stopshapedparagraph
```

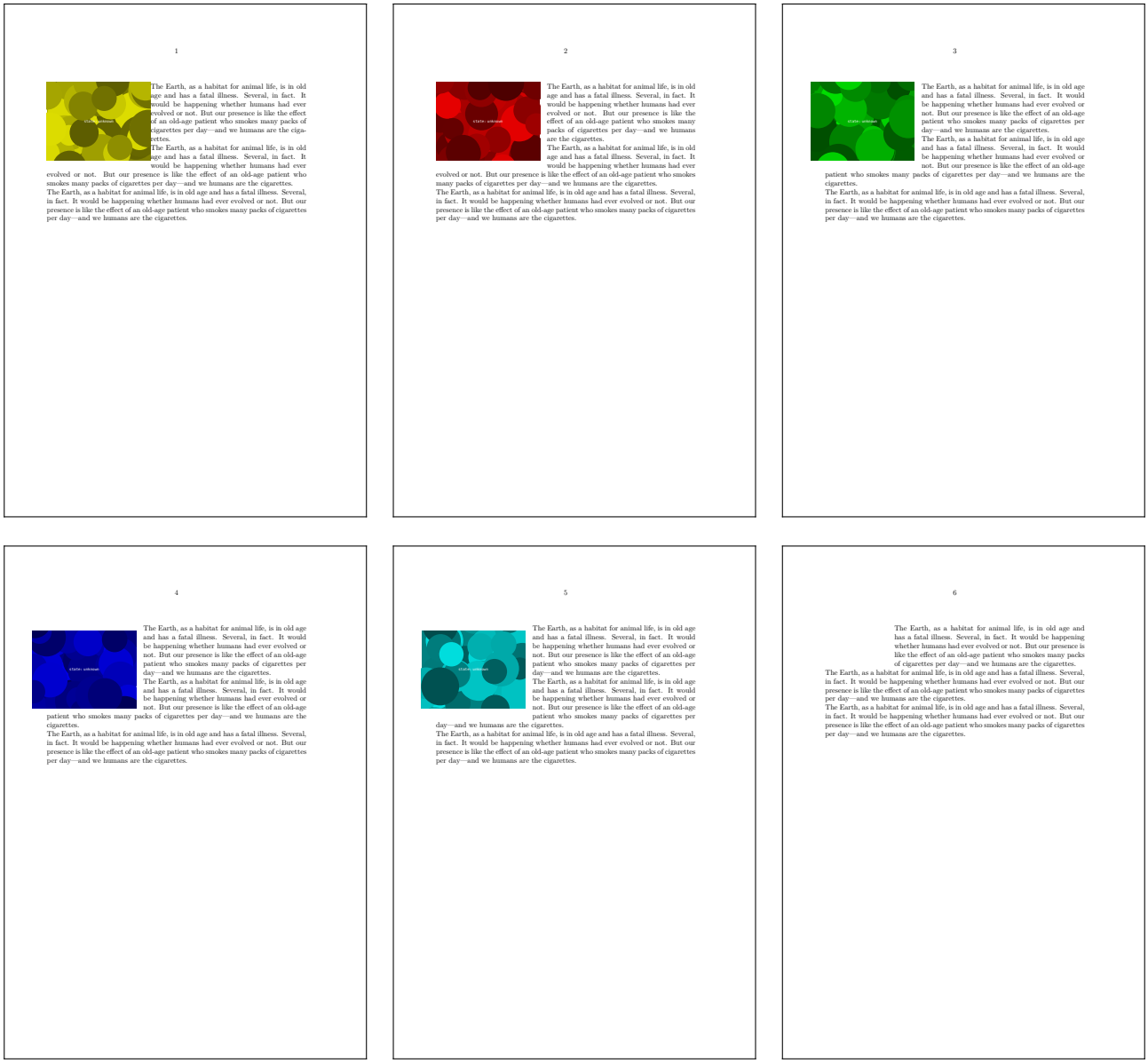


Figure 11.8 Flow around something

11.7 Modes

*todo:_some_of_the_side_effects_of_so_called_modes_*

11.8 Leaders

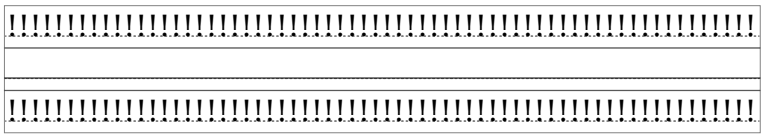
Leaders are a basic feature that users probably never run into directly. They repeat content till it fits the specified width which can be stretched out. The content is typeset once and it is the backend that does the real work of repetition.

```
\strut\leaders \hbox{!}\hfill\strut
\strut\xleaders\hbox{!}\hfill\strut
\strut\cleaders\hbox{!}\hfill\strut
\strut\gleaders\hbox{!}\hfill\strut
```

Here `\leaders` starts at the left edge and are repeats the box as long as it fits, `\xleaders` till the edges and `\cleaders` centers the lot. The `\gleaders` primitive (which is not in orginal TEX) takes the outer box as reference and further behaves like `\cleaders`.

The leader primitives take box or rule but in LuaMetaTEX a glyph can also be specified, wh

```
\ruledvbox \bgroup \hsize 10cm
    \strut\cleaders\hbox{!}\hfill\strut
\egroup

\ruledvbox \bgroup \hsize 10cm
    \strut\cleaders\hrule\hfill\strut
\egroup

\ruledvbox \bgroup \hsize 10cm
    \strut\cleaders\glyph`!\hfill\strut
\egroup
```



The LuaMetaTEX engine also introduced `\uleaders`

We show three boxes, a regular one first (red):

```
x xx xxx xxxx
```

```
\ruledhbox{L\hss R}\space
x xx xxx xxxx
```

The_second_one_(blue)_is_also_a_box_but_one_that_stretches_upto_100pt_and_is_in_a_later_stage,_when_the_paragraph_has_been_built,_is_repackaged_to_the_effective_width._The_third_example_(green)_leaves_out_the_background._

In_ConTEXt_we_have_wrapped_this_feature_in_the_adaptive_box_mechanism,_so_here_a_few_a_f

```
\startsetups adaptive:test:a
    \setbox\usedadaptivebox\vbox to \usedadaptivetotal \bgroup
        \externalfigure
          [cow.pdf]
          [width=\framedmaxwidth,
           frame=on,
           height=\usedadaptivetotal]%
   \egroup
\stopsetups

\startsetups adaptive:test:b
    \setbox\usedadaptivebox\vbox to \usedadaptivetotal \bgroup
        \externalfigure
          [cow.pdf]
          [width=\usedadaptivewidth,
           frame=on,
           height=\usedadaptivetotal]%
   \egroup
\stopsetups
```

We_use_this_as_follows_(see_figure_11.9_for_the_result):_

```
\framed[height=18cm,align=middle,adaptive=yes,top=,bottom=] {%
    \begstrut \samplefile{tufte} \endstrut
    \par
```

```
    \adaptivevbox
      [strut=yes,setups=adaptive:test:a]
      {\showstruts\strut\hsize5cm\hss}%
    \par
    \adaptivevbox
      [strut=yes,setups=adaptive:test:b]
      {\showstruts\strut\hsize5cm\hss}%
    \par
    \begstrut \samplefile{tufte} \endstrut
}
```

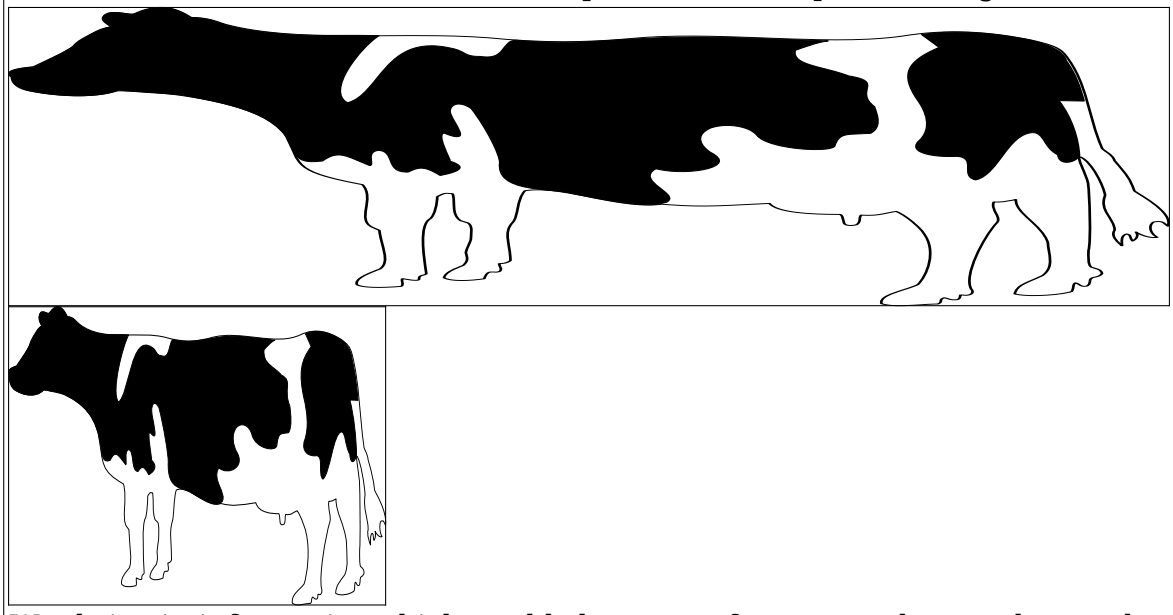
Here_is_one_that_you_can_test_yourself:_

```
\startsetups adaptive:test
    \setbox\usedadaptivebox\vbox to \usedadaptivetotal \bgroup
        \externalfigure
          [cow.pdf]
          [width=\usedadaptivewidth,
           height=\usedadaptivetotal]%
   \egroup
\stopsetups

\ruledvbox to \textheight {
    \par \begstrut \samplefile{tufte} \endstrut \par
    \adaptivevbox[strut=yes,setups=adaptive:test]{\hsize\textwidth\hss}
    \par \begstrut \samplefile{tufte} \endstrut
}
```

The_next_example_comes_from_the_test_suite_(where_it_runs_over_many_pages_in_order_to_illustrate_the_idea):_

```
\startMPdefinitions
    def TickTock =
        interim linecap := squared;
        save p ; path p ;
        p := fullsquare xysized(AdaptiveWidth,.9(AdaptiveHeight+AdaptiveDepth))
;
        fill p withcolor AdaptiveColor ;
        draw bottomboundary (p enlarged (-AdaptiveThickness) )
            withdashes (3*AdaptiveThickness)
            withpen pencircle scaled AdaptiveThickness
```

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.



We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 11.9

```
            withcolor white ;
    enddef ;
\stopMPdefinitions

\startsetups adaptive:test
    \setbox\usedadaptivebox\hbox
```

```
        to         \usedadaptivewidth
        yoffset -.9\usedadaptivedepth
    \bgroup
        \hss
        \startMPcode
            TickTock ;
        \stopMPcode
        \hss
   \egroup
\stopsetups

\definecolor[adaptive:tick][.25(blue,green)]
\definecolor[adaptive:tock][.75(blue,green)]

\defineadaptive
  [tick]
  [setups=adaptive:test,
   color=adaptive:tick,
   foregroundcolor=white,
   foregroundstyle=\infofont,
   strut=yes]

\defineadaptive
  [tock]
  [tick]
  [color=adaptive:tock]

\dostepwiserecurse{8}{12}{1}{%
    \dostepwiserecurse{5}{15}{1}{%
        this~#1.##1 is~#1.##1 test~#1.##1
        \ifodd##1\relax
            \adaptivebox[tick]{\hss tick #1.##1\hss}
        \else
            \adaptivebox[tock]{\hss tock #1.##1\hss}
        \fi
    }
}
```

this 8.5_is 8.5_test 8.5_ tick_8.5 _this 8.6_is 8.6_test 8.6_ tock_8.6 _this 8.7_is 8.7_test 8.7_ _this 8.8_is 8.8_test 8.8_ tock_8.8 _this 8.9_is 8.9_test 8.9_ tick_8.9 _this 8.10_is 8.10_test 8.10_ _this 8.11_is 8.11_test 8.11_ tick_8.11 _this 8.12_is 8.12_test 8.12_ tock_8.12 _this 8.13_is 8.13_test 8.13_

_this 8.14_is 8.14_test 8.14_  _this 8.15_is 8.15_test 8.15_  __this 9.5_is 9.5_test 9.5_
_this 9.6_is 9.6_test 9.6_  _this 9.7_is 9.7_test 9.7_  _this 9.8_is 9.8_test 9.8_
_this 9.9_is 9.9_test 9.9_  _this 9.10_is 9.10_test 9.10_  _this 9.11_is 9.11_test 9.11_
_this 9.12_is 9.12_test 9.12_  _this 9.13_is 9.13_test 9.13_  _this 9.14_is 9.14_test 9.14_
_this 9.15_is 9.15_test 9.15_  __this 10.5_is 10.5_test 10.5_  _this 10.6_is 10.6_test 10.6_
_this 10.7_is 10.7_test 10.7_  _this 10.8_is 10.8_test 10.8_  _this 10.9_is 10.9_test 10.9_
_this 10.10_is 10.10_test 10.10_  _this 10.11_is 10.11_test 10.11_  _this 10.12_is 10.12_te
_this 10.13_is 10.13_test 10.13_  _this 10.14_is 10.14_test 10.14_  _this 10.15_is 10.15_te
__this 11.5_is 11.5_test 11.5_  _this 11.6_is 11.6_test 11.6_  _this 11.7_is 11.7_test 11.7_
_this 11.8_is 11.8_test 11.8_  _this 11.9_is 11.9_test 11.9_  _this 11.10_is 11.10_test 11.10_
_this 11.11_is 11.11_test 11.11_  _this 11.12_is 11.12_test 11.12_  _this 11.13_is 11.13_te
_this 11.14_is 11.14_test 11.14_  _this 11.15_is 11.15_test 11.15_  __this 12.5_is 12.5_test
_this 12.6_is 12.6_test 12.6_  _this 12.7_is 12.7_test 12.7_  _this 12.8_is 12.8_test 12.8_
_this 12.9_is 12.9_test 12.9_  _this 12.10_is 12.10_test 12.10_  _this 12.11_is 12.11_test 1
_this 12.12_is 12.12_test 12.12_  _this 12.13_is 12.13_test 12.13_  _this 12.14_is 12.14_te
_this 12.15_is 12.15_test 12.15____

In_the_next_example_the_graphics_adapt_to_the_available_space:_

```
\startsetups adaptive:test
    \setbox\usedadaptivebox\hbox
        to       \usedadaptivewidth
        yoffset -\usedadaptivedepth
    \bgroup
        \externalfigure
          [cow.pdf]
          [width=\usedadaptivewidth,
           height=\dimexpr\usedadaptivetotal\relax]%
    \egroup
\stopsetups

\dostepwiserecurse{1}{50}{1}{%
    this~#1 is~#1 test~#1
    {\adaptivebox[strut=yes,setups=adaptive:test]{}}
}
```

this 1_is 1_test 1_  _this 2_is 2_test 2_  _this 3_is 3_test 3_  _this 4_is 4_test 4_
_this 5_is 5_test 5_  _this 6_is 6_test 6_  _this 7_is 7_test 7_  _this 8_is 8_test 8_
_this 9_is 9_test 9_  _this 10_is 10_test 10__this 11_is 11_test 11__this 12_is 12_test 12_
_this 13_is 13_test 13_  _this 14_is 14_test 14_  _this 15_is 15_test 15_
_this 16_is 16_test 16_  _this 17_is 17_test 17_  _this 18_is 18_test 18_

_this 19_is 19_test 19_  _this 20_is 20_test 20_  _this 21_is 21_test 21_
_this 22_is 22_test 22_  _this 23_is 23_test 23_  _this 24_is 24_test 24_
_this 25_is 25_test 25_  _this 26_is 26_test 26_  _this 27_is 27_test 27_
_this 28_is 28_test 28_  _this 29_is 29_test 29_  _this 30_is 30_test 30_
_this 31_is 31_test 31_  _this 32_is 32_test 32_  _this 33_is 33_test 33_
_this 34_is 34_test 34_  _this 35_is 35_test 35_  _this 36_is 36_test 36_
_this 37_is 37_test 37_  _this 38_is 38_test 38_  _this 39_is 39_test 39_
_this 40_is 40_test 40_  _this 41_is 41_test 41_  _this 42_is 42_test 42_
_this 43_is 43_test 43_  _this 44_is 44_test 44_  _this 45_is 45_test 45_
_this 46_is 46_test 46_  _this 47_is 47_test 47_  _this 48_is 48_test 48_
_this 49_is 49_test 49_  _this 50_is 50_test 50_  __

11.9 Prevdepth

The_depth_of_a_box_is_normally_positive_but_rules_can_have_a_negative_depth_in_order_to_get_a_rule_above_the_baseline._When_TEX_was_written_the_assumption_was_that_a_nega
MetaTEX._However,_as_dealing_with_the_property_is_somewhat_special_in_the_engine_you_shou

```
line 1\par line 2 \par \nointerlineskip line 3 \par
```

Assuming_that_we_haven't_set_any_inter_paragraph_spacing_this_gives:_

line_1
line_2
line_3

Here_`\nointerlineskip`_is_(normally)_defined_as:_

```
\prevdepth-1000pt
```

although_in_ConTEXt_we_use_`\ignoredepthcriterion`_instead_of_the_hard_coded_dimension._W

```
\ruledhbox \bgroup
    \PrevTest{-10.0pt}\quad
    \PrevTest{-20.0pt}\quad
    \PrevTest{-49.9pt}\quad
    \PrevTest{-50.0pt}\quad
    \PrevTest{-50.1pt}\quad
    \PrevTest{-60.0pt}\quad
    \PrevTest{-80.0pt}%
\egroup
```

In_this_example_we_set_\ignoredepthcriterion_to_−50.0pt_instead_of_the_normal_−1000pt._T
helper_is_defined_as:_

```
\def\PrevTest#1%
  {\setbox0\ruledhbox{\strut$\tf#1$}%
   \dp0=#1
   \vbox\bgroup\hsize4em
     FIRST\par
     \unhbox0\par
     LAST\par
   \egroup}
```

or_

```
\def\PrevTest#1%
  {\setbox0\ruledhbox{\strut$\tf#1$}%
   \dp0=#1
   \vbox\bgroup
     \ruledhbox{FIRST}\par
     \box0\par
     \ruledhbox{LAST}\par
   \egroup}
```

The_result_is_shown_in_figures_11.10_and_11.11._The_first_case_is_what_we_normally_have_in_

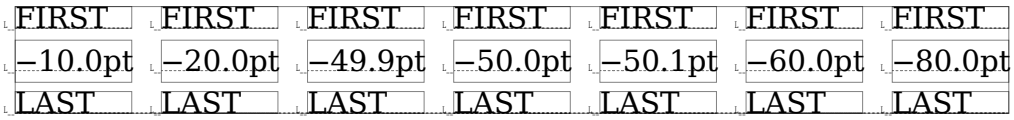


Figure 11.10

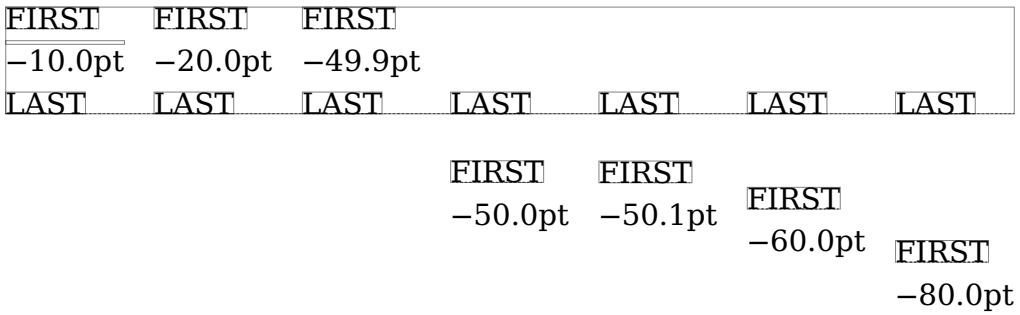


Figure 11.11

I'm_sure_one_can_use_this_effect_otherwise_than_intended_but_I_doubt_is_any_user_is_
willing_to_do_this_but_the_fact_that_we_can_lower_the_criterion_makes_for_nice_exper-
iments._Just_for_the_record,_in_figure_11.12_you_see_what_we_get_with_positive_values:_

```
\ruledhbox \bgroup
    \PrevTest{10.0pt}\quad
    \PrevTest{20.0pt}\quad
    \PrevTest{49.9pt}\quad
    \PrevTest{50.0pt}\quad
    \PrevTest{50.1pt}\quad
    \PrevTest{60.0pt}\quad
    \PrevTest{80.0pt}%
\egroup
```

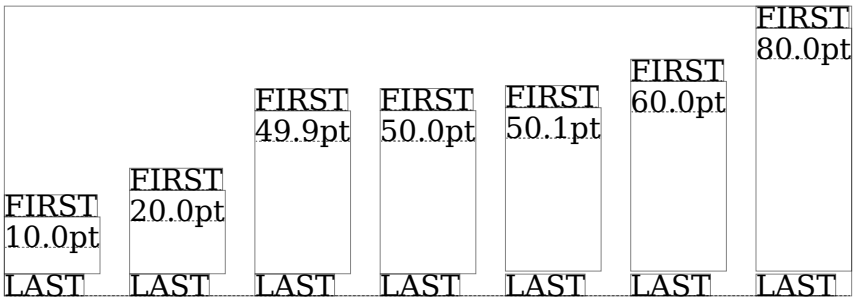


Figure 11.12

Watch_the_interline_skip_kicking_in_when_we_make_the_depth_larger_than_in_`\ignore-depthcriterion`_being_50pt._

11.10 Normalization

*todo:_users_don't_need_to_bother_about_this_but_it_might_be_interesting_anyway_*

11.11 Dirty tricks

*todo:_explain_example_for_combining_paragraphs_*

11.12 Penalties

In_figiure_11.13_we_demonstrate_the_(accumulated)_effect_of_a_few_penalty_arrays_that_you_c

```
\interlinepenalty 0
\clubpenalty      0
\widowpenalty     0
\orphanpenalty    0
\shapingpenalty   0
```

```
\clubpenalties      5   1000   2000   3000   4000   5000 % 6 -> 0
\widowpenalties     5     10     20     30     40     50 % 6 -> 0
\orphanpenalties    5      1      2      3      4      5 % 6 -> 0
\interlinepenalties 5 100000 200000 300000 400000 500000 % 6 -> 0
```

It_actually_makes_sense_to_explicitly_zero_the_last_entry_because_as_you_can_see_in_the_figure_the_last_one_gets_used_when_we_run_out_of_entries._

Can_you_guess_what_the_next_specification_does?_

```
\widowpenalties 3 options \largestspecificationoptioncode 3000 2000 1000
\clubpenalties  3 options \largestspecificationoptioncode   30   20   10
```

11.13 Par passes

Everything_comes_together_in_what_we_call_par_passes._Before_we_explain_them_first_something_about_a_feature_that_makes_setting_up_for_instance_`\widowpenalties`_easier._Here_are_a_few_definitions:_

```
\specificationdef\strictwidowpenalties      \widowpenalties  \plusthree
    \maxcount \maxcount \zerocount \relax
\specificationdef\strictwidowpenaltiestwo   \widowpenalties  \plustwo
    \maxcount \zerocount \relax
\specificationdef\strictwidowpenaltiesthree \widowpenalties  \plusthree
    \maxcount \maxcount \zerocount \relax
\specificationdef\strictwidowpenaltiesfour  \widowpenalties  \plusfour
    \maxcount \maxcount \maxcount \zerocount \relax
```

These_are_defined_in_the_core_and_hooked_into_the_alignment_interface:_

```
\installaligncommand{strictwidows}  {\strictwidowpenalties     }
\installaligncommand{strictwidows:2}{\strictwidowpenaltiestwo  }
\installaligncommand{strictwidows:3}{\strictwidowpenaltiesthree}
\installaligncommand{strictwidows:4}{\strictwidowpenaltiesfour }
```

We_also_have_four_such_‘strict’_definitions_for_club_but_only_one_for_broken_penalties._For_orphan_penalties_we_have_four_‘less’_orphan_penalties_but_for_widow,_club_and_broken_we_have_only_one._So_we_end_up_with_`lessorphans`,_`lessorphans:2`,_`lessorphans:3`,_`lessorphans:4`,_`defaultwidows`,_`defaultclubs`,_`defaultbroken`,_`strictwidows`,_`strictwidows:2`,_`strictwidows:3`,_`strictwidows:4`,_`strictclubs`,_`strictclubs:2`,_`strictclubs:3`,_`strictclubs:4`_and_`strictbroken`._

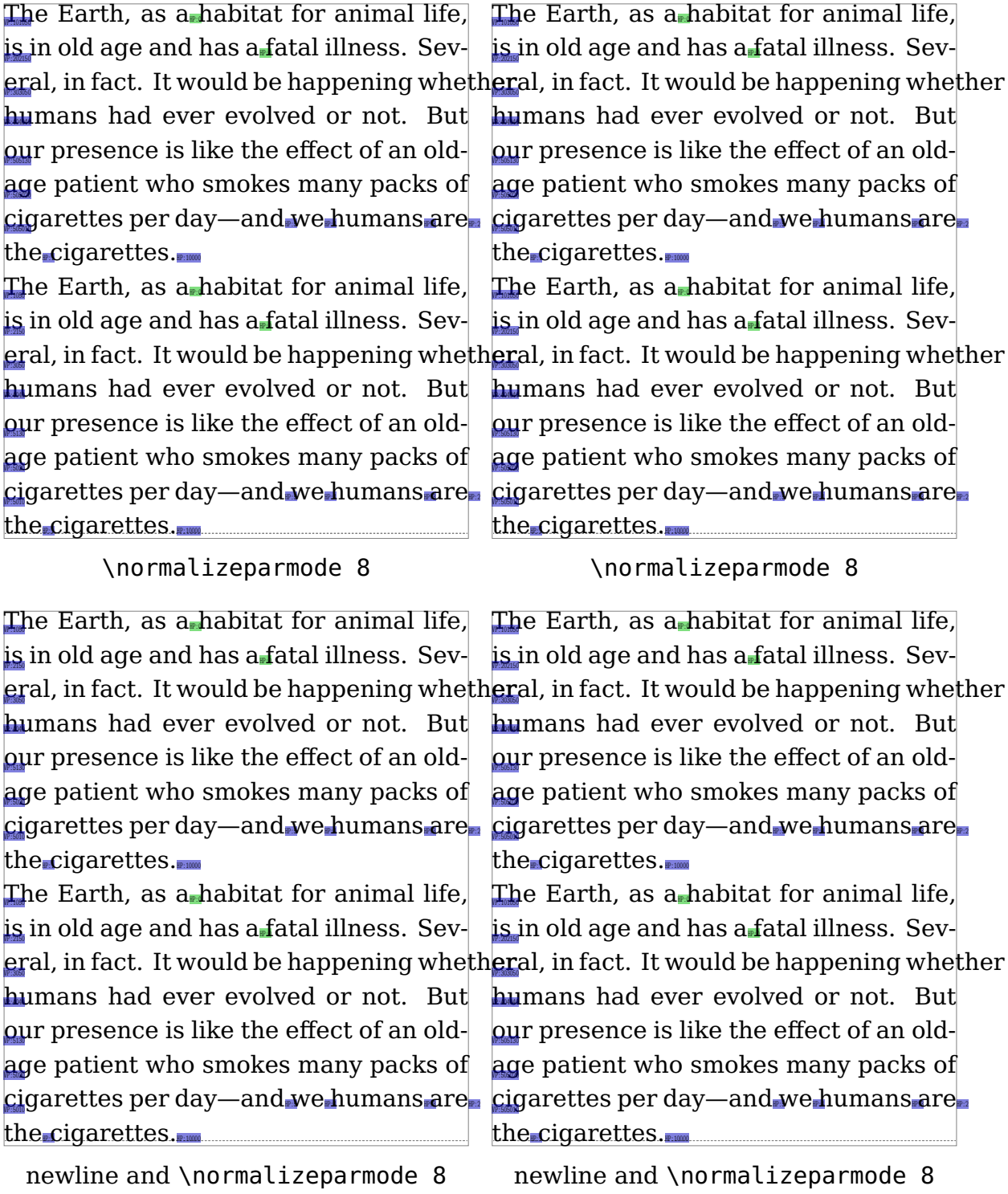


Figure 11.13 Penalty lists

You can also use `\specificationdef` for other constructs that have this multiple variable setup. Now to par passes. This is a mechanism unique to LuaMetaTEX that permits mo

todo: copy some from article when published

11.13 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

12 Alignments

low level TEX

alignments

Contents

12.1 Introduction

TEX_has_a_couple_of_subsystems_and_alignments_is_one_of_them._This_mechanism_is_used_to_

```
\halign {
        \alignmark\hss \aligntab
    \hss\alignmark\hss \aligntab
    \hss\alignmark     \cr
    1.1     \aligntab 2,2     \aligntab 3=3     \cr
    11.11   \aligntab 22,22   \aligntab 33=33   \cr
    111.111 \aligntab 222,222 \aligntab 333=333 \cr
}
```

That_one_doesn't_look_too_complex_and_comes_out_as:_

```
1.1_       2,2_       3=3_
11.11_    22,22_    33=33_
111.111_222,222_333=333_
```

This_is_how_the_previous_code_comes_out_when_we_use_one_of_the_ConTEXt_table_mechanism

```
\starttabulate[|l|c|r|]
  \NC 1.1     \NC 2,2     \NC 3=3     \NC \NR
  \NC 11.11   \NC 22,22   \NC 33=33   \NC \NR
  \NC 111.111 \NC 222,222 \NC 333=333 \NC \NR
\stoptabulate
```

```
1.1          2,2          3=3
11.11       22,22       33=33
111.111   222,222   333=333
```

That_one_looks_a_bit_different_with_respect_to_spaces,_so_let's_go_back_to_the_low_level_variant:_

```
\halign {
        \alignmark\hss \aligntab
    \hss\alignmark\hss \aligntab
    \hss\alignmark     \cr
  1.1\aligntab     2,2\aligntab     3=3\cr
  11.11\aligntab   22,22\aligntab   33=33\cr
  111.111\aligntab 222,222\aligntab 333=333\cr
}
```

Here_we_don't_have_spaces_in_the_content_part_and_therefore_also_no_spaces_in_the_result:_

```
1.1        2,2        3=3
11.11    22,22    33=33
111.111222,222333=333
```

You_can_automate_dealing_with_unwanted_spacing:_

```
\halign {
      \ignorespaces\alignmark\unskip\hss \aligntab
  \hss\ignorespaces\alignmark\unskip\hss \aligntab
  \hss\ignorespaces\alignmark\unskip     \cr
  1.1     \aligntab 2,2     \aligntab 3=3     \cr
  11.11   \aligntab 22,22   \aligntab 33=33   \cr
  111.111 \aligntab 222,222 \aligntab 333=333 \cr
}
```

We_get:_

```
1.1_        2,2_        3=3_
11.11_    22,22_    33=33_
111.111_222,222_333=333_
```

By_moving_the_space_skipping_and_cleanup_to_the_so_called_preamble_we_don't_need_to_deal_with_it_in_the_content_part._We_can_also_deal_with_inter-column_spacing_there:_

```
\halign {
      \ignorespaces\alignmark\unskip\hss \tabskip 1em \aligntab
  \hss\ignorespaces\alignmark\unskip\hss \tabskip 1em \aligntab
  \hss\ignorespaces\alignmark\unskip     \tabskip 0pt \cr
  1.1     \aligntab 2,2     \aligntab 3=3     \cr
  11.11   \aligntab 22,22   \aligntab 33=33   \cr
  111.111 \aligntab 222,222 \aligntab 333=333 \cr
}
```

```
1.1_         2,2_         3=3_
11.11_      22,22_      33=33_
111.111_  222,222_  333=333_
```

If_for_the_moment_we_forget_about_spanning_columns_(`\span`)_and_locally_ignoring_preamble_entries_(`\omit`)_these_basic_commands_are_not_that_complex_to_deal_with._Here_we_use_`\alignmark`_but_that_is_just_a_primitive_that_we_use_instead_of_#_while_`\aligntab`_is_the_same_as_&,_but_using_the_characters_instead_also_assumes_that_they_have_the_category_code_that_relates_to_a_parameter_and_alignment_tab_(and_in_ConTEXt_that_is_not_the_case)._This_we_have-book._

12.2 Between the lines

The_individual_rows_of_a_horizontal_alignment_are_treated_as_lines._This_means_that,_as_we_see_in_the_previous_section,_the_interline_spacing_is_okay._However,_that_also_means_that_when_we_mix_the_lines_with_rules,_the_normal_TEX_habits_kick_in._Take_this:_

```
\halign {
      \ignorespaces\alignmark\unskip\hss \tabskip 1em \aligntab
  \hss\ignorespaces\alignmark\unskip\hss \tabskip 1em \aligntab
  \hss\ignorespaces\alignmark\unskip     \tabskip 0pt \cr
  \noalign{\hrule}
  1.1     \aligntab 2,2     \aligntab 3=3     \cr
  \noalign{\hrule}
  11.11   \aligntab 22,22   \aligntab 33=33   \cr
  \noalign{\hrule}
  111.111 \aligntab 222,222 \aligntab 333=333 \cr
  \noalign{\hrule}
}
```

The_result_doesn't_look_pretty_and_actually,_when_you_see_documents_produced_by_TEX_using_alignments_you_should_not_be_surprised_to_notice_rather_ugly_spacing._The_user_(c

1.1	2,2	3=3
11.11	22,22	33=33
111.111	222,222	333=333

The solution is often easy:

```
\halign {
      \ignorespaces\strut\alignmark\unskip\hss \tabskip 1em \aligntab
  \hss\ignorespaces\strut\alignmark\unskip\hss \tabskip 1em \aligntab
  \hss\ignorespaces\strut\alignmark\unskip     \tabskip 0pt \cr
  \noalign{\hrule}
  1.1     \aligntab 2,2     \aligntab 3=3     \cr
  \noalign{\hrule}
  11.11   \aligntab 22,22   \aligntab 33=33   \cr
  \noalign{\hrule}
  111.111 \aligntab 222,222 \aligntab 333=333 \cr
  \noalign{\hrule}
}
```

1.1	2,2	3=3
11.11	22,22	33=33
111.111	222,222	333=333

The user will not notice it but alignments put some pressure on the general TEX scanner. A

So let's summarize what happens:

1. scan the preamble that defines the cells (where the last one is repeated when needed)
2. check for `\cr`, `\noalign` or a right brace; when a row is entered scan for cells in parallel the preamble so that cell specifications can be applied (then start again)
3. package the preamble based on information with regards to the cells in a column
4. apply the preamble packaging information to the columns and also deal with pending cell spans
5. flush the result to the current list, unless packages in a box a `\halign` is seen as paragraph and rows as lines (such a table can split)

The second (repeated) step is complicated by the fact that the scanner has to look ahead for a `\noalign`, `\cr`, `\omit` or `\span` and when doing that it has to expand what comes. This can give side effects and often results in obscure error messages. When for instance an `\if` is seen and expanded, the wrong branch can be entered. And when you use protected macros embedded alignment commands are not seen at all; of course they still need to produce valid operations in the current context.

All_these_side_effects_are_to_be_handled_in_a_macro_package_when_it_wraps_alignments_in_a_high_level_interface_and_ConTEXt_does_that_for_you._But_because_the_code_doesn't_alway MetaTEX_the_alignment_mechanism_has_been_extended_a_bit_over_time._

Nesting_`\noalign`_is_normally_not_permitted_(but_one_can_redefine_this_primitive_such_that_a_macro_package_nevertheless_handles_it)._The_first_extension_permits_nested_usage_of_`\noalign`._This_has_resulted_of_a_little_reorganization_of_the_code._A_next_extension_showed_up_when_overload_protection_was_introduced_and_extra_prefixes_were_added._We_can_signal_the_scanner_that_a_macro_is_actually_a_`\noalign`_variant:_[19]_

```
\noaligned\protected\def\InBetween{\noalign{...}}
```

Here_the_`\InBetween`_macro_will_get_the_same_treatment_as_`\noalign`_and_it_will_not_trigger_an_error._This_extension_resulted_in_a_second_bit_of_reorganization_(think_of_internal_command_codes_and_such)_but_still_the_original_processing_of_alignments_was_there._

A_third_overhaul_of_the_code_actually_did_lead_to_some_adaptations_in_the_way_alignments_are_constructed_so_let's_move_on_to_that._

12.3 Pre-, inter- and post-tab skips

The_basic_structure_of_a_preamble_and_row_is_actually_not_that_complex:_it_is_a_mix_of_tab_skip_glue_and_cells_(that_are_just_boxes):_

```
\tabskip 10pt
\halign {
  \strut\alignmark\tabskip 12pt\aligntab
  \strut\alignmark\tabskip 14pt\aligntab
  \strut\alignmark\tabskip 16pt\cr
  \noalign{\hrule}
  cell 1.1\aligntab cell 1.2\aligntab cell 1.3\cr
  \noalign{\hrule}
  cell 2.1\aligntab cell 2.2\aligntab cell 2.3\cr
  \noalign{\hrule}
}
```

The_tab_skips_are_set_in_advance_and_apply_to_the_next_cell_(or_after_the_last_one)._

[19] One can argue for using the name `\peekaligned` because in the meantime other alignment primitives also can use this property.

TB:10.000 cell_1.1 TB:12.000 cell_1.2 TB:14.000 cell_1.3 TB:16.000

TB:10.000 cell_2.1 TB:12.000 cell_2.2 TB:14.000 cell_2.3 TB:16.000

In the ConTEXt table mechanisms the value of \tabskip is zero in most cases. As in:

```
\tabskip 0pt
\halign {
  \strut\alignmark\aligntab
  \strut\alignmark\aligntab
  \strut\alignmark\cr
  \noalign{\hrule}
  cell 1.1\aligntab cell 1.2\aligntab cell 1.3\cr
  \noalign{\hrule}
  cell 2.1\aligntab cell 2.2\aligntab cell 2.3\cr
  \noalign{\hrule}
}
```

When these ships are zero, they still show up in the end:

cell_1.1 cell_1.2 cell_1.3

cell_2.1 cell_2.2 cell_2.3

Normally, in order to achieve certain effects there will be more align entries in the preamble than cells in the table, for instance because you want vertical lines between cells. When these are not used, you can get quite a bit of empty boxes and zero skips. Now course this is seldom a problem, but when you have a test document where you want to show font properties in a table and that font supports a script with some ten thousand glyphs, you can imagine that it accumulates and in LuaTEX (and LuaMetaTEX) nodes a TEXt we get messages on the console that node memory is bumped. [20]

[20] I suppose it was a coincidence that a few weeks after these features came available a user consulted the mailing list about a few thousand page table that made the engine run out of memory, something that could be cured by enabling these new features.

After_playing_a_bit_with_stripping_zero_tab_skips_I_found_that_the_code_would_not_really_benefit_from_such_a_feature:_lots_of_extra_tests_made_it_quite_ugly._As_a_result_a_first_alternative_was_to_just_strip_zero_skips_before_an_alignment_got_flushed._At_least_we're_then_a_bit_leaner_in_the_processes_that_come_after_it._This_feature_is_now_available_as_one_of_the_normalizer_bits._

But,_as_we_moved_on,_a_more_natural_approach_was_to_keep_the_skips_in_the_preamble,_because_that_is_where_a_guaranteed_alternating_skip/box_is_assumed._It_also_makes_that_the_original_documentation_is_still_valid._However,_in_the_rows_construction_we_can_be_lean._This_is_driven_by_a_keyword_to_`\halign`:_

```
\tabskip 0pt
\halign noskips {
  \strut\alignmark\aligntab
  \strut\alignmark\aligntab
  \strut\alignmark\cr
  \noalign{\hrule}
  cell 1.1\aligntab cell 1.2\aligntab cell 1.3\cr
  \noalign{\hrule}
  cell 2.1\aligntab cell 2.2\aligntab cell 2.3\cr
  \noalign{\hrule}
}
```

No_zero_tab_skips_show_up_here:_

cell_1.1	cell_1.2	cell_1.3
cell_2.1	cell_2.2	cell_2.3

When_playing_with_all_this_the_LuaMetaTEX_engine_also_got_a_tracing_option_for_alignments._= 2_results_in:_

```
<preamble>
\glue[ignored][...] 0.0pt
\alignrecord
..{\strut }
..<content>
```

```
..{\endtemplate }
\glue[ignored][...] 0.0pt
\alignrecord
..{\strut }
..<content>
..{\endtemplate }
\glue[ignored][...] 0.0pt
\alignrecord
..{\strut }
..<content>
..{\endtemplate }
\glue[ignored][...] 0.0pt
```

The ignored subtype is (currently) only used for these alignment tab skips and it triggers a check later on when the rows are constructed. The <content> is what get injected in the cell (represented by \alignmark). The pseudo primitives are internal and not public.

12.4 Cell widths

Imagine this:

```
\halign {
  x\hbox to 3cm{\strut    \alignmark\hss}\aligntab
  x\hbox to 3cm{\strut\hss\alignmark\hss}\aligntab
  x\hbox to 3cm{\strut\hss\alignmark    }\cr
  cell 1.1\aligntab cell 1.2\aligntab cell 1.3\cr
  cell 2.1\aligntab cell 2.2\aligntab cell 2.3\cr
}
```

which renders as:

xcell 1.1	x cell 1.2	x cell 1.3
xcell 2.1	x cell 2.2	x cell 2.3

A reason to have boxes here is that it enforces a cell width but that is done at the cost of an extra wrapper. In LuaMetaTEX the hlist nodes are rather large because we have

```
\halign {
  x\tabsize 3cm\strut    \alignmark\hss\aligntab
```

```
  x\tabsize 3cm\strut\hss\alignmark\aligntab
  x\tabsize 3cm\strut\hss\alignmark\hss\cr
  cell 1.1\aligntab cell 1.2\aligntab cell 1.3\cr
  cell 2.1\aligntab cell 2.2\aligntab cell 2.3\cr
}
```

If_you_look_carefully_you_will_see_that_this_time_we_don't_have_the_embedded_boxes:_

xcell_1.1	x	cell_1.2x cell_1.3
xcell_2.1	x	cell_2.2x cell_2.3

So,_both_the_sparse_skip_and_new_\tabsize_feature_help_to_make_these_extreme_tables_(spanning_hundreds_of_pages)_not_consume_irrelevant_memory_and_also_make_that_later_on_we_don't_have_to_consult_useless_nodes._

12.5 Plugins

Yet_another_LuaMetaTEX_extension_is_a_callback_that_kicks_in_between_the_preamble_preroll_TEXt_table_mechanisms._

```
\starttabulate[|lG{.}|cG{,}|rG{=}|cG{x}|]
\NC 1.1     \NC 2,2     \NC 3=3     \NC a 0xFF   \NC \NR
\NC 11.11   \NC 22,22   \NC 33=33   \NC b 0xFFF  \NC \NR
\NC 111.111 \NC 222,222 \NC 333=333 \NC c 0xFFFF \NC \NR
\stoptabulate
```

The_tabulate_mechanism_in_ConTEXt_is_rather_old_and_stable_and_it_is_the_preferred_way_to_c

1.1	2,2	3=3	a 0xFF
11.11	22,22	33=33	b 0xFFF
111.111	222,222	333=333	c 0xFFFF

Let's_make_clear_that_this_is_*not*_an_engine_feature_but_a_ConTEXt_one._It_is_however_made_e

```
\halign noskips \alignmentcharactertrigger \bgroup
    \tabskip2em
        \setalignmentcharacter.\ignorespaces\alignmark\unskip\hss \aligntab
    \hss\setalignmentcharacter,\ignorespaces\alignmark\unskip\hss \aligntab
    \hss\setalignmentcharacter=\ignorespaces\alignmark\unskip     \aligntab
    \hss                       \ignorespaces\alignmark\unskip\hss \cr
      1.1   \aligntab   2,2   \aligntab   3=3   \aligntab \setalignmentcharacter{.}\relax 4.4\cr
     11.11  \aligntab  22,22  \aligntab  33=33  \aligntab \setalignmentcharacter{,}\relax 44,44\cr
    111.111 \aligntab 222,222 \aligntab 333=333 \aligntab \setalignmentcharacter{!}\relax 444!444\cr
```

```
       x   \aligntab     x   \aligntab     x   \aligntab \setalignmentcharacter{/}\relax /\cr
      .1   \aligntab    ,2   \aligntab    =3   \aligntab \setalignmentcharacter{?}\relax ?4\cr
      .111 \aligntab    ,222 \aligntab    =333 \aligntab \setalignmentcharacter{=}\relax 44=444\cr
\egroup
```

This_rather_verbose_setup_renders_as:_

```
  1.1_         2,2_         3=3_         4 . 4
 11.11_       22,22_       33=33_       44 , 44
111.111_    222,222_    333=333_    444 ! 444
x_              x_              x_        /
   .1_          ,2_          =3_          ?4
   .111_        ,222_        =333_     44=444
```

Using_a_high_level_interface_makes_sense_but_local_control_over_such_alignment_too,_so_here_follow_some_more_examples._Here_we_use_different_alignment_characters:_

```
\starttabulate[|lG{.}|cG{,}|rG{=}|cG{x}|]
\NC 1.1     \NC 2,2     \NC 3=3     \NC a 0xFF   \NC \NR
\NC 11.11   \NC 22,22   \NC 33=33   \NC b 0xFFF  \NC \NR
\NC 111.111 \NC 222,222 \NC 333=333 \NC c 0xFFFF \NC \NR
\stoptabulate
```

```
  1.1       2,2       3=3     a 0xFF
 11.11     22,22     33=33    b 0xFFF
111.111  222,222  333=333  c 0xFFFF
```

In_this_example_we_specify_the_characters_in_the_cells._We_still_need_to_add_a_specifier_in_the_preamble_definition_because_that_will_trigger_the_plugin._

```
\starttabulate[|lG{}|rG{}|]
\NC left                                          \NC right                                \NC\NR
\NC \showglyphs \setalignmentcharacter{.}1.1      \NC \setalignmentcharacter{.}1.1         \NC\NR
\NC \showglyphs \setalignmentcharacter{,}11,11    \NC \setalignmentcharacter{,}11,11       \NC\NR
\NC \showglyphs \setalignmentcharacter{=}111=111  \NC \setalignmentcharacter{=}111=111     \NC\NR
\stoptabulate
```

```
left              right
   1␣.␣1          1 . 1
  11␣,␣11       11 , 11
111␣=␣111     111=111
```

You_can_mix_these_approaches:_

```
\starttabulate[|lG{.}|rG{}|]
\NC left    \NC right                              \NC\NR
```

```
\NC 1.1     \NC \setalignmentcharacter{.}1.1     \NC\NR
\NC 11.11   \NC \setalignmentcharacter{.}11.11   \NC\NR
\NC 111.111 \NC \setalignmentcharacter{.}111.111 \NC\NR
\stoptabulate
```

left right
1.1 1.1
11.11 11.11
111.111 111.111

Here_the_already_present_alignment_feature,_that_at_some_point_in_tabulate_might_use_this_new_feature,_is_meant_for_numbers,_but_here_we_can_go_wild_with_words,_although_of_course_you_need_to_keep_in_mind_that_we_deal_with_typeset_text,_so_there_may_be_no_match._

```
\starttabulate[|lG{.}|rG{.}|]
\NC foo.bar \NC foo.bar \NC \NR
\NC  oo.ba  \NC  oo.ba  \NC \NR
\NC   o.b   \NC   o.b   \NC \NR
\stoptabulate
```

foo.bar foo.bar
oo.ba oo.ba
o.b o.b

This_feature_will_only_be_used_in_know_situations_and_those_seldom_involve_advanced_typesetting._However,_the_following_does_work:_[21]_

```
\starttabulate[|cG{d}|]
\NC \smallcaps abcdefgh \NC \NR
\NC             xdy     \NC \NR
\NC \sl         xdy     \NC \NR
\NC \tttf       xdy     \NC \NR
\NC \tfd         d      \NC \NR
\stoptabulate
```

abc d efgh
x d y
x d y

[21] Should this be an option instead?

x d y
d

As always with such mechanisms, the question is "Where to stop?" But it makes for nice demos and as long as little code is needed it doesn't hurt.

12.6 Pitfalls and tricks

The next example mixes bidirectional typesetting. It might look weird at first sight but the result conforms to what we discussed in previous paragraphs.

```
\starttabulate[|lG{.}|lG{}|]
\NC \righttoleft 1.1   \NC \righttoleft \setalignmentcharacter{.}1.1   \NC\NR
\NC              1.1   \NC              \setalignmentcharacter{.}1.1   \NC\NR
\NC \righttoleft 1.11  \NC \righttoleft \setalignmentcharacter{.}1.11  \NC\NR
\NC              1.11  \NC              \setalignmentcharacter{.}1.11  \NC\NR
\NC \righttoleft 1.111 \NC \righttoleft \setalignmentcharacter{.}1.111 \NC\NR
\NC              1.111 \NC              \setalignmentcharacter{.}1.111 \NC\NR
\stoptabulate
```

```
  1.1     1.1
1.1     1.1
 11.1    11.1
1.11    1.11
111.1   111.1
1.111   1.111
```

In case of doubt, look at this:

```
\starttabulate[|lG{.}|lG{}|lG{.}|lG{}|]
\NC \righttoleft 1.1   \NC \righttoleft \setalignmentcharacter{.}1.1   \NC
                 1.1   \NC              \setalignmentcharacter{.}1.1   \NC\NR
\NC \righttoleft 1.11  \NC \righttoleft \setalignmentcharacter{.}1.11  \NC
                 1.11  \NC              \setalignmentcharacter{.}1.11  \NC\NR
\NC \righttoleft 1.111 \NC \righttoleft \setalignmentcharacter{.}1.111 \NC
                 1.111 \NC              \setalignmentcharacter{.}1.111 \NC\NR
\stoptabulate
```

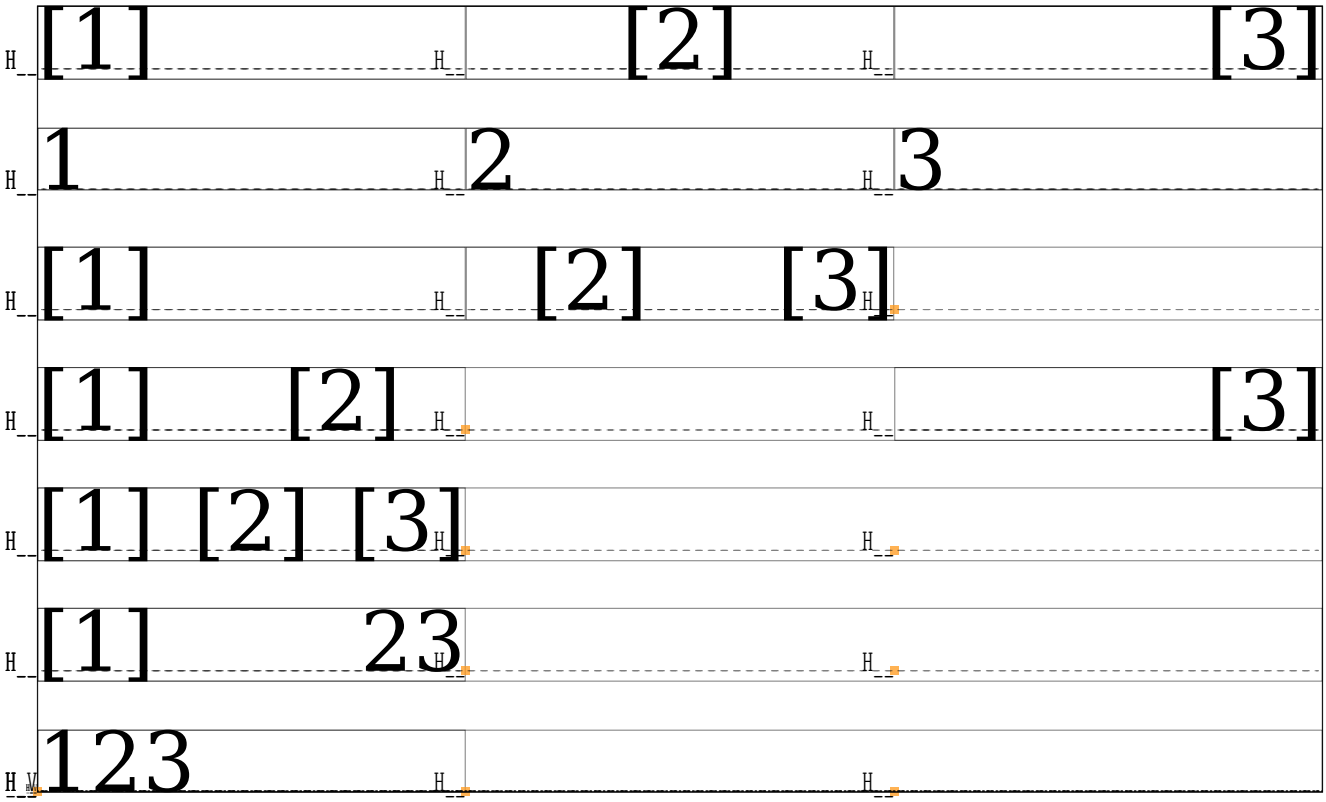
```
  1.1     1.1  1.1     1.1
 11.1    11.1  1.11    1.11
111.1   111.1  1.111   1.111
```

The next example shows the effect of `\omit` and `\span`. The first one makes that in this cell the preamble template is ignored.

```
\halign \bgroup
```

```
    \tabsize 2cm\relax     [\alignmark]\hss \aligntab
    \tabsize 2cm\relax \hss[\alignmark]\hss \aligntab
    \tabsize 2cm\relax \hss[\alignmark]\cr
           1\aligntab       2\aligntab       3\cr
     \omit 1\aligntab \omit 2\aligntab \omit 3\cr
           1\aligntab       2\span           3\cr
           1\span           2\aligntab       3\cr
           1\span           2\span           3\cr
           1\span \omit     2\span \omit     3\cr
     \omit 1\span \omit     2\span \omit     3\cr
\egroup
```

Spans_are_applied_at_the_end_so_you_see_a_mix_of_templates_applied._



When_you_define_an_alignment_inside_a_macro,_you_need_to_duplicate_the_`\alignmark`_signals._This_is_similar_to_embedded_macro_definitions._But_in_LuaMetaTEX_we_can_get_around_

```
\halign \bgroup
    \tabsize 2cm\relax     \aligncontent\hss \aligntab
    \tabsize 2cm\relax \hss\aligncontent\hss \aligntab
    \tabsize 2cm\relax \hss\aligncontent\cr
    1\aligntab 2\aligntab 3\cr
```

```
    A\aligntab B\aligntab C\cr
\egroup
```

1 2 3
A B C

In this example we still have to be verbose in the way we align but we can do this:

```
\halign \bgroup
    \tabsize 2cm\relax \aligncontentleft  \aligntab
    \tabsize 2cm\relax \aligncontentmiddle\aligntab
    \tabsize 2cm\relax \aligncontentright \cr
    1\aligntab 2\aligntab 3\cr
    A\aligntab B\aligntab C\cr
\egroup
```

Where the helpers are defined as:

```
\noaligned\protected\def\aligncontentleft
  {\ignorespaces\aligncontent\unskip\hss}

\noaligned\protected\def\aligncontentmiddle
  {\hss\ignorespaces\aligncontent\unskip\hss}

\noaligned\protected\def\aligncontentright
  {\hss\ignorespaces\aligncontent\unskip}
```

The preamble scanner see such macros as candidates for a single level expansion so it will inject the meaning and see the `\aligncontent` eventually.

1 2 3
A B C

The same effect could be achieved by using the `\span` prefix:

```
\def\aligncontentleft{\ignorespaces\aligncontent\unskip\hss}

\halign { ... \span\aligncontentleft ...}
```

One of the reasons for not directly using the low level `\halign` command is that it's a lot of work but by providing a set of helpers like here might change that a bit. Keep in mind that much of the above is not new in the sense that we could not achieve the same already, it's just a bit programmer friendly.

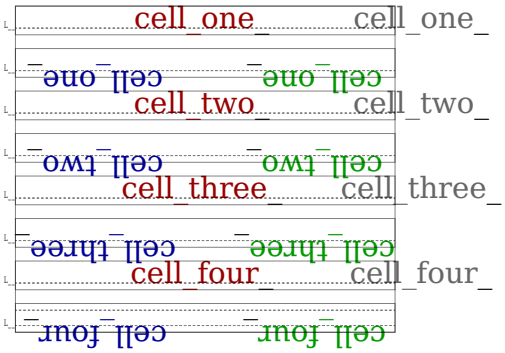
12.7 Rows

Alignment_support_is_what_the_documented_source_calls_'interwoven'._When_the_engine_scans_for_input_it_processing_text,_math_or_alignment_content._While_doing_alignments_it_collects_rows,_and_inside_these_cells_but_also_deals_with_material_that_ends_up_in_between._In_LuaMetaTEX_I_tried_to_isolate_the_bits_and_pieces_as_good_as_possible_but_set'_state._

Scanning_starts_with_interpreting_the_preamble,_and_then_grabbing_rows._There_is_some_nasty_lookahead_involved_for_\noalign,_\span,_\omit,_\cr_and_\crcr_and_that_is_not_code_one_wants_to_tweak_too_much_(although_we_did_in_LuaMetaTEX)._This_means_for_start_a_row_here'_primitive_is_sort_of_tricky_(but_it_might_happen_some_day)_which_in_turn_means_that_it_is_not_really_possible_to_set_row_properties._As_an_experiment_we_can_set_some_properties_now_by_hijacking_\noalign_and_storing_them_on_the_alignment_stack_(indeed:_at_the_cost_of_some_extra_overhead_and_memory)._This_permits_the_following:_

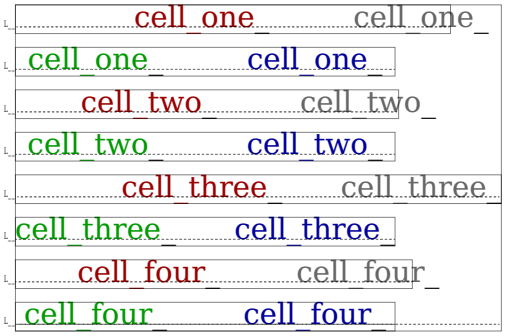
```
\halign {
    \hss
    \ignorespaces \alignmark \removeunwantedspaces
    \hss
    \quad \aligntab \quad
    \hss
    \ignorespaces \alignmark \removeunwantedspaces
    \hss
    \cr
    \noalign xoffset 40pt {}
    {\darkred   cell one}   \aligntab {\darkgray cell one}   \cr
    \noalign orientation "002 {}
    {\darkgreen cell one}   \aligntab {\darkblue cell one}   \cr
    \noalign xoffset 40pt {}
    {\darkred   cell two}   \aligntab {\darkgray cell two}   \cr
    \noalign orientation "002 {}
    {\darkgreen cell two}   \aligntab {\darkblue cell two}   \cr
    \noalign xoffset 40pt {}
    {\darkred   cell three} \aligntab {\darkgray cell three} \cr
    \noalign orientation "002 {}
    {\darkgreen cell three} \aligntab {\darkblue cell three} \cr
    \noalign xoffset 40pt {}
    {\darkred   cell four}  \aligntab {\darkgray cell four}  \cr
    \noalign orientation "002 {}
```

```
    {\darkgreen cell four}  \aligntab {\darkblue cell four}  \cr
}
```



The supported keywords are similar to those for boxes: source, target, anchor, orientatic
dimensions can be prefixed by add and reset wipes all. Here is another example:

```
\halign {
    \hss
    \ignorespaces \alignmark \removeunwantedspaces
    \hss
    \quad \aligntab \quad
    \hss
    \ignorespaces \alignmark \removeunwantedspaces
    \hss
    \cr
    \noalign xmove 40pt {}
    {\darkred   cell one}   \aligntab {\darkgray cell one}   \cr
    {\darkgreen cell one}   \aligntab {\darkblue cell one}   \cr
    \noalign xmove 20pt {}
    {\darkred   cell two}   \aligntab {\darkgray cell two}   \cr
    {\darkgreen cell two}   \aligntab {\darkblue cell two}   \cr
    \noalign xmove 40pt {}
    {\darkred   cell three} \aligntab {\darkgray cell three} \cr
    {\darkgreen cell three} \aligntab {\darkblue cell three} \cr
    \noalign xmove 20pt {}
    {\darkred   cell four}  \aligntab {\darkgray cell four}  \cr
    {\darkgreen cell four}  \aligntab {\darkblue cell four}  \cr
}
```



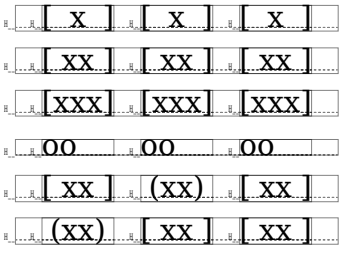
Some_more_features_might_be_added_in_the_future_as_is_it_an_interesting_playground._It_is_to_be_seen_how_this_ends_up_in_ConTEXt_high_level_interfaces_like_tabulate._

12.8 Templates

The_`\omit`_command_signals_that_the_template_should_not_be_applied._But_what_if_we_actually_want_something_at_the_left_and_right_of_the_content?_Here_is_how_it's_done:_

```
\tabskip10pt \showboxes
\halign\bgroup
    [\hss\aligncontent\hss]\aligntab
    [\hss\aligncontent\hss]\aligntab
    [\hss\aligncontent\hss]\cr
            x\aligntab                          x\aligntab         x\cr
           xx\aligntab                         xx\aligntab        xx\cr
          xxx\aligntab                        xxx\aligntab       xxx\cr
    \omit  oo\aligntab\omit                    oo\aligntab\omit   oo\cr
           xx\aligntab\realign{\hss(}{)\hss}xx\aligntab          xx\cr
    \realign{\hss(}{)\hss}xx\aligntab xx\aligntab xx\cr
\egroup
```

The_`\realign`_command_is_like_an_`omit`_but_it_expects_two_token_lists_that_will_for_this_cell_be_used_instead_of_the_ones_from_the_preamble._While_`\omit`_also_skips_insertion_of_`\everytab`,_here_it_is_inserted,_just_like_with_normal_preambles._



It_will_probably_take_a_while_before_I'll_apply_this_in_ConTEXt_because_changing_existing_(sta

12.9 Pitfalls

Alignment_have_a_few_properties_that_can_catch_you_off-guard._One_is_the_use_of_\everycr._T next_example_demonstrates_that_it_is_also_injected_after_the_preamble_definition._

```
\everycr{\noalign{\hrule}}
\halign\bgroup \hsize 5cm \strut \alignmark\cr one\cr two\cr\egroup
```

This_makes_sense_because_it_is_one_way_to_make_sure_that_for_instance_a_rule_gets_ the_width_of_the_cell._

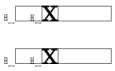
The_sam_eis_of_course_true_for_a_vertical_align:_

```
\everycr{\noalign{\vrule}}
\valign\bgroup \hsize 4cm \strut \aligncontent\cr one\cr two\cr\egroup
```

We_set_the_width_because_otherwise_the_current_text_width_is_used._

Something_similar_happens_with_a_\tabskip:_the_value_set_before_the_alignment_is_ used_left_of_the_first_cell._

```
\tabskip10pt
\halign\bgroup \tabskip20pt\relax\aligncontent\cr x\cr x\cr \egroup
```



The_\tabskip_outside_the_alignment_is_an_internal_glue_register_so_you_can_for_instance_ use_the_prefix_\global._However,_in_a_preamble_it_is_more_a_directive:_the_given_value_ is_stored_with_the_cell._This_means_that_the_next_code_is_invalid:_

```
\tabskip10pt
\halign\bgroup \global\tabskip20pt\relax\aligncontent\cr x\cr x\cr \egroup
```

The_parser_looks_at_tokens_in_the_preamble,_sees_the_\global_and_appends_it_to_the_ current_pre-part_of_the_cell's_template._Then_it_sees_a_\tabskip_and_assigns_the_value_ after_it_to_the_cell's_skip._The_token_and_its_value_just_disappear,_they_are_not_appended_ to_the_template._Now,_when_the_template_is_injected_(and_interpreted)_this_\global_ex- pects_a_variable_next_and_in_our_case_the_x_doesn't_qualify._The_next_snippet_however_ works_okay:_

```
\scratchcounter0
\halign\bgroup
    \global\tabskip40pt\relax\advance\scratchcounter\plusone\aligncontent
      \cr
    x:\the\scratchcounter\cr
    x:\the\scratchcounter\cr
    x:\the\scratchcounter\cr
\egroup
```

Here_the_\global_is_applied_to_the_advance_because_the_skip_definition_is_*not*_in_the_preamb

x:1
x:2
x:3

Here_is_a_variant:_

```
\scratchcounter0
\halign\bgroup
   \global\tabskip10pt\relax\aligncontent\cr
   \advance\scratchcounter\plusone x:\the\scratchcounter\cr
   \advance\scratchcounter\plusone x:\the\scratchcounter\cr
   \advance\scratchcounter\plusone x:\the\scratchcounter\cr
\egroup
```

Again_the_\global_stays_and_this_time_if_ends_up_before_the_content_which_starts_with_an_\advance._
x:1
x:2
x:3

Normally_you_will_not_need_the_next_trickery_but_it_shows_that_cells_are_grouped:_

```
\halign\bgroup\aligncontent\cr
    1\atendofgrouped{A}\atendofgrouped{B}\cr
    2\aftergrouped  {A}\aftergrouped  {B}\cr
    3                                 \cr
\egroup
```

Maybe_at_some_point_I'll_add_something_a_bit_more_tuned_for_dealing_with_cells,_but_here_is_what_you_get_with_the_above:_
1AB
2

AB3_

12.10 Remark

It_can_be_that_the_way_alignments_are_interfaced_with_respect_to_attributes_is_a_bit_different_between_LuaTEX_and_LuaMetaTEX_but_because_the_former_is_frozen_(in_order_not_to_TEXt_LMTX._

In_principle_we_can_have_hooks_into_the_rows_for_pre_and_post_material_but_it_doesn't_really_pay_of_as_grouping_will_still_interfere._So_for_now_I_decided_not_to_add_these._

12.10 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

13 Marks

low level TEX

marks

Contents

13.1 Introduction

Marks_are_one_of_the_subsystems_of_TEX,_as_are_for_instance_alignments_and_math_as_well_a

In_MkII_marks_are_used_to_keep_track_of_colors,_transparencies_and_more_properties_ that_work_across_page_boundaries._It_permits_picking_up_at_the_top_of_a_page_from_ where_one_left_at_the_bottom_of_the_preceding_one._When_MkII_was_written_there_ was_only_one_mark_so_on_top_of_that_a_multiple_mark_mechanism_was_implemented_ that_filtered_specific_marks_from_a_collection._Later,_ε-TEX_provided_mark_classes_so_that_me

But,_marks_have_some_nasty_limitations,_so_from_the_ConTEXt_perspective_there_always_was_s TEX_engine_has_a_`\clearmarks`_primitive_but_that_works_global._In_LuaMetaTEX_a_proper_mar

In_MkIV_the_engine's_marks_were_not_used_at_all_and_an_alternative_mechanism_was_ written_using_Lua._It_actually_is_one_of_the_older_MkIV_features._It_doesn't_have_the_ side_effects_that_native_marks_have_but_it_comes_at_the_price_of_more_overhead,_al- though_that_is_bearable._

In_this_document_we_discuss_marks_but_assume_that_LuaMetaTEX_is_used_with_Con- TEXt_LMTX._There_we_experiment_with_using_the_native_marks,_complemented_by_a_few_Lua_

13.2 The basics

Although_the_original_TEX_primitives_are_there,_the_plural_ε-TEX_mark_commands_are_to_be_u

```
\marks0{This is mark 0} % equivalent to: \mark{This is mark 0}
\marks4{This is mark 4}
```

[22] This is probably true for most LuaTEX and LuaMetaTEX extensions, maybe example usage create retrospective demand. But one reason for picking up on engine development is that in the ConTEXt perspective we actually had some demands.

When a page has been split off, you can (normally this only makes sense in the output routine) access marks with:

```
\topmarks  4
\firstmarks4
\botmarks  4
```

A 'top' mark is the last one on the previous page(s), the 'first' and 'bottom' refer to the current page. A mark is a so called node, something that ends up in the current list and the token list is stored with it. The accessors are just commands and they fetch the token list from a separately managed storage. When you set or access a mark that has not yet been used, the storage is bumped to the right size, so it doesn't make sense to use e.g. `\marks 999` when there are no 998 ones too: it not only takes memory, it also makes TEX run over all these mark stores when synchronization

```
\newmarks\MyMark
```

Currently the first 16 marks are skipped so this makes `\MyMark` become mark 17. The reason is that we want to make sure that users who experiment with marks have some scratch marks available and don't overload system defined ones. Future versions of ConTEXt might become more restrictive.

Marks can be cleared with:

```
\clearmarks 4
```

which clears the storage that keeps the top, first and bot marks. This happens immediately. You can delay this by putting a signal in the list:

```
\flushmarks 4
```

This (LuaMetaTEX) feature makes it for instance easy to reset marks that keep track of sect
TEXt always had that.

The current, latest assigned, value of a mark is available too:

```
\currentmarks 4
```

Using this value in for instance headers and footers makes no sense because the last node set can be on a following page.

13.3 Migration

In the introduction we mentioned that LuaMetaTEX has migration built in. In MkIV we have

Migrated_marks_end_up_in_the_postmigrated_sublist_of_a_box._In_other_lowlevel_manuals_we_discuss_these_pre-_and_postmigrated_sublists._As_example_we_use_this_definition:_

```
\setbox0\vbox\bgroup
test \marks 4 {mark 4.1}\par
test \marks 4 {mark 4.1}\par
test \marks 4 {mark 4.1}\par
\egroup
```

When_we_turn_migration_on_(officially_the_second_bit):_

```
\automigrationmode"FF \showbox0
```

we_get_this:_

```
> \box0=
2:4: \vbox[normal][...], width 483.69687, height 63.43475, depth 0.15576, direction l2r
2:4: .\list
2:4: ..\hbox[line][...], width 483.69687, height 7.48193, depth 0.15576, glue 459.20468fil, direction l2r
2:4: ...\list
2:4: ....\glue[left hang][...] 0.0pt
2:4: ....\glue[left][...] 0.0pt
2:4: ....\glue[parfillleft][...] 0.0pt
2:4: ....\par[newgraf][...], hangafter 1, hsize 483.69687, pretolerance 100, tolerance 200, adjdemerits 10000, linepenalty 10, doublehyphendemerits 10000,
  finalhyphendemerits 5000, clubpenalty 2000, widowpenalty 2000, brokenpenalty 100, parfillskip 0.0pt plus 1.0fil, hyphenationmode 499519
2:4: ....\glue[indent][...] 0.0pt
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000065 e
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000073 s
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glue[space][...] 3.49658pt plus 1.74829pt minus 1.16553pt, font 8
2:4: ....\penalty[line][...] 10000
2:4: ....\glue[parfill][...] 0.0pt plus 1.0fil
2:4: ....\glue[right][...] 0.0pt
2:4: ....\glue[right hang][...] 0.0pt
2:4: ..\glue[par][...] 11.98988pt plus 3.99663pt minus 3.99663pt
2:4: ..\glue[baseline][...] 8.34883pt
2:4: ..\hbox[line][...], width 483.69687, height 7.48193, depth 0.15576, glue 459.20468fil, direction l2r
2:4: ...\list
2:4: ....\glue[left hang][...] 0.0pt
2:4: ....\glue[left][...] 0.0pt
2:4: ....\glue[parfillleft][...] 0.0pt
2:4: ....\par[newgraf][...], hangafter 1, hsize 483.69687, pretolerance 100, tolerance 200, adjdemerits 10000, linepenalty 10, doublehyphendemerits 10000,
  finalhyphendemerits 5000, clubpenalty 2000, widowpenalty 2000, brokenpenalty 100, parfillskip 0.0pt plus 1.0fil, hyphenationmode 499519
2:4: ....\glue[indent][...] 0.0pt
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000065 e
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000073 s
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glue[space][...] 3.49658pt plus 1.74829pt minus 1.16553pt, font 8
2:4: ....\penalty[line][...] 10000
2:4: ....\glue[parfill][...] 0.0pt plus 1.0fil
2:4: ....\glue[right][...] 0.0pt
2:4: ....\glue[right hang][...] 0.0pt
2:4: ..\glue[par][...] 11.98988pt plus 3.99663pt minus 3.99663pt
2:4: ..\glue[baseline][...] 8.34883pt
2:4: ..\hbox[line][...], width 483.69687, height 7.48193, depth 0.15576, glue 459.20468fil, direction l2r
2:4: ...\list
2:4: ....\glue[left hang][...] 0.0pt
2:4: ....\glue[left][...] 0.0pt
2:4: ....\glue[parfillleft][...] 0.0pt
2:4: ....\par[newgraf][...], hangafter 1, hsize 483.69687, pretolerance 100, tolerance 200, adjdemerits 10000, linepenalty 10, doublehyphendemerits 10000,
  finalhyphendemerits 5000, clubpenalty 2000, widowpenalty 2000, brokenpenalty 100, parfillskip 0.0pt plus 1.0fil, hyphenationmode 499519
2:4: ....\glue[indent][...] 0.0pt
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000065 e
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000073 s
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glue[space][...] 3.49658pt plus 1.74829pt minus 1.16553pt, font 8
2:4: ....\penalty[line][...] 10000
2:4: ....\glue[parfill][...] 0.0pt plus 1.0fil
2:4: ....\glue[right][...] 0.0pt
2:4: ....\glue[right hang][...] 0.0pt
```

```
2:4: .\postmigrated
2:4: ..\mark[4][...]
2:4: ..{mark 4.1}
2:4: ..\mark[4][...]
2:4: ..{mark 4.1}
2:4: ..\mark[4][...]
2:4: ..{mark 4.1}
```

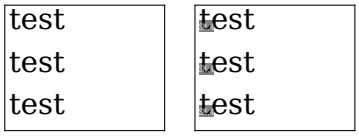
When_we_don't_migrate,_enforced_with:_

```
\automigrationmode"00 \showbox0
```

the_result_is:_

```
> \box0=
2:4: \vbox[normal][...], width 483.69687, height 63.43475, depth 0.15576, direction l2r
2:4: .\list
2:4: ..\hbox[line][...], width 483.69687, height 7.48193, depth 0.15576, glue 459.20468fil, direction l2r
2:4: ...\list
2:4: ....\glue[left hang][...] 0.0pt
2:4: ....\glue[left][...] 0.0pt
2:4: ....\glue[parfillleft][...] 0.0pt
2:4: ....\par[newgraf][...], hangafter 1, hsize 483.69687, pretolerance 100, tolerance 200, adjdemerits 10000, linepenalty 10, doublehyphendemerits 10000,
  finalhyphendemerits 5000, clubpenalty 2000, widowpenalty 2000, brokenpenalty 100, parfillskip 0.0pt plus 1.0fil, hyphenationmode 499519
2:4: ....\glue[indent][...] 0.0pt
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000065 e
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000073 s
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glue[space][...] 3.49658pt plus 1.74829pt minus 1.16553pt, font 8
2:4: ....\penalty[line][...] 10000
2:4: ....\glue[parfill][...] 0.0pt plus 1.0fil
2:4: ....\glue[right][...] 0.0pt
2:4: ....\glue[right hang][...] 0.0pt
2:4: ..\mark[4][...]
2:4: ..{mark 4.1}
2:4: ..\glue[par][...] 11.98988pt plus 3.99663pt minus 3.99663pt
2:4: ..\glue[baseline][...] 8.34883pt
2:4: ..\hbox[line][...], width 483.69687, height 7.48193, depth 0.15576, glue 459.20468fil, direction l2r
2:4: ...\list
2:4: ....\glue[left hang][...] 0.0pt
2:4: ....\glue[left][...] 0.0pt
2:4: ....\glue[parfillleft][...] 0.0pt
2:4: ....\par[newgraf][...], hangafter 1, hsize 483.69687, pretolerance 100, tolerance 200, adjdemerits 10000, linepenalty 10, doublehyphendemerits 10000,
  finalhyphendemerits 5000, clubpenalty 2000, widowpenalty 2000, brokenpenalty 100, parfillskip 0.0pt plus 1.0fil, hyphenationmode 499519
2:4: ....\glue[indent][...] 0.0pt
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000065 e
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000073 s
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glue[space][...] 3.49658pt plus 1.74829pt minus 1.16553pt, font 8
2:4: ....\penalty[line][...] 10000
2:4: ....\glue[parfill][...] 0.0pt plus 1.0fil
2:4: ....\glue[right][...] 0.0pt
2:4: ....\glue[right hang][...] 0.0pt
2:4: ..\mark[4][...]
2:4: ..{mark 4.1}
2:4: ..\glue[par][...] 11.98988pt plus 3.99663pt minus 3.99663pt
2:4: ..\glue[baseline][...] 8.34883pt
2:4: ..\hbox[line][...], width 483.69687, height 7.48193, depth 0.15576, glue 459.20468fil, direction l2r
2:4: ...\list
2:4: ....\glue[left hang][...] 0.0pt
2:4: ....\glue[left][...] 0.0pt
2:4: ....\glue[parfillleft][...] 0.0pt
2:4: ....\par[newgraf][...], hangafter 1, hsize 483.69687, pretolerance 100, tolerance 200, adjdemerits 10000, linepenalty 10, doublehyphendemerits 10000,
  finalhyphendemerits 5000, clubpenalty 2000, widowpenalty 2000, brokenpenalty 100, parfillskip 0.0pt plus 1.0fil, hyphenationmode 499519
2:4: ....\glue[indent][...] 0.0pt
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000065 e
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000073 s
2:4: ....\glyph[32768][...], language (n=1,l=2,r=3), hyphenationmode "79F3F, options "80, font <8: DejaVuSerif @ 11.0pt>, glyph U+000074 t
2:4: ....\glue[space][...] 3.49658pt plus 1.74829pt minus 1.16553pt, font 8
2:4: ....\penalty[line][...] 10000
2:4: ....\glue[parfill][...] 0.0pt plus 1.0fil
2:4: ....\glue[right][...] 0.0pt
2:4: ....\glue[right hang][...] 0.0pt
2:4: ..\mark[4][...]
2:4: ..{mark 4.1}
```

When_you_say_`\showmakeup`_or_in_this_case_`\showmakeup[mark]`_the_marks_are_visualized:_



enabled disabled

Here_sm_means_‘set_mark’_while_rm_would_indicate_a_‘reset_mark’._Of_course_migrated_marks_don't_show_up_because_these_are_bound_to_the_box_and_thereby_have_become_a_a_specific_box_property_as_can_be_seen_in_the_above_trace._

13.4 Tracing

The_LuaMetaTEX_engine_has_a_dedicated_tracing_option_for_marks._The_fact_that_the_traditio

```
\tracingmarks1
\tracingonline2
```

When_tracing_is_set_to_1_we_get_a_list_of_marks_for_the_just_split_of_page:_

```
2:7: <mark class 51, top := bot>
2:7: ..{sample 9.1}
2:7: <mark class 51: first := mark>
2:7: ..{sample 10.1}
2:7: <mark class 51: bot := mark>
2:7: ..{sample 10.1}
2:7: <mark class 51, page state>
2:7: ..top {sample 9.1}
2:7: ..first {sample 10.1}
2:7: ..bot {sample 10.1}
```

When_tracing_is_set_to_2_you_also_get_details_we_get_a_list_of_marks_of_the_analysis:_

```
1:9: <mark class 51, top := bot>
1:9: ..{sample 5.1}
1:9: <mark class 51: first := mark>
1:9: ..{sample 6.1}
1:9: <mark class 51: bot := mark>
1:9: ..{sample 6.1}
1:9: <mark class 51: bot := mark>
1:9: ..{sample 7.1}
1:9: <mark class 51: bot := mark>
1:9: ..{sample 8.1}
```

```
1:9: <mark class 51: bot := mark>
1:9: ..{sample 9.1}
1:9: <mark class 51, page state>
1:9: ..top {sample 5.1}
1:9: ..first {sample 6.1}
1:9: ..bot {sample 9.1}
```

13.5 High level commands

I_think_that_not_that_many_users_define_their_own_marks._They_are_useful_for_showing_section_related_titles_in_headers_and_footers_but_the_implementation_of_that_is_hidden._The_native_mark_references_are_top,_first_and_bottom_but_in_the_ConTEXt_interface

ConTEXt	TEX	column	page
previous	top	last before sync	last on previous page
top	first	first in sync	first on page
bottom	bot	last in sync	last on page
first	top	first not top in sync	first on page
last	bot	last not bottom in sync	last on page
default		the same as first	
current		the last set value	

In_order_to_separate_marks_in_ConTEXt_from_those_in_TEX,_the_term_'marking'_is_used._In_MkIV_the_regular_marks_mechanism_is_of_course_there_but,_as_mentioned,_not_used._By_using_a_different_namespace_we_could_make_the_transition_from_MkII_to_MkIV_(the_same_is_true_for_some_more_mechanisms)._

A_marking_is_defined_with_

```
\definemarking[MyMark]
```

A_defined_marking_can_be_set_with_two_equivalent_commands:_

```
\setmarking[MyMark]{content}
\marking   [MyMark]{content}
```

The_content_is_not_typeset_but_stored_as_token_list._In_the_sectioning_mechanism_that_uses_markings_we_don't_even_store_titles,_we_store_a_reference_to_a_title._In_order_to_use_that_(deep_down)_we_hook_in_a_filter_command._By_default_that_command_does_nothing:_

```
\setupmarking[MyMark][filtercommand=\firstofoneargument]
```

The token list does *not* get expanded by default, unless you set it up:

```
\setupmarking[MyMark][expansion=yes]
```

The current state of a marking can be cleared with:

```
\clearmarking[MyMark]
```

but because that en is not synchronized the real deal is:

```
\resetmarking[MyMark]
```

Be aware that it introduces a node in the list. You can test if a marking is defined with (as usual) a test macro. Contrary to (most) other test macros this one is fully expandable:

```
\doifelsemarking {MyMark} {
    defined
} {
    undefined
}
```

Because there can be a chain involved, we can relate markings. Think of sections below chapters and subsections below sections:

```
\relatemarking[MyMark][YourMark]
```

When a marking is set its relatives are also reset, so setting `YourMark` will reset `MyMark`. It is this kind of features that made for marks being wrapped into high level commands very early in the ConTEXt development (and one can even argue that this is why a package TEXt exists in the first place).

The rest of the (relatively small) repertoire of commands has to do with fetching markings. The general command is `\getmarking` that takes two or three arguments:

```
\getmarking[MyMarking][first]
\getmarking[MyMarking][page][first]
\getmarking[MyMarking][page][first]
\getmarking[MyMarking][column:1][first]
```

There are (normally) three marks that can be fetched so we have three commands that do just that:

High level commands

```
\fetchonemark [MyMarking][which one]
\fetchtwomarks[MyMarking]
\fetchallmarks[MyMarking]
```

You_can_setup_a_separator_key_which_by_default_is:_

```
\setupmarking[MyMarking][separator=\space\emdash\space]
```

Injection_is_enabled_by_default_due_to_this_default:_

```
\setupmarking[MyMarking][state=start]
```

The_following_three_variants_are_(what_is_called)_fully_expandable:_

```
\fetchonemarking [MyMarking][which one]
\fetchtwomarkings[MyMarking]
\fetchallmarkings[MyMarking]
```

13.6 Pitfalls

The_main_pitfall_is_that_a_(re)setting_a_mark_will_inject_a_node_which_in_vertical_mode_can_interfere_with_spacing._In_for_instance_section_commands_we_wrap_them_with_the_title_so_there_it_should_work_out_okay._

13.6 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

14 Inserts

low level TEX

inserts

Contents

14.1 Introduction

This_document_is_a_mixed_bag._We_do_discuss_inserts_but_also_touch_elements_of_the_page_builder_because_inserts_and_regular_page_content_are_handled_there._Examples_of_mechanisms_that_use_inserts_are_footnotes._These_have_an_anchor_in_the_running_text_and_some_content_that_ends_up_(normally)_at_the_bottom_of_the_page._When_considering_a_page_break_the_engine_tries_to_make_sure_that_the_anchor_(reference)_and_the_content_end_up_on_the_same_page._When_there_is_too_much,_it_will_distribute_(split)_the_content_over_pages._

We_can_discuss_page_breaks_in_a_(pseudo)_scientific_way_and_explore_how_to_optimize_this_process,_taking_into_accounts_also_inserts_that_contain_images_but_it_doesn't_make_much_sense_to_do_that_because_in_practice_we_can_encounter_all_kind_of_interferences._Theory_and_practice_are_too_different_because_a_document_can_contain_a_wild_mix_of_text,_figures,_formulas,_notes,_have_backgrounds_and_location_dependent_processing._It_get_seven_more_complex_when_we_are_dealing_with_columns_because_TEX_doesn't_really_know_tha

I_will_therefore_stick_to_some_practical_aspects_and_the_main_reason_for_this_document_is_that_I_sort_of_document_engine_features_and_at_the_same_time_give_an_impression_of_what_we_deal_with._I_will_do_that_in_the_perspective_of_LuaMetaTEX,_which_has_a_few_mor

Currently_this_document_is_mostly_for_myself_to_keep_track_of_the_state_of_inserts_and_the_page_builder_in_LuaMetaTEX_and_ConTEXt_LMTX._The_text_is_not_yet_corrected_and_can_h

14.2 The page builder

When_your_document_is_processed_content_eventually_gets_added_to_the_so_called_main_vertical_list_(mvl)._Content_first_get_appended_to_the_list_of_contributions_and_at_specific_moments_it_will_be_handed_over_to_the_mvl._This_process_is_called_page_building._There_we_can_encounter_the_following_elements_(nodes):_

`glue`	a vertical skip
`penalty`	a vertical penalty
`kern`	a vertical kern
`vlist`	a a vertical box
`hlist`	a horizontal box (often a line)
`rule`	a horizontal rule
`boundary`	a boundary node
`whatsit`	a node that is used by user code (often some extension)
`mark`	a token list (as used for running headers)
`insert`	a node list (as used for notes)

The_engine_itself_will_not_insert_anything_other_than_this_but_Lua_code_can_mess_up_ the_contribution_list_and_the_mvl_and_that_can_trigger_an_error._Handing_over_the_contributions_is_done_by_the_page_builder_and_that_one_kicks_in_in_several_places:_

- When_a_penalty_gets_inserted_it_is_part_of_evaluating_if_the_output_routine_should_ be_triggered._This_triggering_can_be_enforced_by_values_equal_or_below_10.000_that_ then_can_be_checked_in_the_set_routine._
- The_builder_is_*not*_exercised_when_a_glue_or_kern_is_injected_so_there_can_be_multiple_of_
- Adding_a_box_triggers_the_builder_as_does_the_result_of_an_alignment_which_can_ be_a_list_of_boxes._
- When_the_output_routine_is_finished_the_builder_is_executed_because_the_routine_ can_have_pushed_back_content._
- When_a_new_paragraph_is_triggered_by_the_`\par`_command_the_builder_kicks_in_ but_only_when_the_engine_was_able_to_enter_vertical_mode._
- When_the_job_is_finished_the_builder_will_make_sure_that_pending_content_is_handled._
- An_insert_and_vadjust_*can*_trigger_the_builder_but_only_at_the_nesting_level_zero_which_no
- At_the_beginning_of_a_paragraph_(like_text),_before_display_math_is_entered,_and_ when_display_math_ends_the_builder_is_also_activated._

At_the_TEX_the_builder_is_triggered_automatically_in_the_mentioned_cases_but_at_the_Lua_end

The_properties_that_relate_to_the_page_look_like_counter_and_dimension_registers_but_ they_are_not._These_variables_are_global_and_managed_differently._

`\pagegoal`	the available space
`\pagetotal`	the accumulated space
`\pagestretch`	the possible zero order stretch
`\pagefilstretch`	the possible one order stretch
`\pagefillstretch`	the possible second order stretch
`\pagefilllstretch`	the possible third order stretch

\pageshrink	the possible shrink
\pagedepth	the current page depth
\pagevsize	the initial page goal

When the first content is added to an empty page the \pagegoal gets the value of \vsize a gets frozen but the value is diminished by the space needed by left over inserts. These inserts are managed via a separate list so they don't interfere with the page that itself of course can have additional inserts. The \pagevsize is just a (LuaMetaTEX) status va

Another variable is \deadcycles that registers the number of times the output routine is called without returning result.

14.3 Inserts

We now come to inserts. In traditional TEX an insert is a data structure that runs on top of

In LuaMetaTEX you can set \insertmode to 1 and that is what we do in ConTEXt. In that m TEXt MkIV we use the range 127 upto 255 in order to avoid a clash with registers. In LMT

A consequence of this approach is that we use dedicated commands to set the insert properties:

\insertdistance	glue	the space before the first instance (on a page)
\insertmultiplier	count	a factor that is used to calculate the height used
\insertlimit	dimen	the maximum amount of space on a page to be taken
\insertpenalty	count	the floating penalty (used when set)
\insertmaxdepth	dimen	the maximum split depth (used when set)
\insertstorage	count	signals that the insert has to be stored for later
\insertheight	dimen	the accumulated height of the inserts so far
\insertdepth	dimen	the current depth of the inserts so far
\insertwidth	dimen	the width of the inserts

These commands take a number and an integer, dimension or glue specification. They can be set and queried but setting the dimensions can have side effects. The accumulated height of the inserts is available in \insertheights (which can be set too). The \f able determines the penalty applied when a split is needed.

In the output routine the original TEX variable \insertpenalties is a counter that keeps th

The LuaMetaTEX specific storage mode \insertstoring variable is explained in the next sec

14.4 Storing

This_feature_is_kind_of_special_and_still_experimental._When_`\insertstoring`_is_set_1,_all_inserts_that_have_their_storage_flag_set_will_be_saved._Think_of_a_multi_column_setup_where_inserts_have_to_end_up_in_the_last_column._If_there_are_three_columns,_the_first_two_will_store_inserts._Then_when_the_last_column_is_dealt_with_`\insertstoring`_can_be_set_to_2_and_that_will_signal_the_builder_that_we_will_inject_the_inserts._In_both_cases,_the_value_of_this_register_will_be_set_to_zero_so_that_it_doesn't_influence_further_processing._

14.5 Synchronizing

The_page_builder_can_triggered_by_(for_instance)_a_penalty_but_you_can_also_use_`\pagebound`
will_trigger_the_page_builder_but_not_leave_anything_behind._(This_is_experimental.)_

14.6 Migration

Todo,_nothing_new_there,_so_no_hurry._

14.7 Callbacks

Todo,_nothing_new_there,_so_no_hurry._

14.7 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

15 Localboxes

low level TEX

localboxes

Contents

15.1 Introduction

The_LuaTEX_engine_inherited_a_few_features_from_other_engines_and_adding_local_boxes_to_p
MetaTEX_specific_\localmiddlebox_primitives._WHen_these_primitives_are_used_in_vertical_mo

The_original_(Omega)_idea_was_that_local_boxes_are_used_for_repetitive_punctuation_
(like_quotes)_at_the_left_and/or_right_end_of_the_lines_that_make_up_a_paragraph._That_
means_that_when_these_primitives_inject_nodes_they_actually_introduce_states_so_that_
a_stretch_of_text_can_be_marked._

When_this_mechanism_was_cleaned_up_in_LuaMetaTEX_I_decided_to_investigate_if_other_usage
MetaTEX_is_not_compatible_with_LuaTEX._

This_is_a_preliminary,_uncorrected_manual._

15.2 The basics

This_mechanism_uses_a_mix_of_setting_(pseudo_horizontal)_box_registers_that_get_as-
sociated_with_(positions_in_a)_paragraph._When_the_lines_resulting_from_breaking_the_
list_gets_packaged_into_an_horizontal_(line)_box,_the_local_left_and_right_boxes_get_prepended
and_appended_to_the_textual_part_(inside_the_left,_right_and_parfills_kips_and_left_or_
right_hanging_margins)._When_assigning_the_current_local_boxes_to_the_paragraph_node(s)_re
erences_to_the_pseudo_registers_are_used_and_the_packaging_actually_copies_them._This_
mix_of_referencing_and_copying_is_somewhat_tricky_but_the_engine_does_it_best_to_hide_
this_for_the_user._

This_mechanism_is_rather_useless_when_not_wrapped_into_some_high_level_mechanism_
because_by_default_setting_these_boxes_wipes_the_existing_value._In_LuaMetaTEX_you_can_act
TEX,_in_LuaMetaTEX_the_local_box_registers_have_a_linked_lists_of_local_boxes_tagged_by_inde
TEX_here_because_there_we_don't_have_access._This_is_why_usage_as_in_LuaTEX_will_also_wor
MetaTEX._

This_mechanism_obeys_grouping_as_is_demonstrated_in_the_next_three_examples._The_
first_example_is:_

```
\start
    \dorecurse{10}{test #1.1 }
    \localleftbox{\blackrule[width=2em,color=darkred] }
    \dorecurse{20}{test #1.2 }
    \removeunwantedspaces
    \localrightbox{ \blackrule[width=3em,color=darkblue]}
    \dorecurse{20}{test #1.3 }
\stop
    \dorecurse{20}{test #1.4 }
    % par ends here
```

The_next_example_differs_in_a_subtle_way:_watch_the_keep_keyword,_it_makes_the_setting_retain_after_the_group_ends._

```
\start
    \start
        \dorecurse{10}{test #1.1 }
        \localleftbox keep {\blackrule[width=2em,color=darkred] }
        \dorecurse{20}{test #1.2 }
        \removeunwantedspaces
        \localrightbox { \blackrule[width=3em,color=darkblue]}
        \dorecurse{20}{test #1.3 }
    \stop
        \dorecurse{20}{test #1.4 }
\stop
% par ends here
```

The_third_example_has_two_times_keep._This_option_is_LuaMetaTEX_specific._

```
\start
    \start
        \dorecurse{10}{test #1.1 }
        \localleftbox keep {\blackrule[width=2em,color=darkred] }
        \dorecurse{20}{test #1.2 }
        \removeunwantedspaces
        \localrightbox keep { \blackrule[width=3em,color=darkblue]}
        \dorecurse{20}{test #1.3 }
    \stop
        \dorecurse{20}{test #1.4 }
\stop
% par ends here
```

test 1.1 test 2.1 test 3.1 test 4.1 test 5.1 test 6.1 test 7.1 test 8.1 test 9.1 test 10.1 test 1.2 test 2.2 test 3.2 test 4.2 test 5.2 test 6.2 test 7.2 test 8.2 test 9.2 test 10.2 test 11.2 test 12.2 test 13.2 test 14.2 test 15.2 test 16.2 test 17.2 test 18.2 test 19.2 test 20.2 test 1.3 test 2.3 test 3.3 test 4.3 test 5.3 test 6.3 test 7.3 test 8.3 test 9.3 test 10.3 test 11.3 test 12.3 test 13.3 test 14.3 test 15.3 test 16.3 test 17.3 test 18.3 test 19.3 test 20.3 test 1.4 test 2.4 test 3.4 test 4.4 test 5.4 test 6.4 test 7.4 test 8.4 test 9.4 test 10.4 test 11.4 test 12.4 test 13.4 test 14.4 test 15.4 test 16.4 test 17.4 test 18.4 test 19.4 test 20.4

Example 1

test 1.1 test 2.1 test 3.1 test 4.1 test 5.1 test 6.1 test 7.1 test 8.1 test 9.1 test 10.1 test 1.2 test 2.2 test 3.2 test 4.2 test 5.2 test 6.2 test 7.2 test 8.2 test 9.2 test 10.2 test 11.2 test 12.2 test 13.2 test 14.2 test 15.2 test 16.2 test 17.2 test 18.2 test 19.2 test 20.2 test 1.3 test 2.3 test 3.3 test 4.3 test 5.3 test 6.3 test 7.3 test 8.3 test 9.3 test 10.3 test 11.3 test 12.3 test 13.3 test 14.3 test 15.3 test 16.3 test 17.3 test 18.3 test 19.3 test 20.3 test 1.4 test 2.4 test 3.4 test 4.4 test 5.4 test 6.4 test 7.4 test 8.4 test 9.4 test 10.4 test 11.4 test 12.4 test 13.4 test 14.4 test 15.4 test 16.4 test 17.4 test 18.4 test 19.4 test 20.4

Example 2

test 1.1 test 2.1 test 3.1 test 4.1 test 5.1 test 6.1 test 7.1 test 8.1 test 9.1 test 10.1 test 1.2 test 2.2 test 3.2 test 4.2 test 5.2 test 6.2 test 7.2 test 8.2 test 9.2 test 10.2 test 11.2 test 12.2 test 13.2 test 14.2 test 15.2 test 16.2 test 17.2 test 18.2 test 19.2 test 20.2 test 1.3 test 2.3 test 3.3 test 4.3 test 5.3 test 6.3 test 7.3 test 8.3 test 9.3 test 10.3 test 11.3 test 12.3 test 13.3 test 14.3 test 15.3 test 16.3 test 17.3 test 18.3 test 19.3 test 20.3 test 1.4 test 2.4 test 3.4 test 4.4 test 5.4 test 6.4 test 7.4 test 8.4 test 9.4 test 10.4 test 11.4 test 12.4 test 13.4 test 14.4 test 15.4 test 16.4 test 17.4 test 18.4 test 19.4 test 20.4

Example 3

Figure 15.1

One_(nasty)_side_effect_is_that_when_you_set_these_boxes_ungrouped_they_are_applied_to_whatever_follows,_which_is_why_resetting_them_is_built_in_the_relevant_parts_of_ConTEXt._The_next_examples_are_typeset_grouped_an_demonstrate_the_use_of_indices:_

```
\dorecurse{20}{before #1 }
\localleftbox{\bf \darkred L 1 }%
\localleftbox{\bf \darkred L 2 }%
\dorecurse{20}{after #1 }
```

before_1_before_2_before_3_before_4_before_5_before_6_before_7_before_8_before_9_before_10_before_11_before_12_before_13_before_14_before_15_before_16_before_17_before_18_before_19_before_20__after_1_after_2_after_3_after_4_after_5_after_6_after_7_after_8_a

Indices_can_be_set_for_both_sides:_

```
\dorecurse{5}{\localrightbox index #1{ \bf \darkgreen R #1}}%
\dorecurse{20}{before #1 }
\dorecurse{5}{\localleftbox index #1{\bf \darkred L #1 }}%
\dorecurse{20}{after #1 }
```

before_1_before_2_before_3_before_4_before_5_before_6_before__**R_1_R_2_R_3_R_4_R_5**
7_before_8_before_9_before_10_before_11_before_12_before__**R_1_R_2_R_3_R_4_R_5**
13_before_14_before_15_before_16_before_17_before_18_before__**R_1_R_2_R_3_R_4_R_5**
19_before_20__after_1_after_2_after_3_after_4_after_5_after_6_after_7_after_8_after_9_after_10_

We_can_instruct_this_mechanism_to_hook_the_local_box_into_the_main_par_node_by_using_the `par` keyword._Keep_in_mind_that_these_local_boxes_only_come_into_play_when_the_lines_are_broken,_so_till_then_changing_them_is_possible._

```
\dorecurse{3}{\localrightbox index #1{ \bf \darkgreen R #1}}%
\dorecurse{20}{before #1 }
\dorecurse{2}{\localleftbox par index #1{\bf \darkred L #1 }}%
\dorecurse{20}{after #1 }
```

L_1_L_2_before_1_before_2_before_3_before_4_before_5_before_6_before_7_be-**_R_1_R_2_R_3**
L_1_L_2_fore_8_before_9_before_10_before_11_before_12_before_13_before__**R_1_R_2_R_3**
L_1_L_2_14_before_15_before_16_before_17_before_18_before_19_before_20__af-**_R_1_R_2_R_3**
L_1_L_2_ter_1_after_2_after_3_after_4_after_5_after_6_after_7_after_8_after__**R_1_R_2_R_3**
L_1_L_2_9_after_10_after_11_after_12_after_13_after_14_after_15_after_16_af-**_R_1_R_2_R_3**
L_1_L_2_ter_17_after_18_after_19_after_20___**R_1_R_2_R_3**

15.3 The interface

The_interface_described_here_is_experimental._

Because_it_is_hard_to_foresee_if_this_mechanism_will_be_used_at_all_the_ConTEXt_interface_is_

```
\definelocalboxes
  [lefttext]
  [location=lefttext,width=3em,color=darkblue]
\definelocalboxes
```

```
  [lefttextx]
  [location=lefttext,width=3em,color=darkblue]

\definelocalboxes
  [righttext]
  [location=righttext,width=3em,color=darkyellow]
\definelocalboxes
  [righttextx]
  [location=righttext,width=3em,color=darkyellow]
```

The_order_of_definition_matters!_Here_the_x_variants_have_a_larger_index_number._There_can_(currently)_be_at_most_256_indices._The_defined_local_boxes_are_triggered_with_\localbo

```
\startnarrower
\dorecurse{20}{before #1 }%
\localbox[lefttext]{[L] }%
\localbox[lefttextx]{[LL] }%
\localbox[righttext]{ [RR]}%
\localbox[righttextx]{ [R]}%
\dorecurse{20}{ after #1}%
\stopnarrower
```

Watch_how_we_obey_the_margins:_

> before_1_before_2_before_3_before_4_before_5_before_6_before_7_before_8_before_9_before_10_before_11_before_12_before_13_before_14_before_15_before_16_before_17_before_18_before_19_before_20__after_1_after_2_after_3_after_4_after_5_after_6_afte

Here_these_local_boxes_have_dimensions._The_predefined_margin_variants_are_virtual._Here_we_set_up_the_style_and_color:_

```
\setuplocalboxes
  [leftmargin]
  [style=\bs,
   color=darkgreen]
\setuplocalboxes
  [rightmargin]
  [style=\bs,
   color=darkred]

\dorecurse{2}{
    \dorecurse{10}{some text #1.##1 }%
    KEY#1.1%
```

```
    \localmargintext[leftmargin]{L #1.1}%
    \localmargintext[rightmargin]{R #1.1}%
    \dorecurse{10}{some text #1.##1 }%
    KEY#1.2%
    \localmargintext[leftmargin]{L #1.2}%
    \localmargintext[rightmargin]{R #1.2}%
    \dorecurse{10}{some text #1.##1 }%
    \blank
}
```

You can also use leftedge and rightedge but using them here would put them outside the page.

some text 1.1 some text 1.2 some text 1.3 some text 1.4 some text 1.5 some text 1.6 some text 1.7 some text 1.8 some text 1.9 some text 1.10 KEY1.1some text 1.1 some te

L 1.2

R 1.2

some text 2.1 some text 2.2 some text 2.3 some text 2.4 some text 2.5 some text 2.6 some text 2.7 some text 2.8 some text 2.9 some text 2.10 KEY2.1some text 2.1 some te

L 2.2

R 2.2

In previous examples you can see that setting something at the left will lag behind so deep down we use another trick here: \localmiddlebox. When these boxes get placed a callback can be triggered and in ConTEXt we use that to move these middle boxes

Next we implement line numbers. Watch out: this will not replace the existing mechanisms, it's just an alternative as we have alternative table mechanisms. We have a repertoire of helpers for constructing the result:

```
\definelocalboxes
  [linenumberleft]
  [command=\LeftNumber,
   location=middle,
   distance=\leftmargindistance,
   width=3em,
   style=\bs,
   color=darkred]

\definelocalboxes
  [linenumberright] % [linenumberleft]
  [command=\RightNumber,
   location=middle,
   distance=\rightmargindistance,
   width=3em,
```

```
   style=\bf,
   color=darkgreen]

\definecounter[MyLineNumberL]
\definecounter[MyLineNumberR]

\setupcounter
  [MyLineNumberL]
  [numberconversion=characters]

\setupcounter
  [MyLineNumberR]
  [numberconversion=romannumerals]

\def\LineNumberL
  {\incrementcounter[MyLineNumberL]%
   \convertedcounter[MyLineNumberL]}

\def\LineNumberR
  {\incrementcounter[MyLineNumberR]%
   \convertedcounter[MyLineNumberR]}

\protected\def\LeftNumber
  {\setbox\localboxcontentbox\hbox
     to \localboxesparameter{width}
     {(\LineNumberL\hss\strut)}%
   \localmarginlefttext\zeropoint}

\protected\def\RightNumber
  {\setbox\localboxcontentbox\hbox
     to \localboxesparameter{width}
     {(\strut\hss\LineNumberR)}%
   \localmarginrighttext\zeropoint}

\localbox[linenumberleft]{}%
\localbox[linenumberright]{}%
\dorecurse{2}{
    \samplefile{tufte}
    \par
}
\resetlocalbox[linenumberleft]%
\resetlocalbox[linenumberright]%
```

We_use_our_tufte_example_to_illustrate_the_usage:_

(15.a) We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_capac- (15.i)
(15.b) ity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_syn (15.ii)
(15.c) thesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_cla (15.iii)
(15.d) sify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pi (15.iv)
(15.e) geonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_ab (15.v)
(15.f) erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip (15.vi)
(15.g) through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_win (15.vii)
(15.h) now_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__ (15.viii)

(15.i) We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_capac- (15.ix)
(15.j) ity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_syn (15.x)
(15.k) thesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_cla (15.xi)
(15.l) sify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pi (15.xii)
(15.m) geonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_ab (15.xiii)
(15.n) erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip (15.xiv)
(15.o) through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_win (15.xv)
(15.p) now_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__ (15.xvi)

For_convenience_we_support_ranges_like_this_(we've_reset_the_line_number_counters_here):_

```
\startlocalboxrange[linenumberleft]%
\startlocalboxrange[linenumberright]%
\dorecurse{2}{
    \samplefile{tufte}
    \par
}
\stoplocalboxrange
\stoplocalboxrange
```

(15.a) We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_capac- (15.i)
(15.b) ity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_syn (15.ii)
(15.c) thesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_cla (15.iii)
(15.d) sify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pi (15.iv)
(15.e) geonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_ab (15.v)
(15.f) erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip (15.vi)
(15.g) through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_win (15.vii)
(15.h) now_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__ (15.viii)

(15.i) We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_capac- **(15.ix)**
(15.j) ity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_synth- **(15.x)**
(15.k) thesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_clas- **(15.xi)**
(15.l) sify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pi- **(15.xii)**
(15.m) geonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_aver- **(15.xiii)**
(15.n) erage,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_ **(15.xiv)**
(15.o) through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_win- **(15.xv)**
(15.p) now_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__ **(15.xvi)**

15.4 The helpers

For_the_moment_we_have_these_helpers:_

```
\localboxindex             integer
\localboxlinenumber        integer
                           \localboxlinewidthdimension
\localboxlocalwidth        dimension
\localboxprogress          dimension
\localboxleftoffset        dimension
\localboxrightoffset       dimension
                           \localboxleftskip dimension
\localboxrightskip         dimension
\localboxlefthang          dimension
\localboxrighthang         dimension
                           \localboxindent   dimension
\localboxparfillleftskip   dimension
\localboxparfillrightskip  dimension
\localboxovershoot         dimension
```

The_progress_and_offsets_are_accumulated_values_of_the_normalized_indent,_hangs,_skips_etc._The_line_number_is_the_position_in_the_paragraph._In_the_callback_we_set_the_box_register_`\localboxcontentbox`_and_use_it_after_the_command_has_been_applied._In_the_line_number_example_you_can_see_how_we_set_its_final_content,_so_these_boxes_are_sort_of_dynamic._Normally_in_the_middle_case_no_content_is_passed_and_in_the_par_builder_a_middle_is_not_taken_into_account_when_calculating_the_line_width._

15.4 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

16 Loops

low level TEX

loops

Contents

16.1 Introduction

I_have_hesitated_long_before_I_finally_decided_to_implement_native_loops_in_LuaMeta-TEX._Among_the_reasons_against_such_a_feature_is_that_one_can_define_macros_that_do_loops_ MetaTEX_it_is_actually_possible_to_use_the_local_control_mechanism_to_hide_loop_counter_adva

In_the_next_sections_we_describe_the_new_native_loop_primitives_in_LuaMetaTEX_as_well_as_th TEXt_loop_helpers._

16.2 Primitives

Because_MetaPost,_which_is_also_a_macro_language,_has_native_loops,_it_makes_sense_ to_also_have_native_loops_in_TEX_and_in_LuaMetaTEX_it_was_not_that_hard_to_add_it._One_vari

```
\localcontrolledloop 1 100000 1 {%
    % body
}
```

Here_is_an_example_of_usage:_

```
\localcontrolledloop 1 5 1 {%
    [\number\currentloopiterator]
    \localcontrolledloop 1 10 1 {%
        (\number\currentloopiterator)
    }%
    [\number\currentloopiterator]
    \par
}
```

The_\currentloopiterator_is_a_numeric_token_so_you_need_to_explicitly_serialize_it_ with_\number_or_\the_if_you_want_it_to_be_typeset:_

[1]_(1)_(2)_(3)_(4)_(5)_(6)_(7)_(8)_(9)_(10)_[1]_
[2]_(1)_(2)_(3)_(4)_(5)_(6)_(7)_(8)_(9)_(10)_[2]_
[3]_(1)_(2)_(3)_(4)_(5)_(6)_(7)_(8)_(9)_(10)_[3]_
[4]_(1)_(2)_(3)_(4)_(5)_(6)_(7)_(8)_(9)_(10)_[4]_
[5]_(1)_(2)_(3)_(4)_(5)_(6)_(7)_(8)_(9)_(10)_[5]_

Here_is_another_example._This_time_we_also_show_the_current_nesting:_

```
\localcontrolledloop 1 100 1 {%
    \ifnum\currentloopiterator>6\relax
        \quitloop
    \else
        [\number\currentloopnesting:\number\currentloopiterator]
        \localcontrolledloop 1 8 1 {%
            (\number\currentloopnesting:\number\currentloopiterator)
        }\par
    \fi
}
```

Watch_the_`\quitloop`:_it_will_end_the_loop_at_the_*next*_iteration_so_any_content_after_it_will_s

[1:1]_(2:1)_(2:2)_(2:3)_(2:4)_(2:5)_(2:6)_(2:7)_(2:8)_
[1:2]_(2:1)_(2:2)_(2:3)_(2:4)_(2:5)_(2:6)_(2:7)_(2:8)_
[1:3]_(2:1)_(2:2)_(2:3)_(2:4)_(2:5)_(2:6)_(2:7)_(2:8)_
[1:4]_(2:1)_(2:2)_(2:3)_(2:4)_(2:5)_(2:6)_(2:7)_(2:8)_
[1:5]_(2:1)_(2:2)_(2:3)_(2:4)_(2:5)_(2:6)_(2:7)_(2:8)_
[1:6]_(2:1)_(2:2)_(2:3)_(2:4)_(2:5)_(2:6)_(2:7)_(2:8)_

The_three_loop_variants_all_perform_differently:_

```
l:\testfeatureonce {1000} {\localcontrolledloop 1 2000 1 {\relax}} %
  \elapsedtime
e:\testfeatureonce {1000} {\expandedloop        1 2000 1 {\relax}} %
  \elapsedtime
u:\testfeatureonce {1000} {\unexpandedloop      1 2000 1 {\relax}} %
  \elapsedtime
```

An_unexpanded_loop_is_(of_course)_the_fastest_because_it_only_collects_and_then_feeds_back_the_lot._In_an_expanded_loop_each_cycle_does_an_expansion_of_the_body_and_collects_the_result_which_is_then_injected_afterwards,_and_the_controlled_loop_just_expands_the_body_each_iteration._

`l:_0.091`

```
e:_0.089
u:_0.028
```

The_different_behavior_is_best_illustrated_with_the_following_example:_

```
\edef\TestA{\localcontrolledloop 1 5 1 {A}} % out of order
\edef\TestB{\expandedloop        1 5 1 {B}}
\edef\TestC{\unexpandedloop      1 5 1 {C\relax}}
```

We_can_show_the_effective_definition:_

```
\meaningasis\TestA
\meaningasis\TestB
\meaningasis\TestC
```

```
A: \TestA
B: \TestB
C: \TestC
```

Watch_how_the_first_test_pushes_the_content_in_the_main_input_stream:_

```
AAAAA_
\def_\TestA_{}
\def_\TestB_{BBBBB}
\def_\TestC_{C\relax_C\relax_C\relax_C\relax_C\relax_}
```

```
A:_
B:_BBBBB
C:_CCCCC
```

Here_are_some_examples_that_show_what_gets_expanded_and_what_not:_

```
\edef\whatever
  {\expandedloop 1 10 1
     {(\number\currentloopiterator)
      \scratchcounter=\number\currentloopiterator\relax}}

\meaningasis\whatever
```

```
\def_\whatever_{(1)_\scratchcounter_=1\relax_(2)_\scratchcounter_=2\relax_
(3)_\scratchcounter_=3\relax_(4)_\scratchcounter_=4\relax_(5)_\scratchcounter_=5\relax_
(6)_\scratchcounter_=6\relax_(7)_\scratchcounter_=7\relax_(8)_\scratchcounter_=8\relax_
(9)_\scratchcounter_=9\relax_(10)_\scratchcounter_=10\relax_}
```

A_local_control_encapsulation_hides_the_assignment:_

```
\edef\whatever
  {\expandedloop 1 10 1
     {(\number\currentloopiterator)
      \beginlocalcontrol
      \scratchcounter=\number\currentloopiterator\relax
      \endlocalcontrol}}

\meaningasis\whatever
```

```
\def_\whatever_{(1)_(2)_(3)_(4)_(5)_(6)_(7)_(8)_(9)_(10)_}
```

Here_we_see_the_assignment_being_retained_but_with_changing_values:_

```
\edef\whatever
  {\unexpandedloop 1 10 1
     {\scratchcounter=1\relax}}

\meaningasis\whatever
```

```
\def_\whatever_{\scratchcounter_=1\relax_\scratchcounter_=1\relax_\scratchcounter_
=1\relax_\scratchcounter_=1\relax_\scratchcounter_=1\relax_\scratchcounter_=1\relax_
\scratchcounter_=1\relax_\scratchcounter_=1\relax_\scratchcounter_=1\relax_
\scratchcounter_=1\relax_}
```

We_get_no_expansion_at_all:_

```
\edef\whatever
  {\unexpandedloop 1 10 1
     {\scratchcounter=\the\currentloopiterator\relax}}

\meaningasis\whatever
```

```
\def_\whatever_{\scratchcounter_=0\relax_\scratchcounter_=0\relax_\scratchcounter_
=0\relax_\scratchcounter_=0\relax_\scratchcounter_=0\relax_\scratchcounter_=0\relax_
\scratchcounter_=0\relax_\scratchcounter_=0\relax_\scratchcounter_=0\relax_
\scratchcounter_=0\relax_}
```

And_here_we_have_a_mix:_

```
\edef\whatever
  {\expandedloop 1 10 1
     {\scratchcounter=\the\currentloopiterator\relax}}
```

```
\meaningasis\whatever
```

```
\def_\whatever_{\scratchcounter_=1\relax_\scratchcounter_=2\relax_\scratchcounter_
=3\relax_\scratchcounter_=4\relax_\scratchcounter_=5\relax_\scratchcounter_=6\relax_
\scratchcounter_=7\relax_\scratchcounter_=8\relax_\scratchcounter_=9\relax_
\scratchcounter_=10\relax_}
```

There_is_one_feature_worth_noting._When_you_feed_three_numbers_in_a_row,_like_here,_there_is_a_danger_of_them_being_seen_as_one:_

```
\expandedloop
  \number\dimexpr1pt
  \number\dimexpr2pt
  \number\dimexpr1pt
  {}
```

This_gives_an_error_because_a_too_large_number_is_seen._Therefore,_these_loops_permit_leading_equal_signs,_as_in_assignments_(we_could_support_keywords_but_it_doesn't_make_much_sense):_

```
\expandedloop =\number\dimexpr1pt =\number\dimexpr2pt =\number\dimexpr1pt{}
```

16.3 Wrappers

We_always_had_loop_helpers_in_ConTEXt_and_the_question_is:_“What_we_will_gain_when_we_replace_the_definitions_with_ones_using_the_above?”._The_answer_is:_“We_have_little_performance_but_not_as_much_as_one_expects!”._This_has_to_do_with_the_fact_that_we_support_#1_as_iterator_and_#2_as_(verbose)_nesting_values_and_that_comes_with_some_overhead._It_is_also_the_reason_why_these_loop_macros_are_protected_(unexpandable)._However,_using_the_primitives_might_look_somewhat_more_natural_in_low_level_TEX_code._

Also,_replacing_their_definitions_can_have_side_effects_because_the_primitives_are_(and_will_be)_still_experimental_so_it's_typically_a_patch_that_I_will_run_on_my_machine_for_a_while._

Here_is_an_example_of_two_loops._The_inner_state_variables_have_one_hash,_the_outer_one_extra:_

```
\dorecurse{2}{
    \dostepwiserecurse{1}{10}{2}{
        (#1:#2) [##1:##2]
```

```
    }\par
}
```

We_get_this:_

(1:1)_[1:2]__(1:1)_[3:2]__(1:1)_[5:2]__(1:1)_[7:2]__(1:1)_[9:2]_
(2:1)_[1:2]__(2:1)_[3:2]__(2:1)_[5:2]__(2:1)_[7:2]__(2:1)_[9:2]_

We_can_also_use_two_state_macro_but_here_we_would_have_to_store_the_outer_ones:_

```
\dorecurse {2} {
    /\recursedepth:\recurselevel/
    \dostepwiserecurse {1} {10} {2} {
        <\recursedepth:\recurselevel>
    }\par
}
```

That_gives_us:_

/1:1/__<2:1>__<2:3>__<2:5>__<2:7>__<2:9>_
/1:2/__<2:1>__<2:3>__<2:5>__<2:7>__<2:9>_

An_endless_loop_works_as_follows:_

```
\doloop {
    ...
    \ifsomeconditionismet
        ...
        \exitloop
    \else
        ...
    \fi
  % \exitloopnow
    ...
}
```

Because_of_the_way_we_quit_there_will_not_be_a_new_implementation_in_terms_of_the_loop_primitives._You_need_to_make_sure_that_you_don't_leave_in_the_middle_of_an_ongoing_condition._The_second_exit_is_immediate._

We_also_have_a_(simple)_expanded_variant:_

```
\edef\TestX{\doexpandedrecurse{10}{!}} \meaningasis\TestX
```

This_helper_can_be_implemented_in_terms_of_the_loop_primitives_which_makes_them_ a_bit_faster,_but_these_are_not_critical:_

\def_\TestX_{!!!!!!!!!!}

A_variant_that_supports_#1_is_the_following:_

```
\edef\TestX{\doexpandedrecursed{10}{#1}} \meaningasis\TestX
```

So:_

\def_\TestX_{12345678910}

16.4 About quitting

You_can_quit_a_local_and_expanded_loop_at_the_next_iteration_using_`\quitloop`._With_ `\quitloopnow`_you_immediately_leave_the_loop_but_you_need_to_beware_of_side_effects,_like_ not_ending_a_condition_properly._Keep_in_mind_that_a_macro_language_like_TEX_is_not_that_fri

16.5 Simple repeaters

For_simple_iterations_we_have_`\localcontrolledrepeat`,_`\expandedrepeat`,_`\unexpand-edrepeat`._These_take_one_integer_instead_of_three:_the_final_iterator_value._

16.6 Endless loops

There_are_three_endless_loop_primitives:_`\localcontrolledendless`,_`\expandedend-less`,_`\unexpandedendless`._These_will_keep_running_till_you_quit_them._The_loop_counter_ can_overflow_the_maximum_integer_value_and_will_then_start_again_at_1._

16.7 Loop variables

The_following_example_shows_how_we_can_access_the_current,_parent_and_grand_par-ent_loop_iterator_values_using_a_parameter_like_syntax:_

```
\localcontrolledloop 1 4 1 {%
    \localcontrolledloop 1 3 1 {%
        \localcontrolledloop 1 2 1 {%
  \edef\foo{[#G,#P,#I]}\foo
  \def \oof{<#G,#P,#I>}\oof
```

```
            (#G,#P,#I)\space
        }
        \par
    }
}
```

```
[1,1,1]<1,1,1>(1,1,1)_[1,1,2]<1,1,2>(1,1,2)__
[1,2,1]<1,2,1>(1,2,1)_[1,2,2]<1,2,2>(1,2,2)__
[1,3,1]<1,3,1>(1,3,1)_[1,3,2]<1,3,2>(1,3,2)__
[2,1,1]<2,1,1>(2,1,1)_[2,1,2]<2,1,2>(2,1,2)__
[2,2,1]<2,2,1>(2,2,1)_[2,2,2]<2,2,2>(2,2,2)__
[2,3,1]<2,3,1>(2,3,1)_[2,3,2]<2,3,2>(2,3,2)__
[3,1,1]<3,1,1>(3,1,1)_[3,1,2]<3,1,2>(3,1,2)__
[3,2,1]<3,2,1>(3,2,1)_[3,2,2]<3,2,2>(3,2,2)__
[3,3,1]<3,3,1>(3,3,1)_[3,3,2]<3,3,2>(3,3,2)__
[4,1,1]<4,1,1>(4,1,1)_[4,1,2]<4,1,2>(4,1,2)__
[4,2,1]<4,2,1>(4,2,1)_[4,2,2]<4,2,2>(4,2,2)__
[4,3,1]<4,3,1>(4,3,1)_[4,3,2]<4,3,2>(4,3,2)__
```

Another_way_to_access_a(ny)_parent_is:_

```
\localcontrolledloop 1 4 1 {%
    \localcontrolledloop 1 3 1 {%
        \localcontrolledloop 1 2 1 {%
            (\the\previousloopiterator2,%
             \the\previousloopiterator1,%
             \the\currentloopiterator)
        }
        \par
    }
}
```

These_methods_make_that_one_doesn't_have_to_store_the_outer_loop_variables_for_usage_inside_the_inner_loop._Watch_out_with_the_`\edef`:_

```
\edef\foo{[#G,#P,#I]}
\def \oof{<#G,#P,#I>}

\localcontrolledloop 1 4 1 {%
    \localcontrolledloop 1 3 1 {%
        \localcontrolledloop 1 2 1 {%
        %
```

```
        % I iterator    \currentloopiterator
        % P parent      \previousloopiterator1
        % G grandparent \previousloopiterator2
        %
            \edef\ofo{[#G,#P,#I]}%
            \foo\oof\ofo(#G,#P,#I)\space
        %
        }
        \par
    }
}
```

```
[0,0,0]<1,1,1>[1,1,1](1,1,1)_[0,0,0]<1,1,2>[1,1,2](1,1,2)__
[0,0,0]<1,2,1>[1,2,1](1,2,1)_[0,0,0]<1,2,2>[1,2,2](1,2,2)__
[0,0,0]<1,3,1>[1,3,1](1,3,1)_[0,0,0]<1,3,2>[1,3,2](1,3,2)__
[0,0,0]<2,1,1>[2,1,1](2,1,1)_[0,0,0]<2,1,2>[2,1,2](2,1,2)__
[0,0,0]<2,2,1>[2,2,1](2,2,1)_[0,0,0]<2,2,2>[2,2,2](2,2,2)__
[0,0,0]<2,3,1>[2,3,1](2,3,1)_[0,0,0]<2,3,2>[2,3,2](2,3,2)__
[0,0,0]<3,1,1>[3,1,1](3,1,1)_[0,0,0]<3,1,2>[3,1,2](3,1,2)__
[0,0,0]<3,2,1>[3,2,1](3,2,1)_[0,0,0]<3,2,2>[3,2,2](3,2,2)__
[0,0,0]<3,3,1>[3,3,1](3,3,1)_[0,0,0]<3,3,2>[3,3,2](3,3,2)__
[0,0,0]<4,1,1>[4,1,1](4,1,1)_[0,0,0]<4,1,2>[4,1,2](4,1,2)__
[0,0,0]<4,2,1>[4,2,1](4,2,1)_[0,0,0]<4,2,2>[4,2,2](4,2,2)__
[0,0,0]<4,3,1>[4,3,1](4,3,1)_[0,0,0]<4,3,2>[4,3,2](4,3,2)__
```

16.7 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

17 Tokens

low level TEX

tokens

Contents

17.1 Introduction

Most_users_don't_need_to_know_anything_about_tokens_but_it_happens_that_when_TEXies_meet

That_said,_because_in_documents_about_TEX_the_word_‘token’_does_pop_up_I_will_try_to_give_a_little_insight_here._But_for_using_TEX_it's_mostly_irrelevant._The_descriptions_below_kens_as_I_see_them.”_

17.2 What are tokens

Both_the_words_‘node’_and_‘token’_are_quite_common_in_programming_and_also_rather_old_which_is_proven_by_the_fact_that_they_also_are_used_in_the_TEX_source._A_node_is_a_storacter’_nodes_(or_in_LuaTEX_speak_‘glyph’_nodes)_with_properties_like_the_font_and_the_character_referred_to._But_before_that_happens,_the_three_characters_in_the_input_t,_e_and_x interpreted_as_in_this_case_being_just_that:_characters._When_you_enter_\TeX_the_input_processors_first_sees_a_backslash_and_because_that_has_a_special_meaning_in_TEX_it_will_you_enter_$_TEX_will_look_ahead_for_a_second_one_in_order_to_determine_display_math,_push_

A_token_is_internally_just_a_32_bit_number_that_encodes_what_TEX_has_seen._It_is_the_assemb our_example_gets_tagged_as_such_and_encoded_in_this_number_in_a_way_that_the_intention_can_be_derived_later_on._

Now,_the_way_TEX_looks_at_these_tokens_can_differ._In_some_cases_it_will_just_look_at_this_(32

token	=	cmd	chr

[23] Talking about fashion: it would be more impressive to talk about TEX and friends as a software stack than calling it a distribution. Today, it's all about marketing.

Back to the three characters: these become tokens where the command code indicates that it is a letter and the char code stores what letter we have at hand and in the case of LuaTEX and LuaMetaTEX these are Unicode values. Contrary to the traditional code engines an utf sequence is read, but these multiple bytes still become one number that code has plenty of characters slots you can imagine that combining 16 catcode commands with code values makes a large repertoire of tokens.

There are more commands than the 16 basic characters related ones, in LuaMetaTEX we have just over 150 command codes (LuaTEX has a few more but they are also organized MetaTEX the first one has sub command code 9 (`vbox code`) and the second one has code 1

Now, before we move on it is important to know that al these codes are in fact abstract numbers. Although it is quite likely that engines that are derived from each other have similar numbers (just more) this is not the case for LuaMetaTEX. Because the interTEX) the command and char codes have been reorganized in a such a way that exposure is TEXt LMTX and LuaMetaTEX were pretty useable during the process), but also had to (re)co

So, input is converted into tokens, in most cases one-by-one. When a token is assembled, it either gets stored (deliberately or as part of some look ahead scanning), or it immediately gets (what is called:) expanded. Depending on what the command is, some action is triggered. For instance, a character gets appended to the node list immediately. An `\hbox` command will start assembling a box which its own node list that then gets some treatment: if this primitive was a follow up on `\setbox` it will get stored, otherwise it might end up in the current node list as so called hlist node. Commands that relate to registers have `0xFFFF` char codes because that is how many registers we have per category.

When a token gets stored for later processing it becomes part of a larger data structure, a so called memory word. These memory words are taken from a large pool of words and they store a token and additional properties. The info field contains the token value, the mentioned command and char. When there is no linked list, the link can actually be used to store a value, something that in LuaMetaTEX we actually do.

1	info	link
2	info	link
3	info	link
n	info	link

When for instance we say `\toks 0 {tex}` the scanner sees an escape, followed by 4 letters (`toks`) and the escape triggers a lookup of the primitive (or macro or `...`) with tha `0` will push back a copy of this list into the input.

In_addition_to_the_token_memory_pool,_there_is_also_a_table_of_equivalents._That_one_is_part_of_a_larger_table_of_memory_words_where_TEX_stores_all_it_needs_to_store._The_16_gr

main hash	null control sequence
	128K hash entries
	frozen control sequences
	special sequences (undefined)
registers	17 internal & 64K user glues
	4 internal & 64K user mu glues
	12 internal & 64K user tokens
	2 internal & 64K user boxes
	116 internal & 64K user integers
	0 internal & 64K user attribute
	22 internal & 64K user dimensions
specifications	5 internal & 0 user
extra hash	additional entries (grows dynamic)

So,_a_letter_token_t_is_just_that,_a_token._A_token_referring_to_a_register_is_again_just_a_number,_but_its_char_code_points_to_a_slot_in_the_equivalents_table._A_macro,_which_we_haven't_discussed_yet,_is_actually_just_a_token_list._When_a_name_lookup_happens_the_hash_table_is_consulted_and_that_one_runs_in_parallel_to_part_of_the_table_of_equivalents._When_there_is_a_match,_the_corresponding_entry_in_the_equivalents_table_points_to_a_token_list._

1	string index	equivalents or (next > n) index
2	string index	equivalents or (next > n) index
n	string index	equivalents or (next > n) index
n + 1	string index	equivalents or (next > n) index
n + 2	string index	equivalents or (next > n) index
n + m	string index	equivalents or (next > n) index

It_sounds_complex_and_it_actually_also_is_somewhat_complex._It_is_not_made_easier_by_the_fact_that_we_also_track_information_related_to_grouping_(saving_and_restoring),_need_reference_counts_for_copies_of_macros_and_token_lists,_sometimes_store_information_directly_instead_of_via_links_to_token_lists,_etc._And_again_one_cannot_compare_LuaMetaTEX_with_the_other_engines._Because_we_did_away_with_some_of_the_limitations_of_the_t

1	level	type	flag	value
2	level	type	flag	value
3	level	type	flag	value
n	level	type	flag	value

So, here LuaMetaTEX differs from other engines because it combines two tables, which is po
MetaTEX macros can have extra properties (flags) and these also need one byte. Contrary to

Because a macro starts with a reference count we have some room in the info field
to store information about it having arguments or not. It is these details that make
LuaMetaTEX a bit more efficient in terms of memory usage and performance than its ancest
TEX. But as with the other changes, it was a very stepwise process in order to keep the sys

17.3 Some implementation details

Sometimes there is a special head token at the start. This makes for easier append-
ing of extra tokens. In traditional TEX node lists are forward linked, in LuaTEX they are dou

For various reasons original TEX uses global variables temporary lists. This is for instance n
TEX we often just serialize lists and using local variables makes more sense. One of the first
MetaTEX was to group all global variables in (still global) structures but well isolated. That

Because TEX had to run on machines that we nowadays consider rather limited, it had to be
MetaTEX we stay as close to original TEX as possible but there have been some improvemen
TEX we store much more in nodes (each has a prev pointer and an attribute list pointer and

17.4 Other data management

There is plenty going on in TEX when it processes your input, just to mention a few:

- Grouping is handled by a nesting stack.
- Nested conditionals (`\if...`) have their own stack.
- The values before assignments are saved on the save stack.
- Also other local changes (housekeeping) ends up in the save stack.
- Token lists and macro aliases have references pointers (reuse).
- Attributes, being linked node lists, have their own management.

[24] On the agenda of LuaMetaTEX is to use this property in the underlying code, that doesn't yet profit from this and therefore keep previous pointers in store.

In all these subsystems tokens or references to tokens can play a role. Reading a single character from the input can trigger a lot of action. A curly brace tagged as begin group command will push the grouping level and from then on registers and some other quantities that are changed will be stored on the save stack so that after the group ends they can be restored. When primitives take keywords, and no match happens, tokens are pushed back into the input which introduces a new input level (also some stack). When numbers are read a token that represents no digit is pushed back too and macro packages use numbers and dimensions a lot. It is a surprise that TEX is so fast.

17.5 Macros

There is a distinction between primitives, the build in commands, and macros, the commands defined by users. A primitive relates to a command code and char code but macros are, unless they are made an alias to something else, like a `\countdef` or `\let` sically pointers to a token list. There is some additional data stored that makes it possible to parse and grab arguments.

When we have a control sequence (macro) `\controlsequence` the name is looked up in the hash table. When found its value will point to the table of equivalents. As mentioned, that table keeps track of the cmd and points to a token list (the meaning). We saw that this table also stores the current level of grouping and flags.

If we say, in the input, `\hbox to 10pt {x\hss}`, the box is assembled as we go and when it is appended to the current node list there are no tokens left. When scanning this, the engine literally sees a backslash and the four letters `hbox`. However when we have this:

```
\def\MyMacro{\hbox to 10pt {x\hss}}
```

the `\hbox` has become one memory word which has a token representing the `\hbox` primitive plus a link to the next token. The space after a control sequence is gobbled so the next two tokens, again stored in a linked memory word, are letter tokens, followed by two other and two letter tokens for the dimensions. Then we have a space, a brace, a letter, a primitive and a brace. The about 20 characters in the input became a dozen memory words each two times four bytes, so in terms of memory usage we end up with quite a bit more. However, when TEX runs over that list it only has to interpret the tol

17.6 Looking at tokens

When you say `\tracingall` you will see what the engine does: read input, expand primitives and macros, typesetting etc. You might need to set `\tracingonline` to get

a_bit_more_output_on_the_console._One_way_to_look_at_macros_is_to_use_the_\meaning_command,_so_if_we_have:_

```
\permanent\protected\def\MyMacro#1#2{Do #1 or #2!}
```

we_can_say_this:_

```
\meaning    \MyMacro
\meaningless\MyMacro
\meaningfull\MyMacro
```

and_get:_

```
protected_macro:#1#2->Do_#1_or_#2!
#1#2->Do_#1_or_#2!
permanent_protected_macro:#1#2->Do_#1_or_#2!
```

You_get_less_when_you_ask_for_the_meaning_of_a_primitive,_just_its_name._The_\meaningfull_itive_gives_the_most_information._In_LuaMetaTEX_protected_macros_are_first_class_commands:_MetaTEX._Instead_we_have_\tolerant_macros_but_that's_another_story._The_flags_that_were_m

For_the_above_definition,_the_\showluatokens_command_will_show_a_meaning_on_the_console._

```
\showluatokens\MyMacro
```

This_gives_the_next_list,_where_the_first_column_is_the_address_of_the_token,_the_second_one_the_command_code,_and_the_third_one_the_char_code._When_there_are_arguments_involved,_the_list_of_what_needs_to_get_matched_is_shown._

```
permanent protected control sequence: MyMacro
501263  19   49  match                argument 1
501087  19   50  match                argument 2
385528  20    0  end match
--------------
501090  11   68  letter               D (U+00044)
 30833  11  111  letter               o (U+0006F)
500776  10   32  spacer
385540  21    1  parameter reference
112057  10   32  spacer
431886  11  111  letter               o (U+0006F)
 30830  11  114  letter               r (U+00072)
 30805  10   32  spacer
```

```
500787  21    2  parameter reference
213412  12   33  other char           ! (U+00021)
```

In_the_next_subsections_I_will_give_some_examples._This_time_we_use_helper_defined_in_a_module:_

```
\usemodule[system-tokens]
```

17.6.1 Example 1: in the input

```
\luatokentable{1 \bf{2} 3\what {!}}
```

given token list:
<no tokens>

17.6.2 Example 2: in the input

```
\luatokentable{a \the\scratchcounter b \the\parindent \hbox to 10pt{x}}
```

given token list:
<no tokens>

17.6.3 Example 3: user registers

```
\scratchtoks{foo \framed{\red 123}456}

\luatokentable\scratchtoks
```

token register: scratchtoks
<no tokens>

17.6.4 Example 4: internal variables

```
\luatokentable\everypar
```

internal token variable: everypar
<no tokens>

17.6.5 Example 5: macro definitions

```
\protected\def\whatever#1[#2](#3)\relax
  {oeps #1 and #2 & #3 done ## error}

\luatokentable\whatever
```

protected control sequence: whatever						
598089	19	49	match			argument 1
598349	12	91	other char	[	U+0005B	
599415	19	50	match			argument 2
599100	12	93	other char	]	U+0005D	
595936	12	40	other char	(	U+00028	
598262	19	51	match			argument 3
598917	12	41	other char	)	U+00029	
595878	16	0	relax			relax
595733	20	0	end match			
593269	11	111	letter	o	U+0006F	
587738	11	101	letter	e	U+00065	
600492	11	112	letter	p	U+00070	
598845	11	115	letter	s	U+00073	
593073	10	32	spacer			
598500	21	1	parameter reference			
597484	10	32	spacer			
599562	11	97	letter	a	U+00061	
599661	11	110	letter	n	U+0006E	
596204	11	100	letter	d	U+00064	
598773	10	32	spacer			
600467	21	2	parameter reference			
579268	10	32	spacer			
596122	12	38	other char	&	U+00026	
599474	10	32	spacer			
597243	21	3	parameter reference			
597483	10	32	spacer			
597299	11	100	letter	d	U+00064	
600394	11	111	letter	o	U+0006F	
597957	11	110	letter	n	U+0006E	
595744	11	101	letter	e	U+00065	
595941	10	32	spacer			
596059	6	35	parameter			
596007	10	32	spacer			
597590	11	101	letter	e	U+00065	
598117	11	114	letter	r	U+00072	
596634	11	114	letter	r	U+00072	
596695	11	111	letter	o	U+0006F	
599663	11	114	letter	r	U+00072	

17.6.6 Example 6: commands

```
\luatokentable\startitemize
```

\luatokentable\stopitemize

frozen instance protected control sequence: startitemize						
521045	147	0	tolerant protected call			startitemgroup
521046	12	91	other char	[	U+0005B	
521047	11	105	letter	i	U+00069	
521048	11	116	letter	t	U+00074	
521049	11	101	letter	e	U+00065	
521050	11	109	letter	m	U+0006D	
521051	11	105	letter	i	U+00069	
521052	11	122	letter	z	U+0007A	
521053	11	101	letter	e	U+00065	
521054	12	93	other char	]	U+0005D	

frozen instance protected control sequence: stopitemize				
431358	143	0	protected call	stopitemgroup

17.6.7 Example 7: commands

\luatokentable\doifelse

permanent protected control sequence: doifelse				
55414	19	49	match	argument 1
55415	19	50	match	argument 2
55416	20	0	end match	
55417	137	29	if test	iftok
55418	1	123	left brace	
55419	21	1	parameter reference	
55420	2	125	right brace	
55421	1	123	left brace	
55422	21	2	parameter reference	
55423	2	125	right brace	
55424	131	0	expand after	expandafter
55425	142	0	call	firstoftwoarguments
55426	137	3	if test	else
55427	131	0	expand after	expandafter
55428	142	0	call	secondoftwoarguments
55429	137	2	if test	fi

17.6.8 Example 8: nothing

\luatokentable\relax

primitive control sequence: relax
<no tokens>

17.6.9 Example 9: hashes

```
\edef\foo#1#2{(#1)(\letterhash)(#2)}  \luatokentable\foo
```

control sequence: foo						
599661	19	49	match			argument 1
598439	19	50	match			argument 2
598535	20	0	end match			
596552	12	40	other char	(	U+00028	
595598	21	1	parameter reference			
600434	12	41	other char	)	U+00029	
479070	12	40	other char	(	U+00028	
599476	12	35	other char	#	U+00023	
592765	12	41	other char	)	U+00029	
596051	12	40	other char	(	U+00028	
594042	21	2	parameter reference			
599778	12	41	other char	)	U+00029	

17.6.10 Example 10: nesting

```
\def\foo#1{\def\foo##1{(#1)(##1)}}  \luatokentable\foo
```

control sequence: foo						
600076	19	49	match			argument 1
599372	20	0	end match			
596593	128	1	def			def
600484	142	0	call			foo
598583	6	35	parameter			
600394	12	49	other char	1	U+00031	
598770	1	123	left brace			
294	12	40	other char	(	U+00028	
594065	21	1	parameter reference			
600128	12	41	other char	)	U+00029	
597494	12	40	other char	(	U+00028	
596964	6	35	parameter			
599514	12	49	other char	1	U+00031	
595000	12	41	other char	)	U+00029	
598464	2	125	right brace			

17.6.11 Remark

In_all_these_examples_the_numbers_are_to_be_seen_as_abstractions._Some_command_codes_and_sub_command_codes_might_change_as_the_engine_evolves._This_is_why_the_LuaMetaTEX_engine_has_lots_of_Lua_functions_that_provide_information_about_what_number_r

17.6.11 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

18 Buffers

low level TEX

buffers

Contents

18.1 Preamble

Buffers_are_not_that_low_level_but_it_makes_sense_to_discuss_them_in_this_perspective_ because_it_relates_to_tokenization,_internal_representation_and_manipulating._

In_due_time_we_can_describe_some_more_commands_and_details_here._This_is_a_start._Feel_ free_to_tell_me_what_needs_to_be_explained._

18.2 Encoding

Normally_processing_a_document_starts_with_reading_from_file._In_the_past_we_were_ talking_single_bytes_that_were_then_maps_onto_a_specific_input_encoding_that_itself_ matches_the_encoding_of_a_font._When_you_enter_an_'a'_its_(normally_ascii)_number_ 97_becomes_the_index_into_a_font._That_same_number_is_also_used_in_the_hyphenator_ which_is_why_font_encoding_and_hyphenation_are_strongly_related._If_in_an_eight_bit_ TEX_engine_you_need_a_precomposed_'ä'_you_have_to_use_an_encoding_that_has_that_ character_in_some_slot_with_again_matching_fonts_and_patterns._The_actually_used_font_ can_have_the_*shapes*_in_different_slots_and_remapping_is_then_done_in_the_backend_code_usin

In_eight_bit_environments_all_this_brings_a_bit_of_a_resource_management_nightmare_ along_with_complex_installation_of_new_fonts._It_also_puts_strain_on_the_macro_package,_especially_when_you_want_to_mix_different_input_encodings_onto_different_font_ encodings_and_thereby_pattern_encodings_in_the_same_document._You_can_compare_ this_with_code_pages_in_operating_system,_but_imagine_them_potentially_being_mixed_ in_one_document,_which_can_happen_when_you_mix_multiple_languages_where_the_accumulated_number_of_different_characters_exceeds_256._You_end_up_switching_between_

encodings._One_way_to_deal_with_it_is_making_special_characters_active_and_let_their_meaning_differ_per_situation._That_is_for_instance_how_in_MkII_we_handled_utf8_and_thereby_got_around_distributing_multiple_pattern_files_per_language_as_we_only_needed_to_encoding_them_in_utf_and_then_remap_them_to_the_required_encoding_when_loading_patterns._A_mental_exercise_is_wondering_how_to_support_cjk_scripts_in_an_eight_bit_MkII,_something_that_actually_can_be_done_with_some_effort._

The_good_news_is_that_when_we_moved_from_MkII_to_MkIV_we_went_exclusively_utf8_because_that_is_what_the_LuaTEX_engine_expects._Upto_four_bytes_are_read_in_and_translated_in code_character._The_internal_representation_is_a_32_bit_integer_(four_bytes)_instead_of_a_singl TEXt)_but_I'm_pretty_sure_that_nowadays_no_one_uses_input_other_than_utf8._While_ConTEXt_is_normally_quite_upward_compatible_this_is_one_area_where_there_were_fundamental_ch

There_is_still_some_interpretation_going_on_when_reading_from_file:_for_instance,_we_need_to_normalize_the_Unicode_input,_and_we_feed_the_engine_separate_lines_on_demand._Ap

- *Example_code_has_to_be_typeset_as-is,_so_braces_etc._are_just_that.*_This_means_that_we_ha
- *Content_is_collected_and_used_later.*_A_separation_of_content_and_usage_later_on_often_help ping_a_table_in_a_buffer”_and_“including_that_buffer_when_a_table_is_placed”_using_the_placement_macros._
- *Embedded_MetaPost_and_Lua_code.*_These_languages_come_with_different_interpretation_of
- *The_content_comes_from_a_different_source.*_Examples_are_xml_files_where_angle_brackets_
- *The_content_is_generated.*_It_can_for_instance_come_from_Lua,_where_bytes_(representing_u

For_these_reasons_ConTEXt_always_had_ways_to_store_data_in_ways_that_makes_this_possible._

18.3 Performance

When_TEX_came_around,_the_bottlenecks_in_running_TEX_were_the_processor,_memory_and_dis `[2]..`_to_show_up._It_was_possible_to_run_TEX_on_a_personal_computer_but_it_was_somewhat pass)_trickery_was_demanding._

When_processors_became_faster_and_memory_plenty_the_disk_became_the_bottleneck,_but_that_changed_when_ssd's_showed_up._Combined_with_already_present_file_caching_that_had_some_impact._We_are_now_in_a_situation_that_cpu_cores_don't_get_that_much_faster_(at_least_not_twice_as_fast_per_iteration)_and_with_TEX_being_a_single_core_byte_cruncher_we' TEXt_code,_combined_with_LuaMetaTEX_will_give_you_what_you_need_with_a_reasonable_perfor takes’_are_made._Inefficient_Lua_and_TEX_code_has_way_more_impact_than_storing_a_few_mor

18.4 Files

Nearly_always_files_are_read_once_per_run._The_content_(mixed_with_commands)_is_scanned_and_macros_are_expanded_and/or_text_is_typeset_as_we_go._Internally_the_LuaMetaTEX_engine_is_in_“scanning_from_file”,_“scanning_from_token_lists”,_or_“scanning_from_Lua_output”_mode._The_first_mode_is_(in_principle)_the_slowest_because_utf_sequences_are_converted_to_tokens_(numbers)_but_there_is_no_way_around_it._The_second_method_is_fast_because_we_already_have_these_numbers,_but_we_need_to_take_into_account_where_the_linked_list_of_tokens_comes_from._If_it_is_converted_runtime_from_for_instance_file_input_or_macro_expansion_we_need_to_add_the_involved_overhead._But_scanning_a_stored_macro_body_is_pretty_efficient_especially_when_the_macro_is_part_of_the_loaded_macro_package_(format_file)._The_third_method_is_comparable_with_reading_from_file_but_here_we_need_to_add_the_overhead_involved_with_storing_the_Lua_output_into_data_structures_suitable_for_TEX's_input_mechanism,_which_can_involve_m
MetaTEX_we_even_went_a_bit_further,_also_because_we_know_what_kind_of_input,_processing_a

When_reading_from_file_or_Lua_output_we_interpret_bytes_turned_utf_numbers_and_that_is_when_catcode_regimes_kick_in:_characters_are_interpreted_according_to_the_catcode_properties:_escape_character_(backslash),_curly_braces_(grouping_and_arguments),_dollars_(math),_etc._While_with_reading_from_token_lists_these_catcodes_are_already_taken_care_of_and_we're_basically_interpreting_meanings_instead_of_characters._By_changing_the_catcode_regime_we_can_for_instance_typeset_content_verbatim_from_files_and_Lua_strings_but_when_reading_from_token_lists_we're_sort_of_frozen._There_are_tricks_to_reinterpret_the_token_list_but_that_comes_with_overhead_and_limitations._

18.5 Macros

A_macro_can_be_seen_as_a_named_token_with_a_meaning_attached._In_LuaMetaTEX_macros_ca
code_character_doesn't_need_all_four_bytes_of_an_integer_and_because_in_the_engine_numbers

```
\def\foo{abc} \foo \foo \foo
```

When_the_engine_fetches_a_token_from_a_list_it_will_interpret_the_command_and_when_it_fetches_from_file_it_will_create_tokens_on_the_fly_and_then_interpret_those._When_a_file_or_list_is_exhausted_the_engine_pops_the_stack_and_continues_at_the_previous_level._Because_macros_are_already_tokenized_they_are_more_efficient_than_file_input._For_more_about_macros_you_can_consult_the_low_level_document_about_them._

The_more_you_use_a_macro,_the_more_it_pays_off_compared_to_a_file._However_don't_overestimate_this,_because_in_the_end_the_typesetting_and_expanding_all_kind_of_other_involved_macros_might_reduce_the_file_overhead_to_noise._

18.6 Token lists

A_token_list_is_like_a_macro_but_is_part_of_the_variable_(register)_system._It_is_just_a_list_(so_no_arguments)_and_you_can_append_and_prepend_to_that_list._

```
\toks123={abc}    \the\toks123
\scratchtoks{abc} \the\scratchtoks
```

Here_`\scratchtoks`_is_defined_with_`\newtoks`_which_creates_an_efficient_reference_to_a_list_so_that,_contrary_to_the_first_line,_no_register_number_has_to_be_scanned._There_are_low_level_manuals_about_tokens_and_registers_that_you_can_read_if_you_want_to_know_more_about_this._As_with_macros_the_list_in_this_example_is_three_tokens_long._Contrary_to_macros_there_is_no_macro_overhead_as_there_is_no_need_to_check_for_arguments._[25]_

Because_they_use_more_or_less_the_same_storage_method_macros_and_token_list_registers_perform_the_same._The_power_of_registers_comes_from_some_additional_manipulators_in_LuaTEX_(and_LuaMetaTEX)_and_the_fact_that_one_can_control_expansion_with_`\the`,

18.7 Buffers

Buffers_are_something_specific_for_ConTEXt_and_they_have_always_been_part_of_this_system._A

```
\startbuffer[one]
line 1
line 2
\stopbuffer
```

Among_the_operations_on_buffers_the_next_two_are_used_most_often:_

```
\typebuffer[one]
\getbuffer[one]
```

Scanning_a_buffer_at_the_TEX_end_takes_a_little_effort_because_when_we_start_reading_the_ca

```
\startluacode
buffers.assign("one",[[
line 1
line 2
]])
```

[25] In LuaMetaTEX a macro without arguments is also quite efficient.

```
\stopluacode
```

Always keep in mind that buffers eventually are read as files: character by character, and at that time the content gets (as with other files) tokenized. A buffer name is optional. You can nest buffers, with and without names.

Because ConTeXt is very much about re-use of content and selective processing we have an

The `\definebuffer` command defines a new buffer environment. When you set buffers in Lua you don't need to define a buffer because likely you don't need the `\start` and `\sto` mands. Instead of `\getbuffer` you can also use `\getdefinedbuffer` with defined buffers. In that case the `before` and `after` keys of that specific instance are used.

The `\getinlinebuffer` command, which like the getters takes a list of buffer names, ignores leading and trailing spaces. When multiple buffers are flushed this way, spacing between buffers is retained.

The most important aspect of buffers is that the content is *not* interpreted and tokenized: th

```
\definebuffer[MyBuffer]

\startMyBuffer
\bold{this is
a buffer}
\stopMyBuffer

\typeMyBuffer \getMyBuffer
```

These commands result in:

```
\bold{this is
a buffer}
```

this is a buffer

There are not that many parameters that can be set: `before`, `after` and `strip` (when set to `no` leading and trailing spacing will be kept. The `\stop...` command, in our example `\stopMyBuffer`, can be defined independent to so something after the buffer has be read and stored but by default nothing is done.

You can test if a buffer exists with `\doifelsebuffer` (expandable) and `\doifelsebufferempty` (unexpandable). A buffer is kept in memory unless it gets wiped clean with `resetbuffer`.

```
\savebuffer      [MyBuffer][temp]     % gets name: jobname-temp.tmp
\savebufferinfile[MyBuffer][temp.log] % gets name: temp.log
```

You can also stepwise fill such a buffer:

```
\definesavebuffer[slide]

\startslide
    \starttext
\stopslide
\startslide
    slide 1
\stopslide
text 1 \par
\startslide
    slide 2
\stopslide
text 2 \par
\startslide
    \stoptext
\stopslide
```

After this you will have a file `\jobname-slide.tex` that has the two lines wrapped as text. You can set up a 'save buffer' to use a different filename (with the `file` key), a different prefix using `prefix` and you can set up a `directory`. A different name is set with the `list` key.

You can assign content to a buffer with a somewhat clumsy interface where we use the delimiter `\endbuffer`. The only restriction is that this delimiter cannot be part of the content:

```
\setbuffer[name]here comes some text\endbuffer
```

For more details and obscure commands that are used in other commands you can peek into the source.

Using buffers in the cld interface is tricky because of the catcode magick that is involved but there are setters and getters:

function	**arguments**
`buffers.assign`	`name, content [,catcodes]`
`buffers.erase`	`name`

```
buffers.prepend     name, content
buffers.append      name, content
buffers.exists      name
buffers.empty       name
buffers.getcontent  name
buffers.getlines    name
```

There are a few more helpers that are used in other (low level) commands. Their functionality might adapt to their usage there. The `context.startbuffer` and `context.stopbuffer` are somewhat differently defined than regular cld commands.

18.8 Setups

A setup is basically a macro but is stored and accessed in a namespace separated from ordinary macros. One important characteristic is that inside setups newlines are ignored.

```
\startsetups MySetupA
    This is line 1
    and this is line 2
\stopsetups

\setup{MySetupA}
```

This is line 1and this is line 2

A simple way out is to add a comment character preceded by a space. Instead you can also use `\space`:

```
\startsetups [MySetupB]
    This is line 1 %
    and this is line 2\space
    while here we have line 3
\stopsetups

\setup[MySetupB]
```

This is line 1 and this is line 2 while here we have line 3

You can use square brackets instead of space delimited names in definitions and also in calling up a (list of) setup(s). The `\directsetup` command takes a single setup name and is therefore more efficient.

Setups_are_basically_simple_macros_although_there_is_some_magic_involved_that_comes_from_their_usage_in_for_instance_xml_where_we_pass_an_argument._That_means_we_can_do_the_following:_

```
\startsetups MySetupC
    before#1after
\stopsetups

\setupwithargument{MySetupC}{ {\em and} }
```

before_*and*_after_

Because_a_setup_is_a_macro,_the_body_is_a_linked_list_of_tokens_where_each_token_takes_8_bytes_of_memory,_so_MySetupC_has_12_tokens_that_take_96_bytes_of_memory_(plus_some_overhead_related_to_macro_management)._

18.9 xml

Discussing_xml_is_outside_the_scope_of_this_document_but_it_is_worth_mentioning_that_once_an_xml_tree_is_read_is,_the_content_is_stored_in_strings_and_can_be_filtered_into_TEX,_where_it_is_interpreted_as_if_coming_from_files_(in_this_case_Lua_strings)._If_needed_the_

18.10 Lua

As_mentioned_already,_output_from_Lua_is_stored_and_when_a_Lua_call_finishes_it_ends_up_on_the_so_called_input_stack._Every_time_the_engine_needs_a_token_it_will_fetch_from_the_input_stack_and_the_top_of_the_stack_can_represent_a_file,_token_list_or_Lua_output._Interpreting_bytes_from_files_or_Lua_strings_results_in_tokens._As_a_side_note:_Lua_output_can_also_be_already_tokenized,_because_we_can_actually_write_tokens_and_nodes_from_Lua,_but_that's_more_an_implementation_detail_that_makes_the_Lua_input_stack_entries_a_bit_more_complex._It_is_normally_not_something_users_will_do_when_they_use_Lua_in_their_documents._

18.11 Protection

When_you_define_macros_there_is_the_danger_of_overloading_some_defined_by_the_system._Best_use_CamelCase_so_that_you_stay_away_from_clashes._You_can_enable_some_checking:_

```
\enabledirectives[overloadmode=warning]
```

or_when_you_want_to_quit_on_a_clash:_

```
\enabledirectives[overloadmode=error]
```

When_these_trackers_are_enabled_you_can_get_around_the_check_with:_

```
\pushoverloadmode
  ...
\popoverloadmode
```

But_delay_that_till_you're_sure_that_redefining_is_okay._

18.11 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

19 Accuracy

low level TEX

accuracy

Contents

19.1 Introduction

This_is_work_in_progress,_uncorrected._

When_you_look_at_TEX_and_MetaPost_output_the_accuracy_of_the_rendering_stands_out,_unless
portable)_floating_point_calculations,_it_does_all_with_32_bit_integers,_except_in_the_
backend_where_glue_calculations_are_used_for_finalizing_the_glue_values._It_all_changed_
a_bit_when_we_added_Lua_because_there_we_mix_integers_and_doubles_but_in_prac-
tice_it_works_out_okay._

When_looking_at_floating_point_(and_posits)_one_can_end_up_in_discussions_about_which_
one_is_better,_what_the_flaws_fo_each_are,_etc._Here_we're_only_interested_in_the_fact_
that_posits_are_more_accurate_in_the_ranges_where_TEX_and_MetaPost_operate,_as_well_as_th

When_you_work_with_dimensions_like_points,_they_get_converted_to_an_integer_num-
ber_(the_sp_unit)_and_from_that_it's_just_integer_calculations._The_maximum_dimension_
is_16383.99998pt,_which_already_shows_a_rounding_issue._Of_course_when_one_goes_
precise_for_sure_there_is_some_loss,_but_on_the_average_we're_okay._So,_in_the_next_
example_the_two_last_rows_are_equivalent:_

```
       .1pt  0.1pt   6554sp
       .2pt  0.2pt  13107sp
       .3pt  0.3pt  19661sp
.1pt + .2pt  0.3pt  19661sp
```

When_we're_at_the_Lua_end_things_are_different,_there_numbers_are_mapped_onto_64_bit_
floating_point_variables_(doubles)_and_not_all_numbers_map_well._This_is_what_we_get_
when_we_work_with_doubles_in_Lua:_

```
     .1                   0.1
     .2                   0.2
     .3                   0.3
.1 + .2  0.30000000000000004
```

The_serialization_looks_as_if_all_is_okay_but_when_we_test_for_equality_there_is_a_problem:_

```
     .3 == .3  true
.1 + .2 == .3  false
```

This_means_that_a_test_like_this_can_give_false_positives_or_negatives_unless_one_tests_the_difference_against_the_accuracy_(in_MetaPost_we_have_the_eps_variable_for_that)._In_TEX_clipping_of_the_decimal_fraction_influences_equality._

```
     \iflua{ .3 == .3 } Y\else N\fi  different
\iflua{ .1 + .2 == .3 } Y\else N\fi  different
```

The_serialization_above_misguides_us_because_the_number_of_digits_displayed_is_limited._Actually,_when_we_would_compare_serialized_strings_the_equality_holds,_definitely_within_the_accuracy_of_TEX._But_here_is_reality:_

	.3	.1 + .2
%0.10g	0.3	0.3
%0.17g	0.29999999999999999	0.30000000000000004
%0.20g	0.2999999999999999889	0.30000000000000004441
%0.25g	0.29999999999999999888977698	0.30000000000000000444089210

The_above_examples_use_0.1,_0.2_and_0.3_and_on_a_32_bit_float_that_actually_works_out_okay,_but_LuaMetaTEX_is_64_bit._Is_this_really_important_in_practice?_There_are_indeed_ca

19.2 Posits

The_next_table_shows_the_same_as_what_we_started_with_but_with_a_different_serialization._

```
     .1          0.1
     .2          0.2
     .3  0.300000001
.1 + .2  0.300000001
```

And_here_we_get_equality_in_both_cases:_

```
     .3 == .3  true
.1 + .2 == .3  true
```

The_next_table_shows_what_we_actually_are_dealing_with._The_`\if`-test_is_not_a_primitive_but_provided_by_ConTEXt._

```
     \ifpositunum{ .3 == .3 } Y\else N\fi  equal
\ifpositunum{ .1 + .2 == .3 } Y\else N\fi  equal
```

And_what_happens_when_we_do_more_complex_calculations:_

```
math .sin(0.1 + 0.2) == math .sin(0.3)  false
posit.sin(0.1 + 0.2) == posit.sin(0.3)  true
```

Of_course_other_numbers_might_work_out_differently!_I_just_took_the_simple_tests_that_came_to_mind._

So_what_are_these_posits?_Here_it's_enough_to_know_that_they_are_a_different_way_to_store_numbers_with_fractions._They_still_can_loose_precision_but_a_bit_less_on_smaller_values_and_often_we_have_relative_small_values_in_TEX._Here_are_some_links:_

```
https://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf
https://posithub.org/conga/2019/docs/14/1130-FlorentDeDinechin.pdf
```

There_are_better_explanations_out_there_than_I_can_provide_(if_at_all)._When_I_first_read_about_these_unums_(a_review_of_the_2015_book_“The_End_of_Error_Unum_Computing”)_I_was_intrigued_and_when_in_2023_I_read_something_about_it_in_relation_to_RISCV_I_decided_to_just_add_this_playground_for_the_users._After_all_we_also_have_decimal_support._And_interval_based_solutions_might_actually_be_good_for_MetaPost,_so_that_is_why_we_have_it_as_extra_number_model._There_we_need_to_keep_in_mind_that_MetaPost_in_non_scaled_models_also_apply_some_of_the_range_checking_and_clipping_that_happens_in_scaled_(these_magick_4096_tricks)._

For_now_it_is_enough_to_know_that_it's_an_alternative_for_floats_that_*could*_work_better_in_so

But_how_about_TEX?_Per_April_2023_the_LuaMetaTEX_engine_has_native_support_for_floats_(th
TEXt)._How_that_works_can_be_demonstrated_with_some_examples._The_float_related_comman

```
\scratchdimen=1.23456pt
\scratchfloat=1.23456
```

We_now_use_these_two_variables_in_an_example:_

```
\setbox0\hbox to \scratchdimen {x}\the\wd0
\scratchdimen \dimexpr \scratchdimen * 2\relax
\setbox0\hbox to \scratchdimen {x}\the\wd0
```

[26] Are 64 bit posits actually being worked on in softposit? There are some commented sections. We also need to patch some unions to make it compile as C.

```
\advance \scratchdimen \scratchdimen
\setbox0\hbox to \scratchdimen {x}\the\wd0
\multiply\scratchdimen by 2
\setbox0\hbox to \scratchdimen {x}\the\wd0
```

```
1.23456pt
2.46912pt
4.93823pt
9.87646pt
```

When_we_use_floats_we_get_this:_

```
\setbox0\hbox to \scratchfloat {x}\the\wd0
\scratchfloat \floatexpr \scratchfloat * 2\relax
\setbox0\hbox to \scratchfloat {x}\the\wd0
\advance \scratchfloat \scratchfloat
\setbox0\hbox to \scratchfloat {x}\the\wd0
\multiply\scratchfloat by 2
\setbox0\hbox to \scratchfloat {x}\the\wd0
```

```
1.23456pt
2.46912pt
4.93823pt
9.87648pt
```

So_which_approach_is_more_accurate?_At_first_sight_you_might_think_that_the_dimensions_are_better_because_in_the_last_two_lines_they_indeed_duplicate._However,_the_next_example_shows_that_with_dimensions_we_lost_some_between_steps._

```
                                                \the\scratchfloat
\scratchfloat \floatexpr \scratchfloat * 2\relax \the\scratchfloat
\advance \scratchfloat \scratchfloat             \the\scratchfloat
\multiply\scratchfloat by 2                      \the\scratchfloat
```

```
1.2345599979162216187
2.4691199958324432373
4.9382399916648864746
9.8764799833297729492
```

One_problem_with_accuracy_is_that_it_can_build_up._So_when_one_eventually_does_some_comparison_the_expectations_can_be_wrong._

```
\dimen0=1.2345pt
```

```
\dimen2=1.2345pt

\ifdim           \dimen0=\dimen2 S\else D\fi \space +0sp: [dim]
\ifintervaldim0sp\dimen0 \dimen2 O\else D\fi \space +0sp: [0sp]

\advance\dimen2 1sp

\ifdim             \dimen0=\dimen2 S\else D\fi \space +1sp: [dim]
\ifintervaldim 1sp \dimen0 \dimen2 O\else D\fi \space +1sp: [1sp]
\ifintervaldim 1sp \dimen2 \dimen0 O\else D\fi \space +1sp: [1sp]
\ifintervaldim 2sp \dimen0 \dimen2 O\else D\fi \space +1sp: [2sp]
\ifintervaldim 2sp \dimen2 \dimen0 O\else D\fi \space +1sp: [2sp]

\advance\dimen2 1sp

\ifintervaldim 1sp \dimen0\dimen2 O\else D\fi \space +2sp: [1sp]
\ifintervaldim 1sp \dimen2\dimen0 O\else D\fi \space +2sp: [1sp]
\ifintervaldim 5sp \dimen0\dimen2 O\else D\fi \space +2sp: [5sp]
\ifintervaldim 5sp \dimen2\dimen0 O\else D\fi \space +2sp: [5sp]
```

Here we show a test for overlap in values, the same can be done with integer numbers (counts) and floats. This interval checking is an experiment and we'll see it if it gets used.

```
S +0sp: [dim] O +0sp: [0sp]
D +1sp: [dim] O +1sp: [1sp] O +1sp: [1sp] O +1sp: [2sp] O +1sp: [2sp]
D +2sp: [1sp] D +2sp: [1sp] O +2sp: [5sp] O +2sp: [5sp]
```

There are also `\ifintervalfloat` and `\ifintervalnum`. Because I have worked around these few scaled point rounding issues for decades, it might actually take some time before we see the interval tests being used in ConTEXt. After all, there is no reason to touch

To come back to posits, just to be clear, we use 32 bit posits and not 32 bit floats, which we could have but that way we gain some accuracy because less bits are used by default for the exponential.

In ConTEXt we also provide a bunch of pseudo primitives. These take one float: `\pfsin`, `\pf`

19.3 MetaPost

In addition to the instances `metafun` (double in LMTX), `scaledfun`, `doublefun`, `decimalfun` we now also have `positfun`. Because we currently use 32 bit posits in the new number system there is no real gain over the already present 64 bit doubles. When 64 bit posits show up we might move on to that.

19.4 Lua

We_support_posits_in_Lua_too._Here_we_need_to_create_a_posit_user_data_object._The_usual_metatable_magick_kicks_in:_

```
local p = posit.new(123.456)
local q = posit.new(789.123)
local r = p + q
```

Here_we_just_mention_what_is_currently_interface._The_management_functions_are:_new,_copy
umber,_integer,_rounded,_toposit_and_fromposit._The_usual_operators_are_also_supported:_+,_-,_*,_/,_^,_as_well_as_the_binary_|._&,_~,_<<_and_>>._We_can_compare_with_==,_>=,_
more_verbose_bor,_bxor,_band,_shift,_rotate_are_there_too._

There_is_a_subset_of_math_provided:_min,_max,_abs,_conj,_modf,_acos,_asin,_atan,_ceil,_cos,_
what_special_are_NaN_and_NaR._

Currently_integer_division_(//)_and_modulo_(%)_are_not_available,_but_that_might_happen_at_some_time._

19.4 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

20 Balancing

low level TEX

balancing

Contents

20.1 Introduction

This_is_work_in_progress_as_per_end_2024_these_mechanisms_are_still_in_flux._We_expect_them_to_be_stable_around_the_ConTEXt_meeting_in_2025._The_text_is_not_corrected,_so_fe

This_manual_is_about_a_new_(sort_of_fundamental)_feature_that_got_added_to_LuaMetaTEX_when_we_started_upgrading_column_sets._In_TEX_we_have_a_par_builder_that_does_a_multpass_optimization_where_it_considers_various_solutions_based_on_tolerance,_penalties,_demerits_etc._The_page_builder_on_the_other_hand_is_forward_looking_and_backtracks_to_a_previous_break_when_there_is_an_overflow._The_balancing_mechanism_discussed_here_is_basically_a_page_builder_operating_like_the_par_builder:_it_looks_at_the_whole_picture._

In_order_to_make_this_a_useful_mechanism_the_engine_also_permits_intercepting_the_main_vertical_list,_so_we_start_by_introducing_this._

20.2 Intercepting the MVL

When_content_gets_processed_it's_added_to_a_list._We_can_be_in_horizontal_mode_or_vertical_mode_(let's_forget_about_math_mode)._In_vertical_mode_we_can_be_in_a_box_context_(say_`\vbox`)_or_in_what_is_called_the_main_vertical_list:_the_one_that_makes_the_page._But_what_is_page?_When_TEX_has_collected_enough_to_match_the_criteria_set_by_\p

For_various_mechanisms_it_matters_if_they_are_used_inside_a_contained_boxed_environment_or_in_the_more_liberal_main_vertical_list_(from_now_on_called_mvl)._That's_why_we_can_intercept_the_mvl_and_use_it_later._Intercepting_works_as_follows:_

```
\beginmvl 1
```

```
various content
\endmvl

\beginmvl 2
various content
\endmvl
```

When_at_some_point_you_want_this_content,_you_can_do_this:_

```
\setbox\scratchboxone\flushmvl 2
\setbox\scratchboxtwo\flushmvl 1
```

and_then_do_whatever_is_needed._You_can_see_what_goes_on_with:_

```
\tracingmvl 1
```

There_is_not_much_more_to_say_other_than_that_this_is_the_way_to_operate_on_content_as_if_it_were_added_to_the_page_which_can_be_different_from_collecting_something_in_a_vertical_box._Think_of_various_callbacks_that_can_differ_for_the_mvl_and_a_box._

The_`\beginmvl`_primitive_takes_a_number_or_a_set_of_keywords,_as_in:_

```
\beginmvl
    index   1
    options \numexpr "01 + "04\relax
\relax
```

There_is_of_course_some_possible_interference_with_mechanism_that_check_the_page_properties_like_`\pagegoal`._If_needed_one_can_check_this:_

```
\ifcase\mvlcurrentlyactive
  % main mvl
\or
  % first one
\else
  % other ones
\fi
```

Possible_applications_of_this_mechanism_are_the_mentioned_columns_and_parallel,_independent,_streams._However_for_that_we_need_to_be_able_to_manipulate_the_collected_content._Actually,_the_next_manipulator_preceded_the_capturing,_because_we_first_wanted_to_make_sure_that_what_we_had_in_mind_made_sense._

The_beginmvl_also_accepts_keywords._You_can_specify_an_index_(an_integer),_a_prevdepth_(d
mensions)_and_options_(an_integer_bitset)._Possible_option_bit_related_values_are:_

0x1	ignore prevdepth	\ignoreprevdepthmvloptioncode
0x2	no prevdepth	\noprevdepthmvloptioncode
0x4	discard top	\discardtopmvloptioncode
0x8	discard bottom	\discardbottommvloptioncode

Here_the_last_column_is_a_numeric_alias_available_in_ConTEXt._More_options_are_likely_to_sho

```
\beginmvl
    index     1
    prevdepth 0pt
    options  \discardtopmvloptioncode
\relax
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{1\quad}\flushmvl 1}
```

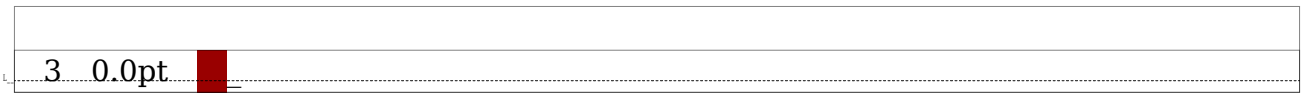
1 1 0.0pt

```
\beginmvl
    index  2
    options \numexpr
                \ignoreprevdepthmvloptioncode
              + \discardtopmvloptioncode
            \relax
\relax
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{2\quad}\flushmvl 2}
```

2 2 -1000.0pt

```
\beginmvl 3 % when no keywords are used we expect a number
\scratchdimen\prevdepth
```

```
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{3\quad}\flushmvl 3}
```




```
\beginmvl index 4 options 1
\scratchdimen\prevdepth
\dontleavehmode
\quad\the\mvlcurrentlyactive\quad\the\scratchdimen
\quad\blackrule[height=\strutht,depth=\strutdp,color=darkred]
\endmvl

\ruledhbox {\llap{4\quad}\flushmvl 4}
```

20.3 Balancing

Balancing is not referring to balancing columns but to 'a result that looks well balanced'. Just like we want lines in a paragraph to look consistent with each other, something that is reflected in the (adjacent) demerits, we want the same with vertical split of pieces. For this purpose we took elements of the par builders to construct a (page) snippet builder. Here are some highlights:

- Instead of a pretolerance, tolerance and emergency pass we only enable the last two. In the par builder the pretolerance pass is the one without hyphenation.
- We seriously considered vertical discretionaries but eventually rejected the idea: we just don't expect users to go through the trouble of adding lots of split related pre, post and replace content. It's not hard to support it but in the end it also interfered with other demands that we had. We kept the code around for a while but then removed it. To mention one complication: if we add some new node we also need to intercept it in various callbacks that we already have in place in ConTEXt. As with horizontal discretionaries, we then need to go into the components and son
- As with the par builder, TEX will happily produce an overfull box when no solution is pos

- In_many_cases_there_is_no_stretch_available._There_are_also_widow,_club,_shape_and_orphan_penalties_that_can_limit_the_solution_space._
- When_we_look_at_splitting_pages_(and_boxes)_we_see_(split)_top_skip_kick_in._This_is_something_that_we_need_to_provide_one_way_ot_the_other._And_as_we_have_to_do_that,_we_can_as_well_provide_support_for_bottom_skip._A_horizontal_analogue_is_protrusion,_something_that_also_has_to_be_taken_into_account_in_a_rather_dynamic_way,_at_the_beginning_or_end_of_the_currently_analyzed_line._
- There_is_no_equivalent_of_hanging_indentation_but_a_shape_makes_sense._Here_the_shape_defines_heights,_top_and_bottom_skips_and_maybe_more_in_the_future._For_that_reason_we_use_a_keyword_driven_shape._
- Because_we_have_so_called_par_passes,_it_made_sense_to_have_something_similar_for_balancing._This_gives_is_the_opportunity_to_experiment_with_various_variables_that_drive_the_process._
- For_those_who_read_what_we_wrote_about_the_par_builder,_it_will_not_come_as_surprise_that_we_also_added_extensive_tracing_and_a_callback_for_intercepting_the_results._This_makes_it_possible_to_show_the_same_detailed_output_as_we_can_do_for_par_passes._

It's_about_time_for_some_examples_but_before_we_come_to_that_it_is_good_to_roughly_explain_how_the_page_builder_works._When_the_page_builder_is_triggered_it_will_take_elements_from_the_contributions_list_and_add_them_to_the_page._When_doing_that_it_keeps_track_of_the_height_and_depth_as_contributed_by_boxes_and_rules._Because_it_will_discard_glue_and_kerns_it_does_some_checking_there._An_important_feature_is_that_the_depth_is_added_in_a_next_iteration._The_routine_also_needs_to_look_at_inserts._The_variables_`\pagegoal`_(original_`\vsize`_minus_accumulated_insert_heights)_and_`\pagetotal`_are_compared_and_when_we_run_over_the_target_height_the_accumulated_stretch_and_shrink_in_glue_(when_present)_will_be_used_to_determine_how_bad_this_break_is._If_it_is_too_bad,_the_previous_best_break_will_be_taken._Penalties_can_make_a_possible_break_more_or_less_attractive._When_the_output_routine_gets_a_split_of_page,_the_total_is_not_reliable_because_we_can_have_backtracked_to_the_previous_break._In_LuaMetaTEX_we_have_som

In_order_to_make_the_first_lines_align_properly_relative_to_the_top_of_the_page_there_is_a_variable_`\topskip`._The_height_of_the_first_line_is_at_least_that_amount._The_correction_is_calculated_when_the_first_contribution_happens:_a_box_or_rule._

When_we_look_at_the_balancer_it_is_good_to_keep_in_mind_that_where_the_page_builder_stepwise_adds_and_checks,_the_balancer_looks_at_the_whole_picture._The_page_builder_does_a_decent_job_but_is_less_sophisticated_than_the_par_builder._There_is_a_badness_

calculation,_penalties_are_looked_at,_glue_is_taken_into_account_but_there_are_no_demerits._

We_want_the_balancer_to_work_well_with_column_sets_that_are_very_much_grid_based._But_in_getting_there_we_had_some_hurdles_to_take._Because_the_algorithm_(like_the_par_builder)_happily_results_in_overfull_boxes_unless_emergency_stretch_is_set,_pages_can_overflow._When_there_is_no_stretch_and/or_shrink_using_emergency_stretch_can_give_an_underfull_page._

The_way_out_of_this_is_to_have_non_destructive_trial_passes_and_decrease_the_number_of_lines._Of_course_we_can_get_short_pages_but_when_for_instance_it_concerns_a_section_title_that_gets_moved_this_is_no_big_deal._In_a_similar_fashion_splitting_a_multiline_formula_is_also_okay._

- Collect_the_content_in_an_mvl_list_and_after_that's_done_put_the_result_in_a_box._
- Set_up_a_balance_shape_that_specifies_the_slots_in_in_columns_(normally_a_column_is_just_a_blob_of_text)._
- Perform_a_trial_balance_run._As_soon_as_an_overfull_page_is_seen,_adapt_the_balance_shape_and_do_a_new_trial_run._
- When_we're_fine,_either_because_we_reached_the_end_without_overfull_column_or_by_passing_the_set_deadcycles_value,_quit_the_trial_process_and_balance_the_original_list_using_the_most_recent_balance_shape._
- Flush_the_result_by_fetching_the_topmost_from_the_result_split_collection_and_feed_it_into_the_page_flow._The_boxed_pseudo_page_will_happily_trigger_the_output_routine_that_in_turn_construct_the_final_page._

At_some_point_we_decided_to_support_multiple_mvl_streams_and_therefore_changed_the_last_mentioned_step._Because_we_store_the_whole_column_set_we_can_as_well_also_store_the_assembled_page_bodies._This_way_we_can_flush_different_streams_into_the_same_result._

- Flush_the_result_by_fetching_the_topmost_from_the_result_split_collection_and_feed_it_into_the_page_flow._Do_this_for_every_saved_(mvl)_stream._
- When_we're_done,_the_boxed_pseudo_pages_will_be_flushed_as_pages._In_the_process,_for_every_page_we_identify_marks._

We_are_now_ready_to_look_at_some_examples._Here_we_also_show_what_balance_shapes_do._These_basically_describe_a_sequence_of_slots_to_be_filled._The_last_specification_is_used_when_we_exceed_the_number_of_defined_slots._These_are_just_examples_of_simple_situations,_for_real_applications_more_code_is_needed._

We start with some content in a box. This can of course be a flushed mvl but here we just set it directly:

```
\setbox\scratchboxone\vbox\bgroup
    \hsize.30\hsize
    \samplefile{tufte}
\egroup
```

We will split this box in columns. If you are familiar with TEX you might know that a parag

```
\balanceshape 3
    vsize      12\lineheight
    topskip    \strutht
    bottomskip \strutdp
next
    vsize       5\lineheight
    topskip    \strutht
    bottomskip \strutdp
next
    vsize      8\lineheight
    topskip    \strutht
    bottomskip \strutdp
\relax

\setbox\scratchboxtwo\vbalance\scratchboxone
```

Contrary to a `\parshape`, a `\balanceshape` is not wiped after the work is done. It also expects keys and values. As with `\parpasses` each step is separated by `next`. This makes it an extensible mechanism. Finally we will split the box according to this shape:

```
\hbox \bgroup
    \localcontrolledendless {%
        \ifvoid\scratchboxtwo
            \expandafter\quitloop
        \else
            \setbox\scratchbox\ruledhbox\bgroup
                \vbalancedbox\scratchboxtwo
            \egroup
            \vbox to 12\lineheight \bgroup
                \box\scratchbox
                \vfill
            \egroup
```

```
            \hskip1em
        \fi
    }\unskip
\egroup
```

The_result_is_shown_here:_

We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_capacity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_synthesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__

Like_the_par_builder_we_can_end_up_with_overfull_boxes_but_we_can_deal_with_that_by_using_trial_runs._

```
\setbox\scratchboxtwo\vbalance\scratchboxone trial
```

In_that_case_the_result_is_made_from_empty_boxes_so_the_original_is_not_disturbed._Here_we_show_an_overflow,_so_in_the_first_resulting_box_you_can_compare_the_height_with_the_requested_one_and_when_it's_larger_you_can_decide_to_decrease_the_first_height_in_the_shape_and_try_again._

Many_readers_will_skim_over_formulas_on_their_first_reading_of_your_exposition._Therefore,_your_sentences_should_flow_smoothly_when_all_but_the_simplest_formulas_are_replaced_by_“blah”_or_some_other_grunting_noise.__

test

Many_readers_will_skim_over_formulas_on_their_first_reading_of_your_exposition._Therefore,_your_sentences_should_flow_smoothly_when_all_but_

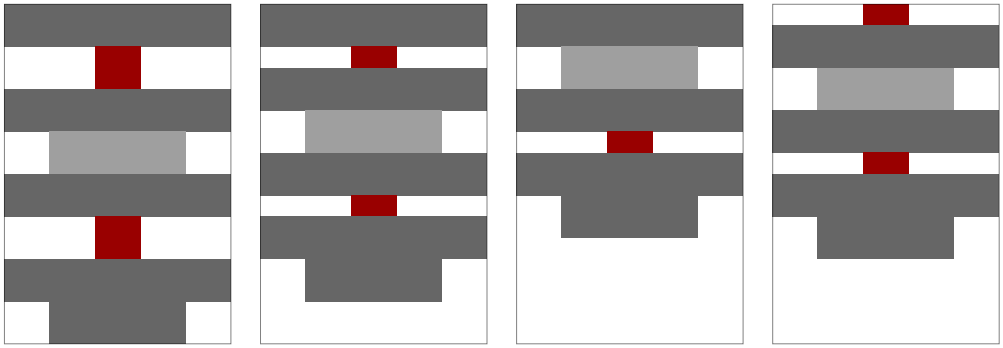
the_simplest_formulas_are_replaced_by_“blah”_or_some_other_grunting_noise.__

Of_course_that_involves_some_juggling_of_the_shape_but_after_all_we_have_Lua_at_our_disposal_so_in_the_end_it's_all_quite_doable._

	real	target
1	169.89122pt	156.95874pt
2	65.39948pt	65.39948pt
3	51.17216pt	104.63916pt

Because the balancer can produce what otherwise the page builder produces, we need to handle the equivalent of top skip which is what the already shown `top` keyword takes care of. This means that the current slice (think current line in the par builder) has to take that into account. This can be compared to the left- and right protrusion in the par builder. When we typeset on a grid we have an additional demand.

When we surround (for instance a formula) with halfline spacing, we eventually have to return on the grid. One complication is that when we are in grid mode and use half line vertical spacing, we can end up in a situation where the initial half line space is on a previous page. That means that we need to use a larger top skip. This is not something that we want to burden the balancer with but we have ways to trick it into taking that compensation into account.



However, when we split in the middle of that segment, we can end up with a half line skip in a next slot because TEX will remove glue at the edge. So we end up with what

20.4 Forcing breaks

Because the initial application of balancing was in column sets, we also need the ability to goto a next slot (step in a shape), column (possibly more steps), page (depending on the page state), and spread (for instance if we are doubles ided). For this we use `\balanceboundary`. It takes two values and when the boundary node triggers a callback in the builder these are passed along with a shape identifier and current shape slot. That callback can then signal back that we need to try a break here with a given penalty. Assuming that at the Lua end we know at which slot we have a slot, column, page or spread break. Multiple slots can be skipped by multiple boundaries. There is one pitfall: we need something in a slot in order to break at all, so one ends up with for instance:

```
\balanceboundary 3 1\relax
\vskip\zeropoint
\balanceboundary 3 0\relax
\vskip\zeropoint
\balanceboundary 3 0\relax
```

Here_the_3_is_just_some_value_that_the_callback_can_use_to_determine_its_action_(like_goto_a_next_page)_and_the_second_value_provides_a_detail._Of_course_all_depends_on_the_intended_usage._By_using_a_callback_we_can_force_breaks_while_not_burdening_the_engine_with_some_hard_coded_solution._For_example,_in_ConTEXt_we_used_these_(the_valu

first	second	action	user interface
1	1 or 0	goto next spread (1 initial, 0 follow up)	`\page[spread]`
2	1 or 0	goto next page (idem)	`\page`
3	1 or 0	goto next column (idem)	`\column`
4	1 or 0	goto next slot (idem)	`\column[slot]`
5	n	next slot when more than n lines	`\testroom[5]`
6	s	next slot when more than s scaled points	`\testroom[80pt]`

20.5 Marks

It_is_possible_to_synchronize_the_marks_with_those_in_the_results_of_balanced_segments_with_a_few_Lua_helpers_that_do_the_same_as_the_page_builder_does_at_the_start_of_a_page,_while_packaging_the_page_and_when_wrapping_it_up._So,_instead_of_split_marks_we_can_have_real_marks._

20.6 Inserts

Before_we_go_into_detail,_we_want_to_point_out_that_when_implementing_a_(balancing)_mechanism_as_introduced_above,_decisions_have_to_be_made._In_traditional_TEX_there_is_for_instan
TEX_and_LuaMetaTEX_it's_(ConTEXt)_user_demands_and_challenges_that_drives_what_gets_impl

Users_on_the_other_hand_have_come_up_with_demands_for_columns,_typesetting_on_the_grid,_multiple_notes,_balancing,_and_parallel_content_streams._The_picture_we_get_from_that_makes_us_confident_that_what_we_provide_is_generally_enough_and_as_users_understand_the_issues_at_hand_(maybe_as_side_effect_of_struggling_with_solutions)_it's_not_that_hard_to_explain_why_constraints_are_in_place._It_makes_more_sense_to_have_a_limited_reliable_mechanism_that_deals_with_the_kind_of_(foot)notes_that_known_users_need_than_to_cook_up_some_complex_mechanism_that_caters_potential_specific_demands_by_potential_users._Of_course_we_have_our_own_challenges_to_deal_with,_even_if_the_resulting_features_will_probably_not_be_used_that_often._So_here_are_the_criteria_that_make_sense:_

- We_can_assume_a_reasonable_amount_of_notes._
- These_are_normally_small_with_no_(vertical)_whitespace._
- Notes_taking_multiple_lines_may_split._
- But_we_need_to_obey_widow_and_club_penalties._
- There_can_be_math_formulas_but_mostly_inline._
- We_need_to_keep_them_close_to_where_they_are_referred_from._

But,_

- We_can_ignore_complex_conflicting_demands._
- As_long_as_we_get_some_result,_we're_fine._
- So_users_have_to_check_what_comes_out._
- We_don't_assume_fully_automated_unattended_usage._

And_of_course:_

- Performance_should_be_acceptable._
- User_interfaces_should_be_intuitive._
- Memory_consumption_should_be_reasonable._

We_have_users_who_use_multiple_note_classes_so_that_also_has_to_be_handled_but_again_we_don't_need_to_come_up_with_solutions_that_solve_all_possible_demands._We_can_assume_that_when_a_book_is_published_that_needs_them,_the_author_will_operate_within_the_constraints._

We_mentioned_footnotes_being_handled_by_the_page_builder_so_how_about_them_in_these_balanced_slots?_Given_the_above_remarks,_we_assume_sane_usage,_so_for_instance_columns_that_have_a_single_slot_with_possibly_fixed_content_at_the_top_or_bottom_(and_maybe_as_part_of_the_stream)._The_balancer_handles_notes_by_taking_their_height_into_account_and_when_a_result_is_used_one_can_request_the_embedded_inserts_and_deal_with_them._Again_this_is_very_macro_package_dependent._Among_the_features_dealt_with_are_space_above_and_between_a_set_of_notes,_which_means_that_we_need_to_identify_the_first_and_successive_notes_in_a_class._Given_how_the_routine_works,_this_is_a_dynamic_feature_of_a_line:_the_amount_of_space_needed_depends_on_how_many_inserts_are_within_a_slot._When_we_did_some_extreme_tests_with_several_classes_of_notes_and_multiple_per_column_we_saw_runtime_increasing_because_instead_of_a_few_passes_we_got_a_few_hundred._In_an_extreme_case_of_800_passes_to_balance_the_result_we_noticed_over_four_million_checks_for_note_related_spacing._We_could_bring_that_down_to_one_tenth_so_in_the_end_we_are_still_slower_but_less_noticeable._Here_are_the_helper_primitives_for_inserts:_

```
<state> = \boxinserts <box>
```

```
<box>   = \vbalancedinsert <box> <class>
<state> = \boxinserts <box>
```

A_(foot)note_implementation_is_very_macro_package_dependent_so_the_next_example_ is_just_that:_an_example_of_using_the_available_primitive._We_start_by_populating_a_ mvl_with_a_sample_text_and_a_single_footnote._

```
\begingroup
    \forgetall
    \beginmvl
        index 5
        options \numexpr
            \ignoreprevdepthmvloptioncode
          + \discardtopmvloptioncode
        \relax
    \relax
        \hsize .4tw
        Line 1 \par Line 2 \footnote {Note 1} \par Line 3 \par
        Line 4 \footnote {Note 2} \par Line 5 \par Line 6 \par
    \endmvl
\endgroup
```

We_fetch_the_footnote_number,_which_is_one_of_many_possible_defined_inserts_

```
\cdef\currentnote{footnote}%
\scratchcounter\currentnoteinsertionnumber
```

The_quick_and_dirty_balancer_uses_a_simple_shape_of_5_lines_with_normal_strut_properties._From_the_balanced_result_we_take_two_columns._We_test_if_there_is_an_insert_ and_take_action_when_there_is._Here_we_just_filter_the_footnotes_but_there_can_of_course_ be_more._We_overlay_these_notes_over_(under)_the_column_that_has_them._So_we_work_ per_column._

```
\begingroup
    \setbox\scratchboxone\flushmvl 5
    \balanceshape 1
        vsize      5lh
        topskip    1sh
        bottomskip 1sd
    \relax
    \setbox\scratchboxtwo\vbalance\scratchboxone
    \ruledhbox \bgroup
```

```
        \localcontrolledrepeat 2 {
          \ifnum\currentloopiterator > 1
            \hskip2\emwidth
          \fi
          \setbox\scratchboxthree\vbalancedbox\scratchboxtwo \relax
          \ifnum\boxinserts\scratchboxthree > 3
            \setbox\scratchboxfour\vbalancedinsert
                \scratchboxthree\scratchcounter
            \wd\scratchboxfour 0pt
            \box\scratchboxfour
          \fi
          \box\scratchboxthree
        }\unskip
    \egroup
\endgroup
```

The_result_is:_

Line 1	Line 4[28]
Line 2[27]	Line 5
Line 3	Line 6

[27] Note 1 [28] Note 2

As_we_progressed_we_realized_that_the_'balancer'_used_in_column_sets_can_also_be_used_for_single_columns_and_we_can_even_support_a_mix_of_single_and_multi_columns._There_is_however_a_problem:_within_a_mvl_we_can_deal_with_spacing_but_we_can't_do_that_reliable_across_mvl's_and_especially_when_we_cross_a_page_it_becomes_hard_to_identify_if_some_(vertical)_spacing_is_needed;_we_don't_want_it_at_the_bottom_or_top_of_a_page._This_feature_is_too_experimental_to_be_discussed_right_now._

We_assumed_reasonable_notes_to_be_used_but_even_if_a_user_tries_to_keep_notes_small_and_avoid_too_many,_there_are_cases_where_they_might_look_like_a_paragraph_and_when_there_are_more_in_a_row,_it_might_be_that_a_column_overflows._This_is_why_we_have_some_support_for_split_notes._This_is_accomplished_by_two_additional_commands:_

```
\setbox\scratchboxone\vbalance\scratchboxone\relax
\vbalanceddeinsert\scratchboxone\relax
```

Here_we_convert_inserts_in_such_a_way_that_they_are_taken_into_account_by_the_balancer_so_that_multi-slot_optimization_takes_place._Afterwards,_when_we_loop_over_the_result_we_can_reconstruct_the_inserts:_

```
\setbox\scratchboxtwo\vbalancedbox\scratchboxone
\vbalancedreinsert\scratchboxtwo\relax
```

Among the reasons that these are explicit actions, is that we want to experiment but also be able to see the effect by selectively enabling it. You can get better results by forcing depth correction.

```
\setbox\scratchboxone\vbalance\scratchboxone
\vbalanceddeinsert\scratchboxone forcedepth\relax
```

This will use the depth as defined by `\insertlinedepth` which is an insert class specific parameter, but discussing details of inserts is not what we do here. The reason for using a `\relax` in the above examples is that we want to stress that when keywords are involved, you need to prevent look-ahead, especially when an `\if...` or expandable loop follows, which is not uncommon when we balance.

It is possible to define top and bottom inserts but of course these need to be filtered and placed at the TEX end, so this is macro package specific. Here we just mention that it i

20.7 Discardables

This is a preliminary explanation.

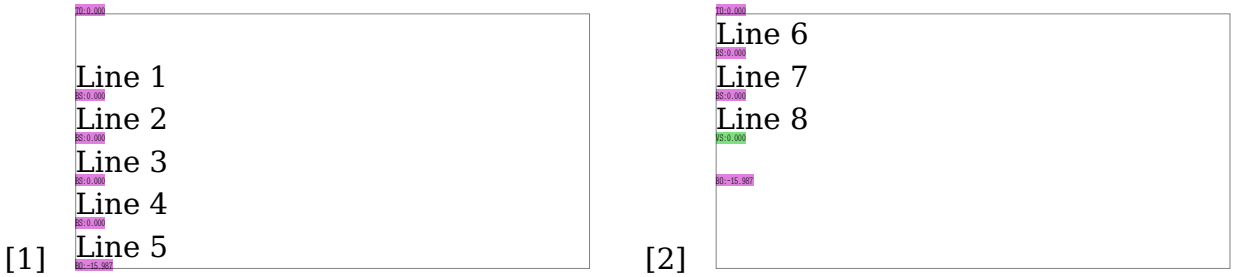
```
\begingroup
    \beginmvl
        index 5
        options \numexpr
            \ignoreprevdepthmvloptioncode
          + \discardtopmvloptioncode
        \relax
    \relax
        \hsize .4tw
        \par
        \vskip0pt
        {\darkred \hrule discardable height 1sh depth 1sd width 1em}
        \par
        % we need the strut because the rule obscures it .. todo
        \dorecurse{8}{\strut Line #1 \par}
        \vskip\zeropoint
        {\darkblue \hrule discardable height 1sh depth 1sd width 1em}
        \par
```

```
    \endmvl
\endgroup

\setbox\scratchboxone\flushmvl 5
\balanceshape 1
    vsize       5lh
    topskip     1sh % see comment above
    bottomskip  1sd
    options     3
\relax
\setbox\scratchboxtwo\vbalance\scratchboxone\relax % lookhead

\hpack \bgroup
    \localcontrolledrepeat 3 {
        \ifvoid\scratchboxtwo\else
            \setbox\scratchboxthree\vbalancedbox\scratchboxtwo
            \ifvoid\scratchboxthree\else
                \dontleavehmode\llap{[\the\currentloopiterator]\quad}%
                \ruledhpack{\box\scratchboxthree}\par
            \fi
            \hskip 4em
        \fi
    }\unskip
\egroup
```

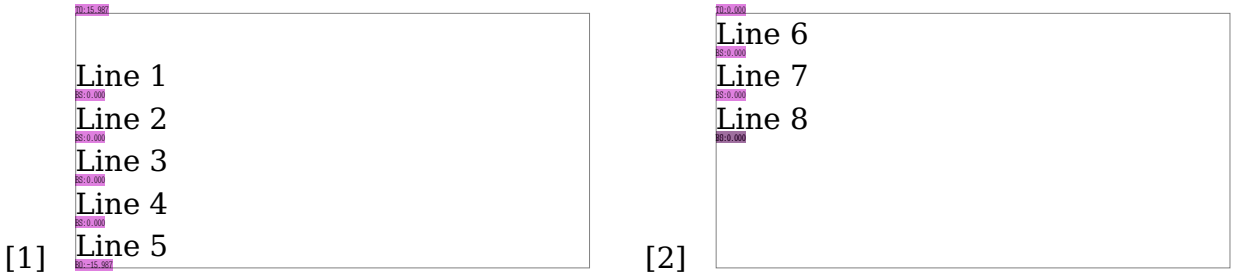
[1]

[2]

When at the top, the rule will be ignored and basically sticks out. When at the bottom the rule might end up in a zero dimension box. With `\vbalanceddiscard\scratchboxtwo` they will become an `\nohrule`. Basically we're talking of optional content. The optic set in the shape definition tells if we have a top (1) and/or bottom (2), here we have both



Here_we_actually_still_have_the_rule_but_marked_as_invisible._So,_topskip_has_a_negative_amount._In_the_next_case_the_remove_keyword_makes_the_rule_go_away_in_which_case_we_also_adapt_the_topskip_accordingly._



You_need_to_juggle_a_bit_with_skips_and_penalties_to_get_this_working_as_you_like._Instead_of_rules_you_can_also_use_boxes,_for_example_before:_

```
\vskip\zeropoint
\ruledvbox discardable {\hpack{\strut BEFORE}}
\par
```

and_after:_

```
\forgetall \par \vskip\zeropoint
\ruledvbox discardable {\hpack{\strut AFTER}}%
\penalty\minusone % !
\par
```

It_currently_is_a_playground_so_it_might_(and_probably_will)_evolve._Although_it_was_also_made_for_a_specific_issue_it_might_have_other_usage._

20.8 Passes

*todo*_

```
\showmakeup[vpenalty,line]
\balancefinalpenalties 6 10000 9000 8000 7000 6000 5000\relax
\balancevsize 5\lineheight
```

```
\setbox\scratchbox\vbox{\dorecurse{1}{\samplefile{tufte}\footnote{!}\par}}
\vbalance\scratchbox
```

20.9 Passes

In LuaMetaTEX the par builder has been extended with additional features (like orphan, tod

```
% threshold
% tolerance
% looseness
% adjdemerits
% originalstretch
% emergencystretch
% emergencyfactor
% emergencypercentage
```

20.9 Colofon

Author	Hans Hagen
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

21 Lines

low level TEX

lines

Contents

21.1 Introduction

There_is_no_doubt_that_TEX_does_an_amazing_job_of_“breaking_paragraphs_into_lines”_where_a_paragraph_is_a_sequence_of_words_in_the_input_separated_by_spaces_or_its_equivalents_(single_line_endings_turned_space)._The_best_descriptions_of_how_that_is_done_can_be_found_in_Don_Knuths_“The_TEX_Book”,_“TEX_The_Program”_and_“Digital_Typography”._Reading_and_rereading_the_relevant_portions_of_those_texts_is_a_good_exercise_in_humility._

That_said,_whatever_follows_here_builds_upon_what_Knuth_gave_us_and_in_no_way_we_pretend_to_do_better_than_that._It_started_out_as_a_side_track_of_improving_rendering_math_in_combination_with_more_control_over_breaking_inline_math._It_pretty_much_about_having_fun_with_the_par_builder_but_in_the_end_can_also_help_make_your_results_look_better._This_is_especially_true_for_proze._

Trying_to_describe_the_inner_working_of_the_par_builder_makes_no_sense._Not_only_is_it_kind_of_complex,_riddled_with_magic_constants_and_heuristics,_but_there_is_a_good_chance_for_us_to_talk_nonsense_thanks_to_misunderstanding._However,_some_curious_aspects_will_be_brought_up._Consider_what_follows_a_somewhat_naive_approach_and_whatever_goes_wrong,_blame_the_authors,_not_TEX._

If_you're_one_of_those_reader_who_love_to_complain_about_the_bad_manuals,_you_can_stop_reading_here._There_is_plenty_said_in_the_mentioned_books_but_you_can_also_consult_Viktor_Eijkhouts_excellent_“TEX_by_Topic”_(just_search_the_web_for_how_the_get_these_books)._If_you're_curious_and_in_for_some_adventure,_keep_reading._

21.2 Warning

This_is_a_first_version._What_is_described_here_will_stay_but_is_still_experimental_and_how_it_evolves_also_depends_on_what_demands_we_get_from_the_users._We_have_defined_some_experimental_setups_in_ConTEXt._We_wil_try_to_improve_the_explanations_in_ways_

```
\setupalign[granular]
```

We_will_explain_below_what_that_means,_but_let_us_already_now_make_clear_that_this_will_likely_become_the_default!_As_far_as_we_can_see,_due_to_the_larger_solution_space,_the_inter-word_spacing_is_more_even_but_that_also_means_that_some_paragraphs_can_become_one_line_less_or_more._

21.3 Constructing paragraphs

There_are_several_concepts_at_work_when_TEX_breaks_a_paragraph_into_lines._Here_we_assun

- The_spaces_between_words_can_stretch_or_shrink._We_don't_want_that_to_be_too_inconsistent_(visible)_between_two_lines._This_is_where_the_terms_loose_and_tight_come_into_play._
- Words_can_be_hyphenated_but_we_don't_want_that_to_happen_too_often._We_also_discourage_neighboring_lines_to_have_hyphens._Hyphenating_the_(pre)_final_line_is_also_sort_of_bad._
- We_definitely_don't_want_words_to_stick_out_in_the_margin._If_we_have_to_choose,_stretching_is_preferred_over_shrinking._If_spaces_become_too_small_words,_start_to_blur._
- If_needed_glyphs_can_stretch_or_shrink_a_little_in_order_to_get_rid_of_excessive_spacing._But_we_really_want_to_keep_it_minimal,_and_avoid_it_when_possible._Usually_we_permit_more_stretch_than_shrink._Not_all_scripts_(and_fonts_for_that_matter)_might_work_well_with_this_feature._
- As_a_last_resort_we_can_stretch_spaces_so_that_we_get_rid_of_any_still_sticking_out_word._When_TEX_reports_an_overfull_box_(often_a_line)_you_have_to_pay_attention!_

When_TEX_decides_where_to_break_and_when_to_finish_doing_so_it_uses_a_system_of_penalties

Here_we_shortly_summarize_the_parameters_that_play_a_role_in_calculating_what_TEX_calls_th

```
\ruledhbox to 20 ts{left \hss right}
\ruledhbox to 40 ts{left \hss right}
\ruledhbox to  5 ts{left \hss right}
```

```
\ruledhbox to  5 ts{left      right}
\ruledhbox to  5 es{%
    left
    \hskip 1ts plus 0.5ts\relax
    middle
    \hskip 1ts plus 1.5ts\relax
    right%
}
```

These_boxes_show_a_bit_what_happens_with_spacing_that_can_stretch_of_shrink._The_first_three_cases_are_not_bad_because_it's_what_we_ask_for_with_the_wildcard_`\hss`._[29]_

left right
left right
left right
left right
left middle right

TEX_will_run_over_each_paragraph_at_most_three_times._On_each_such_run,_it_will_choose_diff

The_process_is_primarily_controlled_by_these_parameters:_

- `\pretolerance`:_This_number_determines_the_success_of_the_first,_not_hyphenated_pass._Often_the_value_is_set_to_the_plain_TEX_value_of_100._If_TEX_finds_a_possible_division
- `\tolerance`:_This_number_determines_the_success_of_the_second,_hyphenated_pass._Often_the_value_is_set_to_the_plain_TEX_value_of_200._
- `\emergencystretch`:_This_dimension_kicks_in_when_the_second_pass_is_not_successful._In_ConTEXt_we_often_set_it_to_2`\bodyfontsize`._

When_we_are_(in_ConTEXt_speak)_`tolerant`,_we_have_a_value_of_3000,_while_`verytolerant`_b

We_now_come_to_the_other_relevant_parameters._You_need_to_keep_in_mind_that_the_demerits_are_made_from_penalty_values_that_get_squared_which_is_why_parameters_with_demerits_in_their_name_have_high_values:_a_penalty_of_50_squared_has_to_relate_to_a_demerit_of_5000,_so_we_might_have_$2500 + 5000$_at_some_point._

The_formula_(most_often)_used_to_calculate_the_demerits_d_is_

$$d = (l + b + p)^2 + e$$

[29] We use this opportunity to promote the new `ts` and `es` units.

Here l is the \linepenalty, set to 10 in plain, b is the badness of the line, and p is the penalty of the current break (for example, added by hyphenation, or by breaking an inline formula). The e stands for extra non-local demerits, that do not depend on only the current line, like the \doublehyphendemerits that is added if two lines in a row are hyphenated.

The badness reflects how the natural linewidth relates to the target width and uses a cubic function. A badness of zero is of course optimal, but a badness of 99 is pretty bad. A magic threshold is 12 (around that value a line is considered decent). If you look at the formula above you can now understand why the line penalty defaults to the low value of 10.

- \hyphenpenalty: When a breakpoint occurs at a discretionary this one gets added. In LuaMetaTEX we store penalties in the discretionary nodes but user defined \discretiona
- \linepenalty: Normally this is set to 10 and it is the baseline for a breakpoint. This is again a small value compared to for instance the penalties that you find in inline math. There we need some breakpoints and after binary and relation symbols such an opportunity is created. The specific penalties are normally 500 and 700. One has to keep in mind, as shown in the formula above, that the penalties are not acting on a linear scale when the demerits are calculated. Math spacing and penalty control is discussed in the (upcoming) math manual.
- \doublehyphendemerits: Because it is considered bad to have two hyphens in a row this is often set pretty high, many thousands. These are treated as demerits (so outside of the squared part of the above formula).
- \finalhyphendemerits: The final (pre last) line having a hyphen is also considered bad. The last line is handled differently anyway, just because it gets normally flushed left.
- \adjdemerits: lines get rated in terms of being loose, decent, tight, etc. When two lines have a different rating we bump the total demerits.
- \looseness: it is possible to force less or more lines but to what extend this request is honored depends on for instance the possible (emergency) stretch in the spaces (or any glue for that matter). `

It is worth noticing that you can set \lastlinefit such that the spaces in the last line will be comparable to those in the preceding line. This is a feature that ε-TEX brought

The next one is a flag that triggers expansion (or compression) of glyphs to kick in. Those get added to the available stretch and/or shrink of a line:

- \adjustspacing: Its value determines if expansion kicks in: glyphs basically get a stretch and shrink value, something that helps filling our lines. We only have zero, two and three (and not the pdfTEX value of two): three means 'only glyphs' and two means 'font kerns and glyphs'.

In LuaMetaTEX we also have:

- \linebreakcriterion: The normal distinction between loose, decent and tight in TEX uses 12 for 0.5 and 99 for about 1.0, but because we have more granularity (.25) we get it', hardly any user will change these values. One can think of the 100 squared becomes a 10000 (at least this helps relating these numbers) and 10000 is pretty bad in TEXs perception.

- \adjustspacingstep: When set this one is are used instead of the font bound value which permits local control without defining a new font instance.

- \adjustspacingstretch: idem.

- \adjustspacingshrink: idem.

- \orphanpenalty: This penalty will be injected before the last word of a paragraph.

- \orphanpenalties: Alternatively a series of penalties can be defined. This primitive expects a count followed by that number of penalties. These will be injected starting from the end.

The shape of a paragraph is determined by \hangindent, \hangafter, \parshape and \parindent. The width is controlled by \hsize, \leftskip, \rightskip. In addition there are \parinitleftskip, \parinitrightskip, \parfillleftskip and \parfillrightskip that control first and last lines.

We also have these:

- \linebreakpasses: When set to one, the currently set \parpasses will be applied.

- \parpasses: This primitive defined a set of sub passes that kick in when the second pass is finished. This basically opens up the par builder. It is still experimental and will be improved based upon user feedback. Although it is a side effect of improving the breaking of extensive mixes of math and text, it is also quite useful for text only (think novels).

In the next sections we will explain how these can improve the look and feel of what you typeset.

21.4 Subpasses

In_LuaTEX_and_therefore_also_in_LuaMetaTEX_a_paragraph_is_constructed_in_steps:_

- The_list_of_nodes_that_makes_the_paragraph_is_hyphenated:_words_become_a_mixture_of_glyphs_and_discretionaries._
- That_list_is_processed_by_a_font_handler_that_can_remove,_add_or_change_glyphs_depending_on_how_glyphs_interact._This_depends_on_the_language_and_scripts_used._
- The_result_is_fed_into_the_par_builder_that_applies_up_to_three_passes_as_mentioned_before._

In_traditional_TEX_these_three_actions_are_combined_into_one_and_the_overhead_is_shared._In_

Here_is_an_example_of_a_setup._We_set_a_low_tolerance_for_the_first_pass_and_second_pass._We_can_do_that_because_we_don't_need_to_play_safe_nor_need_to_compromise._

```
\pretolerance  75
\tolerance    150
\parpasses      3
    threshold           0.025pt
    classes             \indecentparpassclasses
    tolerance           150
  next
    threshold           0.025pt
    classes             \indecentparpassclasses
    tolerance           200
    emergencystretch    .25\bodyfontsize
  next
    threshold           0.025pt
    classes             \indecentparpassclasses
    tolerance           200
    optional            1
    emergencystretch    .5\bodyfontsize
\relax
\linebreakpasses 1
```

Because_we_want_to_retain_performance_we_need_to_test_efficiently_if_we_really_need_the_(here_upto_three)_additional_passes,_so_let's_see_how_it_is_done._When_a_pass_list_is_defined,_and_line_break_passes_are_enabled,_the_engine_will_check_*after*_the_second_pass_i

- overflow_:_the_maximum_value_found,_this_is_something_really_bad._
- underflow_:_the_maximum_value_found,_this_is_something_we_can_live_with._
- verdict_:_what_is_the_worst_badness_of_lines_in_this_paragraph._
- classified_:_what_classes_are_assigned_to_lines,_think_looseness,_decent_and_tight._

There_are_two_cases_where_the_engine_will_continue_with_the_applying_passes:_there_is_an_overflow_or_there_is_a_verdict_(max_badness)_larger_than_zero._When_we_tested_this_on_some_large_documents_we_noticed_that_this_is_nearly_always_true,_but_by_checking_we_save_a_few_unnecessary_passes._

Next_we_test_if_a_pass_is_really_needed,_and_if_not_we_check_the_next_pass._When_a_pass_is_done,_we_pick_up_where_we_left,_but_we_test_for_the_overflow_or_badness_every_sub_pass._The_next_checks_make_us_run_a_pass:_

- overfull_exceeds_threshold_
- verdict_exceeds_badness_
- classified_overlaps_classes_

Here_`threshold`,_`badness`_and_`classes`_are_options_in_a_pass_section._Which_test_makes_sense_depends_a_bit_on_how_TEX_sees_the_result._Internally_TEX_uses_numbers_for_its_classifi

		indecent	**almostdecent**	**loose**	**tight**
1	**veryloose**	+	+	+	
2	**loose**	+	+	+	
4	**semiloose**	+		+	
8	**decent**				
16	**semitight**	+			+
32	**tight**	+	+		+

The_semiloose_and_semitight_values_are_something_LuaMetaTEX._In_ConTEXt_we_have_these_f

The_sections_in_a_par_pass_setup_are_separated_by_`next`._For_testing_purposes_you_can_add_`skip`_and_`quit`._The_`threshold`_tests_against_the_overfull_value,_the_`badness`_against_the_verdict_and_`classes`_checks_for_overlap_with_encountered_classes,_the_classification._

You_can_specify_an_`identifier`_in_the_first_segment_that_then_will_be_used_in_tracing_but_it_is_also_passed_to_callbacks_that_relate_to_this_feature._Discussing_these_callback_is_outside_the_scope_fo_this_wrapup._

You need to keep in mind that parameters are not reset to their original values between two subpasses of a paragraph. We have `tolerance` and `emergencystretch` which are handy for simple setups. When we start with a small tolerance we often need to bump that one. The stretch is likely a last resort. The usual demerits can be set too: `doublehyphendemerits`, `finalhyphendemerits` and `adjdemerits`. We have `extrahyphenpenalty` that gets added to the penalty in a discretionary. You can also set `linepenalty` to a different value than it normally gets.

The `looseness` can be set but keep in mind that this only makes sense in very special cases. It's hard to be loose when there is not much stretch or shrink available. The `linebreakcriterion` parameter can best be left untouched and is mostly there for testing purposes.

The LuaMetaTEX specific `orphanpenalty` gets injected before the last word in a paragraph.

The next four parameters are related to expansion: `adjustspacing`, `adjustspacingstep`, `adjustspacingshrink` and `adjustspacingstretch`. Here we have several scenarios.

- Fonts are set up for expansion (in ConTEXt for instance with the quality specifier). When
- When we don't enable it, the par pass can do it by setting `adjustspacing` (to 3).
- When the other parameters are set these will overload the ones in the font, but used with the factors in there, so different characters get scaled differently. You can set the step to one to get more granular results.
- When expansion is *not* set on the font, setting the options in a pass will activate expansi

When a font is not set up to use expansion, you can do something like this:

```
\parpasses    6
    classes              \indecentparpassclasses
    threshold            0.025pt
    tolerance             250
    extrahyphenpenalty     50
    orphanpenalty        5000
  % font driven
  next ifadjustspacing
    threshold            0.025pt
    classes              \tightparpassclasses
    tolerance             300
    adjustspacing           3
```

```
    orphanpenalty        5000
  next ifadjustspacing
    threshold            0.025pt
    tolerance            350
    adjustspacing           3
    adjustspacingstep       1
    adjustspacingshrink    20
    adjustspacingstretch   40
    orphanpenalty        5000
    emergencystretch     .25\bodyfontsize
  % otherwise, factors 1000
  next
    threshold            0.025pt
    classes              \tightparpassclasses
    tolerance             300
    adjustspacing           3
    adjustspacingstep       1
    adjustspacingshrink    10
    adjustspacingstretch   15
    orphanpenalty        5000
  next
    threshold            0.025pt
    tolerance             350
    adjustspacing           3
    adjustspacingstep       1
    adjustspacingshrink    20
    adjustspacingstretch   40
    orphanpenalty        5000
    emergencystretch     .25\bodyfontsize
  % whatever
  next
    threshold            0.025pt
    tolerance            3000
    orphanpenalty        5000
    emergencystretch     .25\bodyfontsize
\relax
```

With ifadjustspacing you ignore steps that expect the font to be setup, so you don't waste time if that is not the case.

Subpasses

There_is_also_a `callback` parameter_but_that_one_is_experimental_and_used_for_special_purposes_and_testing._We_don't_expect_users_to_mess_with_that._

A_really_special_feature_is_optional_content._Here_we_use_as_example_a_quote_from_Digital_Typography:_

```
Many readers will skim over formulas on their first reading of your
exposition. Therefore, your sentences should flow smoothly when all but
the simplest formulas are replaced by \quotation {blah} or some other
\optionalword {1} {grunting }noise.
```

Here_the `grunting` (with_embedded_space)_is_considered_optional._When_you_set `\linebreakoptional` to 1_this_word_will_be_typeset._However,_when_you_set_the_pass_parameter `linebreakoptional` to 0_it_will_be_skipped._There_can_be_multiple_optional_words_with_different_numbers._The_numbers_are_actually_bits_in_a_bit_set_so_plenty_is_possible._However,_normally_these_two_values_are_enough,_if_used_at_all._

21.5 Definitions

The_description_above_is_rather_low_level_and_in_practice_users_will_use_a_bit_higher_level_interface._Also,_in_practice_only_a_subset_of_the_parameters_makes_sense_in_general_usage._It_is_not_that_easy_to_decide_on_what_parameter_subset_will_work_out_well_but_it_can_be_fun_to_play_with_variants._After_all,_this_is_also_what_TEX_is_about:_look,_feel_a

Some_users_praise_the_ability_of_recent_TEX_engines_to_provide_expansion_and_protrusion._Th
MetaTEX_that_normally_can_be_neglected),_it_also_makes_the_output_files_larger._Some_find_it_

The_traditional_(MkIV)_way_to_set_up_expansion_is_to_add_this_to_the_top_of_the_document,_or_at_least_before_fonts_get_loaded._

```
\scratchcounter 1
\bgroup
\advance\scratchcounter 1
\egroup
\the\scratchcounter
```

and_later_on_to_enable_it_with:_

```
\setupalign[hz]
```

However,_par_passes_make_it_possible_to_be_more_selective._Take_the_following_two_definitions:_

```
\startsetups align:pass:quality:1
    \pretolerance 50
    \tolerance    150
    \parpasses    6
        identifier           \parpassidentifier{quality:1}
        threshold            0.025pt
        tolerance            175
      next
        threshold            0.025pt
        tolerance            200
      next
        threshold            0.025pt
        tolerance            250
      next
        classes              \almostdecentparpassclasses
        tolerance            300
        emergencystretch     .25\bodyfontsize
      next ifadjustspacing
        classes              \indecentparpassclasses
        tolerance            300
        adjustspacing          3
        emergencystretch     .25\bodyfontsize
      next
        threshold            0.025pt
        tolerance            3000
        emergencystretch     2\bodyfontsize
    \relax
\stopsetups

\startsetups align:pass:quality:2
    \pretolerance 50
    \tolerance    150
    \parpasses    5
        identifier           \parpassidentifier{quality:2}
        threshold            0.025pt
        tolerance            175
      next
        threshold            0.025pt
        tolerance            200
      next
```

```
        threshold            0.025pt
        tolerance            250
      next ifadjustspacing
        classes              \indecentparpassclasses
        tolerance            300
        adjustspacing          3
        emergencystretch     .25\bodyfontsize
      next
        threshold            0.025pt
        tolerance            3000
        emergencystretch     2\bodyfontsize
    \relax
\stopsetups
```

You can now enable one of these:

```
\setupalignpass[quality:1]
```

The result is shown in figure 21.1 where you can see that expansion is applied selectively;

21.6 Tracing

There are several ways to see what goes on. The engine has a tracing option that is set with `\tracingpasses`. Setting it to 1 reports the passes on the console, and a value of 2 also gives some details.

There is a also a tracker, `paragraphs.passes` that can be enabled. This gives a bit more information:

```
\enabletrackers[paragraphs.passes]
\enabletrackers[paragraphs.passes=summary]
\enabletrackers[paragraphs.passes=details]
```

If you want to see where expansion kicks in, you can use:

```
\showmakeup[expansion]
```

This is just one of the options, `spaces`, `penalties`, `glue` are are useful when you play with passes, but if you are really into the low level details, this is what you want:

```
\startnarrower[5*right]
\startshowbreakpoints[option=margin,offset=\dimexpr{.5\emwidth-\rightskip}]
```

Figure 21.1 Two different passes applied to `tufte.tex`.

```
\samplefile{tufte}
\stopshowbreakpoints
\stopnarrower
```

We_thrive_in_information–thick_worlds_because_of_our_marvelous_and_everyday_capacity_to_select,_edit,_single_out,_structure,_highlight,_group,_pair,_merge,_harmonize,_synthesize,_focus,_organize,_condense,_reduce,_boil_down,_choose,_categorize,_catalog,_classify,_list,_abstract,_scan,_look_into,_idealize,_isolate,_discriminate,_distinguish,_screen,_pigeonhole,_pick_over,_sort,_integrate,_blend,_inspect,_filter,_lump,_skip,_smooth,_chunk,_average,_approximate,_cluster,_aggregate,_outline,_summarize,_itemize,_review,_dip_into,_flip_through,_browse,_glance_into,_leaf_through,_skim,_refine,_enumerate,_glean,_synopsize,_winnow_the_wheat_from_the_chaff_and_separate_the_sheep_from_the_goats.__

[01] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=1 b=10001
[02] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=2 b=10001
[03] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=3 b=10001
[04] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=4 b=10001
[05] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=5 b=10001
[06] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=6 b=10001
[07] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=7 b=10001
[08] b=1000000 d=0pt p=0.000 r=1.000 (dt=100) (0pt) from s=8 b=10001
[09] b=0 d=7.50644pt p=1.552 r=7.506 (dt=0) (0pt) from s=9 b=0

You_can_see_the_chosen_solutions_with_

```
\showbreakpoints[n=1]
```

```
1    1  0  10001  0  verytight  disc
2    2  1  10001  0  verytight  disc
3    3  2  10001  0  verytight  disc
4    4  3  10001  0  verytight  disc
5    5  4  10001  0  verytight  disc
6    6  5  10001  0  verytight  disc
7    7  6  10001  0  verytight  disc
8    8  7  10001  0  verytight  disc

1
2  1
3  2 1
4  3 2 1
5  4 3 2 1
6  5 4 3 2 1
7  6 5 4 3 2 1
8  7 6 5 4 3 2 1

pass       : 3  demerits   : 0
subpass    : T  looseness : 0
subpasses : 0
```

When_we_started_playing_with_the_par_builder_in_the_perspective_of_math,_we_side_tracked_and_ended_up_with_a_feature_that_can_ge_used_in_controlled_situations._Currently_we_only_have_a_low_level_ConTEXt_interface_for_this_(see_figure_21.2)._

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

`\tracinglousiness 1 \lousiness 0`

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

`\lousiness 1 11 0`

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

`\silliness 11`

Figure 21.2 Influencing the way TEX breaks lines applied to `ward.tex`.

21.7 Criterion

The_granular_alignment_option_will_configure_the_linebreakcriterion_to_work_with_0.25_steps_instead_of_0.50_steps_which_means_that_successive_lines_can_become_a_bit_closer_in_spacing._There_is_no_real_impact_on_performance_because_testing_happens_anyway._In_figure_21.3_you_see_some_examples,_where_in_some_it_indeed_makes_a_difference._

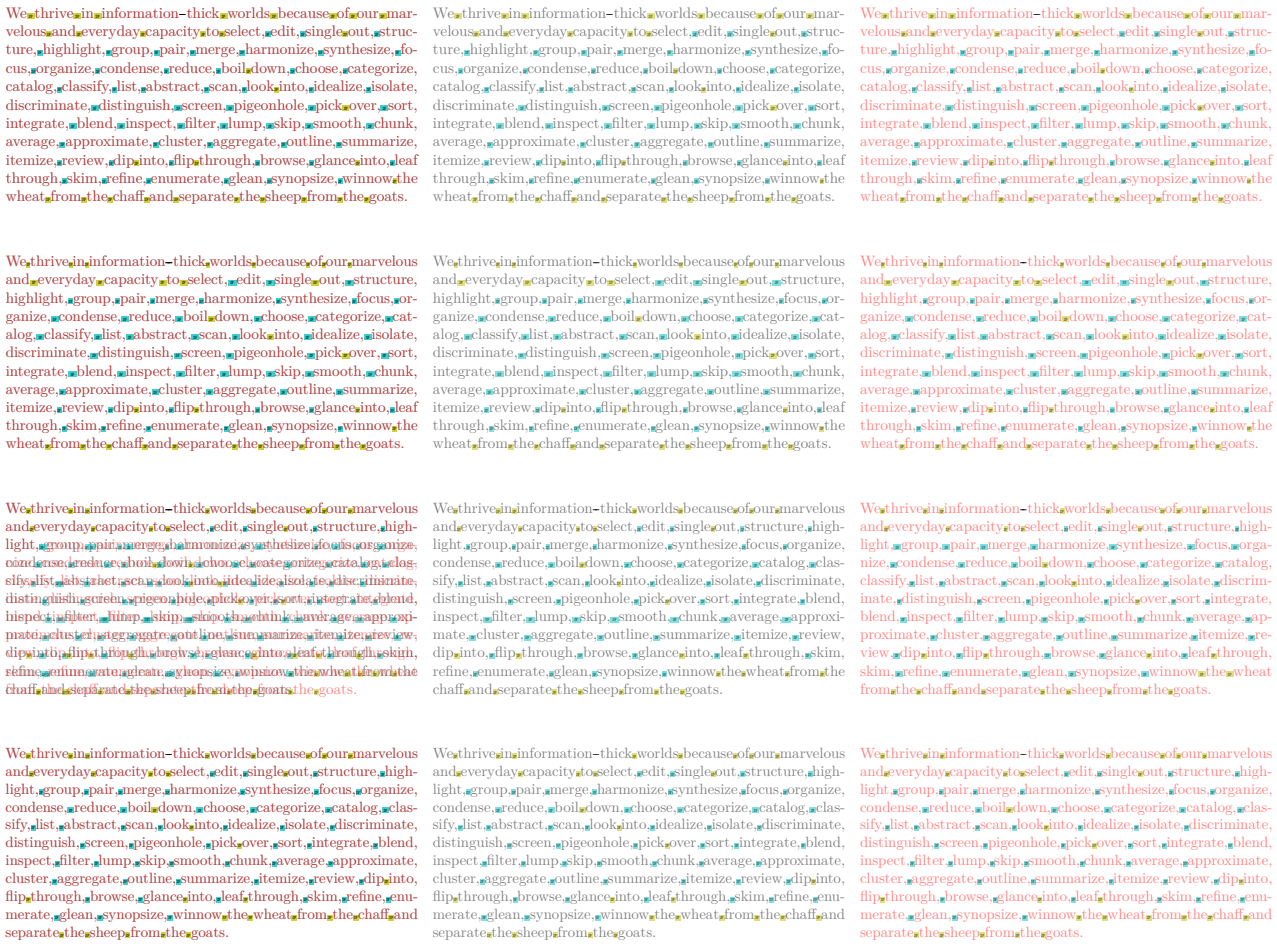


Figure 21.3 More granular interline criteria.

21.8 Examples

The_ConTEXt_distribution_comes_with_a_few_test_setups:_spac-imp-tests.mkxl._Once_we_have

Currently_we_provide_the_following_predefined_passes_that_you_can_enable_with_\setupalignpass:_decent,_quality,_test1,_test2,_test3,_test4,_test5._We_hope_that_users_are_willing_to_test_these._

21.9 Pages

While the par builder does multiple passes, the page builder is a single pass progressive routine. Every time something gets added to the (so called) main vertical list the page state gets updated and when the page overflows what has been collected gets passed to the output routine. It is to a large extend driven by glue (with stretch and shrink) and penalties and when content (boxes) is added the process is somewhat complicated by inserts as these needs to be taken into account too.

You can get pages that run from top to bottom by adding stretch between lines but by default in ConTEXt we prefer to fill up the bottom with white space.

It can be hard to make decisions at the TEX end around a potential page break because in

Penalties play an important role and because these are used to control for instance widows and clubs high values can lead to underfull pages so if we want to influence that we need to cheat. For this we have three experimental mechanisms:

- tweaking the page goal: `\pageextragoal`
- initializing the state quantities: `\initialpageskip`
- adapting the state quantities as we go: `\additionalpageskip`

The first tweak is for me to play with, and when a widow or club is seen the extra amount can kick in. This feature is likely to be replaced by a more configurable one.

The second tweak lets the empty page start out with some given height, stretch and shrink. This variable is persistent over pages. This is not true for the third tweak: it kicks in when the page gets initialized *or* as we go, but after it has been applied the value

Adapting the layout (within the regular text area) is done with `\setpagelooseness` an demonstrated in figure 21.4 and figure 21.5. Possible parameters are `lines`, `height`, `str`

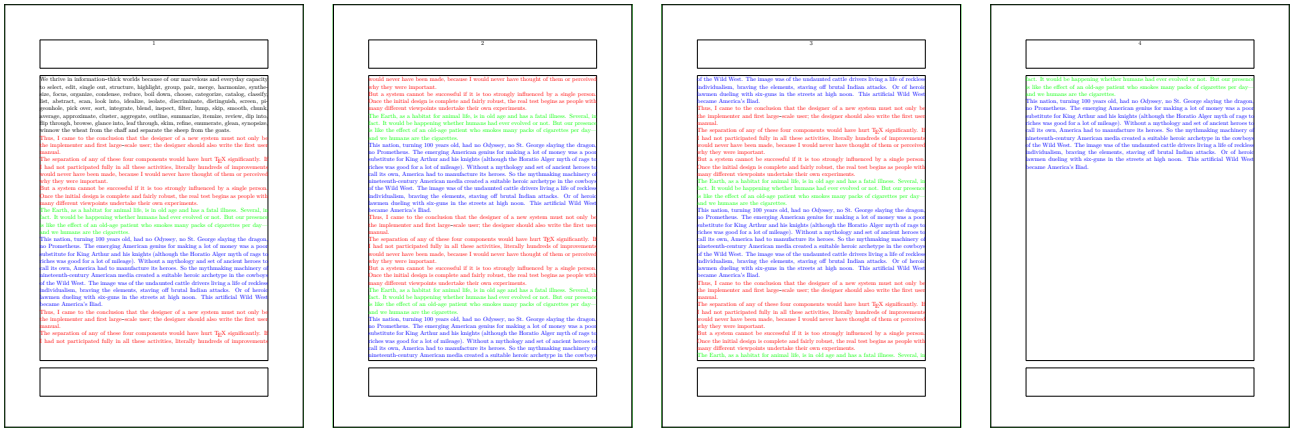


Figure 21.4 Cheating with page dimensions: `[lines=2]`.

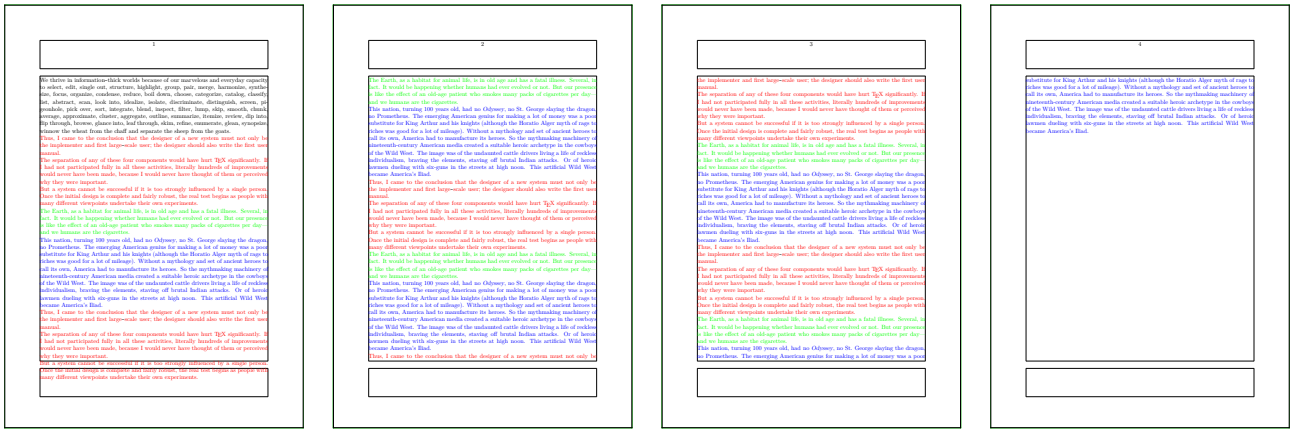


Figure 21.5 Cheating with page dimensions: `[-3]`.

It_is_not_that_trivial_to_fulfill_the_wide_range_of_user_demands_but_over_time_the_\setupalign_commands_has_gotten_plenty_of_features._Getting_for_instance_windows_and_clubs_right_in_the_kind_of_mixed_usage_that_is_common_in_ConTEXt_is_not_always_easy._One_c

The_examples_in_figure_21.6,_21.7_and_21.8_scale_vertically_in_order_ti_fill_up_the_text_area;_

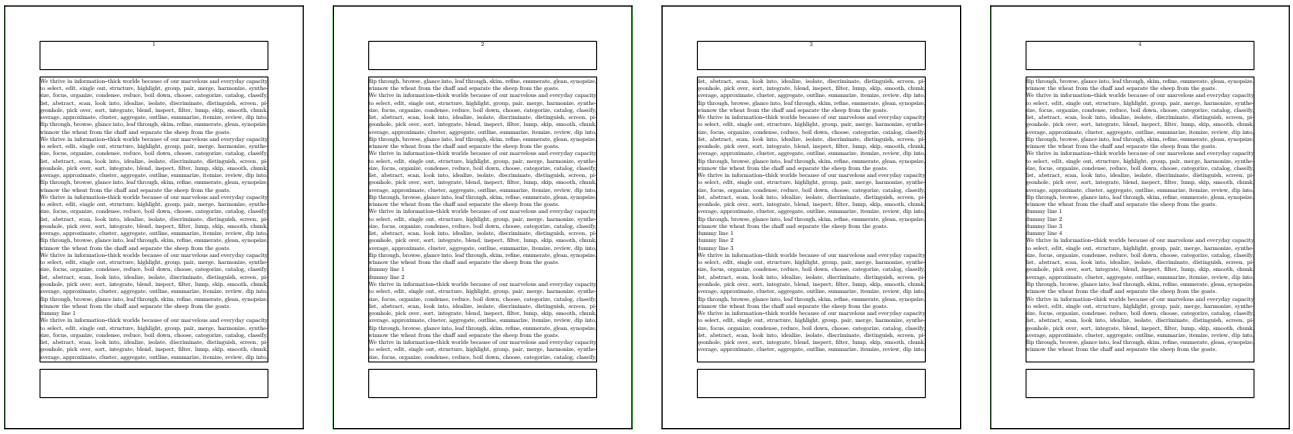


Figure 21.6 Cheating with vertical expansion: `[vz=no]`.

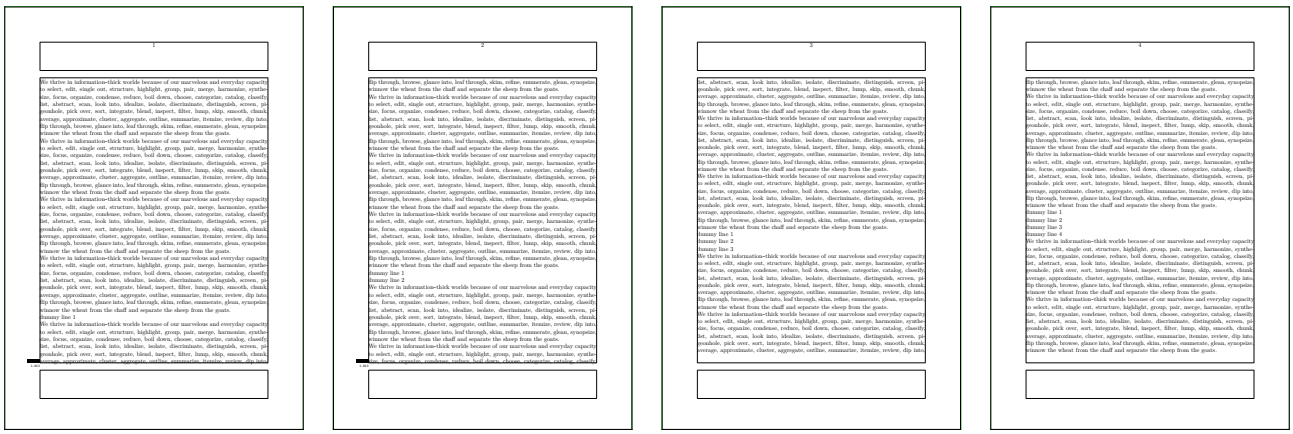


Figure 21.7 Cheating with vertical expansion: `[vz=yes]`.

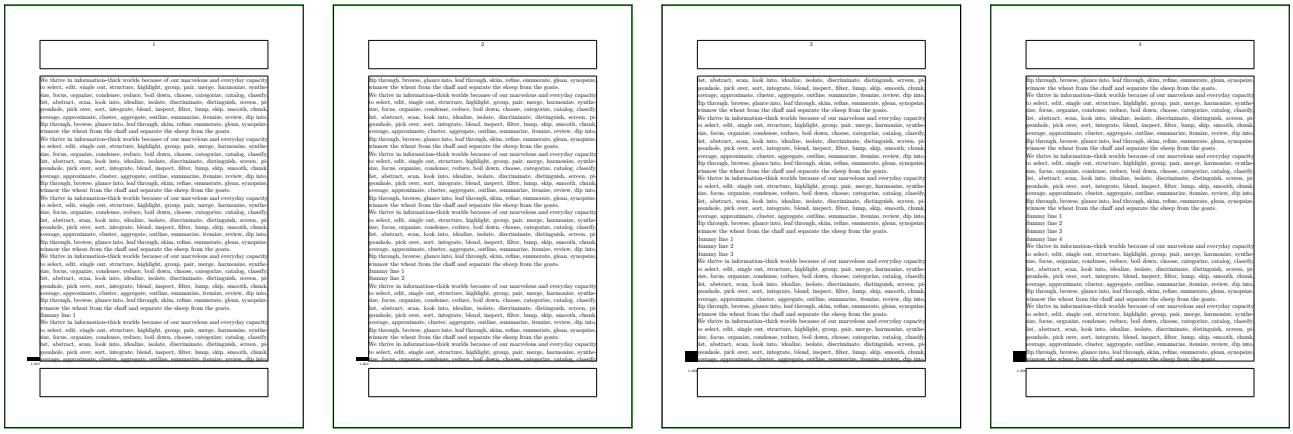


Figure 21.8 Cheating with vertical expansion: `[vz=2]`.

There are a few other tweaks but these one can wonder about these. We can add stretch and shrink to the baseline skip, something that can also be triggered with the 'spread' option to `\setupalign`, assuming that also `height` is given). An alternative is to permit an extra line and accept a visual overflow, assuming that the layout is set up to make sure that the footer line doesn't overlap. None of this guarantees that a whole document with plenty of graphics and special constructs will come out well, but for text only it might work okay. Figures 21.9, 21.10 and 21.11 show some of

Figure 21.9 Cheating: just high penalties.

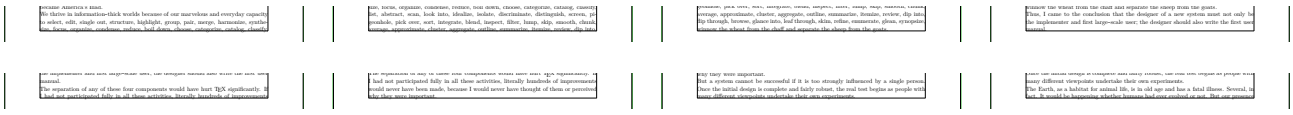


Figure 21.10 Cheating: `\baselineskip 1\baselineskip plus 1pt minus .1pt`.

Figure 21.11 Cheating: `\pageextragoal\lineheight`.

21.10 Profiles

You_can_have_a_paragraph_with_lines_that_exceed_the_maximum_height_and/or_depth_or_where_spaces_end_up_in_a_way_that_create_so_called_rivers._Rivers_are_more_a_curiosity_than_an_annoyance_because_any_attempt_to_avoid_them_is_likely_to_result_in_a_worse_looking_result._The_unequal_line_distances_can_be_annoying_too_but_these_can_be_avoided_when_bringing_lines_closer_together_doesn't_lead_to_clashes._This_can_be_done_without_reformatting_the_paragraph_by_passing_the_`profile`_option_to_`\setupalign`._It_comes_at_the_cost_of_a_little_more_runtime_and_(as_far_as_we_observed)_it_kicks_in_seldom,_for_instance_when_inline_math_is_used_that_has_super-_or_subscripts,_radicals,_fractions_or_other_slightly_higher_constructs._

21.10 Colofon

Author	Hans Hagen & Mikael Sundqvist
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

22 Debugging

low level TEX

debugging

Contents

22.1 Introduction

Below there will be some examples of how you can see what TEX is doing. We start with so
MetaTEX engine. More details about what is possible can be found in documents in the Con
TEXt 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), hy-
phenation, font handling, and more and each has its own specific ways of tracing, ei-
ther 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 languag

22.2 Token lists

There are two main types of linked lists in TEX: token lists and node lists. Token lists relate
TEX 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				
535297	19	49	match	argument 1
595913	20	0	end match	
593996	1	123	left brace	bgroup
600374	143	0	protected call	bf
600289	21	1	parameter reference	
472023	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_to

```
\tolerant\permanent\protected\def\test[#1]#:#2%
  {{\iftok{#1}{sl}\bs\else\bf\fi#2}}
```

Gives_us:_

permanent tolerant protected control sequence: test						
23082	12	91	other char	[	U+0005B	
619498	19	49	match			argument 1
619503	12	93	other char	]	U+0005D	
597323	19	58	match			argument :

597877	19	50	match		argument 2
599747	20	0	end match		
618784	1	123	left brace		
610242	137	29	if test		iftok
599394	1	123	left brace		
250459	21	1	parameter reference		
598918	2	125	right brace		
599457	1	123	left brace		
599992	11	115	letter	s U+00073	
618770	11	108	letter	l U+0006C	
599822	2	125	right brace		
595750	143	0	protected call		bs
50594	137	3	if test		else
597246	143	0	protected call		bf
614066	137	2	if test		fi
618783	21	2	parameter reference		
597337	2	125	right brace		

If you are familiar with TEX and spend some time looking at this you will start recognizing 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

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 MetaTEX we have tried to improve these traces a bit but that's of course subjective and eve

22.3 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_a

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._T

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._

22.4 Visual debugging

In the early days of ConTEXt, in the mid 90's of the previous century, one of the first preser
TEXt MkIV it was implemented in a different (less intrusive) way and it got gradually extend

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

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

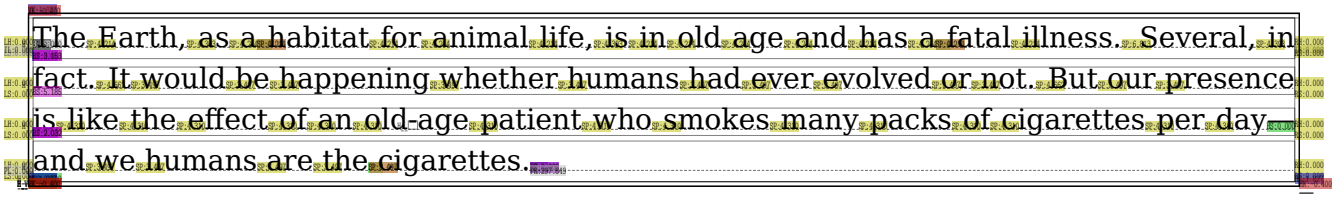
Of course not all information is well visible simply because it can be overlayed by what follows, but one gets the idea. Also, when you have a layer capable pdf viewer you can turn on and off categories, so you can decide to only show glue. You can also do that immediately, with `\showmakeup[glue]`.

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:


The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day and we humans are the cigarettes.

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}}
```

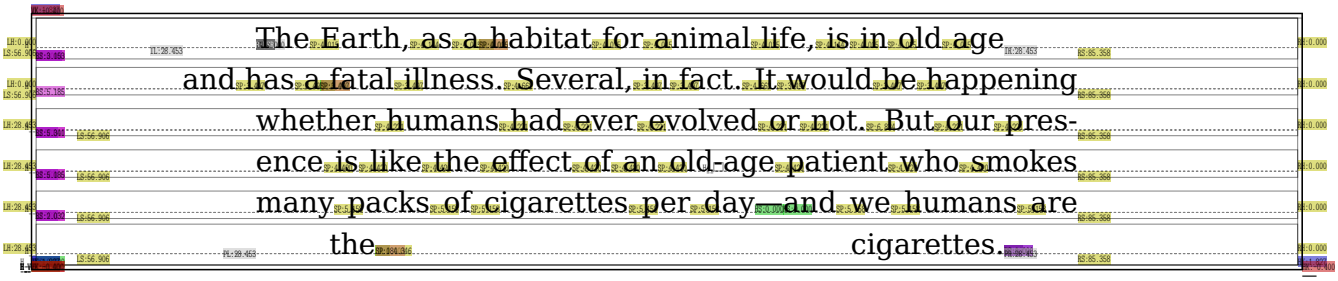
Here we only see lines, regular spaces and spaces that are determined by the space factor that is driven by punctuation.

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

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 TEX and LuaMetaTEX we have to take care of that.

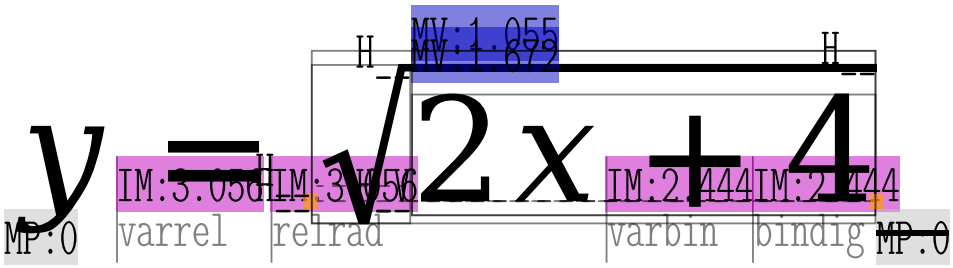
Another characteristic is that the paragraph stores these (and many more) properties in the so called initial par node so that they work well in situations where grouping would interfere with our objectives. As with all extensions, these are things that can be configured in detail but they are enabled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

22.5 Math

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

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

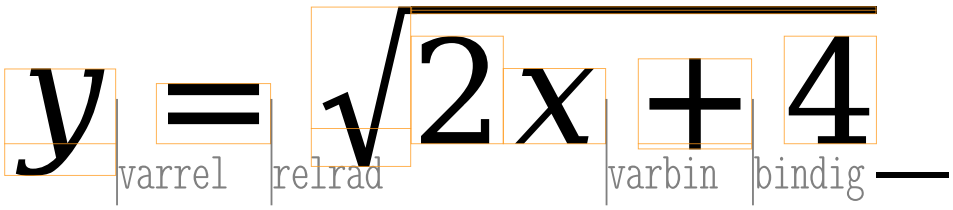
Scaled_up_we_get:_



Instead_of_showing_everything_we_can_again_be_more_selective:_

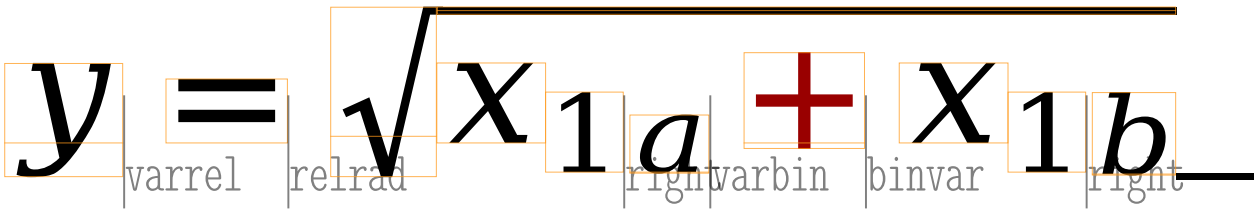
```
\im {
    \showmakeup[mathglue,glyph]
    y = \sqrt {2x + 4}
}
```

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

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



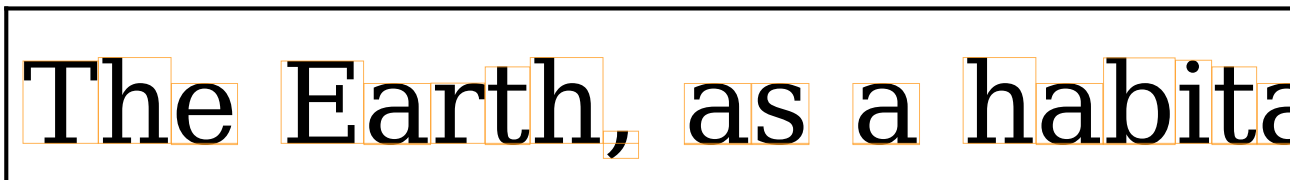
Here_the_variable_class_is_used_for_alphabetic_characters_and_some_more,_contrary_to_the_more_traditional_(often_engine_assigned)_ordinary_class_that_is_now_used_for_the_left-overs._

22.6 Fonts

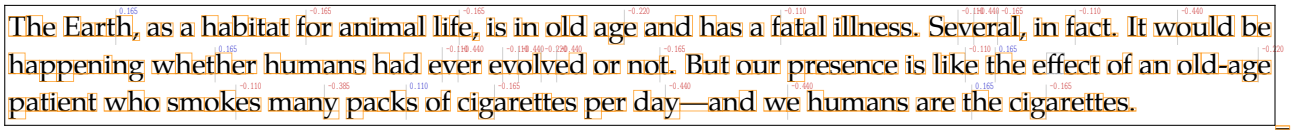
Some_of_the_mentioned_tracing_has_shortcuts,_for_instance_`\showglyphs`._Here_we_show_the_same_sample_paragraph_as_before:_

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

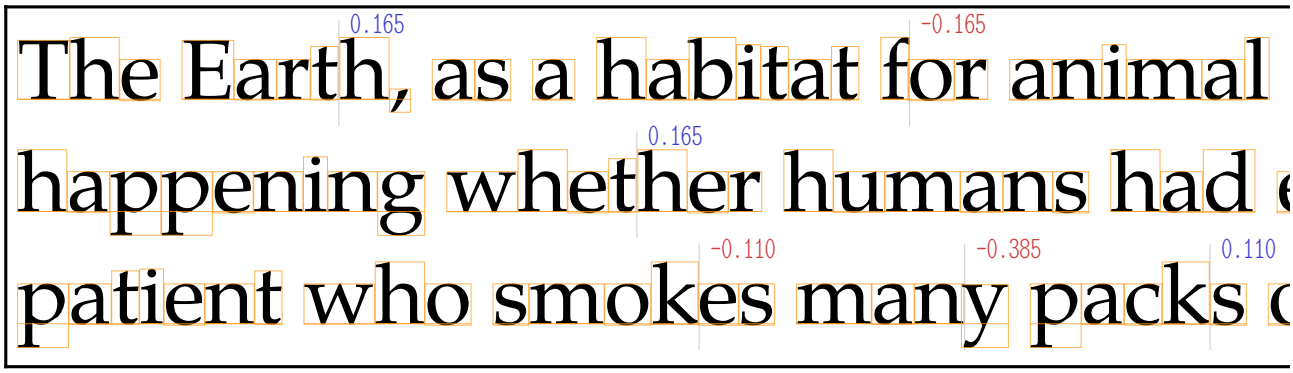
Here_is_the_upper_left_corner_of_the_result:_



What_font_kerns_we_get_depends_on_the_font,_here_we_use_pagella:_

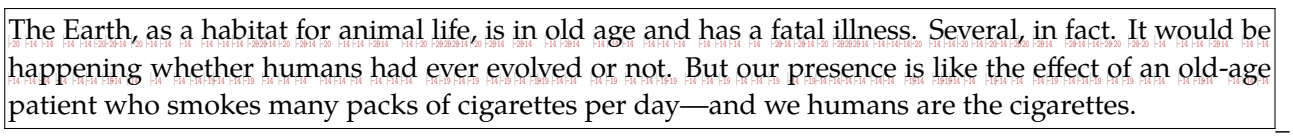


If_we_zoom_in_the_kerns_are_more_visible:_

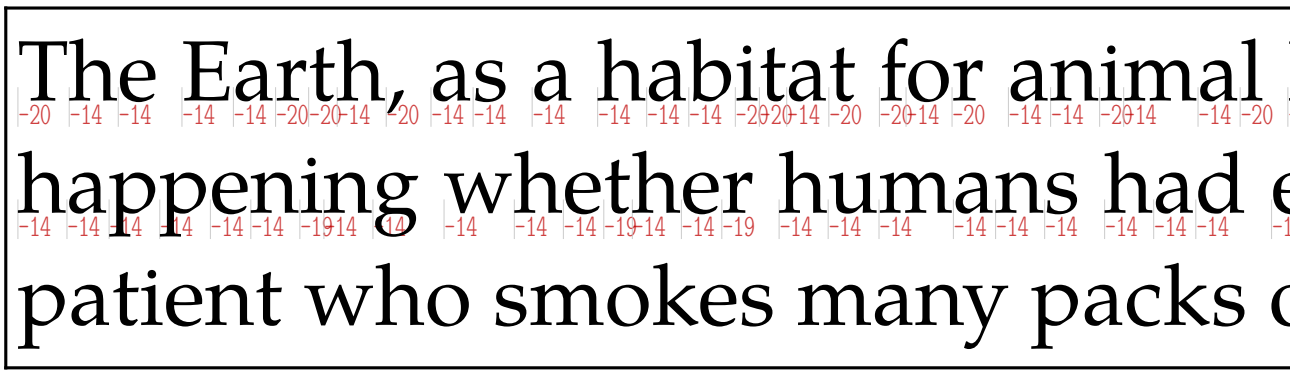


And_here_is_another_one:_

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



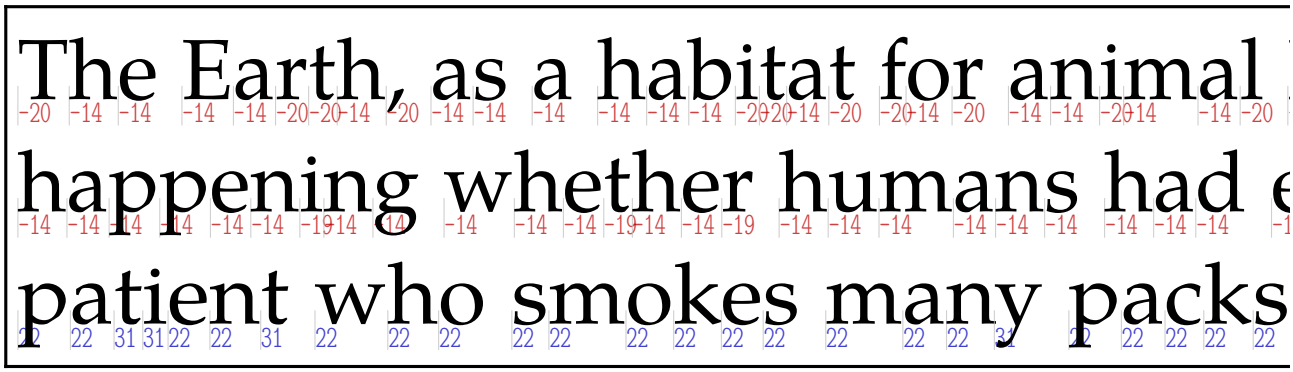
or_blown_up:_



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_ty

22.7 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.

22.8 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.

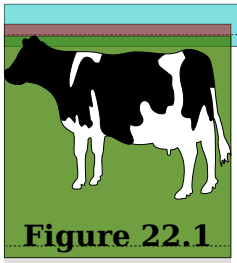


Figure 22.1

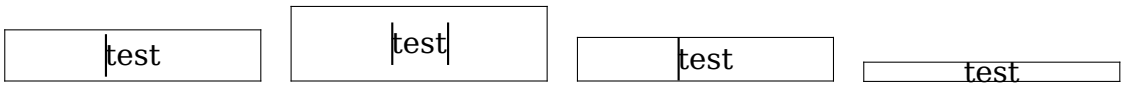
Not only do we have to make sure that the spacing before such a float is as good and consistent as possible, we also need the progression to work out well, that is: the number of lines that we need to indent.

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]
```

22.9 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_p

22.10 Features

Compared_to_the_time_when_TEX_showed_up_the_current_fonts_are_more_complicated,_especia

font 22:_texgyrepagella-regular.otf_@_10.0pt

features **[basic:**_kern=yes,_liga=yes,_mark=yes,_mkmk=yes,_script=dflt**]_[extra:**_ana

step_1 effe_fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f U+65:e [pre: U+5F:_ replace: U+5F:_] 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)

step_2 effe_fietsen U+65:e [pre: U+66:f U+2D:- post: U+66:f replace: U+FB00:ff] U+65:e [pre: U+5F:_ replace: U+5F:_] 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

step_3 effe_fietsen U+65:e [pre: U+66:f U+2D:- post: U+66:f replace: U+FB00:ff] U+65:e [pre: U+5F:_ replace: U+5F:_] 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_-0.1499

result effe_fietsen U+65:e [pre: U+66:f U+2D:- post: U+66:f [kern] replace: U+FB00:ff] U+65:e [pre: U+5F:_ replace: U+5F:_] 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_a

With_Cambria_we_get_a_single_replacement_combined_with_kerning:_

font 23:_cambria.ttc_@_10.0pt

features **[basic:**_kern=yes,_liga=yes,_mark=yes,_mkmk=yes,_script=latn**]_[extra:**_ana

step_1 effe_fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f U+65:e [pre: U+5F:_ replace: U+5F:_] 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',_replaci

step_2 effe_fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f U+65:e [pre: U+5F:_ replace: U+5F:_] 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_

result effe_fietsen U+65:e U+66:f [pre: U+2D:-] U+66:f [kern] U+65:e [pre: U+5F:_ replace: U+5F:_] 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._

22.11 Profiling

There_are_some_features_in_ConTEXt_that_are_nice_but_only_useful_in_some_situations._An_exa

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_coeffic

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_coeffic

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._

22.12 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_i 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_$1 < y < \infty$;_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_$(A,B,C,D) \leftarrow (C,D,A-X_kC,B-X_kD)$,_$k \leftarrow k+1$;_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_im $B)/(Cx_j + D)$,_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      1  0  10001      0  verytight  disc      7  1  10   9  10001   80346  verytight  disc     14   12 11 10 9 8 6 3 2 1
2  2   2  1  10001      0  verytight  disc      8  2  11  10      0  102946  decent     disc     15   13 11 10 9 8 6 3 2 1
3  1   3  2    165  53125  veryloose  disc      9  2  12  11     16  116122  loose      disc     16   12 11 10 9 8 6 3 2 1
   2   4  2      6  22756  decent     disc         2  13  11      0  115546  decent     disc     17   13 11 10 9 8 6 3 2 1
4      5  4    166  76232  veryloose  disc     10     14  12    131  148503  veryloose  disc     18   17 13 11 10 9 8 6 3 2 1
   1   6  3    166  67650  loose      disc            15  13     14  128622  loose      disc
       7  4     16  35932  tight      disc            16  12     14  128722  decent     disc     pass      : 3  demerits  : 129490
5      8  6      4  80346  decent     disc            17  13     28  129490  tight      disc     subpass   : T  looseness :      0
6      9  8  10001  80346  verytight  disc     11     18  17  10001  129490  verytight  penalty  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_a 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._

22.13 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._

22.13 Colofon

Author	Hans Hagen & Mikael Sundqvist
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl

23 Pages

low level TEX pages

Contents

23.1 Introduction

There_are_several_builder_in_the_engine:_paragraphs,_math,_alignments,_boxes_and_if_course_pages._But_where_a_paragraph_is_kind_of_complete_and_can_be_injected_on_a_line_by_line_basis,_a_page_is_less_finished._When_enough_content_is_collected_the_result_so_far_is_handled_over_to_the_output_routine._Calling_it_a_routine_is_somewhat_confusing_because_it_is_not_really_a_routine,_it's_the_token_list_`\output`_that_gets_expanded_and_what_in_there_is_supposed_to_something_with_the_result,_like_adding_inserts_(footnotes,_moved_around_graphics_aka_floats,_etc.),_adding_headers_and_footers,_possibly_using_marks,_and_finally_wrapping_up_and_shipping_out._

The_engine_primarily_offers_a_single_column_page_so_two_or_more_columns_are_done_by_using_tricks,_like_typesetting_on_a_double_height_and_splitting_the_result._If_columns_need_to_be_balanced_some_extra_work_has_to_be_done,_and_it's_definitely_non_trivial_when_we_have_more_that_just_text._

In_this_chapter_we_will_discuss_and_collect_some_mechanisms_that_deal_with_pages_or_operate_at_the_outer_vertical_level._We_might_discuss_some_primitive_but_more_likely_you_will_see_various_solutions_based_on_TEX_macros_and_Lua_magic._

This_is_work_in_progress._

23.2 Rows becoming columns

This_is_an_experimental_mechanism._We_need_to_check/decide_how_to_deal_with_penalties._We_also_need_to_do_more_checking._

Conceptually_this_is_a_bit_strange_feature_but_useful_nevertheless._There_are_several_multi-column_mechanisms_in_ConTEXt_and_each_is_made_for_a_specific_kind_of_usage._You_can

```
\starttabulate[|l|l|]
\NC 1 \NC one   \NC \NR
\NC 2 \NC two   \NC \NR
\NC 3 \NC three \NC \NR
\NC 4 \NC four  \NC \NR
\NC 5 \NC five  \NC \NR
```

```
\stoptabulate
```

but_you_don't_want_to_waste_space._So_you_might_want:_

1	one	4	four
2	two	5	five
3	three		

or_maybe_even_this:_

1	one	3	three	5	five
2	two	4	four		

but_still_wants_to_code_like_this:_

```
\starttabulate[|l|l|]
\NC 1 \NC one   \NC \NR
\NC 2 \NC two   \NC \NR
\NC 3 \NC three \NC \NR
\NC 4 \NC four  \NC \NR
\NC 5 \NC five  \NC \NR
\stoptabulate
```

You_can_do_this:_

```
\startcolumns[n=3]
\getbuffer
\stopcolumns
```

The_(mixed)_columns_mechanism_used_here_normally_works_ok_but_because_of_the_way_columns_are_packaged_they_don't_work_well_with_for_instance_‘vz’._Page_columns_do_a_better_job_but_don't_mix_with_single_columns_that_well._Another_solution_is_this:_

```
\startrows[n=3,before=\blank,after=\blank]
\getbuffer
\stoprows
```

Here_the_result_is_collected_in_a_vertical_box,_post_processed_and_flushed_line_by_line._We_need_to_explicitly_handle_the_before_and_after_spacing_here_because_it_gets_discarded_(if_added_at_all)._When_a_slice_of_the_box_is_part_of_the_shipped_out_page_the_cells_are_swapped_so_that_instead_of_going_horizontal_we_go_vertical._Compare_the_original_

1	one	2	two	3	three

Rows becoming columns

4	four	5	five		

with the swapped one:

1	one	3	three	5	five
2	two	4	four		

This is not really a manual but let's mention a few configuration options. The `n` parameter controls the number of columns. In order to support swapping this mechanism adds empty pseudo cells for as far as needed. By default the `order` is `vertical` but one can set it to `horizontal` instead. In the next example we have set `height` to `2\struth`

1	one	3	three	5	five
2	two	4	four		

When you set `height` and `depth` to `max` all cells will get these dimensions from the tallest cell. Compare:

1	$y = x + 1$	3	$y = \sqrt[n]{x^2} + 1$
2	$y = x^2 + 1$	4	$y = \frac{1}{x^2} + 1$

with:

1	$y = x + 1$	3	$y = \sqrt[n]{x^2} + 1$
2	$y = x^2 + 1$	4	$y = \frac{1}{x^2} + 1$

In the examples with tabulate we honor the original dimensions but you can also set the `width`, combined with a `distance`. Instead of a dimension the `width` parameter can be set to `fit`.

In case one wonders, of	in tables) that can be	lems but when used in
course regular columns	swapped. For as far as	situations where one
can be used, but this is	possible footnotes are	knows what goes in, it
an alternative that actu-	supported but of course	is quite powerful any-
ally gives you balancing	floats are not.	way. It also has a rela-
for free, but of course	So, this rows based	tively simple implemen-
with the limitation that	mechanism is not the	tation.
we have lines (or cells	solution for all prob-	

In the previous rendering we have set the width as mentioned but also set `align` to `verytolerant,stretch` so that we don't overflow lines. The `before` and `after` parameters are set to `\blank`.

Rows becoming columns

23.2 Colofon

Author	Hans Hagen & Mikael Sundqvist
ConTEXt	2025.02.24 16:28
LuaMetaTEX	2.11.07 | 20250226
Support	www.pragma-ade.com
	contextgarden.net
	ntg-context@ntg.nl