



low level

TEX

paragraphs

Contents

1	Introduction	1
2	Paragraphs	1
3	Properties	5
4	Wrapping up	7
5	Hanging	8
6	Shapes	8
7	Modes	26
8	Leaders	26
9	Prevdepth	33
10	Normalization	35
11	Dirty tricks	35

1 Introduction

This manual is mostly discussing a few low level wrappers around low level \TeX features. Its writing is triggered by an update to the MetaFun and LuaMetaFun manuals where we mess a bit with shapes. It gave a good reason to also cover some more paragraph related topics but it might take a while to complete. Remind me if you feel that takes too much time.

Because paragraphs and their construction are rather central to \TeX , you can imagine that the engine exposes dealing with them. This happens via commands (primitives) but only when it's robust. Then there are callbacks, and some provide detailed information about what we're dealing with. However, intercepting node lists can already be hairy and we do that a lot in Con \TeX t. Intercepting and tweaking paragraph properties is even more tricky, which is why we try to avoid that in the core. But ... in the following sections you will see that there are actually a couple of mechanism that do so. Often new features like this are built in stepwise and enabled locally for a while and when they seem okay they get enabled by default.¹

2 Paragraphs

Before we demonstrate some trickery, let's see what a paragraph is. Normally a document source is formatted like this:

¹ 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`.

```
some text (line 1)
some text (line 2)

some more test (line 1)
some more test (line 2)
```

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 LuaMeta \TeX 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:

Paragraphs

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 ConT_EXt anyway.

Back to reality. In LuaT_EX we get a node list that starts with a so called `localpar` node and ends with a `\parfillskip`. The first node is prepended automatically. That list travels through the system: hyphenation, applying font properties, break the effectively one line into lines, wrap them and add them to a vertical list, etc. Each stage can be intercepted via callbacks.

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 LuaMetaT_EX 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 T_EX into so called horizontal mode. When you look at plain T_EX documents you will notice commands like `\noindent` and `\indent`. In ConT_EXt 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` which is a wink to `\leavevmode`, a command that you should never use in ConT_EXt, so when you come from plain or L^AT_EX, it's one of the commands you should wipe from your memory.

When T_EX starts with a paragraph the `\everypar` token list is expanded and again this is a primitive you should not mess with yourself unless in very controlled situations.

Paragraphs

If you change its content, you're on your own with respect to interferences and side effects.

One of the things that \TeX does in injecting the indentation. Even when there is none, it gets added, not as skip but as an empty horizontal box of a certain width. This is easier on the engine when it constructs the paragraph from the one liner: starting with a skip demands a bit more testing in the process (a nice trick so to say). However, in \ConTeXt we enable the \LuaMetaTeX feature that does use a skip instead of a box. It's part of the normalization that is discussed later. Instead of checking for a box with property `indent`, we check for a skip with such property. This is often easier and cleaner.

A bit off topic is the fact that in traditional \TeX empty lines or `\par` primitives can trigger an error. This has to do with the fact that the program evolved in a time where paper terminals were used and runtime could be excessive. So, in order to catch a possible missing brace, a concept of `\long` macros, permitting `\par` or equivalents in arguments, was introduced as well as not permitting them in for instance `display math`. In \ConTeXt MkII most macros that could be sensitive for this were defined as `\long` so that users never had to bother about it and probably were not even aware of it. Right from the start in \LuaTeX 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 about this.

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 each paragraph (also nested ones).

```
>>>> 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 just nicer (and easier on the log when tracing is enabled).

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 end of the paragraph is met are used. So, when you change them in a group and then ends the paragraph after the group, the values you've set in the group are not used.

However, in \LuaMetaTeX we can optionally store them with the paragraph. When that happens the values current at the start are frozen. You can still overload them but that has to be done explicitly then. The advantage is that grouping no longer interferes with the line break algorithm. The magic primitive is `\snapshotpar` which takes a number made from categories mentioned below:

variable	category	code
<code>\hsize</code>	hsize	0x01
<code>\leftskip</code>	skip	0x02
<code>\rightskip</code>	skip	0x02
<code>\hangindent</code>	hang	0x04
<code>\hangafter</code>	hang	0x04
<code>\parindent</code>	indent	0x08
<code>\parfillleftskip</code>	par fill	0x10
<code>\parfillrightskip</code>	par fill	0x10
<code>\adjustspacing</code>	adjust	0x20
<code>\adjustspacingstep</code>	adjust	0x20
<code>\adjustspacingshrink</code>	adjust	0x20
<code>\adjustspacingstretch</code>	adjust	0x20
<code>\protrudechars</code>	protrude	0x40
<code>\pretolerance</code>	tolerance	0x80
<code>\tolerance</code>	tolerance	0x80
<code>\emergencystretch</code>	stretch	0x100
<code>\looseness</code>	looseness	0x200
<code>\lastlinefit</code>	last line	0x400
<code>\linepenalty</code>	line penalty	0x800
<code>\interlinepenalty</code>	line penalty	0x800
<code>\interlinepenalties</code>	line penalty	0x800
<code>\clubpenalty</code>	club penalty	0x1000

<code>\clubpenalties</code>	club penalty	0x1000
<code>\widowpenalty</code>	widow penalty	0x2000
<code>\widowpenalties</code>	widow penalty	0x2000
<code>\displaywidowpenalty</code>	display penalty	0x4000
<code>\displaywidowpenalties</code>	display penalty	0x4000
<code>\brokenpenalty</code>	broken penalty	0x8000
<code>\adjdemerits</code>	demerits	0x10000
<code>\doublehyphendemerits</code>	demerits	0x10000
<code>\finalhyphendemerits</code>	demerits	0x10000
<code>\parshape</code>	shape	0x20000
<code>\baselineskip</code>	line	0x40000
<code>\lineskip</code>	line	0x40000
<code>\lineskiplimit</code>	line	0x40000
<code>\hyphenationmode</code>	hyphenation	0x80000

As you can see here, there are more paragraph related parameters than in for instance pdf \TeX and Lua \TeX and these are (to be) explained in the LuaMeta \TeX manual. You can imagine that keeping this around with the paragraph adds some extra overhead to the machinery but most users won't notice that because is is compensated by gains elsewhere.

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 you like, and the code gives plenty of examples but when you do this, you're on your own because it can interfere with Con \TeX t 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!
```

Properties

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.

4 Wrapping up

In ConT_EXt LMTX we have a mechanism to exercise macros (or content) before a paragraph ends. This is implemented using the `\wrapuppar` primitive. The to be wrapped up material is bound to the current paragraph which in order to get this done has to be started when this primitive is used.

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

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`. In 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 1, the four cases are driven by the sign of the values. If you want to hang into the margin you need to use different tricks, like messing with the `\leftskip`, `\rightskip` or `\parindent` parameters (which then of course can interfere with other mechanisms uses at the same time).

6 Shapes

In ConT_EXt we don't use `\parshape` a lot. It is used in for instance side floats but even there not in all cases. It's more meant for special applications. This means that in MkII and MkIV we don't have some high level interface. However, when MetaFun got upgraded to LuaMetaFun, and the manual also needed an update, one of the examples in that manual that used shapes also got done differently (read: nicer). And that triggered the arrival of a new low level shape mechanism.

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.²

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
```

² Not discussed here is a variant that might end up in LuaMetaT_EX 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.



Figure 1 Hanging indentation

```
\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.

```
\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 determine 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.

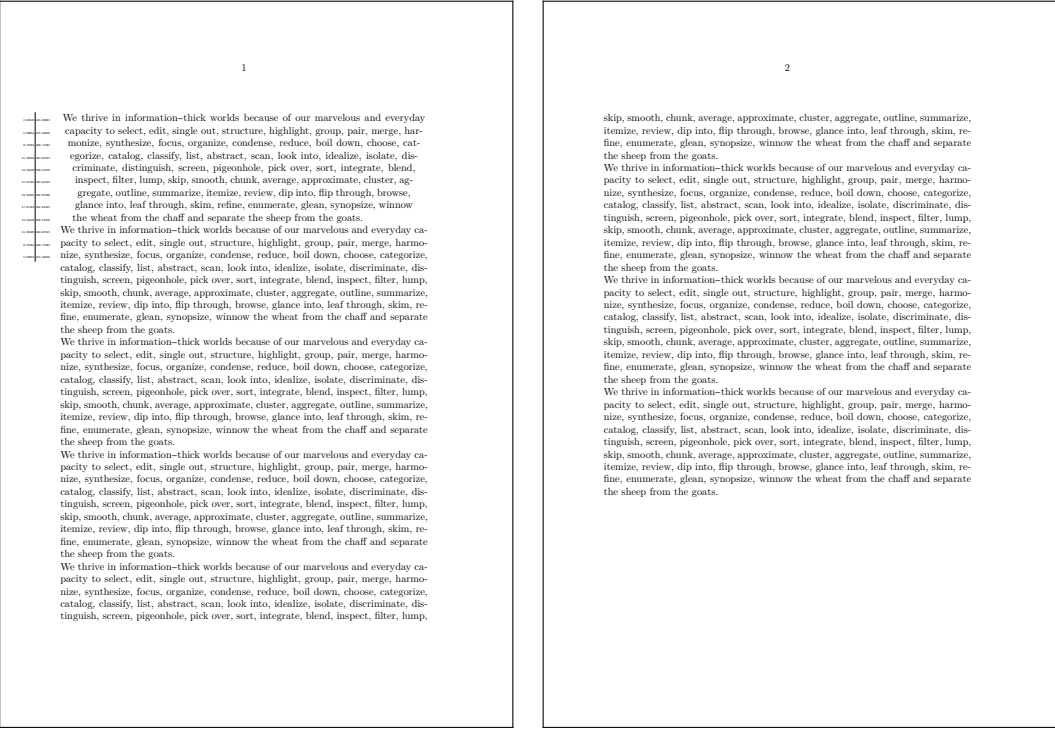
In figure 2 you see the following applied:

```

\startshapedparagraph[list=test]
  \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph

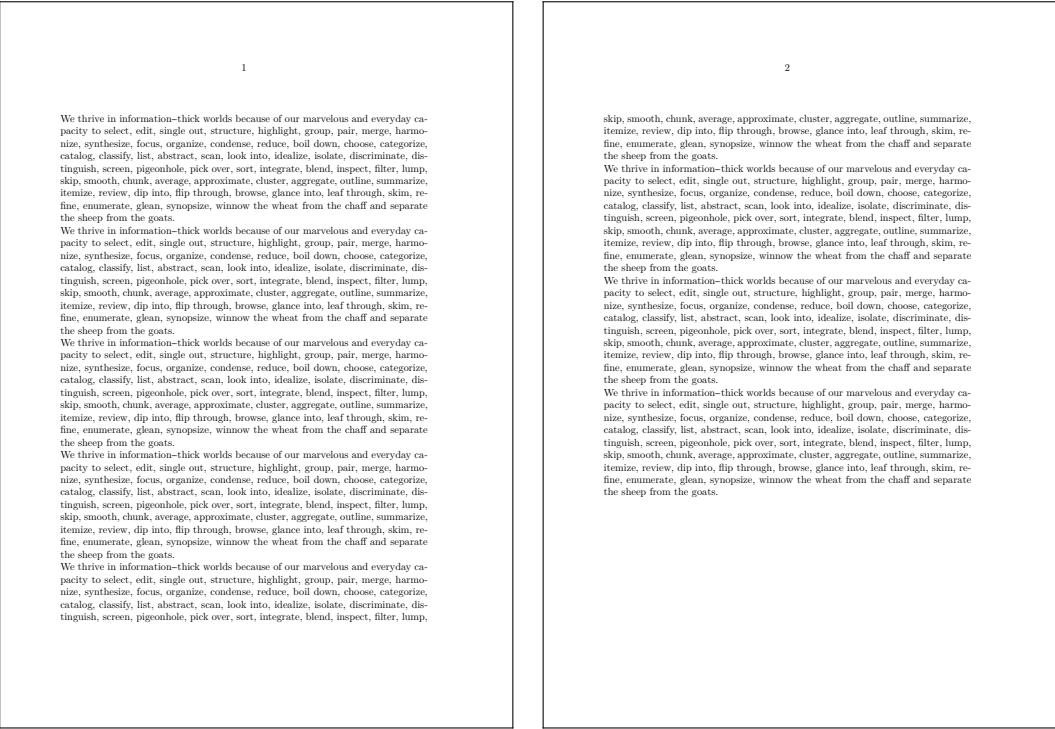
\startshapedparagraph[list=test-repeat]
  \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph

```



discard, finite shape, page 1

discard, finite shape, page 2



discard, repeat in shape, page 1

discard, repeat in shape, page 2

Figure 2 Discarded shaping

In figure 3 we use this instead:

```
\startshapedparagraph[list=test,method=shift]
  \dorecurse{8}{\showparagraphshape\samplefile{tufte} \par}
\stopshapedparagraph
```

Finally, in figure 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

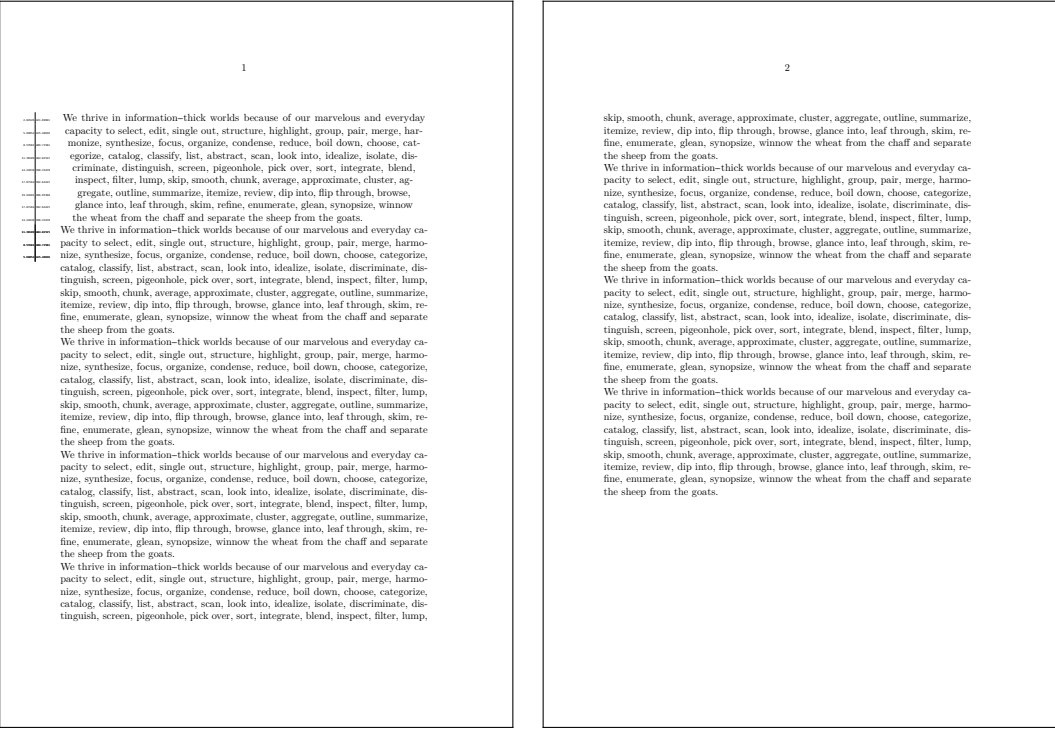
\startshapedparagraph[list=test,method=cycle]
  verse line 1.1\crlf verse line 2.1\crlf
  verse line 3.1\crlf verse line 4.1\par
  verse line 1.2\crlf verse line 2.2\crlf
  verse line 3.2\crlf verse line 4.2\crlf
  verse line 5.2\crlf verse line 6.2\par
\stopshapedparagraph
```

```
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
```

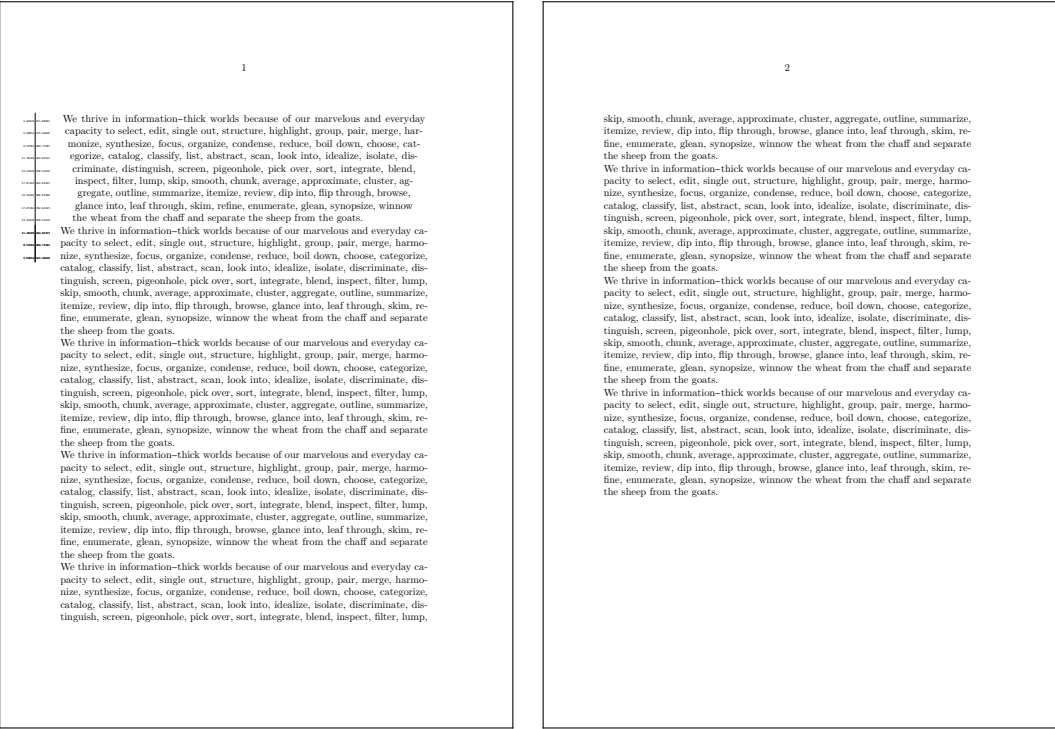
Because the idea for this feature originates in MetaFun, we will now kick in some Meta-Post. The following code creates a shape for a circle. We use a 2mm offset here:

Shapes



shift, finite shape, page 1

shift, finite shape, page 2



shift, repeat in shape, page 1

shift, repeat in shape, page 2

Figure 3 Shifted shaping


```

\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 every-
day capacity to select, edit, single out, structure,
highlight, group, pair, merge, harmonize, synthesize, focus,
organize, condense, reduce, boil down, choose, categorize, cat-
alog, classify, list, abstract, scan, look into, idealize, isolate, discrimi-
nate, 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, synopsise, 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, clus-
ter, aggregate, outline, summarize, itemize, review, dip into, flip through, browse,
glance into, leaf through, skim, refine, enumerate, glean, synopsise, 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

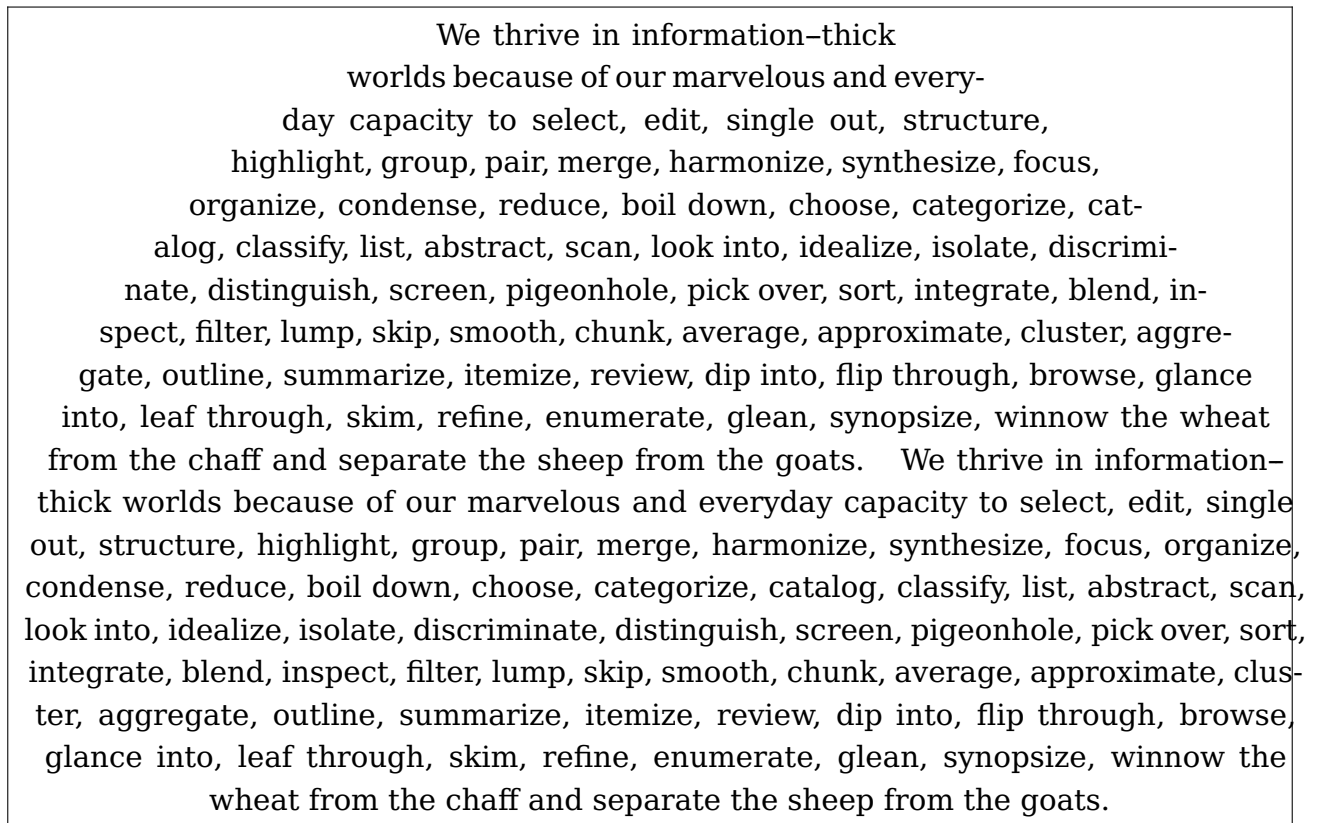
```

The result is shown in figure 5. Because all action happens in the framed environment, we can also use this definition:

```

\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

```



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, synthesize, 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, synthesize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 5 A framed circular shape

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.

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.mkx` can 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 6 we see the result. Watch how for two shapes we have enabled tracing. Of course you need to tweak till all fits well but we're talking of special situations anyway.

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
```

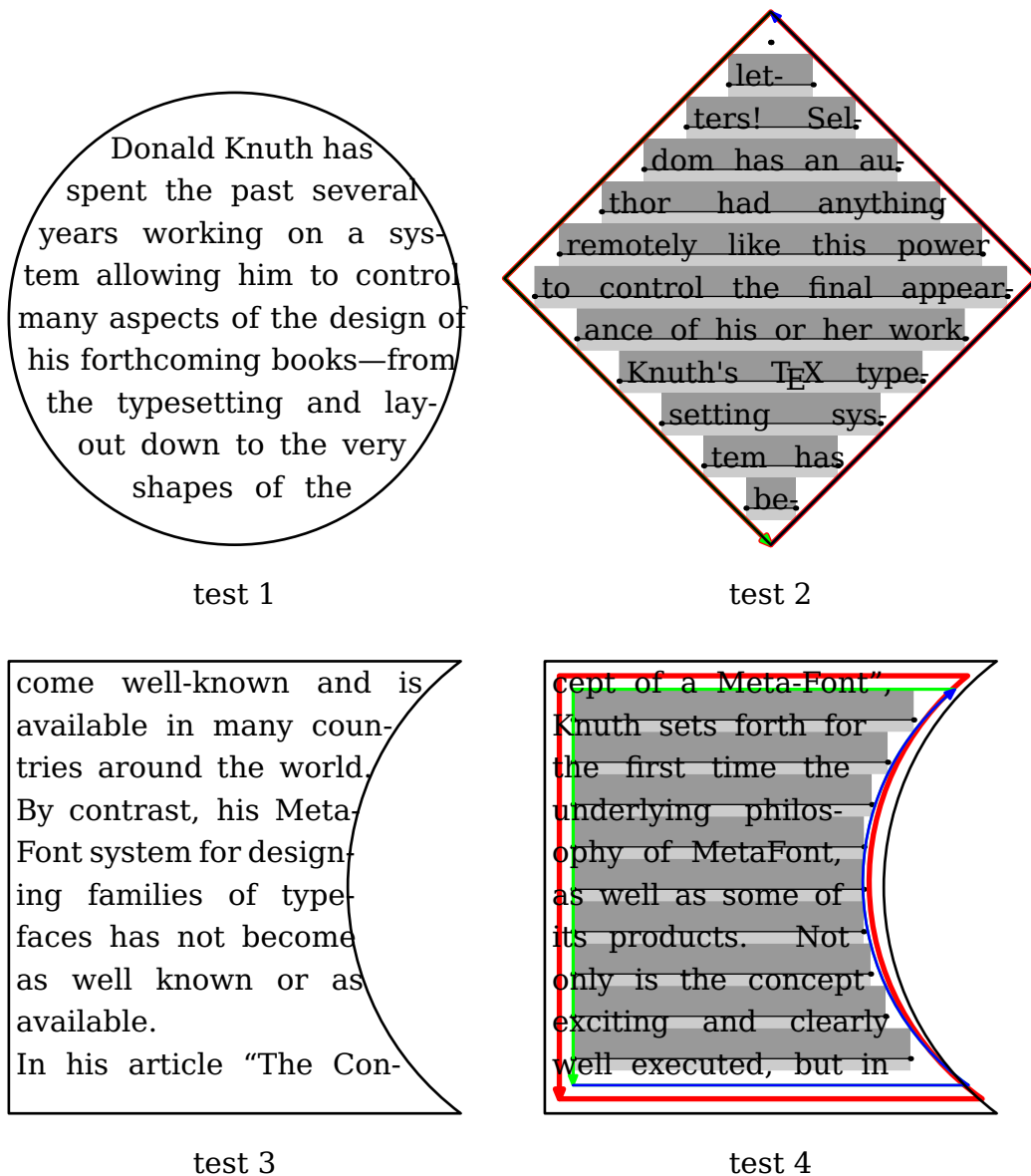


Figure 6

```

{\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 every-
day capacity to select, edit, single out, structure,
highlight, group, pair, merge, harmonize, synthesize, focus,
organize, condense, reduce, boil down, choose, categorize, cat-

alog, 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, synopsise, 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, synopsise, 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, synopsise, 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, synopsise, 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, synopsise, 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, 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, synopsise, 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, synopsise, 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, pigeon-
 hole, pick over, sort, integrate, blend, inspect, filter, lump,
 skip, smooth, chunk, average, approximate, cluster,
 aggregate, outline, summarize, itemize, re-
 view, dip into, flip through, browse,
 glance into, leaf
 through, skim, refine, enumer-
 ate, glean, synopsise, 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, fo-
 cus, organize, condense, reduce, boil down, choose, categorize, catalog,
 classify, list, abstract, scan, look into, idealize, isolate, discriminate, distin-
 guish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump,
 skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summa-
 rize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim,
 refine, enumerate, glean, synopsise, winnow the wheat from the chaff and separate
 the sheep from the goats. We thrive in information-thick worlds because of our mar-
 velous 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, discrimi-
 nate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter,
 lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, sum-
 marize, itemize, review, dip into, flip through, browse, glance into, leaf through,
 skim, refine, enumerate, glean, synopsise, 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 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

```

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 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
```

7 Modes

todo: some of the side effects of so called modes

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` spreads till the edges and `\cleaders` centers the lot. The `\gleaders` primitive (which is not in original T_EX) takes the outer box as reference and further behaves like `\cleaders`.

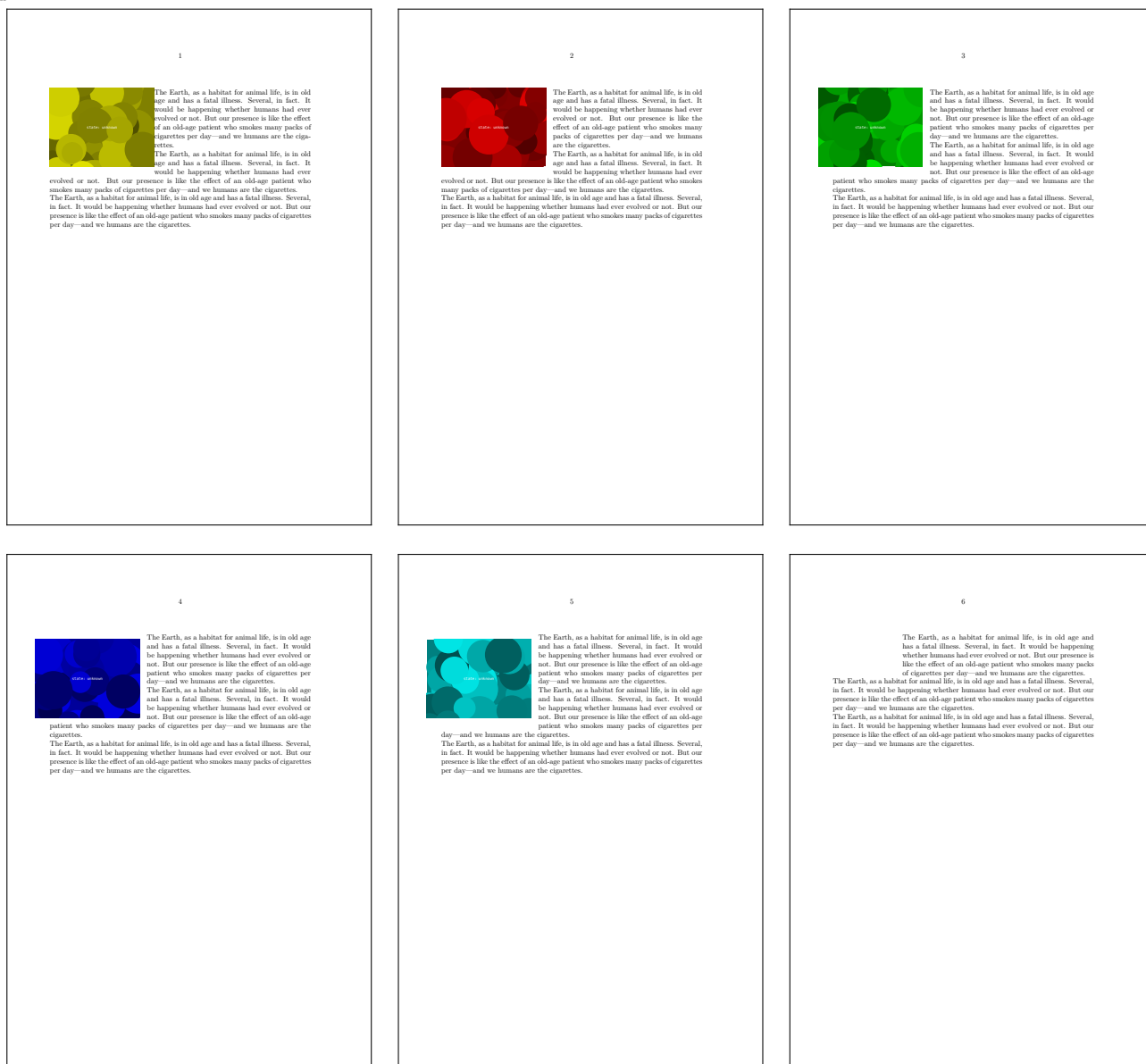
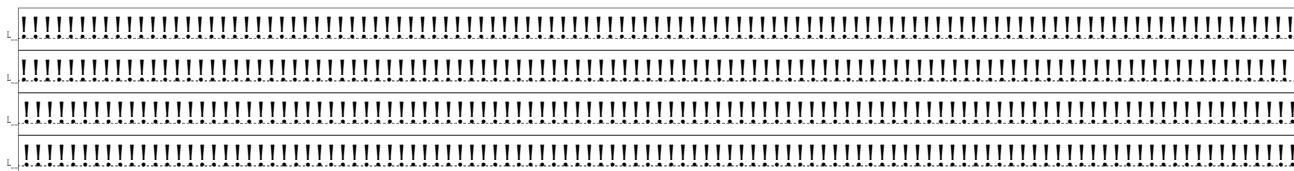


Figure 8 Flow around something



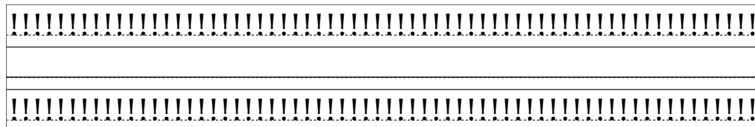
The leader primitives take box or rule but in LuaMetaTeX a glyph can also be specified, which saves wrapping in a box.

```
\ruledvbox \bgroup \hsize 10cm
\strut\cleaders\hbox{!}\hfill\strut
\egroup
```

Leaders

```
\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.

```
x xx xxx xxxx LR x xx xxx xxxx x xx xxx xxxx LR x xx xxx xxxx x xx xxx xxxx LR x xx xxx xxxx x xx xxx xxxx
LR xxx xxx xxx xxx xxx xxx LR LR xxx xxx xxx xxx xxx LR LR LR xxx xxx LR LR LR xxx
xxxx xxx LR LR LR xxx xxx xxx xxx LR LR LR xxx xxx xxx LR LR LR xxx xxx LR LR LR xxx
xxx LR xxx xxx xxx LR xxx LR xxx xxx xxx LR LR LR xxx xxx LR LR LR xxx LR LR LR xxx
xxxx xxx LR LR LR xxx xxx xxx xxx LR LR LR xxx LR LR LR xxx LR LR LR xxx LR LR LR xxx
xxx xxx LR LR LR xxx xxx LR LR LR xxx LR LR LR LR LR xxx xxx
```

In ConTeXt we have wrapped this feature in the adaptive box mechanism, so here a few a few examples:

```
\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 9 for the result):

```

\framed[height=18cm,align=middle,adaptive=yes,top=,bottom=] {%
  \begstrut \samplefile{tufte} \endstrut
  \par
  \adaptivevbox
    [strut=yes,setup=adaptive:test:a]
    {\showstruts\strut\hsize5cm\hss}%
  \par
  \adaptivevbox
    [strut=yes,setup=adaptive:test:b]
    {\showstruts\strut\hsize5cm\hss}%
  \par
  \begstrut \samplefile{tufte} \endstrut
}

```

Here is one that you can test yourself:

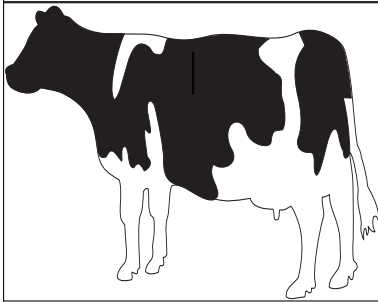
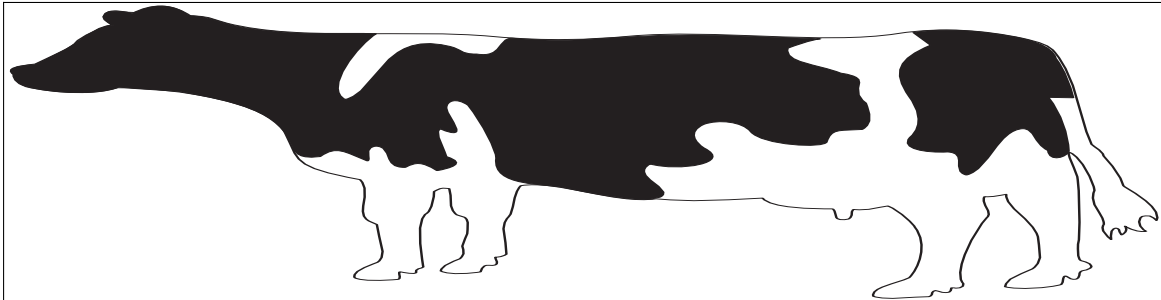
```

\startsetups adaptive:test
  \setbox\usedadaptivebox\vbox to \usedadaptivetotal \bgroup
    \externalfigure
      [cow.pdf]
      [width=\usedadaptivewidth,
        height=\usedadaptivetotal]%
  \egroup
\stopsetups

\rule\vbox to \textheight {
  \par \begstrut \samplefile{tufte} \endstrut \par
  \adaptivevbox[strut=yes,setup=adaptive:test]{\hsize\textrwidth\hss}
  \par \begstrut \samplefile{tufte} \endstrut
}

```

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, synthesize, 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, synthesize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 9

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
        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}{%




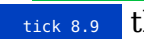


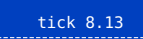
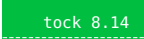
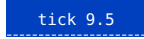


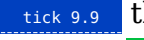


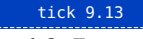
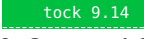



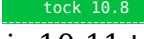




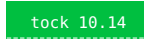




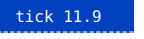
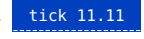


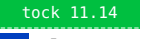
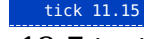
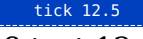


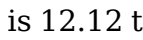

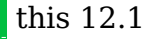

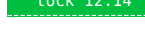

```



```

\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  this 8.6 is 8.6 test 8.6  this 8.7 is 8.7 test 8.7
 this 8.8 is 8.8 test 8.8  this 8.9 is 8.9 test 8.9  this 8.10 is 8.10 test 8.10
 this 8.11 is 8.11 test 8.11  this 8.12 is 8.12 test 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 test 10.12  this 10.13 is 10.13 test 10.13  this 10.14 is 10.14
 test 10.14  this 10.15 is 10.15 test 10.15  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 test 11.13  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 12.5  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 12.11
 this 12.12 is 12.12 test 12.12  this 12.13 is 12.13 test 12.13  this 12.14
 is 12.14 test 12.14  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,setup=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 

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 T_EX was written the assumption was that a negative depth of more than 1000 point made no sense at all. The last depth on a vertical list is registered in the `\prevdepth` variable. This is basically a reference into the current list. In order to illustrate some interesting side effects of setting this `\prevdepth` and especially when we set it to -1000pt. In order to illustrate this this special value can be set to a different value in LuaMetaT_EX. However, as dealing with the property is somewhat special in the engine you should not set it unless you know that the macro package is ware of it.

```
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 ConT_EXt we use `\ignoredepthcriterium` instead of the hard coded dimension. We now give a more extensive example:

In this example we set `\ignoredepthcriterium` to `-50.0pt` instead of the normal `-1000pt`. The helper is defined as:

or

The result of the following example is shown in figures 10 and 11. The first case is what we normally have in text and we haven't set `prevdepth` explicitly between lines so T_EX will just look at the depth of the lines. In the second case the depth is ignored when less than the criterium which is why, when we set the depth of the box to a negative value we get somewhat interesting skips.

FIRST	FIRST	FIRST	FIRST	FIRST	FIRST	FIRST
-10.0pt	-20.0pt	-49.9pt	-50.0pt	-50.1pt	-60.0pt	-80.0pt
LAST	LAST	LAST	LAST	LAST	LAST	LAST

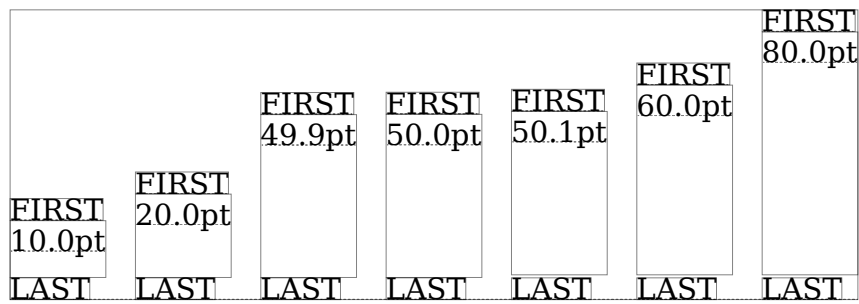
Figure 10

FIRST	FIRST	FIRST
-10.0pt	-20.0pt	-49.9pt
LAST	LAST	LAST

FIRST	FIRST
-50.0pt	-50.1pt
FIRST	FIRST
-60.0pt	-80.0pt

Figure 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 criterium makes for nice experiments. Just for the record, in figure 12 you see what we get with positive values:

**Figure 12**

Watch the interline skip kicking in when we make the depth larger than in `\ignoredepthcriterium` being 50pt.

10 Normalization

todo: users don't need to bother about this but it might be interesting anyway

11 Dirty tricks

todo: explain example for combining paragraphs

11 Colofon

Author	Hans Hagen
ConT _E Xt	2023.04.27 17:04
LuaMetaT _E X	2.1008
Support	www.pragma-ade.com contextgarden.net