low level TEX

inserts

Contents

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 that concept.

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 more options and tracing than other engines.

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 have errors.

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 them before another element triggers the builder.
- 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 normally is not the case (I need an example).
- 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 you can use `tex.triggerbuildpage()` to flush the pending contributions.

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` and 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 variable that hold the initial `\pagegoal` but it might play a role in future extensions.

Another variable is `\deadcycles` that registers the number of times the output routine is called without returning result.

3 Inserts

We now come to inserts. In traditional TEX an insert is a data structure that runs on top of registers: a box, count, dimension and skip. An insert is accessed by a number so for instance insert 123 will use the four registers of that number. Because TEX only offers a command alias mechanism for registers (like `\countdef`) a macro package will implement some allocator management subsystem (like `\newcount`). A `\newinsert` has to be defined in a way that the four registers are not clashing with other allocators. When you start with TEX seeing code that deals with in (in plain TEX) can be puzzling but it follows from the way TEX is set up. But inserts are probably not what you start exploring right away away.

In LuaMetaTEX you can set `\insertmode` to 1 and that is what we do in ConTEXt. In that mode inserts are taken from a pool instead of registers. A side effect is that like the page properties the insert properties are global too but that is normally no problem and can be managed well by a macro package (that probably would assign register the values globally too). The insert pool will grow dynamically on demand so one can just start at 1; in ConTEXt MkIV we use the range 127 upto 255 in order to avoid a clash with registers. In LMTX we start at 1 because there are no clashes.

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 `\floatingpenalty` variable determines the penalty applied when a split is needed.

In the output routine the original TEX variable `\insertpenalties` is a counter that keeps the number of insertions that didn't fit on the page while otherwise if has the accumulated penalties of the split insertions. When `\holdinginserts` is non zero the inserts in the list are not collected for output, which permits the list to be fed back for reprocessing.

The LuaMetaTEX specific storage mode `\insertstoring` variable is explained in the next section.

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.

5 Synchronizing

The page builder can triggered by (for instance) a penalty but you can also use `\pageboundary`. This will trigger the page builder but not leave anything behind. (This is experimental.)

6 Migration

Todo, nothing new there, so no hurry.

7 Callbacks

Todo, nothing new there, so no hurry.

7 Colofon

Author	Hans Hagen
ConTEXt	2023.04.27 17:04
LuaMetaTEX	2.1008
Support	www.pragma-ade.com
	contextgarden.net