

low level

TEX

alignments

## Contents

1	Introduction	1
2	Between the lines	3
3	Pre-, inter- and post-tab skips	5
4	Cell widths	8
5	Plugins	9
6	Pitfalls and tricks	12
7	Rows	15
8	Remark	18

## 1 Introduction

$\text{\TeX}$  has a couple of subsystems and alignments is one of them. This mechanism is used to construct tables or alike. Because alignments use low level primitives to set up and construct a table, and because such a setup can be rather extensive, in most cases users will rely on macros that hide this.

```
\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  $\text{Con}\text{\TeX}t$  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
111.111222,222333=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 catcode that relates to a parameter and alignment tab (and in ConT<sub>E</sub>Xt that is not the case). The T<sub>E</sub>Xbook has plenty alignment examples so if you really want to learn about them, consult that must-have-book.

## 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 T<sub>E</sub>X 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 T<sub>E</sub>X using alignments you should not be surprised to notice rather ugly spacing. The user (or the macropackage) should deal with that explicitly, and this is not always the case.

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  $\text{T}_{\text{E}}\text{X}$  scanner. Actually, the scanner is either scanning an alignment or it expects regular text (including math). When you look at the previous example you see `\noalign`. When the preamble is read,  $\text{T}_{\text{E}}\text{X}$  will pick up rows till it finds the final brace. Each row is added to a temporary list and the `\noalign` will enter a mode where other stuff gets added to that list. It all involves subtle look ahead but with minimal overhead. When the whole alignment is collected a final pass over that list will package the cells and rows (lines) in the appropriate way using information collected (like the maximum width of a cell and width of the current cell. It will also deal with spanning cells then.

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 ConT<sub>E</sub>Xt does that for you. But because the code doesn't always look pretty then, in LuaMetaT<sub>E</sub>X 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:<sup>1</sup>

```
\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.

### 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}
```

<sup>1</sup> One can argue for using the name `\peekaligned` because in the meantime other alignment primitives also can use this property.

```

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).

---

cell 1.1	cell 1.2	cell 1.3
cell 2.1	cell 2.2	cell 2.3

In the ConT<sub>E</sub>Xt 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 skips 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, of 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 are larger so it is one of these cases where in ConTeXt we get messages on the console that node memory is bumped.<sup>2</sup>

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:

---

<sup>2</sup> 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.



---

cell<sub>SP:3 497</sub>.1 cell<sub>SP:3 497</sub>.2 cell<sub>SP:3 497</sub>.3

---

cell<sub>SP:2 467</sub>.1 cell<sub>SP:2 467</sub>.2 cell<sub>SP:2 467</sub>.3

---

When playing with all this the LuaMetaTeX engine also got a tracing option for alignments. We already had one that showed some of the `\noalign` side effects, but showing the preamble was not yet there. This is what `\tracingalignments = 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.

## 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 LuaMetaT<sub>E</sub>X the hlist nodes are rather large because we have more options than in original T<sub>E</sub>X, for instance offsets and orientation. In a table with 10K rows of 4 cells yet get 40K extra hlist nodes allocated. Now, one can argue that we have plenty of memory but being lazy is not really a sign of proper programming.

```

\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 1.2	x	cell 1.3
xcell 2.1	x	cell 2.2	x	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.

## 5 Plugins

Yet another LuaMetaT<sub>E</sub>X extension is a callback that kicks in between the preamble pre-roll and finalizing the alignment. Initially as test and demonstration a basic character

alignment feature was written but that works so well that in some places it can replace (or compliment) the already existing features in the ConT<sub>E</sub>Xt 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 ConT<sub>E</sub>Xt is rather old and stable and it is the preferred way to deal with tabular content in the text flow. However, adding the G specifier (as variant of the g one) could be done without interference or drop in performance. This new G specifier tells the tabulate mechanism that in that column the given character is used to vertically align the content that has this character.

```
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 ConT<sub>E</sub>Xt one. It is however made easy by this callback mechanism. We can of course use this feature with the low level alignment primitives, assuming that you tell the machinery that the plugin is to be kicked in.

```
\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:<sup>3</sup>

```
\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
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.

## 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
```

<sup>3</sup> Should this be an option instead?

```

\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.

H	[1]	H	[2]	H	[3]
H	1	H	2	H	3
H	[1]	H	[2]	H	[3]
H	[1]	[2]	H	H	[3]
H	[1]	[2]	[3]	H	H
H	[1]	23	H	H	H
H	123	H	H	H	H

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 that by using `\aligncontent`. Keep in mind that when the preamble is scanned there is no doesn't expand with the exception of the token after `\span`.

#### `\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.

## 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 it remains complicated (for all good reasons). Cells as well as rows are finalized after the whole alignment has been collected and processed. In the end cells and rows are boxes but till we're done they are in an ‘unset’ state.

Scanning starts with interpreting the preamble, and then grabbing rows. There is some nasty lookahead involved for `\noalign`, `\span`, `\omit`, `\cr` and `\crr` and that is not code



one wants to tweak too much (although we did in LuaMetaTeX). This means for instance that adding ‘let's 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
}

```

cell one	cell one
cell one	cell one
cell two	cell two
cell two	cell two
cell three	cell three
cell three	cell three
cell four	cell four
cell four	cell four

The supported keywords are similar to those for boxes: source, target, anchor, orientation, shift, xoffset, yoffset, xmove and ymove. The 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
}
```

cell one	cell one
cell one	cell one
cell two	cell two
cell two	cell two
cell three	cell three
cell three	cell three
cell four	cell four
cell four	cell four

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 ConT<sub>E</sub>Xt high level interfaces like tabulate.

## 8 Remark

It can be that the way alignments are interfaced with respect to attributes is a bit different between LuaT<sub>E</sub>X and LuaMetaT<sub>E</sub>X but because the former is frozen (in order not to interfere with current usage patterns) this is something that we will deal with deep down in ConT<sub>E</sub>Xt 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.

## 8 Colofon

Author           Hans Hagen  
 ConT<sub>E</sub>Xt        2023.04.27 17:04  
 LuaMetaT<sub>E</sub>X   2.1008  
 Support         www.pragma-ade.com  
                   contextgarden.net