The autotabbing package

Pascal Germroth
pascal@germroth.name

Version 1.0.0, dated 2015/05/05

1 Introduction

This package attempts to combine the way tabstops can be defined inline from `tabbing` with the way column sizes are determined automatically from `tabular`. You don't define columns beforehand but simply insert named tabstops which will be aligned with each other, as determined by the content.
Originally written to pretty print aligned Haskell code in a proportional font: see the autoverbalign package for an attempt to guess these tabstops from monospace-aligned text. Declaring a complex alignment like this is easier than using a `tabbing` environment or a `tabular`.

```
map :: (a → b) → [Maybe a] → [b]    -- this is a longer comment
map   fun          (x:xs)     = case x of Just y   → fun y : map xs
{-          centered comment            -} Nothing → map xs
map   _            []         = []
```

Here is a flexible three-column layout—there is no need to define the columns beforehand or use `\multicolumn` for cell spans.

first	second	third
	this is centered	third
something longer	. . . . .	
this takes	. . . . .	the whole row
	multiple columns	

```
\begin{autotabbing}
  first  & second & third \\
  \hfill this is centered \hfill
    &[2] \hfill third \\
  something longer & \dotfill \\
  this takes \dotfill the whole row \\
  & multiple columns
\end{autotabbing}
```

There are no high-level functions at the moment, i.e. aligning content is done using `\hfill`, there are no borders or backgrounds or cell templates. There is also no support for cells spanning multiple rows.

The environment supports breaking across pages
while content says aligned

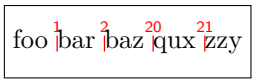
Of course for simple layouts longtable would be more flexible here, but this package doesn't try to provide full-featured tables but instead a very simple automatic alignment function.

2 Usage

This package requires LuaLaTeX, Load the package using `\usepackage[`⟨*kvargs*⟩`]{autotabbing}`. Use the environment: `autotabbing`, `\begin{autotabbing}[`⟨*kvargs*⟩`]` ... `\end{autotabbing}`. Use the optional arguments or `\autotabbingsetup{`⟨*kvargs*⟩`}` to set options.

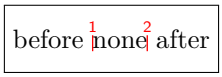
`&` `&[`⟨*id*⟩`]`

Set a tabstop using `&[`⟨*id*⟩`]`. This introduces a new cell. Stop numbers must be given in ascending order within each row, but arbitrary gaps are fine. If no ⟨*id*⟩ is given or a smaller one, it will just use the next available.



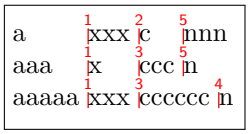

```
\begin{autotabbing}
  foo & bar & baz &[20] qux & zzy
\end{autotabbing}
```

`&` ignores the following spaces but not the preceding ones. To get a space at the start of a cell, hide it:




```
\begin{autotabbing}
  before & none& { after}
\end{autotabbing}
```

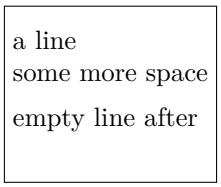
All tabs with the same ID are aligned to the same position; but so may multiple independent tabs (2 and 3 below). While tabs must be used in ascending order, a tab with a lower ID may still end up to the right of one with a higher ID (4 and 5 below). This is intentional and allows for extremely complex layouts.




```
\begin{autotabbing}
  a      & xxx & c      &[5] nnn \\
  aaa    & x   &[3] ccc &[5] n \\
  aaaaa & xxx &[3] cccccc &[4] n
\end{autotabbing}
```

`\\` `\\[⟨skip⟩]`

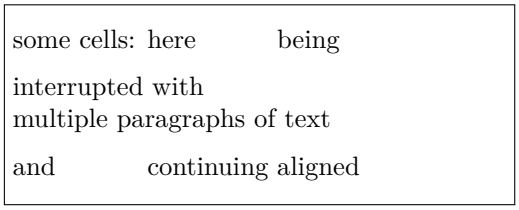
Start a new row using `\\[⟨skip⟩]`. Don't end the last one with `\\`, it will cause an empty row. The optional argument may contain a vertical skip inserted in addition to `row-sep`.




```
\begin{autotabbing}
  a line\\
  some more space\\[.5em]
  empty line after\\
\end{autotabbing}
```

`\intertext` `\intertext{⟨content⟩}`

Like in amsmath, `\intertext{⟨content⟩}` allows interrupting an alignment, aligning across the break. Unlike in the math environments, there should *not* be a `\\` before the `\intertext` as this will cause an empty line. The content may contain paragraph- and page breaks, it will be added directly to the parent vlist.



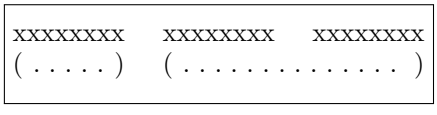

```
\begin{autotabbing}
  some cells: & here & being
  \intertext{
  interrupted with

  multiple paragraphs of text
  }
  and & continuing & aligned
\end{autotabbing}
```

2.1 Spacing

`above-skip`
`below-skip`
`above-intertext-skip`
`below-intertext-skip`
`col-sep`
`row-sep`

Similar to tabbing, the row boxes are laid out like paragraph lines. There are options provided for additional vertical spacing, defaulting to `.5\baselineskip`: For each new row `\\[⟨additional⟩]` extra space can be inserted. `above-skip` and `below-skip` are inserted before and after the first and last row respectively. `above-intertext-skip` and `below-intertext-skip` are inserted before and after an `\intertext`, adjust this instead of using `\\` here. To spread the layout there are `col-sep` and `row-sep`, which default to zero.

Alignment takes `col-sep` into account even when spanning multiple columns:



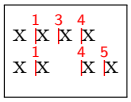

```
\begin{autotabbing}[col-sep=5mm]
  xxxxxxxx& xxxxxxxx& xxxxxxxx\\
  (\dotfill)& (\dotfill)
\end{autotabbing}
```

2.2 Debugging

draft
final

`draft` = ⟨*bool*⟩
`final` = ⟨*bool*⟩

The `draft` option, and its opposite `final` can be used to respectively enable or disable a debug mode where tabs are drawn as zero-sized (i.e. with no impact on the layout) markers showing the tab's ID.




```
\begin{autotabbing}[draft]
  x & x &[3] x & x \\
  x & x &[4] x & x
\end{autotabbing}
```

3 Implementation

3.1 LaTeX-part

```
⟨*package⟩
⟨@@=autotabbing⟩
\ProvidesExplPackage{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}
  {\ExplFileDescription}
```

Load the Lua portion of the package

```
\RequirePackage{luatexbase}
\RequireLuaModule{autotabbing}
\RequirePackage{xparse}
```

3.1.1 Options

We use `l3keys` to define the options.

\l__autotabbing_draft_bool

Opposing `draft` and `final` switches.

```
\keys_define:nn { autotabbing } {
  draft .bool_set:N = \l__autotabbing_draft_bool ,
  draft .initial:n = false ,
  draft .default:n = true ,
  final .bool_set_inverse:N = \l__autotabbing_draft_bool ,
  final .default:n = true ,
}
```

\l__autotabbing_above_skip

Space above block

```
\keys_define:nn { autotabbing } {
  above-skip .skip_set:N = \l__autotabbing_above_skip ,
  above-skip .initial:n = .5\baselineskip ,
}
```

`\l__autotabbing_below_skip` Space below block

```
\keys_define:nn { autotabbing } {
  below-skip .skip_set:N = \l__autotabbing_below_skip ,
  below-skip .initial:n = .5\baselineskip ,
}
```

`\l__autotabbing_above_intertext_skip`

Space above `\intertext`

```
\keys_define:nn { autotabbing } {
  above-intertext-skip .skip_set:N = \l__autotabbing_above_intertext_skip ,
  above-intertext-skip .initial:n = .5\baselineskip ,
}
```

`\l__autotabbing_below_intertext_skip`

Space below `\intertext`

```
\keys_define:nn { autotabbing } {
  below-intertext-skip .skip_set:N = \l__autotabbing_below_intertext_skip ,
  below-intertext-skip .initial:n = .5\baselineskip ,
}
```

`\l__autotabbing_col_sep` Space between cells horizontally

```
\keys_define:nn { autotabbing } {
  col-sep .skip_set:N = \l__autotabbing_col_sep ,
  col-sep .initial:n = 0pt ,
}
```

`\l__autotabbing_row_sep` Space between rows vertically

```
\keys_define:nn { autotabbing } {
  row-sep .skip_set:N = \l__autotabbing_row_sep ,
  row-sep .initial:n = 0pt
}
```

Set from package options.

```
\RequirePackage{l3keys2e}
\ProcessKeysOptions { autotabbing }
```

`\autotabbingsetup` `#1` : key-value list of options

Set the options for the remainder of the context.

```
\NewDocumentCommand \autotabbingsetup { +m } {
  \keys_set:nn { autotabbing } {#1}
}
```

3.1.2 Creating attributed cells

Cells and rows are boxes with special attributes marking their tabstops, which are then re-packed at the correct size using Lua.

`\g__autotabbing_current`

Keep track of the current tabstop.

```
\int_new:N \g__autotabbing_current
```

`\__autotabbing_draw_tab:`

Draft visualisation: show the tabstops and their IDs.

```
\cs_new:Nn \__autotabbing_draw_tab: {
  \bool_if:nT {
    \l__autotabbing_draft_bool
    && \int_compare_p:n { \g__autotabbing_current > 0 }
  } {
    \raisebox{0pt}[0pt][0pt]{% zero height box
      \ifdefined\color\color{red}\fi
      \makebox[0pt][c]{% zero width rule
        \rule[-.1\baselineskip]{.5pt}{.5\baselineskip}}
      \makebox[0pt][c]{% zero width label
        \raisebox{0.5\baselineskip}{
          \fontsize{6pt}{6pt}\sffamily
          \int_to_arabic:n \g__autotabbing_current
        }
      }
    }
  }
}
```

`\__autotabbing_enable_syntax:`

Enable the user syntax inside the current group.

```
\group_begin:
\char_set_catcode_active:N &
\cs_new:Nn \__autotabbing_enable_syntax: {
  \DeclareDocumentCommand & { O{0} } {
    \__autotabbing_next_cell:n {##1}
  }
  \char_set_catcode_active:N \&

  \DeclareDocumentCommand \\ { O{0pt} } {
    \__autotabbing_next_row:n {##1}
  }
  \DeclareDocumentCommand \intertext { +m } {
    \__autotabbing_intertext:n {##1}
  }
}
\group_end:
```

`\__autotabbing_cell:`
`\__autotabbing_cell_end:`

We put the cell's contents into a box, giving it a tag and the ID of the starting tab via attributes. (Just using 0 and 1 should be fine here since the box will be recreated without attributes once its size is known)
The tabular-like syntax is only available inside cells, not i.e `\intertext`.
We ignore the spaces at the beginning to avoid inconsistency with the optional argument of `&` and ease, but allow them at the end where it's easy to choose. The box ends with a glue so the content won't stretch when the width is ajusted, manual alignment using `\hfill` still works.

```
\cs_new:Nn \__autotabbing_cell: {
  \hbox
    attr0 = 2% mark as cell
    attr1 = \int_use:N \g__autotabbing_current
  \bgroup
    \__autotabbing_draw_tab:
    \__autotabbing_enable_syntax:
    \ignorespaces
}
\cs_new:Nn \__autotabbing_cell_end: {
    \hss
  \egroup
}
```

`\__autotabbing_next_cell:n`

`#1` : name of this tabstop
Available as `&` in the environment. To ensure increasing tabs, without or with an invalid argument just advance the counter.

```
\cs_new:Nn \__autotabbing_next_cell:n {
  \__autotabbing_cell_end:
  \int_compare:nTF { #1 > \g__autotabbing_current }
    { \int_gset:Nn \g__autotabbing_current {#1}  }
    { \int_gincr:N \g__autotabbing_current }
  \skip_horizontal:N \l__autotabbing_col_sep
  \__autotabbing_cell:
}
```

`\__autotabbing_row:`
`\__autotabbing_row_end:`

A row starts at tab 0. Mark the row box using attribute. (Should be fine to use `attr0` here as well, the box will be recreated)

```
\cs_new:Nn \__autotabbing_row: {
  \int_gset:Nn \g__autotabbing_current 0
  \hbox attr0 = 1 % mark as row
  \bgroup
    \__autotabbing_cell:
}
\cs_new:Nn \__autotabbing_row_end: {
    \__autotabbing_cell_end:
  \egroup
}
```

`\__autotabbing_next_row:n`

`#1` : additional skip to insert between rows

Available as `\\` in the environment. Closes the current row and immediately opens a new one.

```
\cs_new:Nn \__autotabbing_next_row:n {
  \__autotabbing_row_end:
  \skip_vertical:N \l__autotabbing_row_sep
  \skip_vertical:n {#1}
  \__autotabbing_row:
}
```

`\__autotabbing_intertext:n`

`#1` : the content that should be placed between rows

Interrupt the display. Doesn't check if the last row may have been empty.

```
\cs_new:Nn \__autotabbing_intertext:n {
  \__autotabbing_row_end:
  \skip_vertical:N \l__autotabbing_above_intertext_skip
  #1
  \par
  \skip_vertical:N \l__autotabbing_below_intertext_skip
  \__autotabbing_row:
}
```

`\__autotabbing_tabbing:`

Collect the rows into a temporary vbox, correcting the `baselineskip` glue.

```
\cs_new:Nn \__autotabbing_tabbing: {
  \par\dimen0 = \prevdepth
  \setbox0 = \vbox\bgroup
    \prevdepth = \dimen0
    \__autotabbing_row:
}
```

`\__autotabbing_tabbing_end:`

Use Lua to adjust the cells and insert the vbox's contents into the parent vbox. We unskip the final space here to allow a linebreak before the `\end` without having to use `%` in the line before.

```
\cs_new:Nn \__autotabbing_tabbing_end: {
      \unskip
    \__autotabbing_row_end:
  \egroup
  \directlua{autotabbing.adjust()}
  \skip_vertical:N \l__autotabbing_above_skip
  \unvbox0
  \skip_vertical:N \l__autotabbing_below_skip
}
```

autotabbing `#1` : key-value options

```
\NewDocumentEnvironment {autotabbing} {o}
  {
    \group_begin:
      \IfValueT {#1} { \autotabbingsetup {#1} }
      \__autotabbing_tabbing:
  }
  {
      \__autotabbing_tabbing_end:
    \group_end:
    \noindent
    \ignorespacesafterend
  }
```

⟨/package⟩

3.2 Lua-portion

```
⟨*lua⟩
local err, warn, info, log =
  luatexbase.provides_module({name = 'autotabbing'})
autotabbing = autotabbing or {}

local first_tab = 0
local last_tab = 1000
```

`get_offsets()` Get the horizontal offsets of the tab stops. We measure the cells column-wise (keeping track using the `index` table, since we might not visit a cell in each row every step) by selecting those starting at the current stop (which had its offset calculated in a previous step), and moving the cell's end-stops. A cell may end at any stop with a larger index than its start. There may be gaps in the tab numbers, and their offsets don't have to be in the same order.

```
local function get_offsets(rows)
  local offsets = {[first_tab] = 0}
  local index = {}
  for i = 1, #rows do index[i] = 1 end
  local current = first_tab
  repeat
    for i, row in ipairs(rows) do
      local cell = row.cells[index[i]]
      if cell and cell.from == current then
        -- measure from our left edge to the left edge of the next cell
        local next_box = (row.cells[index[i] + 1] or {}).box
        local width = node.dimensions(cell.box, next_box)
        -- there may be content between cells.
        cell.right_margin = width - cell.box.width
        -- move our right tab stop accordingly
        local left = offsets[cell.from]
```

```
        local right = left + width
        if right >= (offsets[cell.to] or 0) then
          offsets[cell.to] = right
        end
        -- done with this cell
        index[i] = index[i] + 1
      end
    end
    -- find the next tab ID: it's the smallest,
    -- because tab IDs are topologically ordered.
    current = last_tab
    for i, row in ipairs(rows) do
      local cell = row.cells[index[i]]
      if cell and cell.from < current then
        current = cell.from
      end
    end
  until current == last_tab
  return offsets
end
```

`repack_hbox()` Repack the box in place.

```
local function repack_hbox(head, old, ...)
  local new = node.hpack(old.head, ...)
  head = node.insert_before(head, old, new)
  head = node.remove(head, old)
  return head, new
end
```

`adjust_widths()` Alter each cell's widths so they fit between their respective stops.

```
local function adjust_widths(head, rows, offsets)
  for _, row in ipairs(rows) do
    for _, cell in ipairs(row.cells) do
      -- recreate the cell at target width
      local col_width = offsets[cell.to] - offsets[cell.from]
      local width = col_width - cell.right_margin
      row.box.head, cell.box =
        repack_hbox(row.box.head, cell.box, width, 'exactly')
    end
    -- recreate the row at its new natural width
    head, row.box = repack_hbox(head, row.box)
  end
  return head
end
```

`collect_rows()` Collect the cells from the current table.

```
local function collect_rows(table_box)
  local rows = {}
  for row_box in node.traverse(table_box.head) do
```

```
    -- skip glue, intertext, etc, without explicit row tag.
    if node.has_attribute(row_box, 0, 1) then
      local cells = {}
      for cell_box in node.traverse(row_box.head) do
        -- skip glue etc without explicit cell tag
        if node.has_attribute(cell_box, 0, 2) then
          local cell = {
            box = cell_box,
            from = node.has_attribute(cell_box, 1),
            to = last_tab
          }
          -- previous cell ends where this one starts
          local prev_cell = cells[#cells]
          if prev_cell then
            prev_cell.to = cell.from
          end
          -- cell done
          table.insert(cells, cell)
        end
      end
      -- row done
      table.insert(rows, {
        box = row_box,
        cells = cells
      })
    end
  end
  return rows
end
```

`autotabbing.adjust()` Entry point. Find the cells, compute tabstops and adjust the widths. Modifies the box in-place.

```
function autotabbing.adjust()
  local vbox = tex.box[0]
  local rows = collect_rows(vbox)
  local offsets = get_offsets(rows)
  vbox.head = adjust_widths(vbox.head, rows, offsets)
end
```

⟨/lua⟩