rowmantic

A Typst package for row-wise table editing

version	`0.5.0`
import as	`#import "@preview/rowmantic:0.5.0"`
typst universe	https://typst.app/universe/package/rowmantic
repository	https://github.com/typst-community/rowmantic

1 Introduction

Rowmantic introduces the `rowtable` function as a row-oriented "frontend" to the usual `table` function in typst.

The `rowtable` function takes a markup block `[...]` per row, and the markup is split internally[1] on a delimiter which is `&` by default. In all other aspects it works like the usual `table` function, with `stroke`, `fill`, `hline` and so on.

Input:

```
rowtable(
  [A & B],
  [C & D & E])
```

A	B	
C	D	E

Equivalent table:

```
table(columns: 3,
  [A], [B], [],
  [C], [D], [E])
```

For improved table ergonomics, the longest row determines the number of columns, and all rows are effectively completed so that they are of full length. This creates a better editing experience, as rows and columns can be filled out gradually.

There is a corresponding `rowgrid` function with identical interface, but for the usual `grid`.

Contents

[1] But shallowly - not looking into styled or nested content

2 Examples

2.1 Introductory Examples

Document Result

goá	*iáu-boē*	*koat-tēng*	*tang-sî*	*boeh*	*tńg-khì*
goa[1]	iau[1]-boe[3]	koat[2]-teng[3]	tang[7]-si[5]	boeh[2]	tng[1]-khi[3]
goa[2]	iau[2]-boe[7]	koat[4]-teng[7]	tang[1]-si[5]	boeh[4]	tng[2]-khi[3]
I	not-yet	decide	when	want	return.

“I have not yet decided when I shall return.”

Input

```
#import "@preview/rowmantic:0.5.0": rowtable, expandcell
#{
  show regex("\d"): super.with(size: 0.8em, typographic: false)
  show table.cell: it => { set text(size: 0.9em) if it.y >= 1; it }
  show table.cell.where(y: 0): emph
  rowtable(
    separator: ",",   // configurable separator
    stroke: none,     // pass through table arguments, hlines, cells et.c.
    inset: (x: 0em),
    column-gutter: 0.9em,
    // rows are filled to be equal length after collecting cells
    [goá,   iáu-boē,    koat-tēng,    tang-sî,    boeh,   tńg-khì   ],
    [goa1,  iau1-boe3,  koat2-teng3,  tang7-si5,  boeh2,  tng1-khi3 ],
    [goa2,  iau2-boe7,  koat4-teng7,  tang1-si5,  boeh4,  tng2-khi3 ],
    [I,     not-yet,    decide,       when,       want,   return.   ],
    table.hline(),
    // cell that fills remainder of row
    expandcell["I have not yet decided when I shall return."],
  )
}
```

Example from Wikipedia[2]

[2] https://en.wikipedia.org/wiki/Interlinear_gloss

Document Result

Term	Explanation	Assumptions
$\mathbf{X}$	Explanatory variables	Non-random
$\mathbf{Y}$	$Y_1, ..., Y_n$ observations	**Pairwise independent**
β	Model parameters	

Input

```
#import "@preview/rowmantic:0.5.0": rowtable
#{
  set table(stroke: none, inset: 0.8em)
  set table.hline(stroke: 0.5pt)
  show table.cell.where(y: 0): strong
  show table.cell.where(x: 0): x => math.bold(math.upright(x))
  rowtable(
    table.hline(),
    table.header([Term  & Explanation         & Assumptions ]),
    table.hline(),
    [$X$      & Explanatory variables         & Non-random ],
    [$Y$      & $Y_1, ..., Y_n$ observations  & *Pairwise independent*],
    [$beta$   & Model parameters              ],
    table.hline(),
  )
}
```

2.2 Escaping and Various Examples

Document Result

Emphasis &	**Strong &**	Literal &
$\int_{-\infty}^{\infty} f(x)\,dx$	$\int_{0}^{\infty} f(t)\,e^{-st}\,dt$	$X\&Y$
Figure 1: Top A	See Figure 1 & Figure 2	B Figure 2: Bot
1. A 2. B	**A** a **B** b	A B
A	B	C

Escape the separator with \&

Display equations as a row

`set` rules need to be enclosed so that they don't try to style the separator itself.

Lists and other larger elements can be embedded as usual.

Colorfully filled cells

Input

```
#rowtable(
  align: horizon,
  column-width: 1fr,
  row-filler: [N/A],
  stroke: (x, y) => if x == 3 { none } else { 0.5pt },
  [_Emphasis &_   & *Strong &*  & Literal \&  & Escape the separator with `\&`],
  $ integral_(-oo)^oo f(x) thin d x   & integral_0^oo f(t) thin e^(-s t) thin d t
    & X \& Y & "Display equations" \ "as a row" $,
  [
    #{
      set figure.caption(position: top)
      [#figure(rect[A], caption: "Top")<fig1>]
    }
    &
    See @fig1 \& @fig2
    &
    #figure(rect[B], caption: "Bot")<fig2>
    &
    ```typc set``` rules need to be enclosed so that they don't try
    to style the separator itself.
  ],
  [
    + A
    + B
    &
    / A: a
    / B: b
    & #rowtable([A & B])
    & Lists and other larger elements can be embedded as usual.
  ],
  [ #table.cell(fill: yellow)[A]  & #table.cell(fill: orange)[B]  &
    #table.cell(fill: red)[C]     & Colorfully filled cells ],
)
```
```

2.3 Other Separators

Example: A double semicolon. The cell separator is specified as a string with the `separator` argument. It can be more than a single character, like in this example:

Document Result

First	Second; Third
Fourth	This is a literal ;; and ; and , and &

Input

```
#rowtable(
  separator: ";;",
  stroke: 0.5pt,
  [First    ;; Second; Third ],
  [Fourth   ;; This is a literal \;\; and ; and , and & ],
)
```

Example: Using a space as a separator has special behaviour – because of how spaces are treated in Typst markup: multiple spaces, tabs, or a (single) newline are all collapsed into just whitespace. Using a space as separator splits on any such whitespace:

Document Result

First	Second	Third (3rd)
Fourth	Fifth	Sixth (6th)

Input

```
#rowtable(
  separator: " ",
  stroke: 0.5pt,
  [First  Second Third~(3rd)],
  [Fourth Fifth  Sixth~(6th)],
)
```

The recommended way to insert a literal space in this case is to use `~`.

2.4 Using Other Table Functions

Use the `table` argument to let rowtable pass its result to a different table function rather than the standard one, for example `pillar.table` (shown below) or `zero.ztable`.

Document Result

Isotope	Z	N	Half-life	Mass (Da)	Abundance (%)
^{107}Ag	47	60	Stable	106.905 091 5(26)	51.839 ± 0.008
^{109}Ag	47	62	Stable	108.904 755 8(14)	48.161 ± 0.008

Input

```
#import "@preview/pillar:0.3.3"
#set text(font: "Libertinus Serif")
#show table.cell.where(y: 0): strong
#rowtable(
  separator: ",",
  table: pillar.table,
  cols: "rCCCCC",
  format: (auto, ) * 4 + ((uncertainty-mode: "compact"), auto),
  column-gutter: (0.5em, 0pt),
  stroke: (x, y) => if y == 0 { (bottom: 0.75pt) },
  table.header(
  [Isotope,       Z,  N,  Half-life,  Mass (Da),        Abundance (%) ]),
  [#super[107]Ag, 47, 60, Stable,     106.9050915(26),  51.839(8) ],
  [#super[109]Ag, 47, 62, Stable,     108.9047558(14),  48.161(8) ],
)
```

2.5 Table Cells, `rowspan` and `colspan`

- `colspan`: a cell spans multiple columns
- `rowspan`: a cell spans multiple rows

Table cells can be customized with the usual properties (`stroke`, `fill`, et.c.) but also with colspan and rowspan. It's important to include the cells inline inside the rows, i.e like this:

Document Result

1	2	3	4
A	B	Extra Wide	

Input

```
#let cell = table.cell
#rowtable(
  separator: ",",
  column-width: 3em, rows: 3em,
  [1, 2, 3, #cell(fill: yellow)[4]],
  [A, B, #cell(colspan: 2, stroke: 2pt)[Extra Wide]],
)
```

where `[...]` is the markup for the whole row. Then automatic row length computations will continue to work correctly.

The column span is straightforward, because it's contained in the row, but there is also support for `rowspan`, seen below. The rows that follow take the `rowspan`-reserved space into account when computing their effective length.

Document Result

Rs2, Cs2		1	2	**Rs3**	4
		expandc.			**Rs3**
e	f	g	h		
expandcell				ijk	

Input

```
#show table.cell: it => if it.colspan + it.rowspan > 2 { strong(it) } else { it }
#let cell = table.cell
#rowtable(
  separator: ",",
  column-width: 2.5em, rows: 2.5em,
  inset: 0.25em,
  [#cell(rowspan: 2, colspan: 2)[Rs2, Cs2], 1, 2, #cell(rowspan: 3)[Rs3], 4],
  [#expandcell[expandc.], #cell(rowspan: 3)[Rs3]],
  [e, f, g, h],
  [#expandcell[expandcell], ijk],
)
```

2.6 The `row` Function

The `row` function can be used to transform or style a whole row at a time. For example, `row([a & b & c], map: strong)` styles the row using **`strong`**. The row function is entirely optional. It has three optional arguments:

- `map` - apply a function to each element
- `imap` - apply a funtion to each element, and also receive its index
- `cell` - set a dictionary of cell properties, e.g. `fill` and `stroke` on the whole row.

It is recommended to use the style `row([...], map: ..)` with the row markup as the first argument so that rows align well to each other.

Example: Use `map` or `imap` to transform or style the cells of a row.

Document Result

0	1	2	3	4	5
α^0	β^1	γ^2	δ^3	ε^4	ζ^5
a	b	c	d	e	f

Input

```
#import "@preview/rowmantic:0.5.0": rowtable, row
/// Draw cell with a color between `from` and `to`
#let colorcells(index, elt, from: yellow, to: red, step: 20%) = {
  let fr = index * step
  let fill = color.mix((to, fr), (from, 100% - fr)).lighten(50%)
  table.cell(fill: fill, elt)
}
#rowtable(
  separator: ",", align: center, inset: 0.7em,
  row([0,     1,    2,     3,     4,       5   ], map: strong),
  row($alpha, beta, gamma, delta, epsilon, zeta$, imap: (idx, elt) => $elt^idx$),
  row([a,     b,    c,     d,     e,       f   ], imap: colorcells),
)
```

Example: Use `map` to highlight empty entries, and `cell` to set background color.

Document Result

A		C		E	F
G	H		J	K	

Input

```
/// Draw cell with thick stroke if not empty, and red background if empty
#let markempty(elt) = {
  if elt != [] and elt != none {
    table.cell(stroke: 2pt, elt)
  } else {
    table.cell(fill: red.lighten(80%), elt)
  }
}
#rowtable(
  separator: "&", align: center, inset: 0.7em,
  row([ A &   & C &   & E & F ], map: markempty),
  row([ G & H &   & J & K &   ], map: markempty),
  row([ ~ & ], cell: (fill: gray)),
)
```

2.7 Equations

Equations can be treated as rows by themselves, and they split on `&` by default (but it is configurable). Note that `&` can be escaped in equations, but other separator symbols are not as easy to escape[3].

Document Result

A	B	C	D	E
1	x	x^2	$\sum_i^n$	inline equation row
1	x	x^2	$\sum_i^n$	block equation row
1	x	x^2	$\sum_i^n$	markup with embedded equations row

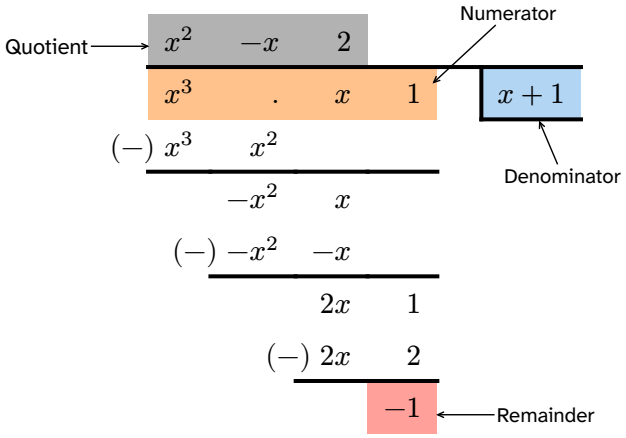
Input

```
#rowtable(
  align: center + horizon,
  stroke: (x, y) => (bottom: int(y == 0) * 1pt),
  separator: ",",      // regular sep
  separator-eq: $&$,  // equation sep
  [A, B, C, D, E],
  $1    & x   & x^2   & sum_i^n & #[inline equation row]$,
  $ 1   & x   & x^2   & sum_i^n & #[block equation row] $,
  [$1$, $x$,  $x^2$,  $ sum_i^n $, markup with embedded \ equations row],
)
```

2.7.1 Long Division

This example shows one way to set up long division. In this case it's polynomial division, computing the result $x^3 + x + 1 = (x^2 - x + 2)(x + 1) - 1$. For this example, the colorful annotations add the most of the complexity. `rowtable` contributes to the example by splitting equations on the separator and filling rows to be equal length.

Document Result



[3]The escape for `&` is just `\&`, for other separators like for example the comma a `box` or `","` is used to escape them.

Input

```
#import "@preview/mannot:0.3.0": annot, mark

/// Set up strokes and gutter for long division table
#let longdiv(..args, grid: std.grid) = {
  let cols = args.at("columns")
  let st = std.stroke(args.at("stroke", default: black))
  let stroke = (x, y) => (
    left: if x == cols - 1 and y == 1 { st },
    bottom: if (
      // Add top and bottom stroke to denominator cell
      y == 1 and x == cols - 1
      // Add bottom stroke every two rows (calc.even check),
      // but for one less column each time
      or y >= 1 and x < cols - 1 and calc.even(y) and x + 1 >= y / 2
    ) {
      st
    }
  )
  grid(..args,
    column-gutter: (0pt,) * (cols - 2) + (1.5em, ),
    stroke: stroke,
    std.grid.hline(y: 1, stroke: st))
}

// Set up marking functions and grid cell backgrounds
#let mark = mark.with(outset: (top: 0em, rest: 0.50em))
#let mkq(..args) = grid.cell(fill: luma(70%), mark(..args))
#let mkn(..args) = grid.cell(fill: orange.lighten(50%), mark(..args))
#let mkdenom(it) = grid.cell(fill: blue.lighten(70%), mark(tag: <denom>, it))
#let mkrem(it)   = grid.cell(fill: red.lighten(50%), mark(tag: <rem>, it))

#let um = math.class("unary", math.minus) // unary minus
#let leftset(x) = box(place(dx: -0.3em, right + bottom, $#x$))
#let rm = math.class("unary", leftset($(-)$)) // row minus

#show: block.with(breakable: false)
#rowgrid(
  align: right,
  table: longdiv.with(stroke: 1.5pt),
  inset: 0.55em,
  $mkq(x^2, tag: #<ans>)& mkq(um x) & mkq(2)    &$,
  $mkn(x^3) & mkn(.)    & mkn(x)    & mkn(1, tag: #<num>) & mkdenom(x + 1)$,
  $rm x^3   & x^2$,

  $         &-x^2       &  x$,
  $         & rm -x^2   & -x$,

  $         &           &   2x  & 1$,
  $         &           &rm 2x  & 2$,

  $         &           &       & mkrem(um 1)$,
)
#annot(<ans>, pos: left + horizon, dx: -2em, dy: -0em, annot-text-props: (fill:
black, size: 0.75em))[Quotient]
#annot(<denom>, pos: right + bottom, dy: 1.5em, dx: -1.0em)[Denominator]
#annot(<num>, pos: right + top, dy: -2.0em, dx: 1.5em)[Numerator]
#annot(<rem>, pos: right + horizon, dy: 0em, dx: 2em)[Remainder]
```

2.7.2 Equations with Alignment

Use a different separator than `&` to use equations with alignment.

Document Result

Matrix Form	Equation System Form
$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$	$(1\text{st})\ y_1 = a_{11}x_1 + a_{12}x_2$ $(2\text{nd})\ y_2 = a_{21}x_1 + a_{22}x_2$

Input

```
#set math.mat(delim: "[")
#rowtable(
  separator: ";",
  stroke: (x, y) => (bottom: int(y == 0) * 1pt),
  [Matrix Form; Equation System Form],
  $
    mat(y_1; y_2) = mat(a_11, a_12; a_21, a_22) mat(x_1; x_2)
    ;
    "(1st)" y_1 &= a_11 x_1 &+ a_12 x_2 \
    "(2nd)" y_2 &= a_21 x_1 &+ a_22 x_2 \
  $
)
```

2.7.3 Sizing Delimiters in Annotated Matrix

This example draws a matrix using `rowgrid` and inserts iteratively sized delimiters in `rowspan` cells. Grid lines are drawn to show how the table is constructed.

Document Result

		(k)	(l)				
(a)	$\left[\right.$	n	$\sum_{i=1}^{n} x_i$	$\left.\right]$	$\left[\right.$	β_0	$\left.\right]$
(b)		$\sum_{i=1}^{n} x_i$	$\sum_{i=1}^{n} x_i^2$			β_1	

Input

```
/// Measure a grid's size
/// Successively remove display of each row
///  (first full table, then remove 0, then remove 0 and 1, and so on).
/// Returns an array of measurements which is a size profile of the grid's rows
/// (or columns if `attr: "x"`)
#let measure-grid(t, min: 0, max: 50, attr: "y") = {
  let wh = (x: "width", y: "height").at(attr)
  let result = ()
  for i in range(max) {
    let sz = measure({
      show grid.cell: it => {
        if it.at(attr) < i { none } else { it }
      }
```

```
      t
    })
    result.push(sz)
    if i > min and (sz.at(wh) == 0pt or result.at(-2, default: none) == sz) {
      break
    }
  }
  result
}

/// Create an element using `func`; then measure it using `measure-func`
/// then create the final version of the element again using func, this
/// time with the size info from the first try.
#let measure-and-make(func, measure-func: measure-grid) = context {
  func(measure-func(func(none)))
}

#measure-and-make(sizeinfo => {
  set text(size: 1.5em)
  let nrows = 2
  // use row height information from size profile with one row removed.
  let row = if sizeinfo != none { sizeinfo.at(1).height / nrows } else { 1em }
  let delimheight = row * 0.90 * nrows
  let delim(p) = grid.cell(rowspan: nrows, $lr(#p, size: #delimheight)$, inset:
0em)
  let open = delim([\[])
  let close = delim([\]])
  rowgrid(
    stroke: 0.10pt + blue,
    align: horizon,
    inset: 0.4em,
    $     &      & (k)     & (l) &&& $,
    $ (a) & open & n       & sum_(i=1)^n x_i   & close & open & beta_0 & close $,
    $ (b) &        sum_(i=1)^n x_i & sum_(i=1)^n x_i^2 &        beta_1 $,
  )
})
```

3 Known Limitations

Properties/Behaviours

- A single *styled* item is not treated as a row, like this: `[*One item*]`. Fix this by inserting a cell separator: `[*One item* & ]`.[4]
- Table cells can be passed outside the rows, and this messes up the row length calculations. Avoid doing this.

Unimplemented

- Multi-row `table.header/footer` are not supported yet.
- `rowspan` is not supported in rows inside `table.header/footer`.
- `expandcell` can collide with `rowspan`ned cells (if they are not placed along the left or right side of the table); use `colspan` as a workaround when necessary.
- `rowtable` does not properly support being used as a front end for `grid`.

[4] Only content sequences and text are treated as rows and split on separators. `[*item*]` ends up being a `strong` element.

4 Function Reference

4.1 rowtable

Table which takes table cell inputs in rows.

Each row is passed as one markup block `[...]` which is split internally on the separator. Rows that are shorter than the longest row (or the configured `columns`) will be filled to be the same length as all other rows.

Rows can also be passed as equations (`$...$`) and they are then split into cells by `separator-eq`.

Leading/trailing spaces are removed from each table element in rows. To preserve such spaces, use `~`.

This function wraps the standard `table` function and passes through all its regular arguments.

Passing `table.cell` outside rows is possible but not recommended. Passing `#table.cell[]` inside a row, between separators, is supported and can be used with `colspan` >= 1 and/or `rowspan >= 1`. Successive rows will take rowspans into account when computing their length.

It is supported to input rows inside `table.header` and `table.footer`.

Parameters

```
rowtable(
  separator: str,
  separator-eq: none auto equation,
  row-filler: none content,
  column-width: length relative fraction array,
  table: function,
  cell-function: function,
  ..args: arguments
) -> table content
```

separator `str`

Configurable cell separator in rows. Good choices are `"&"`, `","` or `";"`. Escape the separator using e.g. `\&`. As a special case `" "` will split on consecutive whitespace (one or more spaces, tabs or a newline). Additionally, using `"  "` (two spaces) will only split on

Default: `"&"`

separator-eq `none` or `auto` or `equation`

Cell separator for equations, must be single symbol. By default depends on `separator` if possible otherwise falls back to `$&$`. Set to `none` to disable splitting equations.

Default: `auto`

row-filler `none` or `content`

Value used to fill rows that are too short.

Default: `none`

column-width `length` or `relative` or `fraction` or `array`

Set column width without specifying number of columns. A single length is repeated for all columns. An array of lengths is repeated by extending with the last item.

Default: `none`

table `function`

Table function to use to build the final table. Intended for use with table wrappers from other packages. (The function `arguments` can be used for argument pass-through.)

Default: `std.table`

cell-function `function`

Cell function to use (either `table.cell` or `grid.cell`). You normally do not need to specify this argument.

Default: `auto`

..args `arguments`

Rows like `[A & B & C]` and other positional or named table function parameters. Arguments to `table` pass through. A `columns` argument to the table is possible but not mandatory.

4.2 rowgrid

Grid which takes grid cell inputs as rows.

The `rowgrid` function has exactly the same interface as the `rowtable` function; refer to it for full documentation. The only difference is the different defaults for the `table` and `cell-function` arguments. Since `rowgrid` forwards to the `grid` function, cells must use `grid.cell` and lines `grid.hline` and so on, when applicable.

The `rowgrid` function produces a `grid`.

Parameters

```
rowgrid(
  table: function,
  cell-function: function,
  ..args: arguments
)
```

table `function`

The table function to use to build the table.

Default: `std.grid`

cell-function `function`

Cell function to use (either `table.cell` or `grid.cell`). You normally do not need to specify this argument.

Default: `std.grid.cell`

..args `arguments`

`rowtable` and `grid` arguments. Refer to the `rowtable` documentation for full description of all arguments.

4.3 row-split

Take a sequence (content) and split it into an array by the given separator. It's split only shallowly, not deeply; the separators must exist in the uppermost sequence's content.

Parameters

```
row-split(
  it: content,
  sep: str,
  strip-space: bool
) -> array
```

it `content`

Text or sequence or other content

sep `str`

Separator

Default: `"&"`

strip-space `bool`

Remove leading/trailing spaces from split sequences

Default: `true`

4.4 expandcell

An expandcell is a `table.cell` that expands its `colspan` to available width. The expandcell can be passed alone as a whole row, or should be placed inside a row markup block to form part of a row.

Parameters

```
expandcell(
  body: content,
  ..args: arguments
) -> content
```

body `content`

Cell body

..args `arguments`

`table.cell` arguments, except `colspan` and `rowspan` which are not permitted.

4.5 row

Style a whole row at once

All arguments are optional. Only one of `map` or `imap` can be passed at the same time.

Cell properties are resolved in this order: 1. cell returned from map/imap, 2. cell properties from `cell`. 3. cell properties from the cell in the row.

Parameters

```
row(
  body,
  map: none function,
  imap: none function,
  cell: dictionary
)
```

map `none` or `function`

apply this function to the content of each cell, after resolving row lengths and padding rows. Passing cell content, signature: function(any) -> any.

Default: `none`

imap `none` or `function`

apply this function to the content of each cell, after resolving row lengths and padding rows. Passing index and cell content, signature: function(int, any) -> any. Note that this is just the index in the row, which does not correspond to the column number in complex layouts.

Default: `none`

cell `dictionary`

set these `table.cell` settings on each cell of the row, after resolving row lengths and padding rows. The properties `x`, `y`, `colspan`, `rowspan` are not allowed here.

Default: `(:)`

API Documentation generated using `tidy`.