rowmantic

A Typst package for row-wise table editing

version	0.2.0
import as	`#import "@preview/rowmantic:0.2.0"`
typst universe	https://typst.app/universe/package/rowmantic
repository	https://github.com/typst-community/rowmantic

1 Introduction

The idea is a row-oriented way to input tables, with just a little less syntactical overhead than 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 table sizes the number of columns by the longest row. All rows are effectively completed so that they are of full length. This creates a better the editing experience, as rows can be filled out gradually.

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

```
#{
  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**
$\mathrm{\beta}$	Model parameters	

Input

```
#{
  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 Difficult Examples

Document Result

Literal &	**Strong**	**X**–*Y*
Equation $\pi = 3.1415...$	$\int_\Omega d\omega$	$X\&Y$
π^1	π^2	π^3
• A • B	1. A 2. B	**A** a **B** b
Figure 1: Top $\boxed{\text{A}}$	See Figure 1 & Figure 2	$\boxed{\text{B}}$ Figure 2: Bot

Nested rowtable
$\boxed{\text{A}}\boxed{\text{B}}$
	Nested table
	$\boxed{\text{A}}\boxed{\text{B}}$
		`table.cell(`
		`rowspan: 2)`

Cell with colspan=2		
Expandcell		
N/A	N/A	N/A

Input

```
#rowtable(
  align: horizon,
  stroke: 0.1pt,
  row-filler: [N/A],
  [Literal \& & *Strong* & *X*--_Y_ ],
  [Equation \ $pi = 3.1415...$ & $ integral_Omega d omega $  & $X \& Y$],
  $ pi^1 & pi^2 & pi^3 $,
  [
    - A
    - B
    &
    + A
    + B
    &
    / A: a
    / B: b
  ],
  [
    #{
      set figure.caption(position: top)
      [#figure(rect[A], caption: "Top")<fig1>]
    }
    &
    See @fig1 \& @fig2
    &
    #figure(rect[B], caption: "Bot")<fig2>
  ],
  {
    [Nested rowtable \ ]
    rowtable([A & B])
    [&]
    [Nested table \ ]
    table(columns: 2, [A], [B])
    [&]
    table.cell(stroke: 1pt + red, rowspan: 2)[`table.cell(` \ `rowspan: 2)`]
  },
  [#table.cell(fill: yellow.lighten(90%), colspan: 2)[Cell with colspan=2] ],
  [#expandcell(fill: yellow.lighten(90%))[Expandcell] & #expandcell[#none]],
  table.footer([]),
)
```

2.3 Double semicolon separator

Document Result

First	This is a literal ;; and ; and , and &
Second; Third	Equation $\pi = 3.1415...$

Input

```
#rowtable(
  separator: ";;",
  stroke: 0.5pt,
  [First           ;; This is a literal \;\; and ; and , and & ],
  [Second; Third   ;; Equation $pi = 3.1415...$],
)
```

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:0.3.2"
#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 Equations

Equations as rows are are 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	$\displaystyle\sum_i^n$	block equation row
1	x	x^2	$\displaystyle\sum_i^n$	markup with embedded equations row

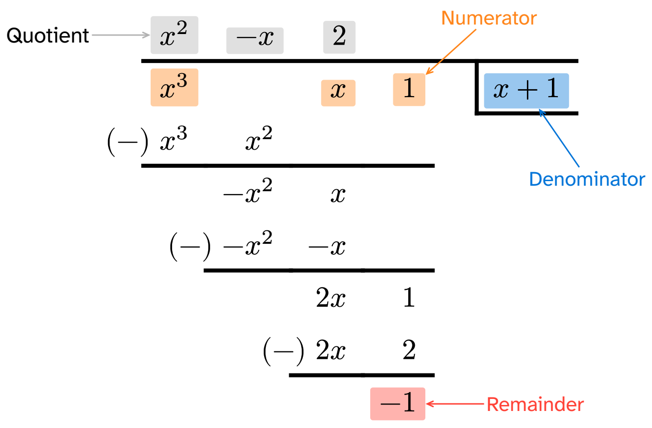
Input

```
#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],
)
```

2.5.1 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



Input

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

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

/// Set up strokes and gutter for long division table
#let longdiv(..args, table: std.table) = {
  let cols = args.at("columns")
  let st = std.stroke(args.at("stroke", default: black))
  let stroke = (x, y) => (
     // Add left stroke to the last column
    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
    }
  )
  table(..args,
    column-gutter: (0pt,) * (cols - 2) + (1.5em, ),
    stroke: stroke,
    std.table.hline(y: 1, stroke: st))
}

// Set up marking functions
#let markhl = markhl.with(outset: (top: 0.15em, rest: 0.30em), radius: 1pt)
#let um = math.class("unary", math.minus) // unary minus
#let mkq = markhl.with(color: luma(70%))
#let mkn = markhl.with()
#let mkdenom = markhl.with(color: blue, tag: <denom>)
#let mkrem = markhl.with(color: red, tag: <rem>)

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

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

  $         &-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: +2em)[Denominator]
#annot(<num>, pos: right + top, dy: -1.5em, dx: 1em)[Numerator]
#annot(<rem>, pos: right + horizon, dy: 0em, dx: 2em)[Remainder]
```

2.5.2 Equations with alignment

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

Document Result

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

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 \
  $
)
```

2.6 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

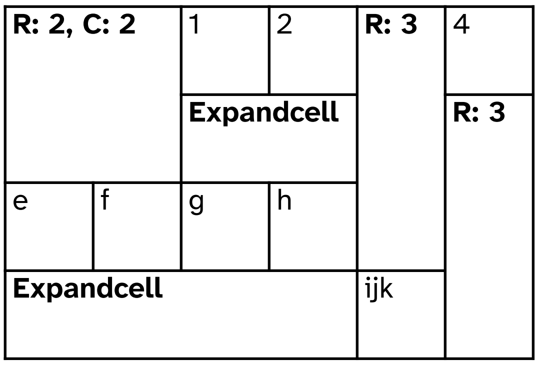
```
#let cell = table.cell
#rowtable(
  separator: ",",
  column-width: 3em,
  rows: 3em,
  [1, 2, 3,      #cell(fill: yellow)[4]],
  [A, B, #cell(colspan: 2, stroke: 2pt)[Extra Wide]],
)
```

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, so support for this was available from `rowmantic`'s first version.

However, there is also support for `rowspan` since version 0.2.0:

Document Result



R: 2, C: 2		1	2	**R: 3**	4
		Expandcell			**R: 3**
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: 3em, rows: 3em,
  inset: 0.25em,
  [#cell(rowspan: 2, colspan: 2)[R: 2, C: 2], 1, 2, #cell(rowspan: 3)[R: 3], 4],
  [#expandcell[Expandcell], #cell(rowspan: 3)[R: 3]],
  [e, f, g, h],
  [#expandcell[Expandcell], ijk],
)
```

3 Known Limitations

- Multi-row `table.header/footer` are not supported yet.
- `rowspan` is not supported in cells 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.
- Table cells can be passed outside the rows, and this removes the usefulness of automatic column lengths. Avoid doing this.
- `rowtable` does not properly support being used as a “front end” for `grid`.

4 Function Reference

4.1 expandcell

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

Parameters

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

body `content`

cell body

..args `arguments`

`table.cell` arguments. colspan and rowspan are not permitted.

4.2 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`

4.3 rowtable

Table which takes table cell inputs in rows.

Each row is passed as one markup block (`{[...]}` syntax) 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 a row. 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: any ,
  column-width: length  relative  array ,
  table: function ,
  ..args: arguments
)
```

separator `str`

configurable cell separator in a row. Good choices are `&`, `,`, or `;`. Escape the separator using e.g. `[\&]`

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 `any`

value used to fill rows that are too short

Default: `none`

column-width `length` or `relative` 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`

..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.

API Documentation generated using `tidy`.