Tabut

Powerful, Simple, Concise

A Typst plugin for turning data into tables.

Outline

Examples

Input Format and Creation

The `tabut` function takes input in "record" format, an array of dictionaries, with each dictionary representing a single "object" or "record".

In the example below, each record is a listing for an office supply product.

```
#let supplies = (
  (name: "Notebook", price: 3.49, quantity: 5),
  (name: "Ballpoint Pens", price: 5.99, quantity: 2),
  (name: "Printer Paper", price: 6.99, quantity: 3),
)
```

Basic Table

Now create a basic table from the data.

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#tabut(
  supplies, // the source of the data used to generate the table
  ( // column definitions
    (
      header: [Name], // label, takes content.
      func: r => r.name // generates the cell content.
    ),
    (header: [Price], func: r => r.price),
    (header: [Quantity], func: r => r.quantity),
  )
)
```

Name	Price	Quantity
Notebook	3.49	5
Ballpoint Pens	5.99	2
Printer Paper	6.99	3

`funct` takes a function which generates content for a given cell corrosponding to the defined column for each record. `r` is the record, so `r => r.name` returns the `name` property of each record in the input data if it has one.

The philosphy of `tabut` is that the display of data should be simple and clearly defined, therefore each column and it's content and formatting should be defined within a single clear column defintion. One consequence is you can comment out, remove or move, any column easily, for example:

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [Price], func: r => r.price), // This column is moved to the front
    (header: [Name], func: r => r.name),
    (header: [Name 2], func: r => r.name), // copied
    // (header: [Quantity], func: r => r.quantity), // removed via comment
  )
)
```

Price	Name	Name 2
3.49	Notebook	Notebook
5.99	Ballpoint Pens	Ballpoint Pens
6.99	Printer Paper	Printer Paper

Table Styling

Any default Table style options can be tacked on and are passed to the final table function.

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [Name], func: r => r.name),
    (header: [Price], func: r => r.price),
    (header: [Quantity], func: r => r.quantity),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Name	Price	Quantity
Notebook	3.49	5
Ballpoint Pens	5.99	2
Printer Paper	6.99	3

Header Formatting

You can pass any content or expression into the header property.

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#let fmt(it) = {
  heading(
    outlined: false,
    upper(it)
  )
}

#tabut(
  supplies,
  (
    (header: fmt([Name]), func: r => r.name ),
    (header: fmt([Price]), func: r => r.price),
    (header: fmt([Quantity]), func: r => r.quantity),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

NAME	PRICE	QUANTITY
Notebook	3.49	5
Ballpoint Pens	5.99	2
Printer Paper	6.99	3

Remove Headers

You can prevent from being generated with the `headers` paramater. This is useful with the `tabut-cells` function as demonstrated in it's section.

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*Name*], func: r => r.name),
    (header: [*Price*], func: r => r.price),
    (header: [*Quantity*], func: r => r.quantity),
  ),
  headers: false, // Prevents Headers from being generated
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none,
)
```

Notebook	3.49	5
Ballpoint Pens	5.99	2
Printer Paper	6.99	3

Cell Expressions and Formatting

Just like the headers, cell contents can be modified and formatted like any content in Typst.

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*Name*], func: r => r.name ),
    (header: [*Price*], func: r => usd(r.price)),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Name	Price
Notebook	$3.49
Ballpoint Pens	$5.99
Printer Paper	$6.99

You can have the cell content function do calculations on a record property.

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*Name*], func: r => r.name ),
    (header: [*Price*], func: r => usd(r.price)),
    (header: [*Tax*], func: r => usd(r.price * .2)),
    (header: [*Total*], func: r => usd(r.price * 1.2)),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Name	Price	Tax	Total
Notebook	$3.49	$0.69	$4.18
Ballpoint Pens	$5.99	$1.19	$7.18
Printer Paper	$6.99	$1.39	$8.38

Or even combine multiple record properties, go wild.

```
#import "@preview/tabut:0.0.1": tabut

#let employees = (
    (id: 3251, first: "Alice", last: "Smith", middle: "Jane"),
    (id: 4872, first: "Carlos", last: "Garcia", middle: "Luis"),
    (id: 5639, first: "Evelyn", last: "Chen", middle: "Ming")
);

#tabut(
  employees,
  (
    (header: [*ID*], func: r => r.id ),
    (header: [*Full Name*], func: r => [#r.first #r.middle.first(), #r.last] ),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

ID	Full Name
3251	Alice J, Smith
4872	Carlos L, Garcia
5639	Evelyn M, Chen

Index

`tabut` automatically adds an `_index` property to each record.

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*\#*], func: r => r._index),
    (header: [*Name*], func: r => r.name ),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

#	Name
0	Notebook
1	Ballpoint Pens
2	Printer Paper

You can also prevent the `index` property being generated by setting it to `none`, or you can also set an alternate name of the index property as shown below.

```
#import "@preview/tabut:0.0.1": tabut
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*\#*], func: r => r.index-alt ),
    (header: [*Name*], func: r => r.name ),
  ),
  index: "index-alt", // set an aternate name for the automatically generated
index property.
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

#	Name
0	Notebook
1	Ballpoint Pens
2	Printer Paper

Transpose

This was annoying to implement, and I don't know when you'd actually use this, but here.

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*\#*], func: r => r._index),
    (header: [*Name*], func: r => r.name),
    (header: [*Price*], func: r => usd(r.price)),
    (header: [*Quantity*], func: r => r.quantity),
  ),
  transpose: true,  // set optional name arg `transpose` to `true`
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

#	0	1	2
Name	Notebook	Ballpoint Pens	Printer Paper
Price	$3.49	$5.99	$6.99
Quantity	5	2	3

Alignment

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  ( // Include `align` as an optional arg to a column def
    (header: [*\#*], func: r => r._index),
    (header: [*Name*], align: right, func: r => r.name),
    (header: [*Price*], align: right, func: r => usd(r.price)),
    (header: [*Quantity*], align: right, func: r => r.quantity),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

#	Name	Price	Quantity
0	Notebook	$3.49	5
1	Ballpoint Pens	$5.99	2
2	Printer Paper	$6.99	3

You can also define Alignment manually as in the the standard Table Function.

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#tabut(
  supplies,
  (
    (header: [*\#*], func: r => r._index),
    (header: [*Name*], func: r => r.name),
    (header: [*Price*], func: r => usd(r.price)),
    (header: [*Quantity*], func: r => r.quantity),
  ),
  align: (auto, right, right, right), // Alignment defined as in standard table
function
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

#	Name	Price	Quantity
0	Notebook	$3.49	5
1	Ballpoint Pens	$5.99	2
2	Printer Paper	$6.99	3

Column Width

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#box(
  width: 300pt,
  tabut(
    supplies,
    ( // Include `width` as an optional arg to a column def
      (header: [*\#*], func: r => r._index),
      (header: [*Name*], width: 1fr, func: r => r.name),
      (header: [*Price*], width: 20%, func: r => usd(r.price)),
      (header: [*Quantity*], width: 1.5in, func: r => r.quantity),
    ),
    fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
    stroke: none,
  )
)
```

#	Name	Price	Quantity
0	Notebook	$3.49	5
1	Ballpoint Pens	$5.99	2
2	Printer Paper	$6.99	3

You can also define Columns manually as in the the standard Table Function.

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#box(
  width: 300pt,
  tabut(
    supplies,
    (
      (header: [*\#*], func: r => r._index),
      (header: [*Name*], func: r => r.name),
      (header: [*Price*], func: r => usd(r.price)),
      (header: [*Quantity*], func: r => r.quantity),
    ),
    columns: (auto, 1fr, 20%, 1.5in),  // Columns defined as in standard table
    fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
    stroke: none,
  )
)
```

#	Name	Price	Quantity
0	Notebook	$3.49	5
1	Ballpoint Pens	$5.99	2
2	Printer Paper	$6.99	3

Get Cells Only

```
#import "@preview/tabut:0.0.1": tabut-cells
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#tabut-cells(
  supplies,
  (
    (header: [Name], func: r => r.name),
    (header: [Price], func: r => usd(r.price)),
    (header: [Quantity], func: r => r.quantity),
  )
)
```

```
(
  columns: (auto, auto, auto),
  align: (auto, auto, auto),
  [Name],
  [Price],
  [Quantity],
  [Notebook],
  styled(child: [([$], [3], [.], [49])], ..),
  [5],
  [Ballpoint Pens],
  styled(child: [([$], [5], [.], [99])], ..),
  [2],
  [Printer Paper],
  styled(child: [([$], [6], [.], [99])], ..),
  [3],
)
```

Use with Tablex

```
#import "@preview/tabut:0.0.1": tabut-cells
#import "usd.typ": usd
#import "example-data/supplies.typ": supplies

#import "@preview/tablex:0.0.8": tablex, rowspanx, colspanx

#tablex(
  auto-vlines: false,
  header-rows: 2,

  /* --- header --- */
  rowspanx(2)[*Name*], colspanx(2)[*Price*], (), rowspanx(2)[*Quantity*],
  (),                  [*Base*], [*W/Tax*], (),
  /* -------------- */

  ..tabut-cells(
    supplies,
    (
      (header: [], func: r => r.name),
      (header: [], func: r => usd(r.price)),
      (header: [], func: r => usd(r.price * 1.3)),
      (header: [], func: r => r.quantity),
    ),
    headers: false
  )
)
```

Name	Price		Quantity
	Base	W/Tax	
Notebook	$3.49	$4.53	5
Ballpoint Pens	$5.99	$7.78	2
Printer Paper	$6.99	$9.08	3

Data Operation Examples

While technically seperate from table display, the following are examples of how to perform operations on data before it is displayed with `tabut`.

Since `tabut` assumes an "array of dictionaries" format, then most data operations can be performed easily with Typst's native array functions. `tabut` also provides several functions to provide additional functionality.

CSV Data

By default, imported CSV gives a "rows" or "array of arrays" data format, which can not be directly used by `tabut`. To convert, `tabut` includes a function `rows-to-records` demonstrated below.

```
#import "@preview/tabut:0.0.1": tabut, rows-to-records
#import "example-data/supplies.typ": supplies

#let titanic = {
  let titanic-raw = csv("example-data/titanic.csv");
  rows-to-records(
    titanic-raw.first(), // The header row
    titanic-raw.slice(1, -1), // The rest of the rows
  )
}
```

Imported CSV data are all strings, so it's usefull to convert them to `int` or `float` when possible.

```
#import "@preview/tabut:0.0.1": tabut, rows-to-records
#import "example-data/supplies.typ": supplies

#let auto-type(input) = {
  let is-int = (input.match(regex("^-?\d+$")) != none);
  if is-int { return int(input); }
  let is-float = (input.match(regex("^-?(inf|nan|\d+|\d*(\.\d+))$")) != none);
  if is-float { return float(input) }
  input
}

#let titanic = {
  let titanic-raw = csv("example-data/titanic.csv");
  rows-to-records( titanic-raw.first(), titanic-raw.slice(1, -1) )
  .map( r => {
    let new-record = (:);
    for (k, v) in r.pairs() { new-record.insert(k, auto-type(v)); }
    new-record
  })
}
```

`tabut` includes a function, `records-from-csv`, to automatically perform this process.

```
#import "@preview/tabut:0.0.1": records-from-csv

#let titanic = records-from-csv("doc/example-snippets/example-data/titanic.csv");
```

Slice

```
#import "@preview/tabut:0.0.1": tabut, records-from-csv
#import "usd.typ": usd
#import "example-data/titanic.typ": titanic

#let classes = (
  "N/A",
  "First",
  "Second",
  "Third"
);

#let titanic-head = titanic.slice(0, 5);

#tabut(
  titanic-head,
  (
    (header: [*Name*], func: r => r.Name),
    (header: [*Class*], func: r => classes.at(r.Pclass)),
    (header: [*Fare*], func: r => usd(r.Fare)),
    (header: [*Survived?*], func: r => ("No", "Yes").at(r.Survived)),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Name	Class	Fare	Survived?
Mr. Owen Harris Braund	Third	$7.25	No
Mrs. John Bradley (Florence Briggs Thayer) Cumings	First	$71.28	Yes
Miss. Laina Heikkinen	Third	$7.92	Yes
Mrs. Jacques Heath (Lily May Peel) Futrelle	First	$53.10	Yes
Mr. William Henry Allen	Third	$8.05	No

Sorting and Reversing

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/titanic.typ": titanic, classes

#tabut(
  titanic
  .sorted(key: r => r.Fare)
  .rev()
  .slice(0, 5),
  (
    (header: [*Name*], func: r => r.Name),
    (header: [*Class*], func: r => classes.at(r.Pclass)),
    (header: [*Fare*], func: r => usd(r.Fare)),
    (header: [*Survived?*], func: r => ("No", "Yes").at(r.Survived)),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Name	Class	Fare	Survived?
Mr. Gustave J Lesurer	First	$512.32	Yes
Mr. Thomas Drake Martinez Cardeza	First	$512.32	Yes
Miss. Anna Ward	First	$512.32	Yes
Mr. Mark Fortune	First	$263.00	No
Miss. Alice Elizabeth Fortune	First	$263.00	Yes

Filter

```
#import "@preview/tabut:0.0.1": tabut
#import "usd.typ": usd
#import "example-data/titanic.typ": titanic, classes

#tabut(
  titanic
  .filter(r => r.Pclass == 1)
  .slice(0, 5),
  (
    (header: [*Name*], func: r => r.Name),
    (header: [*Class*], func: r => classes.at(r.Pclass)),
    (header: [*Fare*], func: r => usd(r.Fare)),
    (header: [*Survived?*], func: r => ("No", "Yes").at(r.Survived)),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Name	Class	Fare	Survived?
Mrs. John Bradley (Florence Briggs Thayer) Cumings	First	$71.28	Yes
Mrs. Jacques Heath (Lily May Peel) Futrelle	First	$53.10	Yes
Mr. Timothy J McCarthy	First	$51.86	No
Miss. Elizabeth Bonnell	First	$26.55	Yes
Mr. William Thompson Sloper	First	$35.50	Yes

Aggregation using Map and Sum

```
#import "usd.typ": usd
#import "example-data/titanic.typ": titanic, classes

#table(
  columns: (auto, auto),
  [*Fare, Total:*], [#usd(titanic.map(r => r.Fare).sum())],
  [*Fare, Avg:*], [#usd(titanic.map(r => r.Fare).sum() / titanic.len())],
  stroke: none
)
```

Fare, Total:	$28,647.15
Fare, Avg:	$32.33

Grouping

```
#import "@preview/tabut:0.0.1": tabut, group
#import "example-data/titanic.typ": titanic, classes

#tabut(
  group(titanic, r => r.Pclass),
  (
    (header: [*Class*], func: r => classes.at(r.value)),
    (header: [*Passengers*], func: r => r.group.len()),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Class	Passengers
First	216
Second	184
Third	486

```
#import "@preview/tabut:0.0.1": tabut, group
#import "usd.typ": usd
#import "example-data/titanic.typ": titanic, classes

#tabut(
  group(titanic, r => r.Pclass),
  (
    (header: [*Class*], func: r => classes.at(r.value)),
    (header: [*Total Fare*], func: r => usd(r.group.map(r => r.Fare).sum())),
    (
      header: [*Avg Fare*],
      func: r => usd(r.group.map(r => r.Fare).sum() / r.group.len())
    ),
  ),
  fill: (_, row) => if calc.odd(row) { luma(240) } else { luma(220) },
  stroke: none
)
```

Class	Total Fare	Avg Fare
First	$18,177.41	$84.15
Second	$3,801.84	$20.66
Third	$6,667.90	$13.71