Package ‘tables’

July 22, 2025

Title Formula-Driven Table Generation

Version 0.9.31

Description Computes and displays complex tables of summary statistics.
Output may be in LaTeX, HTML, plain text, or an R
matrix for further processing.

Maintainer Duncan Murdoch <murdoch.duncan@gmail.com>

License GPL-2

Depends R (>= 2.12.0)

Imports stats, utils, knitr, htmltools

Suggests magrittr, kableExtra (>= 0.9.0), Hmisc, bookdown, rmarkdown,
pkgdown, formatters, tinytable (>= 0.0.5)

VignetteBuilder rmarkdown

URL https://dmurdoch.github.io/tables/

BugReports https://github.com/dmurdoch/tables/issues

SystemRequirements pandoc (>= 1.12.3) for vignettes

NeedsCompilation no

Author Duncan Murdoch [aut, cre]

Repository CRAN

Date/Publication 2024-08-29 14:10:02 UTC

Contents

All *Include all columns of a dataframe.*

Description

This constructs a formula object for all the columns of a dataframe.

Usage

```
All(df, numeric=TRUE, character=FALSE, logical=FALSE, factor=FALSE,
        complex=FALSE, raw=FALSE, other=FALSE,
        texify=getOption("tables.texify", FALSE))
```

Arguments

`df` The dataframe in which to find the columns.

`numeric`, `character`, `logical`, `factor`, `complex`, `raw`
Whether to include columns of specified types. See the Details below.

`other` Whether to include columns that match none of the previous types.

`texify` Whether to escape LaTeX special characters in column names.

Details

This function constructs a formula from the columns of a dataframe. By default, only numeric columns are included. The arguments `numeric`, `character`, `logical`, `factor`, `complex` and `raw` control the inclusion of columns of the corresponding types. The argument `other` controls inclusion of any other columns.

If these arguments are `TRUE`, such columns will be included in the formula.

If a function (or the name of a function given as a character string) is passed, such columns will be transformed by the function before inclusion. For example, `All(df, factor=as.character)` will convert all factor columns into their character representation for inclusion.

In other cases, the columns will be skipped.

Value

Language to insert into the table formula to achieve the desired table.

Examples

```
# Show mean and sd of all numeric columns in the iris data
tabular( Species  ~
       All(iris)*(mean + sd), data=iris )
```

`AllObs` *Display all observations in a table.*

Description

These functions generate the code for a `tabular` table to include all observations in a dataset, possibly divided up according to other factors.

Usage

```
AllObs(data = NULL, show = FALSE, label = "Obsn.", within = NULL)
RowNum(within = NULL, perrow = 5, show = FALSE, label = "Row", data = NULL)
```

Arguments

`data` The full dataset, used only to find the number of observations.

`show` Whether to show the observation number or row number in the table.

`label` The label to use when `show = TRUE`.

`within` A factor or list of factors by which to break up the observations.

`perrow` How many observations per row when `RowNum` is used in the row specification, or per column when it is part of the column specification.

Details

AllObs is used to display all of the observations in a dataset. It generates a (usually undisplayed) factor with a different level for each observation, sets a function to display the value, and calls DropEmpty to suppress display of empty rows, columns or cells.

If the within argument is specified in AllObs, the factor levels are restarted within each grouping. (within is interpreted as the INDEX argument of tapply, with one exception described below.) This may be useful when displaying the observation number, and is definitely useful if AllObs is used as a column specification in the table. It will also save some computation time if the table is very large, since fewer factor levels will be generated and later dropped.

RowNum is unlikely to be useful in a table by itself, but is helpful when displaying large datasets with AllObs. It allows a large number of observations to be broken into several rows and columns.

Because RowNum affects both rows and columns, its use is somewhat unusual. Normally it should be called before calling tabular, and its result saved in a variable. That variable (e.g. rownum) is used in the row specification for the table wrapped in I(), and in the column specification of the table in the within argument to AllObs. (This is the exception mentioned above.)

Despite its name, RowNum can be used as a column specifier, if you'd prefer column-major ordering of the values displayed in the table.

Value

Both AllObs and RowNum return language objects to be used on tabular formulas.

See Also

tabular, DropEmpty

Examples

```
tabular(Factor(cyl)*Factor(gear)*AllObs(mtcars) ~
               rownames(mtcars) + mpg, data=mtcars)

rownum <- with(mtcars, RowNum(list(cyl, gear)))
tabular(Factor(cyl)*Factor(gear)*I(rownum) ~
        mpg * AllObs(mtcars, within = list(cyl, gear, rownum)),
        data=mtcars)
```

Arguments   Arguments *pseudo-function*

Description

The Arguments pseudo-function enables the use of analysis functions that take multiple arguments.

Usage

```
Arguments(...)
```

Arguments

...	Arguments to pass to the analysis function.

Details

The arguments to Arguments are evaluated in full, then those which are length n are subsetted for each cell in the table.

If no analysis variable has been specified, but Arguments() has been, then the analysis function will be called with arguments matching those given in .... If an analysis variable was specified, it will be inserted as an unnamed first argument to the analysis function.

The Arguments() entry will not create a heading.

Only one Arguments() specification may be active in any term in the tabular formula.

Pseudo-functions

This is a "pseudo-function": it takes the form of a function call, but is never actually called: it is handled specially by tabular.

See Also

Percent for a different way to specify a multiple argument analysis function.

Examples

```
# This is the example from the weighted.mean help page
wt <- c(5,  5,  4,  1)/15
x <- c(3.7,3.3,3.5,2.8)
gp <- c(1,1,2,2)
tabular( (Factor(gp) + 1)
  ~ weighted.mean*x*Arguments(w = wt) )
```

---

as.matrix.tabular *Convert tabular object to matrix*

Description

Convert a tabular object to a matrix of the strings that would print, or a matrix of values.

Usage

```
## S3 method for class 'tabular'
as.matrix(x, format = TRUE,
    rowLabels = TRUE, colLabels = TRUE, justification = "n", ...)
```

Arguments

`x` A `"tabular"` object.

`format` How to format; see Details below.

`rowLabels, colLabels`
Whether to include the row or column labels; only used if `format = TRUE`.

`justification` How to justify values; only used if `format = TRUE`.

`...` Other parameters to pass to `format.tabular`.

Details

If `format=TRUE`, then a matrix of formatted strings is produced. If not, then the `format` argument is assumed to be a function (or name of a function passed as a character vector) to convert the list-mode matrix to another mode, e.g. `as.numeric`.

Value

A matrix.

Examples

```
table <-
    tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )

print(table)
as.matrix(table)
as.matrix(table, format=as.numeric)
```

`as.tabular` *Convert matrix or dataframe to tabular object.*

Description

These functions construct or copy labels onto an existing matrix or dataframe.

Usage

```
as.tabular(x, like = NULL)
## Default S3 method:
as.tabular(x, like = NULL)
## S3 method for class 'data.frame'
as.tabular(x, like = NULL)
```

Arguments

`x` The object to convert.

`like` If not `NULL`, should be a tabular object with the same number of rows and columns as `x`. Its labels will be used on the result.

Value

A `tabular` object.

See Also

`as.matrix.tabular`

Examples

```
model <- tabular( (Species + 1) ~ (n=1) + Sepal.Length + Sepal.Width, data=iris )
model
as.tabular(matrix(1:12, 4,3), like=model)
```

`DropEmpty` — `DropEmpty` *pseudo-function*

Description

Pseudo-function to indicate that rows or columns containing no observations should be dropped.

Usage

```
DropEmpty(empty = "", which = c("row", "col", "cell"))
```

Arguments

`empty`	String to use in empty cells.
`which`	A vector indicating what should be dropped. See the Details below.

Details

If the `which` argument contains `"row"`, then any row in the table in which all cells are empty will be dropped. Similarly, if it contains `"col"`, empty columns will be dropped. If it contains `"cell"`, then cells in rows and columns that are not dropped will be set to the `empty` string.

Pseudo-functions

This is a “pseudo-function”: it takes the form of a function call, but is never actually called: it is handled specially by `tabular`.

Examples

```
df <- data.frame(row = factor(1:10), value = rnorm(10))
subset <- df[sample(10, 5),, drop = FALSE]

# Some rows did not get selected, so this looks ugly
tabular(row ~ value*mean, data = subset)

# This only shows rows with data in them
tabular(row*DropEmpty() ~ value*mean, data = subset)

# This shows empty cells as "(empty)"
tabular(row*DropEmpty("(empty)", "cell") ~ value*mean, data = subset)
```

Format   Format *pseudo-function*

Description

Format controls the formatting of the cells it applies to. .Format is mainly for internal use.

Usage

```
Format(...)
.Format(n)
```

Arguments

...	Arguments to pass to a formatting function, or a call to a formatting function.
n	A format number.

Details

The Format pseudo-function changes the formatting of table cells, and it specifies that all values it applies to will be formatted together.

In the first form, the “call” to Format looks like a call to format, but without specifying the argument x. When tabular() formats the output it will construct x from the entries in the table governed by the Format() specification, and pass it to the standard format function along with the other arguments.

In the second form, the “call” to Format contains a call to a function to do the formatting. Again, an argument x will be added to the call, containing the values to be formatted.

In the first form, or if the explicit function is named format, any cells in the table with character values will not be formatted. This is done so that a column can have mixed numeric and character values, and the numerics are not converted to character before formatting.

The pseudo-function .Format is mainly intended for internal use. It takes a single integer argument, saying that data governed by this call uses the same formatting as the format specification indicated by the integer. In this way entries can be commonly formatted even when they are not contiguous. The integers are assigned sequentially as the format specification is parsed; users will likely need trial and error to find the right value in a complicated table with multiple formats.

Pseudo-functions

This is a “pseudo-function”: it takes the form of a function call, but is never actually called: it is handled specially by `tabular`.

Examples

```
# Using the first form
tabular( (Sepal.Length+Sepal.Width) ~
         Format(digits=2)*(mean + sd), data=iris )

# The same table, using the second form
tabular( (Sepal.Length+Sepal.Width) ~
         Format(format(digits=2))*(mean + sd), data=iris )
```

`Heading`   `Heading` *pseudo-function*

Description

The `Heading` pseudo-function normally overrides the automatic heading on the following items in a table. Setting `override=FALSE` is used in automatically generated expressions.

Usage

```
Heading(name = NULL, override = TRUE, character.only = FALSE, nearData = TRUE)
```

Arguments

`name` A legal R variable name, or a character constant.

`override` Whether this heading should override one that is already present.

`character.only` If `TRUE`, the `name` argument will be interpreted as an expression evaluating to a character value.

`nearData` See Details below.

Details

This replaces the automatic heading or row label on the following item with the `name` or string as specified. If no argument is given, the heading or label is suppressed.

An alternative form of `Heading(name)` is `(name=...)`, where `...` is an expression to be displayed in the table.

If `override` = `FALSE`, the label is only supplied if there is no other label. This is used in the code for `Factor`.

The `nearData` argument is rarely used. It affects the situation where `"+"` is used to join tables with different numbers of labels. If `nearData` = `TRUE` (the default), the shorter list of labels are pushed close to the data, i.e. to the right side for row labels, the bottom for column labels. If `FALSE`, they are pushed to the opposite side.

Pseudo-functions

This is a “pseudo-function”: it takes the form of a function call, but is never actually called: it is handled specially by tabular.

Examples

```
tabular( (Sepal.Length+Sepal.Width) ~
         (Heading(Mean)*mean + (S.D.=sd)), data=iris )

heading <- "Variable Heading"
tabular( (Sepal.Length+Sepal.Width) ~
         (Heading(heading, character.only = TRUE)*mean + (S.D.=sd)),
         data=iris )
```

Hline *Add a horizontal line to a LaTeX table.*

Description

This function inserts a LaTeX directive to draw a full or partial line in a table.

Usage

```
Hline(columns, nearData = FALSE)
```

Arguments

columns	Which columns should receive the line?
nearData	See the Details section of Heading.

Details

Hline() is not very flexible: it must be the leftmost header in a row specification for the table, i.e. mean * Hline() is not allowed. Anything to the right of the Hline() factor will be ignored.

Value

Produces an expression to insert a label which will be interpreted by LaTeX as a request for a horizontal line.

Examples

```
toLatex( tabular( Species + Hline() + 1 ~ mean*Sepal.Width, data=iris) )
```

html.tabular *Display a tabular object using HTML.*

Description

This is similar to print.tabular, but it inserts the code to display the table in an HTML table.

Usage

```
toHTML(object, file = "", options = NULL,
                        id = NULL, append = FALSE,
                        browsable = TRUE, ...)
html.tabular(object, ...)
writeCSS(CSS = htmloptions()$CSS, id = NULL)
```

Arguments

object	The tabular object.
file	A filename or connection to which to write the HTML code, or "" to write to the console.
options	A list of options to set for the duration of the call.
id	A unique identifier to set for this table and the associated CSS style, or NULL, for no id.
append	If TRUE, opens file for appending (if it is a filename rather than a connection).
browsable	Should the output be marked as browsable?
...	Settings for default formatting. See Details below.
CSS	A character vector to use as CSS.

Details

The toHTML() function produces HTML output suitable for inclusion in an HTML page.

The html.tabular function is set up to work as a method for the html generic in **Hmisc**, but is not registered as a method, so that **tables** can work when **Hmisc** is not installed.

In HTML, it is mainly the CSS style sheet that determines the look of the table. When formatting a table, html.tabular sets the CSS class according to the table's Justify setting; justifications of c("l", "c", "r") are translated to classes c("left", "center", "right") respectively; other strings will be passed through and used directly as class names. If the id value is not NULL, then it will be used as the CSS id selector when searching for a style. See table_options for a number of options that control formatting, including the default style sheet.

Value

If file = "" (the default), the toHTML() function creates an HTML object using the htmltools::HTML function and returns it. If file is a character value or a connection, the results are written there and the HTML object is returned invisibly.

See Also

print.tabular, toLatex.tabular, html, htmloptions

Examples

```
X <- rnorm(125, sd=100)
Group <- factor(sample(letters[1:5], 125, replace = TRUE))

tab <- tabular( Group ~ (N=1)+Format(digits=2)*X*((Mean=mean) + Heading("Std Dev")*sd) )

save <- table_options()
table_options(rowlabeljustification="c")

f <- tempfile(fileext=".html")
con <- file(f, "wt")

if (interactive())
  toHTML(tab, con, options=htmloptions(head=TRUE, table=FALSE))

writeLines("<p>This table has pad = FALSE.  The centered numbers look
sloppy.<br>", con)

if (interactive())
  toHTML(tab, con, options=htmloptions(head=FALSE, table=TRUE, pad=FALSE))

writeLines("<p>This table has pad = FALSE and justification=\"r\".
The justification makes the columns of numbers look all right (except
for the hyphens used as minus signs), but they are placed poorly
relative to the labels.<br>", con)

if (interactive())
 toHTML(tab, con, options=htmloptions(head=FALSE, table=TRUE, pad=FALSE, justification="r"))

writeLines("<p>This one has pad = TRUE. It looks best, but if you cut
and paste, the spacing characters may cause problems.<br>", con)

if (interactive())
  toHTML(tab, con, options=htmloptions(head=FALSE, table=TRUE, pad=TRUE))

table_options(save)
close(con)
if (interactive())
  browseURL(f)
```

HTMLfootnotes *Construct footnotes*

Description

This function constructs HTML code for footnotes to insert at the bottom of the table.

Usage

```
HTMLfootnotes(tab, ...)
```

Arguments

`tab` A tabular object, used only so that the column width of the footnotes matches.

`...` The footnotes. If named, will be preceded with the name as a superscript.

Value

A character string containing HTML code for the footnotes. Use this in `table_options(HTMLfooter = ...)`

Examples

```
tab <- tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
footnote <- HTMLfootnotes(tab,
                          "This is a footnote with no marker.",
                          "*" = "This is a footnote with an asterisk.")
if (interactive())
  toHTML(tab, options = list(doFooter = TRUE,
                             HTMLfooter = footnote))
```

`Justify` `Justify` *pseudo-function*

Description

The `Justify` pseudo-function sets the justification of the following items in the table.

Usage

```
Justify(labels, data=labels)
```

Arguments

`labels` Justification to use for labels

`data` Justification to use for data.

Details

The justification can be an R name if that is syntactically valid, or a quoted string.

Pseudo-functions

This is a “pseudo-function”: it takes the form of a function call, but is never actually called: it is handled specially by `tabular`.

Examples

```
tabular( Justify(c,l)*Heading(Var)*(Sepal.Length+Sepal.Width) ~
         Justify(c)*(mean + sd), data=iris )
```

knit_print.tabular *Custom printing of* tabular *objects.*

Description

Automatically print tabular objects with formatting when in a **knitr** document.

Usage

```
## S3 method for class 'tabular'
knit_print(x, format = getKnitrFormat(), ...)
```

Arguments

x	A tabular object.
format	Which output format? "latex" and "html" are supported.
...	Other parameters, currently ignored.

Details

This function is not normally called by a user. It is designed to be called by **knitr** while processing a '.Rmd' or '.Rnw' document.

If table_options()$knit_print is TRUE and the output format is supported, this method will prepare output suitable for formatted printing in a **knitr** document using asis_output. Otherwise, the usual unformatted print display will be done by normal_print.

Value

An object marked for printing in a **knitr** document.

Examples

```
tab <- tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
knitr::knit_print(tab)
```

labels	*Retrieve or modify the row or column labels.*

Description

These functions allow the row or column labels of a tabular object to be retrieved or modified.

Usage

```
rowLabels(x)
rowLabels(x) <- value
colLabels(x)
colLabels(x) <- value
## S3 method for class 'tabularRowLabels'
x[i, j, ..., drop = FALSE]
## S3 method for class 'tabularColLabels'
x[i, j, ..., drop = FALSE]
```

Arguments

x	A "tabular", "tabularRowLabels" or "tabularColLabels" object.
value	A replacement
i, j, ..., drop	Arguments used for subsetting the labels. See Details below.

Details

Subsetting the row labels does not allow the number of rows to be changed; likewise, subsetting the column labels does not allow the number of columns to be changed. To change both, subset the original "tabular" object.

Value

rowLabels and the corresponding subsetting method return an object of class "tabularRowLabels".

colLabels and the corresponding subsetting method return an object of class "tabularColLabels".

The assignment functions return "tabular" objects.

See Also

[.tabular

Examples

```
tab <- tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
colLabels(tab)
colLabels(tab) <- colLabels(tab)[1,]
tab
```

labelSubset *Add a label to a logical vector.*

Description

This function is mainly for internal use. It adds a label to a logical vector, so that the `Percent` pseudo-function can ignore it when forming a denominator.

Usage

```
labelSubset(subset, label)
```

Arguments

`subset` A logical vector describing a subset of the dataset.

`label` A character label to use to describe this subset in a call to `Equal` or `Unequal` within `Percent`.

Value

A vector of class `"labelledSubset"` with the label recorded as an attribute.

Author(s)

Duncan Murdoch

See Also

`Percent`

latex.tabular *Display a tabular object using LaTeX.*

Description

This is similar to `print.tabular`, but it inserts the code to display the table in a LaTeX tabular environment. The `toLatex.tabular` method works with the `toLatex` generic from **utils**.

Usage

```
toLatex(object, ...)
## S3 method for class 'tabular'
toLatex(object, file = "", options = NULL, append = FALSE, ...)
latex.tabular(object, ...)
```

Arguments

object	The tabular object.
file	A filename or connection to which to write the LaTeX code, or "" to write to the standard output.
options	A list of options to set for the duration of the call.
append	If TRUE, opens file for appending (if it is a filename rather than a connection).
...	Settings for default formatting. See Details below.

Details

The toLatex() method produces LaTeX output suitable for inclusion in a Sweave document.

The latex.tabular function is set up to work as a method for the latex generic in **Hmisc**, but is not registered as a method, so that **tables** can work when **Hmisc** is not installed.

Value

The toLatex() method returns x invisibly, and prints the LaTeX script to the console.

table_options() and booktabs() return the previous settings.

Note

For historical reasons, the toLatex() function with a non-empty file argument doesn't write to the file until the returned value is printed.

See Also

print.tabular, table_options, toLatex, latex

Examples

```
tab <- tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
toLatex(tab)
save <- booktabs()
toLatex(tab)
table_options(save)
```

latexNumeric *Process numeric LaTeX or HTML values.*

Description

This takes formatted strings as produced by format from numeric values, and modifies them to LaTeX or HTML code that retains the spacing, and renders minus signs properly. The default formatting in tabular uses this to maintain proper alignment.

Usage

```
latexNumeric(chars, minus = TRUE, leftpad = TRUE, rightpad = TRUE, mathmode = TRUE)
htmlNumeric(chars, minus = TRUE, leftpad = TRUE, rightpad = TRUE)
```

Arguments

`chars` A character vector of numeric values.

`minus` Whether to pad cases with no minus sign with spacing of the same width.

`leftpad, rightpad` Whether to pad cases that have leading or trailing blanks with spacing matching a digit width per space.

`mathmode` Whether to wrap the result in dollar signs, so LaTeX renders minus signs properly.

Value

A character vector of the same length as `chars`, with modifications to render properly in LaTeX.

Examples

```
latexNumeric(format(c(1.1,-1,10,-10)))
htmlNumeric(format(c(1.1,-1,10,-10)))
```

latexTable *Create table in full table environment*

Description

The `tabular` function creates a table which usually renders as a `tabular` environment when displayed in LaTeX, not as a "float" with caption, label, etc. This function wraps the `tabular` result in the result of a call to the `knitr::kable` function.

Usage

```
latexTable(table, format = "latex", longtable = FALSE, ...)
```

Arguments

`table` Either a formula to be passed to `tabular`, or the result of a call to that function.

`format` Currently only `"latex"` format output is supported.

`longtable, ...` Additional arguments to be passed to the `knitr::kable` function. See details below.

Details

Rather than redoing all the work of generating LaTeX code to wrap the `tabular` result, this function works by generating a dummy `kable` table, and replaces the `tabular` part with the `tabular` result.

Many of the arguments to `kable` deal with the content of the table, not the wrapper. These will be ignored with a warning. Currently the arguments that will not be ignored, with their defaults, are:

- `caption = NULL` The caption to use on the table.
- `label = NULL` Part of the LaTeX label to use on the table. The full label will have `"tab:"` prepended by `kable`.
- `escape = TRUE` Whether to escape special characters in the caption.
- `booktabs = FALSE, longtable = FALSE` Logical values to indicate that style of table. These must also be specified to `tabular`; see the main vignette for details.
- `position = ""` The instruction to LaTeX about how to position the float in the document.
- `centering = TRUE` Whether to center the table in the float.
- `caption.short = ""` Abbreviated caption to use in TOC.
- `table.envir = if (!is.null(caption)) "table"` Name of outer environment.

These arguments are all defined in the **knitr** package, and may change in the future.

Value

An object of class `"knitr_kable"` suitable to include in a `Sweave` or **knitr** ‘`.Rnw`’ document.

Examples

```
cat(latexTable(tabular((Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris ),
    caption = "Iris sepal data", label = "sepals"))
```

`Literal` *Insert a literal entry into a table margin.*

Description

This allows insertion of arbitrary LaTeX text into a table.

Usage

```
Literal(x, nearData = TRUE)
```

Arguments

`x`	A character string to insert.
`nearData`	See the Details section of `Heading`.

Details

In LaTeX the `literal` string should usually end with a `%` comment character to avoid having a blank line inserted.

Value

Produces an expression to insert a label containing the literal text.

See Also

`Hline`, which uses this to insert lines.

Examples

```
tabular( (Literal("Some text") + Species)  ~
       All(iris)*mean, data=iris )
```

`matrix_form.tabular` *Transform tabular object to matrices printable by formatters package*

Description

The **formatters** package provides methods for displaying and breaking up tables into smaller pieces.

Usage

```
matrix_form.tabular(df)
```

Arguments

`df`	A tabular object.

Details

The **formatters** package provides a `matrix_form` generic function for S4 objects. `"tabular"` objects are S3 objects, so it won't dispatch to this function, which must be fully specified when called.

Author(s)

Duncan Murdoch with help from Gabe Becker.

Examples

```
if (requireNamespace("formatters")) {
  Sex <- factor(sample(c("Male", "Female"), 100, replace = TRUE))
  Status <- factor(sample(c("low", "medium", "high"), 100, replace = TRUE))
  z <- rnorm(100) + 5
  fmt <- function(x) {
  s <- format(x, digits=2)
  even <- ((1:length(s)) %% 2) == 0
  s[even] <- sprintf("(%s)", s[even])
  s
}
  tab <- tabular( Justify(c)*Heading()*z*Sex*Heading(Statistic)*Format(fmt())*(mean+sd)
                  ~ Status )
  mform <- matrix_form.tabular(tab)
  page <- 1
  cat("Page ", page, " Full table\n\n")
  cat(formatters::toString(mform))

  # This shows automatic pagination, breaking up the
  # table by rows
  byrow <- formatters::pag_indices_inner(formatters::mf_rinfo(mform),
                              rlpp = 2,
                              min_siblings = 1)
  for (i in seq_along(byrow)) {
    mform2 <- matrix_form.tabular(tab[byrow[[i]], ])
    page <- page + 1
    cat("\nPage ", page, " Rows", byrow[[i]], "\n\n")
    cat(formatters::toString(mform2))
  }

  # This gives the breaks by columns, counting the
  # row label columns.
  # The formatters::vert_pag_indices function had an incompatible
  # change in v0.5.8
  if ("fontspec" %in% names(formals(formatters::vert_pag_indices)))
    bycol <- formatters::vert_pag_indices(mform, cpp = 30,  rep_cols = 2,
                                        fontspec = NULL)
  else
    bycol <- formatters::vert_pag_indices(mform, cpp = 30,  rep_cols = 2)
  # Display the table with both kinds of breaks
  for (i in seq_along(byrow)) {
    rows <- byrow[[i]]
    for (j in seq_along(bycol)) {
      cols <- bycol[[j]]
      cols <- cols[cols > 2] - 2  # cols includes the row labels
      mform3 <- matrix_form.tabular(tab[rows, cols, drop = FALSE])
      page <- page + 1
      cat("\nPage ", page, "Rows", rows, "column", cols, "\n\n")
      cat(formatters::toString(mform3))
    }
  }
}
```

Paste *Generate terms to paste values together in table.*

Description

This function generates a component of a table formula to output multiple columns with punctuation between. It is designed only for LaTeX output.

Usage

```
Paste(..., head, digits=2, justify="c", prefix="", sep="", postfix="",
      character.only = FALSE)
```

Arguments

`...` Expressions to be displayed in the columns of the table. If they are named, they will get those names as headings, otherwise they will not be labelled.

`head` If not missing, this will be used as a column heading for the combined columns.

`digits` Will be passed to the `format` function. If `digits` is length one, all expressions use a common format; otherwise they are formatted separately.

`justify` One or more justifications to use on the individual columns.

`sep` One or more separators to use between columns.

`prefix, postfix` Additional text to insert before and after the group of columns.

`character.only` If `TRUE`, the `head` argument will be interpreted as an expression evaluating to a character value.

Value

An expression which will produce the requested output in LaTeX.

Examples

```
stderr <- function(x) sd(x)/sqrt(length(x))
lcl <- function(x) mean(x) - qt(0.975, df=length(x)-1)*stderr(x)
ucl <- function(x) mean(x) + qt(0.975, df=length(x)-1)*stderr(x)
toLatex( tabular( (Species+1) ~ All(iris)*
          Paste(lcl, ucl, digits = 2,
                head="95\% CI",
                prefix = "[", sep = ",", postfix = "]"),
          data=iris ) )
```

Percent *Pseudo-function to compute a statistic relative to a reference set.*

Description

The Percent pseudo-function is used to specify a statistic that depends on other values in the table.

Usage

```
Percent(denom = "all", fn = percent)
```

Arguments

denom	How the reference set (the denominator in case of a percentage calculation) should be calculated. See below.
fn	The two argument function to calculate the statistic.

Details

The function fn will be called with two arguments. The first argument is the usual "value vector" of values corresponding to this cell in the table, and the second is another vector of reference values, determined by denom.

The default value of fn is the percent function, defined as function(x, y) 100*length(x)/length(y). This gives the ratio of the number of values in the current cell relative to the reference values, expressed as a percentage. Using fn = function(x, y) 100*sum(x)/sum(y) would give the percentage of the sum of the values in the current cell to the sum in the reference set.

With the default denom = "all", all values of the analysis variable in the dataset are used as the reference. Other possibilities are denom = "row" or denom = "col", for which the values of the variable corresponding to the current row or column subset are used.

The special syntax denom = Equal(...) will record each expression in .... The reference set will be the cases with equal values of all expressions in ... to those of the current cell. The similar form denom = Unequal(...) sets the reference values to be those that differ in any of the ... expressions from the current cell. (In fact, these can be used somewhat more generally; see the vignette for details.)

Finally, other possible denom values are a logical vector, in which case the values marked TRUE are used, or anything else, which will be passed to fn as y without any subsetting. (To pass a variable with subsetting, use the Arguments pseudo-function instead.)

Pseudo-functions

Percent is a "pseudo-function": it takes the form of a function call, but is never actually called: it is handled specially by tabular. Equal and Unequal are also pseudo-functions, but are only special when used in the denom argument to Percent.

See Also

Arguments for a different way to specify a multiple argument analysis function.

Examples

```
x <- factor(sample(LETTERS[1:2], 1000, replace = TRUE))
y <- factor(sample(letters[3:4], 1000, replace = TRUE))
z <- factor(sample(LETTERS[5:6], 1000, replace = TRUE))

# These both do the same thing:
tabular( (x + 1)*(y + 1) ~ (z + 1)*(1+(RowPct=Percent("row"))))
tabular( (x + 1)*(y + 1) ~ (z + 1)*(1+(xyPct=Percent(Equal(x, y)))))
```

`PlusMinus` *Generate* `x +/- y` *terms in table.*

Description

This function generates a component of a table formula to output two columns separated by a `+/-` sign. It is designed only for LaTeX output.

Usage

```
PlusMinus(x, y, head, xhead, yhead, digits = 2,
          character.only = FALSE, ...)
```

Arguments

`x, y` Expressions to be displayed in the columns on the left and right of the `+/-` sign, respectively.

`head` If not missing, this will be used as a column heading for the two columns.

`xhead, yhead` If not missing, these will be used as individual column headings.

`digits, ...` Parameters to pass to the `format` function.

`character.only` If `TRUE`, the `head`, `xhead` and `yhead` arguments will be interpreted as expressions evaluating to character values.

Value

An expression which will produce the requested output in LaTeX.

Examples

```
stderr <- function(x) sd(x)/sqrt(length(x))
toLatex( tabular( (Species+1) ~ Sepal.Length*
         PlusMinus(mean, stderr, digits=1), data=iris ) )
```

RowFactor *Use a variable as a factor to give rows in a table.*

Description

The functions take a variable and treat it as a factor in a table. RowFactor is designed for LaTeX output, adding extra spacing to make the table more readable. Multicolumn also works only in LaTeX, and displays the label in a style with the level on a line by itself, spanning multiple columns.

Usage

```
Factor(x, name = deparse(expr), levelnames = levels(x),
       texify = getOption("tables.texify", FALSE),
       expr = substitute(x), override = TRUE)
RowFactor(x, name = deparse(expr),
  levelnames = levels(x),
          spacing = 3, space = 1, suppressfirst = TRUE,
          nopagebreak = "\\nopagebreak ",
          texify = getOption("tables.texify", FALSE),
          expr = substitute(x),
          override = TRUE)
Multicolumn(x, name = deparse(expr), levelnames = levels(x),
            width=2, first=1, justify="l",
            texify = getOption("tables.texify", FALSE),
            expr = substitute(x),
            override = TRUE)
```

Arguments

x	A variable to be treated as a factor.
name	The display name for the factor.
levelnames	The strings to use as levels of x.
texify	If TRUE, characters that would be interpreted specially by LaTeX are escaped (using latexTranslate) so they will print properly.
expr	The expression to use in evaluating the factor. Generally the same as the expression passed as x, but internal uses may differ.
override	Should the name for the factor override any previously specified Heading() setting?
spacing	Extra spacing will be added before every spacing lines.
space	How much extra space to add, in ex units.
suppressfirst	Whether to suppress the spacing in the first group.
nopagebreak	LaTeX macro to insert to suppress page breaks except between groups.
width	How many columns should the label span?
first	Which is the first column in which this label appears?
justify	How should the label be justified in the columns?

Value

Language to insert into the table formula to achieve the desired table.

Examples

```
tabular( Factor(1:10, "row") ~
       All(iris[1:10,])*Heading()*identity )
toLatex( tabular( RowFactor(1:10, "", 5)  ~
       All(iris[1:10,])*Heading()*identity ))
```

`table_options` *Set or query options for the table formatting.*

Description

These functions set or query options for table formatting in LaTeX or HTML output.

Usage

```
table_options(...)
booktabs(...)
htmloptions(head=TRUE, table=TRUE, pad=FALSE,
            ...)
```

Arguments

`head`	logical; enables all of the HTML header options
`table`	logical; enables output of all parts of the table itself
`pad`	logical; enables all of the HTML padding options
`...`	Any of the options listed in the Details below.

Details

The `table_options()` function sets a number of options that control formatting. Currently the options that affect both LaTeX and HTML output are:

`justification = "c"` Default justification for the data columns in the table.

`rowlabeljustification = "l"` Default justification for row labels.

`doBegin, doHeader, doBody, doFooter, doEnd` These logical values (all defaults are `TRUE`) control the inclusion of specific parts of the output table.

`knit_print = TRUE` Do auto formatting when printing in a **knitr** document.

These options are only used for LaTeX output:

`tabular = "tabular"` The LaTeX environment to use for the table. Other choices such as `"longtable"` might make sense.

toprule, midrule, bottomrule The LaTeX macros to use for the lines in the table. By default they are all "\hline".

titlerule = NULL The LaTeX macro to use to underline multicolumn titles. If NULL, no underlining is done.

latexleftpad, latexrightpad, latexminus, mathmode These control formatting of numbers in the table. If TRUE (the default), blanks in R's formatting are converted to hard spaces in the LaTeX output, and negative signs are rendered properly. Generally this makes output look better, but the '.tex' input may be harder to read.

These options are only used for HTML output:

doHTMLheader, doCSS, doHTMLbody These control output of the material at the top of an HTML page.

HTMLhead, CSS, HTMLbody These are the default strings to output when the corresponding element is selected. If present, the string "CHARSET" will be replaced with the result of localeToCharset() in the HTMLhead. The string "#ID" will be replaced with "#" followed by the id argument to html.tabular (or removed if that is blank).

HTMLcaption This is an optional HTML caption for the table. If NULL, no caption is emitted.

HTMLleftpad, HTMLrightpad, HTMLminus These control formatting of numbers in the table. If TRUE, blanks in R's formatting are converted to hard spaces in the HTML output, and negative signs are rendered properly. Generally this makes output look better, but cut and paste from the table may include these special characters and not be recognized by other software. The default is FALSE.

HTMLattributes This is a string to add to the "<table>" declaration at the top of the table. By default, the attributes are 'frame="hsides" rules="groups"'. These set horizontal rules on the top and bottom of the table and between the header, body, and footer (if present).

HTMLfooter This is NULL for no footer, or HTML code to insert in the table. Note that in HTML the footer should be specified before the body of the table; html.tabular will do this if both are written in the same call.

These may be set persistently by calling table_options(), or just for the duration of the call by passing them in a list via latex(options=list(...)). Additional ... arguments to latex are passed to format.

The booktabs() function sets the table_options() values to different defaults, suitable for use with the **booktabs** LaTeX package.

The htmloptions() function constructs a list suitable for the options argument to html.tabular, with grouping of options that rarely differ from each other.

Note that any LaTeX code can be used in the rule options; for example, see the longtable example in the vignette. Material to go above the headers goes into toprule, material between the headers and the body goes into midrule, and material at the bottom of the table goes into bottomrule.

Value

table_options() and booktabs() return the previous settings.

htmloptions() returns a list of settings without changing the defaults.

See Also

latex.tabular, html.tabular

Examples

```
tab <- tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
toLatex(tab)
save <- booktabs()
toLatex(tab)
table_options(save)


f <- tempfile(fileext = ".html")
if (interactive())
  toHTML(tab, f,
    options=htmloptions(HTMLcaption="Table of Iris Data",
                       pad = TRUE))
```

tabular *Compute complex table*

Description

Computes a table of summary statistics, cross-classified by various variables.

Usage

```
tabular(table, ...)
## Default S3 method:
tabular(table, ...)
## S3 method for class 'formula'
tabular(table, data = NULL, n, suppressLabels = 0, ...)
## S3 method for class 'tabular'
print(x, justification="n", ...)
## S3 method for class 'tabular'
format(x, digits=4, justification="n", latex=FALSE, html=FALSE,
                         leftpad = TRUE, rightpad = TRUE, minus = TRUE,
 mathmode = TRUE, ...)
## S3 method for class 'tabular'
x[i, j, ..., drop=FALSE]
## S3 method for class 'tabular'
cbind(..., deparse.level = 1)
## S3 method for class 'tabular'
rbind(..., deparse.level = 1)
```

Arguments

`table` A table expression. See the Details below.

`data` An optional dataframe, list or environment in which to look for variables in the table.

`n` An optional value giving the length of the data. See the Details below.

`suppressLabels` How many initial labels to suppress?

`x` The object to print, format, or subset.

`digits, ...` In the print and format methods, how many significant digits or other parameters to show by default? See Formatting below.

`justification` The default justification to use in the table.

`latex` If `TRUE`, the `latexNumeric` function will be applied when formatting numeric columns after `format`, to maintain spacing and handle signs properly.

`html` If `TRUE`, the `htmlNumeric` function will be applied when formatting numeric columns after `format`, to maintain spacing and handle signs properly.

`leftpad`, `rightpad`, `minus`, `mathmode`
Options to pass to `latexNumeric` or `htmlNumeric` to control details of formatting. See those pages for details.

`i`, `j`, `drop` The usual arguments for matrix indexing, but see the Details below.

`deparse.level` Ignored. (Present because the generic requires it.)

Details

For the purposes of this function, a "table" is a rectangular array of values, computed using a formula expression. The left hand side of the formula describes the rows of the table, the right hand side describes the columns.

Within the expression for the rows or columns, the operators `+`, `*` and `=` have special meanings.

The `+` operator represents concatenation, so that `x + y ~ z` says to show the rows corresponding to `x` above the rows corresponding to `y`.

The `*` operator represents nesting, so that `x*y ~ z` says to show the rows of `y` within each row corresponding to `x`.

The `=` operator sets a new name for a term; it is an abbreviation for the `Heading()` pseudo-function. ("Pseudo-functions" are described in the `tables` vignette.) Note that `=` has low operator precedence and may be confused by the parser with setting function arguments, so parentheses are usually needed.

Parentheses may be used to group terms in the usual arithmetic way, so `(x + y)*(u + v)` is equivalent to `x*u + x*v + y*u + y*v`.

The names `Format`, `.Format` and `Heading` have special meaning; see the section on Formatting below.

The interpretation of other terms in the formulas depends on how they evaluate.

If the term evaluates to a function, it should be a summary function that produces a scalar value when applied to a vector of values, and that scalar will be displayed in the table. For example, `(mean + var) ~ x` will display the mean of `x` above the variance of `x`. If no function is specified, `length` is assumed, so the table will display counts. (At most one summary function may be specified in any one term, so `mean*var` would be an error.)

If the term evaluates to a logical vector, it is assumed to specify a subset. For example, `~ (x > 3) + (x > 5)` will tabulate the counts of those two subsets.

If the term evaluates to a factor, it generates multiple rows or columns, corresponding to the different levels of the factor. For example if `A` has three levels, then `A ~ mean*x` will calculate the mean of `x` within each level of `A`.

If the term evaluates to a language object, it is treated as a macro, and expanded in place in the formula.

Other terms are assumed to be R expressions producing a vector of values to be summarized in the table. Only one vector of values can be specified in any given term, but different terms can summarize different values. `is.atomic` must evaluate to `TRUE` for these values for them to be recognized.

All logical, factor or other values in the table should be the same length, as if they were columns in a dataframe (but they can be computed values). If `n` is missing but `data` is a dataframe, `n` is set from that. Otherwise, if terms such as `1` appear in a table, the length is assumed to be the same as for previous terms. As a last resort, set the `n` argument in the call to `tabular()` explicitly.

The `"["` method extracts a subset of the table. For indexing, consider the table to consist of a matrix containing the values. If `drop=TRUE`, the labels and attributes are dropped. If `drop=FALSE`, the default, the `i` and `j` indices must select a rectangular block of the table; matrix indexing (using a two column matrix or a full matrix of logical values) is not supported.

Value

An object of S3 class `"tabular"`. This is a matrix of mode list, whose entries are computed summary values, with the following attributes:

`rowLabels` A matrix of labels for the rows. This will have the same number of rows as the main matrix, but may have multiple columns for different nested levels of labels. If a label covers multiple rows, it is entered in the first row, and `NA` is used to fill following rows.

`colLabels` Like `rowLabels`, but labelling the columns.

`table` The original table expression being displayed. A list of the original format specifications are attached as a `"fmtlist"` attribute.

`formats` A matrix of the same shape as the main result, containing `NA` for default formatting, or an index into the format list.

Formatting

The `tabular()` function does no formatting of computed values, but it records requests for formatting. The `format.tabular()`, `print.tabular()` and `latex.tabular()` functions make use of these requests.

By default, columns are formatted using the `format` function, with arguments `digits` and any other arguments passed in `...`. Each column is formatted separately, similarly to how a matrix is printed by default.

To request special formatting, four pseudo-functions are provided. The first is `Format`. Normally it includes arguments to pass to the `format()` function, e.g. `Format(digits=2)`. It may instead include a call to a function, e.g. `Format(sprintf("%.2f")`. In either case the summary values from cells in the table that are nested below the `Format` specification will be passed to that function in an argument named `x`, i.e. in the first example, the values would be formatted using `format(digits=2, x=values)`, and in the second, using `sprintf("%.2f", x=values)`. Users can supply their own function to be used for formatting; it should take values in a named argument `x` and return a character vector of the same length.

This can be used to control formatting in multiple columns at once. For example, `Format(digits=2)*(mean + sd)` will print both the mean and standard deviation in a single call to `format`, guaranteeing that the same number of decimal places is used in both. (The `iris` example below demonstrates this.)

If the `latex` argument to `latex.tabular` is `TRUE`, then an effort is made to retain spacing, and to convert minus signs to the appropriate type of dash using the `latexNumeric` function.

The second pseudo-function `.Format` is mainly intended for internal use. It takes a single integer argument, saying that data governed by this call uses the same formatting as another format specification. In this way entries can be commonly formatted even when they are not contiguous. The integers are assigned sequentially as the format specification is parsed; users will likely need trial and error to find the right value in a complicated table with multiple formats.

A third pseudo-function is `Justify`. It takes character values or names as arguments; how they are treated depends on the output format. In `format.tabular`, coarse justification is done to left, right or center, using `l`, `r` or `c` respectively. For LaTeX formatting, any string acceptable as a justification string to LaTeX will be passed on.

A final pseudo-function is `Heading`. Use this function in the row definitions to set a heading on the following column of row labels. (Internally this is how the headings on factor columns are implemented.) If it is called with no argument, it suppresses the following heading. The `suppressLabels=n` argument to `tabular()` is equivalent to repeating `Heading()` `n` times at the start of the table formula. The `=` operator is an abbreviation for `Heading()`; see above.

`tabular` methods

The default `tabular` method just applies `as.formula` to `table`, and then calls `tabular.formula`.

The `tabular.formula` method is the main workhorse of the package. Other authors who wish to produce tables directly from their own structures should normally create a formula whose environment contains all mentioned variables and call `tabular.formula` with appropriate arguments.

Author(s)

Duncan Murdoch

References

This function was inspired by my 20 year old memories of the SAS TABULATE procedure.

See Also

`table` and `ftable` are base R functions which produce tables of counts. The `tables` vignette has many more examples and displays the formatted output.

Examples

```
tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
         (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )

# This example shows some of the less common options
Sex <- factor(sample(c("Male", "Female"), 100, replace = TRUE))
Status <- factor(sample(c("low", "medium", "high"), 100, replace = TRUE))
z <- rnorm(100)+5
fmt <- function(x) {
  s <- format(x, digits=2)
  even <- ((1:length(s)) %% 2) == 0
  s[even] <- sprintf("(%s)", s[even])
  s
}
tab <- tabular( Justify(c)*Heading()*z*Sex*Heading(Statistic)*Format(fmt())*(mean+sd)
                ~ Status )
tab
tab[1:2, c(2,3,1)]
```

toKable *Convert* tabular *object to* knitr_kable *format.*

Description

Converts the output of the tabular and related functions to a format consistent with the output of the kable function, so that it can be customized using the **kableExtra** package.

Usage

```
toKable(table, format = getKnitrFormat(), booktabs = TRUE, ...)
getKnitrFormat(default = "latex")
```

Arguments

table	An object of class tabular.
format	The type of knitr_kable object desired; currently only "latex" and "html" are supported.
booktabs	Should the table be rendered in booktabs style? This only affects LaTeX output.
...	Additional arguments to pass to html.tabular or latex.tabular.
default	The default type of output if not running in a **knitr** document.

Value

An object of class knitr_kable, suitable for passing to functions in the **kableExtra** package.

See Also

kableExtra-package

Examples

```
if (requireNamespace("kableExtra") &&
    (!requireNamespace("pkgdown") || !pkgdown::in_pkgdown())) {
  tab <-  tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
          (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
  print(kableExtra::kable_styling(toKable(tab), latex_options = "striped"))
  cat("\n")
  toKable(tab, format = "html", options = list(HTMLcaption = "Fisher's iris data"))
}
```

toTinytable   *Convert* tabular *object to* tinytable *format.*

Description

Converts the output of the tabular and related functions to a format consistent with the output of the tt function, so that it can be customized using the **tinytable** package.

Usage

```
toTinytable(table, ...)
```

Arguments

table	An object of class tabular.
...	Additional arguments to pass to tt.

Value

An object of class tinytable, suitable for passing to functions in the **tinytable** package. These tables can be exported to several formats, including LaTeX, HTML, Markdown, Word, Typst, PDF, and PNG.

See Also

tinytable-package

Examples

```
if (requireNamespace("tinytable")) {

  tab <-  tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
          (Sepal.Length + Sepal.Width)*(mean + sd), data=iris )
  tab <- toTinytable(tab, theme = "striped")
  tab <- tinytable::style_tt(tab, i = 1:2, background = "teal", color = "white")
  tab

}
```

useGroupLabels *Format table with groups of lines*

Description

When a factor level applies to multiple lines in a table, normally an extra column of row labels is added to show the levels. This function merges that column into another column in the labels.

Usage

```
useGroupLabels(tab, col = 1,
               indent = "  ",
               newcolumn = 1,
               singleRow = TRUE,
               extraLines = 0)
```

Arguments

tab	A tabular object.
col	Which column defines the groups?
indent	A prefix to add to existing entries in the new column.
newcolumn	Which column gets the group headings?
singleRow	If a group has a single row and there is no entry in newcolumn in that row, put the group label in the same row rather than in a separate row.
extraLines	Add this many blank lines to separate the groups in addition to the line containing the group label.

Details

If newcolumn is less than 1 or greater than the number of row label columns (after removing column col), extra columns will be added.

Value

A tabular object, modified to include header rows as specified.

Examples

```
set.seed(123)
n <- 10
df <- data.frame(a = factor(sample(1:3, n, replace=TRUE)),
                 b = factor(sample(1:3, n, replace=TRUE)),
                 x = rnorm(n))
levels(df$a) <- c("Long name 1", "Long name 2", "Long name 3")
levels(df$b) <- c("a", "abc", "abcdef")
library(tables)
tab <- tabular(a*Heading()*b ~ mean*x, data = df)
tab <- tab[!is.na(tab[,1])]
useGroupLabels(tab)
```

write.csv.tabular *Write table to file in CSV or other format.*

Description

This writes the formatted table into a CSV or other delimeted file, for import into a spreadsheet or other report writer.

Usage

```
write.csv.tabular(x, file="",
    justification = "n", row.names=FALSE, ...)
write.table.tabular(x, file="",
    justification = "n", row.names=FALSE, col.names=FALSE, ...)
```

Arguments

`x`	An object from `tabular`.
`file`	A filename or connection to which to write.
`justification`	Parameter to pass to `format.tabular`.
`row.names`, `col.names`	Parameters to pass to `write.csv` or `write.table`
`...`	Parameters to pass to `format.tabular` or `write.table`; see Details below.

Details

`write.csv.tabular` writes a simple version of the table (similar to what is produced by `print.tabular`) to the given connection in CSV format, using `write.csv`. `write.table.tabular` does similarly using the more general `write.table`.

The optional arguments in `...` are sent to `write.csv`/`write.table` if their names exactly match parameters to `write.table`; otherwise, they are sent to `format.tabular`.

Value

The return value from `write.csv` or `write.table`.

Examples

```
## Not run:
# This writes a table to the clipboard on Windows using tab delimiters, for
# easy import into a spreadsheet.

write.table.tabular(
   tabular( (Species + 1) ~ (n=1) + Format(digits=2)*
          (Sepal.Length + Sepal.Width)*(mean + sd), data=iris ),
    "clipboard", sep="\t")

## End(Not run)
```

Index