

# Package ‘odfWeave’

April 17, 2009

**Type** Package

**Title** Sweave processing of Open Document Format (ODF) files

**Version** 0.7.10

**Date** 2009-01-25

**Author** Max Kuhn, Steve Weaston

**Maintainer** Max Kuhn <max.kuhn@pfizer.com>

**Description** Sweave processing of Open Document Format (ODF) files

**Depends** R (>= 2.3.1), lattice, XML

**License** GPL-2

**Repository** CRAN

**Date/Publication** 2009-01-26 08:53:34

## R topics documented:

adjustImageSize . . . . .	2
announce . . . . .	3
listString . . . . .	3
matrixPaste . . . . .	4
odfCat . . . . .	5
odfFigureCaption . . . . .	6
odfInsertPlot . . . . .	7
odfItemize . . . . .	8
odfPageBreak . . . . .	8
odfSetPageStyle . . . . .	9
odfTable . . . . .	9
odfTableCaption . . . . .	11
odfTmpDir . . . . .	12
odfTranslate . . . . .	13
odfWeave . . . . .	13

odfWeaveControl . . . . .	15
pkgVersions . . . . .	16
RweaveOdf . . . . .	17
setStyles . . . . .	18
tableStyles . . . . .	20

<b>Index</b>	<b>21</b>
--------------	-----------

---

adjustImageSize	<i>Automate image size adjustments</i>
-----------------	--

---

### Description

This function is a wrapper for `getImageDefs` and `setImageDefs` to allow the user to change image sizes in one function call.

### Usage

```
adjustImageSize(x, y, scale = 1, res = 96)
```

### Arguments

<code>x</code>	the display width of an image (in inches)
<code>y</code>	the display height of an image (in inches)
<code>scale</code>	a scaling factor for the image file size
<code>res</code>	resolution for bitmap images

### Details

This function will get the current image specifications, change them and reset them using `setImageDefs`. If the device is either `bmp`, `jpeg` or `png`, the image will be converted to pixel size using the `res` argument.

Optionally, to make bitmap images look a little better, the `scale` argument can be used to scale-up the image file. This also has the effect of reducing the size of text and points in the displayed image.

### Value

No data are returned.

### Author(s)

Sarah Goslee, Max Kuhn

### Examples

```
getImageDefs()
adjustImageSize(2, 5, scale = 1.75)
getImageDefs()
```

---

announce	<i>cat and flush console</i>
----------	------------------------------

---

**Description**

This function is a wrapper for `cat` with an immediate flush of the console

**Usage**

```
announce(verbose = TRUE, ...)
```

**Arguments**

verbose	a logical for printing
...	arguments passed to <code>cat</code>

**Value**

NULL, invisibly

**Author(s)**

Nathan Coulter

**Examples**

```
announce()  
announce(FALSE, "this will not be printed")  
announce(TRUE, "but this will be")
```

---

listString	<i>Convert a vector to a textual list</i>
------------	---

---

**Description**

Takes a vector and produces output like "element1 and element2" or "element1, element2 and element3"

**Usage**

```
listString(x, period = FALSE, verbose = FALSE)
```

**Arguments**

x                    a vector (converted to character if not)  
 period              a logical: should a period end the text?  
 verbose             a logical: how much details are logged

**Value**

a character string

**Author(s)**

Max Kuhn

**Examples**

```
listString(letters[1])
listString(letters[1:2])
listString(letters[1:4])
```

---

matrixPaste

*Element-Wise Paste of Conforming Matrices*

---

**Description**

For a series of character matrices with the same dimensions, paste each element together in the order specified.

**Usage**

```
matrixPaste(..., sep = rep(" ", length(list(...)) - 1))
```

**Arguments**

...                    a set of  $P$  character matrices with the same dimensions  
 sep                    a vector of  $P-1$  separators for each element

**Details**

The matrices are converted to vectors and pasted, then re-dimensioned. Different separators can be used between matrices, but the same separator must be used for all of the elements.

**Value**

a character matrix

**Author(s)**

Max Kuhn

**Examples**

```
mat1 <- matrix(letters[1:6], nrow = 2)
mat2 <- matrix(LETTERS[1:6], nrow = 2)
mat3 <- matrix(paste(1:6), nrow = 2)

matrixPaste(mat1, mat2)
matrixPaste(mat1, mat2, sep = "+")
matrixPaste(mat1, mat2, mat3, sep = c("+", "plus"))
```

odfCat

*Concatenate and Print in Native ODF***Description**

Outputs the objects, concatenating the representations, and sandwiches the output between XML tags.

**Usage**

```
odfCat(..., sep = " ", trim = FALSE,
       digits = max(3, getOption("digits") - 3), nsmall = 0,
       width = NULL, na.encode = TRUE, scientific = NA)
```

**Arguments**

...	R objects (see Details for the types of objects allowed).
sep	character string to insert between the objects to print.
trim	used to convert numeric data to character using <a href="#">format</a>
digits	used to convert numeric data to character using <a href="#">format</a>
nsmall	used to convert numeric data to character using <a href="#">format</a>
width	used to convert numeric data to character using <a href="#">format</a>
na.encode	used to convert numeric data to character using <a href="#">format</a>
scientific	used to convert numeric data to character using <a href="#">format</a>

**Details**

`odfCat` is an analog to [cat](#) and is useful for producing output in user-defined functions. It converts its arguments to character strings, concatenates them, separating them by the given 'sep=' string. It then sandwiches the text between `<text:p>` tags, and then outputs them. Note that this will produce paragraphs and cannot sequentially produce sentences within a paragraph.

[cat](#) uses an internal function, so there will be some differences between [cat](#) and `odfCat`. For example, factors are naively converted to character and numeric data are converted to character.

Since the text is embedded in ODF tags escaped characters (e.g. `\n`) don't have any effect. Exceptions are single- and double-quotes.

The paragraph uses the current paragraph style. The document `formatting.odt` in the package's examples directory illustrates the process of changing the appearance of the paragraph.

**Value**

a character string of class `odfCat`

**Author(s)**

Max Kuhn

**See Also**

`cat`, `format`

**Examples**

```
odfCat("\"hello world\"")
odfCat("these are the first letters", letters[1:5])
odfCat("some random normal data:", rnorm(5))
```

---

`odfFigureCaption` *Provide a Caption for a Figure*

---

**Description**

Provide a numbered caption for a figure. Captions are automatically numbered, and by default using arabic numerals, but letters or roman numerals can also be specified via the `numformat` argument.

**Usage**

```
odfFigureCaption(caption, numformat='1', numlettersync=FALSE, formula='Illustration
```

**Arguments**

<code>caption</code>	the text portion of the caption
<code>numformat</code>	the format to use the figure number
<code>numlettersync</code>	specifies the style of numbering to use if <code>numformat</code> is 'A' or 'a'
<code>formula</code>	the formula to use for computing this figure number from the previous

**Details**

This function should be called no more than once in a code chunk where 'figure' was set to true.

Legal values for `numformat` are 'A', 'a', 'I', 'i', and '1'.

If `numformat` is 'A' or 'a', `numlettersync` specifies what style of numbering to use after the first 26 figures. If `numlettersync` is true, the next 26 figures will be numbered 'AA', 'BB', ..., 'ZZ', 'AAA', 'BBB', etc. If `numlettersync` is false, the subsequent figures will be numbered 'AA', 'AB', ..., 'AZ', 'BA', 'BB', ..., 'BZ', etc.

The default formula, which numbers figures consecutively, is usually desired, but you could specify a formula of 'Illustration+10' to have your figures numbered 1, 11, 21, etc.

**Examples**

```
## Not run:
odfFigureCaption("This is a very interesting figure")
## End(Not run)
```

---

odfInsertPlot      *Write XML for image inseration*

---

**Description**

Writes ODF markup to allow images in documents. Also can copy the image file to the Picture directory.

**Usage**

```
odfInsertPlot(file, height, width, units = "in", anchor =
              c("<text:p>", "</text:p>"), name = paste("graphics",
              floor(runif(1) * 1000), sep = ""), externalFile =
              FALSE, dest = paste(getwd(), "/Pictures", sep = ""),
              caption = NULL)
```

**Arguments**

file	a string for the image file location
height	the display height of the image
width	the display width of the image
units	the units for the display dimensions
anchor	a character vector of length 2. The image markup will be sandwiched between these two elements
name	a name for the figure
dest	the location of the picture directory
externalFile	a logical; was the plot automatically generated by R during Sweaving? If TRUE, then the file is copied to the Pictures directory.
caption	either NULL (for no caption) or the output of odfFigureCaption. Note that specifying a caption results in the image being placed within a frame

**Value**

a character string

**Author(s)**

Max Kuhn

**Examples**

```
odfInsertPlot("plot.png", 4, 4)
```

`odfItemize`*Write XML for one layer lists*

---

**Description**

Creates ODF markup for enumerated or bulleted list one layer deep

**Usage**

```
odfItemize(data, ...)
```

**Arguments**

<code>data</code>	a vector
<code>...</code>	options to pass to <code>format</code>

**Value**

a string of XML markup

**Author(s)**

Max Kuhn

**Examples**

```
odfItemize(levels(iris$Species))
```

---

`odfPageBreak`*Generate a Page Break*

---

**Description**

Generate a page break in an odfWeave document.

**Usage**

```
odfPageBreak()
```

**Details**

This function should be called in a code chunk in an odfWeave document at the point where a page break is desired.

**Examples**

```
## Not run:  
odfPageBreak()  
## End(Not run)
```

---

odfSetPageStyle	<i>Set the Page Style</i>
-----------------	---------------------------

---

**Description**

Insert a page break with a specified page style.

**Usage**

```
odfSetPageStyle(style="Standard")
```

**Arguments**

style	a character string of length one that references a page style that is either contained in the style definitions or already used within the document.
-------	--

**Details**

This function should be called in a code chunk in an `odfWeave` document at the point where a page break with a page style is desired. A common use would be to change the page to landscape mode, perhaps using the "RlandscapePage" style in order to display a table in landscape mode, and then to set it back to the standard style after generating the table.

**Examples**

```
## Not run:  
odfSetPageStyle("RlandscapePage")  
## End(Not run)
```

---

odfTable	<i>Create an Open Document Format table</i>
----------	---

---

**Description**

Create an Open Document Format table from a data frame, matrix or vector

**Usage**

```
odfTable(x, ...)

## S3 method for class 'numeric':
odfTable(x, horizontal = length(x) < 5, colnames = names(x),
  digits = max(3, getOption("digits") - 3),
  name = paste("Table", floor(runif(1) * 1000), sep = ""),
  styles = NULL, ...)

## S3 method for class 'character':
odfTable(x, horizontal = length(x) < 5, colnames = names(x),
  name = paste("Table", floor(runif(1) * 1000), sep = ""),
  styles = NULL, ...)

## S3 method for class 'data.frame':
odfTable(x, colnames = NULL, useRowNames = TRUE,
  digits = max(3, getOption("digits") - 3),
  name = paste("Table", floor(runif(1) * 1000), sep = ""),
  styles = NULL, ...)

## S3 method for class 'matrix':
odfTable(x, colnames = NULL, useRowNames = TRUE,
  digits = max(3, getOption("digits") - 3),
  name = paste("Table", floor(runif(1) * 1000), sep = ""),
  styles = NULL, ...)
```

**Arguments**

<code>x</code>	a vector, matrix or data frame
<code>horizontal</code>	a logical: should the vector be shown as a 1xn table or nx1? This is ignored for other data structures.
<code>colnames</code>	a vector of column names that can be used. Note that if the row names are used in the table, this should contain an extra element for that column.
<code>useRowNames</code>	a logical: should the row names be printed in the final table
<code>digits</code>	number of significant digits passed to <code>format</code>
<code>name</code>	A name for the table. ODF requires a name for each object, so a random name will be used if unspecified.
<code>styles</code>	An optional list of style names for each table element (cells, headers etc). See <a href="#">tableStyles</a>
<code>...</code>	optional arguments that can be passed to <code>format</code>

**Details**

The data structures are converted to character matrices using `format`. The `justify` and `trim` arguments to `format` are usually overridden by the table style options, so those arguments are

automatically set to `justify = "none"` and `trim = TRUE`. However, if values of these arguments are passed using the three dots, `format` will use them (but they probably won't do anything).

When using `odfTable` in a code chunk, the chunk's `results` argument should be set to `xml`.

The document `formatting.odt` in the package's `examples` directory illustrates the process of changing the appearance of the table.

### Value

a list of character string that contain XML markup

### Author(s)

Max Kuhn

### See Also

[tableStyles](#)

### Examples

```
odfTable(iris[1:5,])
```

---

`odfTableCaption`      *Provide a Caption for a Table*

---

### Description

Provide a numbered caption for a table. Captions are automatically numbered, and by default using arabic numerals, but letters or roman numerals can also be specified via the `numformat` argument.

### Usage

```
odfTableCaption(caption, numformat='1', numlettersync=FALSE, formula='Table+1')
```

### Arguments

<code>caption</code>	the text portion of the caption
<code>numformat</code>	the format to use the table number
<code>numlettersync</code>	specifies the style of numbering to use if <code>numformat</code> is 'A' or 'a'
<code>formula</code>	the formula to use for computing this table number from the previous

### Details

This function should be called immediately after a call to `odfTable` in a code chunk in an `odfWeave` document.

Legal values for `numformat` are 'A', 'a', 'I', 'i', and '1'.

If `numformat` is 'A' or 'a', `numlettersync` specifies what style of numbering to use after the first 26 tables. If `numlettersync` is true, the next 26 tables will be numbered 'AA', 'BB', ..., 'ZZ', 'AAA', 'BBB', etc. If `numlettersync` is false, the subsequent tables will be numbered 'AA', 'AB', ..., 'AZ', 'BA', 'BB', ..., 'BZ', etc.

The default formula, which numbers tables consecutively, is usually desired, but you could specify a formula of 'Table+10' to have your tables numbered 1, 11, 21, etc.

### Examples

```
## Not run:  
odfTableCaption("This is a very boring table")  
## End(Not run)
```

---

`odfTmpDir`

*odfWeave working directory*

---

### Description

Create a working directory for `odfWeave`

### Usage

```
odfTmpDir()
```

### Details

This function determines the temp directory and creates a subdirectory. If a subdirectory cannot be created, an error is thrown.

### Value

a character string containing the path

### Author(s)

Max Kuhn

---

odfTranslate                      *Translation of text to XML*

---

**Description**

odfTranslate converts some XML modified characters (such as &gt) to R code (>). This function also tries to mistake proof the code by anticipating characters that might be in UTF-8 encoding to R compliant characters (e.g. OpenOffice may convert some characters. For example, " will become a UTF-8 character, which R will choke on).

**Usage**

```
odfTranslate(x, toR = TRUE)
```

**Arguments**

x	a character vector
toR	a logical. If TRUE, the text is translated from XML to R. The opposite is done if FALSE.

**Value**

a character vector

**Author(s)**

Max Kuhn and Nathan Coulter

**Examples**

```
y <- "\$expr{paste(letters[1:5], <text:s text:c=\"2\"/>collapse = &quot;;,&quot;)}"
odfTranslate(y)
```

---

odfWeave                              *Sweave processing of Open Document Format (ODF) files*

---

**Description**

Sweave processing of Open Document Format files

**Usage**

```
odfWeave(file, dest, workDir = odfTmpDir(), control = odfWeaveControl())
```

## Arguments

<code>file</code>	the ODF file created using OpenOffice V2.0 or above.
<code>dest</code>	path to put the processed file (should include file name and extension)
<code>workDir</code>	a path to a directory where the source file will be unpacked and processed. If it does not exist, it will be created. If it exists, it should be empty, since all its contents will be included in the generated file.
<code>control</code>	a list of control settings. See <code>odfWeaveControl</code> for the names of the settable control values and their effects.

## Details

`odfWeave` can be used to embed R code within a word processing document. The `odfWeave` package was created so that the functionality of `Sweave` can be used within a rich editor like OpenOffice. The generated document can also easily be edited.

The markup language used is the Open Document Format (ODF), which is an open, non–proprietary format that encompasses text documents, presentations and spreadsheets. There are several editors/office suites that can produce ODF files. OpenOffice, as of version 2.0, uses ODF as the default format. `odfWeave` has been tested with OpenOffice to produce text documents. As of the current version, `odfWeave` processing of presentations and spreadsheets should be considered to be experimental (but should be supported in subsequent versions).

Since ODF files are compressed archives of files and directories, R will need to zip and unzip the source file. While R has an unzip utility, it does not have one for re-zipping files, so an external application is needed. `unzip` and `zip` are free utilities located at

<http://www.info-zip.org/>

Also, `jar` can be used. See `odfWeaveControl` for more information on configuring `odfWeave` to use applications other than `zip` and `unzip`.

A few notes about file paths and working directories:

- When specifying the location of the odt file, you cannot use relative paths such as `file = "../file.odt"`. If the input or output files are not in the current working directory, then the absolute path should be used
- `odfWeave` changes the working directory to the location where the odt file is decompressed. If an error occurs within `odfWeave`, the working directory will be changed back to the original path
- Since `odfWeave` changes the working directory when the code chunks are executed, references to files and directories should use absolute paths. For example, if you are using `read.csv` to bring data into R, the file specification should include the whole path since the working directory will have been changed to a temporary location

The functionality of `Sweave` is mostly preserved in `odfWeave`, such as weaving, hooks, figure environments, etc. Some functionality, such as writing output to separate files for each code chunk using the `split` argument, doesn't make sense when using ODF. See `RweaveOdf` for more details about the available options.

`odfWeave` uses the `noweb` convention for R code. In-line R commands should be in `\Sexpr` calls. the `Sexpr` text should all be completely in one visual format. e.g., changing the color of part

of the `Sexpr` could result in an error. Block code chunks should use the `<<>=` syntax (i.e. no LaTeX syntax will currently work).

The image format and sizes are specified using `setImageDefs`. The dimensions of the image file and the dimensions of the rendered image can be set independently. See `setImageDefs` for more details.

The document `formatting.odt` in the package's examples directory illustrates the process of changing the appearance of the various document elements.

### Value

an ODF file with the R output

### Author(s)

Max Kuhn

### See Also

[odfWeaveControl](#), [RweaveOdf](#), [Sweave](#)

### Examples

```
## Not run: vignette("odfWeave")

## Not run:
demoFile <- system.file("examples", "examples.odt", package = "odfWeave")
demoFile <- system.file("examples", "testCases.odt", package = "odfWeave")
demoFile <- system.file("examples", "formatting.odt", package = "odfWeave")
## End(Not run)

demoFile <- system.file("examples", "simple.odt", package = "odfWeave")
outputFile <- gsub("simple.odt", "output.odt", demoFile)

library(odfWeave)
odfWeave(demoFile, outputFile)
```

---

`odfWeaveControl`      *Control odfWeave options*

---

### Description

Allows the user to specify how `odfWeave` operates and style information for the document.

**Usage**

```
odfWeaveControl(
  zipCmd = c("zip -r $$file$$ .", "unzip -o $$file$$"),
  cleanup = TRUE, verbose = TRUE)
```

**Arguments**

`zipCmd` a string for the zipping/unzipping the odt file via a system call. The token `$$file$$` will be gsub'ed with the file name.

`cleanup` a logical: remove the working directory?

`verbose` a logical: should details be printed?

**Value**

a list with element for each of the arguments above.

**Author(s)**

Max Kuhn

**See Also**

[odfWeave](#)

**Examples**

```
odfWeaveControl(cleanup = TRUE)
```

---

pkgVersions

*List packages currently used*

---

**Description**

Creates a string, matrix or data frame that shows the packages and their versions currently used (i.e in the search path).

**Usage**

```
pkgVersions(type = "string", ncol = 4)
```

**Arguments**

`type` a string designating the type of output object: "string", "matrix" or "data frame"

`ncol` integer specification for the number of columns in the output when `type = "matrix"`

**Value**

a string, character matrix or a data frame with columns "Package" and "Version"

**Author(s)**

Max Kuhn, original version by Jim Rogers

**Examples**

```
pkgVersions()  
pkgVersions("matrix")  
pkgVersions("data frame")
```

---

RweaveOdf

*R/ODF Driver for Sweave*

---

**Description**

A driver for [Sweave](#) that translates R code chunks in XML files produced in Open Document Format (ODF) files.

**Usage**

```
RweaveOdf()  
  
RweaveOdfSetup(file, syntax,  
               output=NULL, quiet=FALSE, debug=FALSE, echo=TRUE,  
               eval=TRUE, ...)
```

**Arguments**

<code>file</code>	Name of Sweave source file.
<code>syntax</code>	An object of class <a href="#">SweaveSyntax</a> .
<code>output</code>	Name of output file, default is to remove extension <code>‘.nw’</code> , <code>‘.Rnw’</code> or <code>‘.Snw’</code> and to add extension <code>‘.xml’</code> . Any directory names in <code>file</code> are also removed such that the output is created in the current working directory.
<code>quiet</code>	If TRUE all progress messages are suppressed.
<code>debug</code>	If TRUE, input and output of all code chunks is copied to the console.
<code>echo</code>	set default for option <code>echo</code> , see details below.
<code>eval</code>	set default for option <code>eval</code> , see details below.
<code>...</code>	optional arguments. This is used to pass the control object to the driver

## Supported Options

[RweaveOdf](#) supports the following options for code chunks (the values in parentheses show the default values):

**echo:** logical (TRUE). Include S code in the output file?

**eval:** logical (TRUE). If FALSE, the code chunk is not evaluated, and hence no text or graphical output produced.

**results:** character string (`verbatim`). If `verbatim`, the output of S commands is included in the verbatim-like Soutput environment. If `xml`, the output is taken to be already proper XML markup and included as is. If `hide`, then all output is completely suppressed (but the code executed during the weave).

**print:** logical (FALSE) If TRUE, each expression in the code chunk is wrapped into a `print ()` statement before evaluation, such that the values of all expressions become visible.

**term:** logical (TRUE). If TRUE, visibility of values emulates an interactive R session: values of assignments are not printed, values of single objects are printed. If FALSE, output comes only from explicit `print` or `cat` statements.

**fig:** logical (FALSE), indicating whether the code chunk produces graphical output. Note that only one figure per code chunk can be processed this way.

Note that image options, such as the image type and size, are set using [setImageDefs](#).

## Author(s)

Max Kuhn, based on [RweaveLatex](#) by Friedrich Leisch

## See Also

[Sweave](#), [odfWeave](#), [setImageDefs](#)

---

setStyles

*Style Definitions and Assignments*

---

## Description

Utility functions for declaring and setting styles

## Usage

```
getStyles()
setStyles(style)
```

```
getStyleDefs()
setStyleDefs(def)
```

```
getImageDefs()
setImageDefs(def, verbose = TRUE)
```

**Arguments**

style	a list of style assignments
def	a list of style definitions
verbose	a logical: should warnings be printed?

**Details**

There are two main components to specifying output formats: style definitions and style assignments. The definition has the specific components (such as a table cell) and their format values (e.g. boxed with solid black lines). The function `getStyleDefs` can fetch the pre-existing styles in the package. These can be modified and new definitions can be added. The function `setStyleDefs` “registers” the style changes with the package. When `odfWeave` is called, these definitions are written to the style sections of the XML files. See the example below.

There is a second mechanism to assign styles to specific output elements. The functions `getStyle` and `setStyle` can be used to tell `odfWeave` which style definition to use for a particular output. For example, the `input` and `output` elements control how R code and command-line output look. To change either of these, an existing definition can be assigned to these entries and reset using `setStyle(currentStyle)`. Unlike the style definitions, the style assignments can be modified throughout the R code.

For graphics, `getImageDefs` and `setImageDefs` can be used to specify the type of plot device and its arguments. `getImageDefs` will return a list with elements

**type** a character string for the image type (this is also used to set the file extension). Possible values are "png", "jpeg", "bmp", or "eps" (OpenOffice does not accept pdf or svg graphics)

**device** a character string for the device that should be used to generate the graphics. Some systems may not have png or jpeg devices setup, so `capabilities` is used to make that determination by default

**plotHeight** the height for the image file. For "png", "bmp" and "jpeg" devices, this is in pixels, but for others it is in inches

**plotWidth** similar to `plotHeight`

**dispWidth** the height of the image, in inches, as shown in OpenOffice

**dispHeight** similar to `dispWidth` Since these functions can be called from within code chunks, graphical parameters can be changed during the Sweave process.

**Value**

The get functions return named lists.

**Author(s)**

Max Kuhn

**Examples**

```
currentStyleDefs <- getStyleDefs()
currentStyleDefs$ArialNormal$fontSize <- "10pt"
setStyleDefs(currentStyleDefs)
```

---

tableStyles	<i>Generate Table Styles</i>
-------------	------------------------------

---

**Description**

Based on the current style specifications, create style names for all table elements.

**Usage**

```
tableStyles(x, useRowNames = TRUE, header = NULL)
```

**Arguments**

x	a vector, matrix or data frame. See details below
useRowNames	a logical: should the row names be printed in the final table
header	an optional vector of heading names

**Details**

Based on the dimensions of `x`, this function generates table style names for all of the elements. For example, if the data are an `nxn` matrix, it will create an `nxn` matrix of style names for the text and the table cells.

The arguments of `tableStyles` must be consistent with those specified for `odfTable`, specifically the `useRowNames` and `header` arguments.

Once the appropriate set of style names are generated, the user can programatically alter it. For example, based on some logic, cells can have different text colors etc.

The document `formatting.odt` in the package's examples directory illustrates the process of changing the appearance of the table using `tableStyles`.

**Value**

a list of style names with elements: `table`, `text`, `cell`, `header` and `headerCell`. The `text` and `cell` entries are for the non-header table elements.

**Author(s)**

Max Kuhn

**See Also**

[odfTable](#)

**Examples**

```
irisStyles <- tableStyles(iris, useRowNames = TRUE, header = names(iris))
irisStyles$text[2,3] <- "ttRed"
odfTable(iris, useRowNames = TRUE, styles = irisStyles)
```

# Index

## \*Topic **utilities**

- adjustImageSize, 1
  - announce, 2
  - listString, 3
  - matrixPaste, 4
  - odfCat, 5
  - odfFigureCaption, 6
  - odfInsertPlot, 7
  - odfItemize, 8
  - odfPageBreak, 8
  - odfSetPageStyle, 9
  - odfTable, 9
  - odfTableCaption, 11
  - odfTmpDir, 12
  - odfTranslate, 13
  - odfWeave, 13
  - odfWeaveControl, 15
  - pkgVersions, 16
  - RweaveOdf, 17
  - setStyles, 18
  - tableStyles, 20
  - odfPageBreak, 8
  - odfSetPageStyle, 9
  - odfTable, 9, 20
  - odfTableCaption, 11
  - odfTmpDir, 12
  - odfTranslate, 13
  - odfWeave, 13, 16, 18, 19
  - odfWeaveControl, 14, 15, 15
  - pkgVersions, 16
  - print, 18
  - RweaveLatex, 18
  - RweaveOdf, 14, 15, 17, 18
  - RweaveOdfSetup (*RweaveOdf*), 17
  - setImageDefs, 15, 18
  - setImageDefs (*setStyles*), 18
  - setStyleDefs (*setStyles*), 18
  - setStyles, 18
  - Sweave, 14, 15, 17, 18
  - SweaveSyntax, 17
  - tableStyles, 10, 11, 20
- 
- adjustImageSize, 1
  - announce, 2
  - cat, 5, 6, 18
  - format, 5, 6
  - getImageDefs (*setStyles*), 18
  - getStyleDefs (*setStyles*), 18
  - getStyles (*setStyles*), 18
  - listString, 3
  - matrixPaste, 4
  - odfCat, 5
  - odfFigureCaption, 6
  - odfInsertPlot, 7
  - odfItemize, 8