Package 'XLConnect'

May 26, 2025

Type Package

Title Excel Connector for R

Version 1.2.2

URL https://mirai-solutions.ch
https://github.com/miraisolutions/xlconnect

BugReports https://github.com/miraisolutions/xlconnect/issues

SystemRequirements Java (>= 8)

Depends R (>= 3.6.0)

Imports methods, rJava (>= 1.0-1)

Suggests RUnit, lattice, ggplot2 (>= 0.9.3), zoo

Description Provides comprehensive functionality to read, write and format Excel data.

License GPL-3

Copyright See file COPYRIGHTS

LazyData yes

NeedsCompilation no

Author Mirai Solutions GmbH [aut],
Martin Studer [cre] (ORCID: <https://orcid.org/0009-0004-8250-7303>),
The Apache Software Foundation [ctb, cph] (Apache POI),
Graph Builder [ctb, cph] (Curvesapi Java library),
Brett Woolridge [ctb, cph] (SparseBitSet Java library)

Maintainer Martin Studer <martin.studer@mirai-solutions.com>

Repository CRAN

Date/Publication 2025-05-26 18:50:05 UTC

Contents

XLConnect-package 4
addImage-methods 5
appendNamedRegion-methods 6

appendWorksheet-methods . . . 8
aref . . . 9
aref2idx . . . 10
cellstyle-class . . . 11
clearNamedRegion-methods . . . 12
clearRange-methods . . . 14
clearRangeFromReference-methods . . . 15
clearSheet-methods . . . 16
cloneSheet-methods . . . 17
col2idx . . . 19
configurePOI . . . 19
createCellStyle-methods . . . 21
createFreezePane-methods . . . 23
createName-methods . . . 24
createSheet-methods . . . 26
createSplitPane-methods . . . 27
cref2idx . . . 28
existsCellStyle-methods . . . 29
existsName-methods . . . 30
existsSheet-methods . . . 31
extraction-methods . . . 32
extractSheetName . . . 34
getActiveSheetIndex-methods . . . 35
getActiveSheetName-methods . . . 36
getBoundingBox-methods . . . 37
getCellFormula-methods . . . 38
getCellStyle-methods . . . 39
getCellStyleForType-methods . . . 41
getDefinedNames-methods . . . 42
getForceFormulaRecalculation-methods . . . 43
getLastColumn-methods . . . 44
getLastRow-methods . . . 45
getOrCreateCellStyle-methods . . . 46
getReferenceCoordinates-methods . . . 47
getReferenceCoordinatesForName-methods . . . 48
getReferenceCoordinatesForTable-methods . . . 49
getReferenceFormula-methods . . . 50
getSheetPos-methods . . . 51
getSheets-methods . . . 52
getTables-methods . . . 53
hideSheet-methods . . . 54
idx2aref . . . 55
idx2col . . . 56
idx2cref . . . 57
isSheetHidden-methods . . . 58
isSheetVeryHidden-methods . . . 59
isSheetVisible-methods . . . 61
loadWorkbook . . . 62

mergeCells-methods . . . 63
mirai . . . 64
onErrorCell-methods . . . 65
print-methods . . . 66
readNamedRegion . . . 67
readNamedRegionFromFile . . . 71
readTable . . . 72
readWorksheet-methods . . . 75
readWorksheetFromFile . . . 80
removeName-methods . . . 81
removePane-methods . . . 82
removeSheet-methods . . . 83
renameSheet-methods . . . 84
saveWorkbook-methods . . . 85
setActiveSheet-methods . . . 87
setAutoFilter-methods . . . 88
setBorder-methods . . . 89
setCellFormula-methods . . . 90
setCellStyle-methods . . . 92
setCellStyleForType-methods . . . 94
setColumnWidth-methods . . . 95
setDataFormat-methods . . . 96
setDataFormatForType-methods . . . 98
setFillBackgroundColor-methods . . . 99
setFillForegroundColor-methods . . . 100
setFillPattern-methods . . . 101
setForceFormulaRecalculation-methods . . . 103
setHyperlink-methods . . . 104
setMissingValue-methods . . . 105
setRowHeight-methods . . . 107
setSheetColor-methods . . . 108
setSheetPos-methods . . . 109
setStyleAction-methods . . . 110
setStyleNamePrefix-methods . . . 113
setWrapText-methods . . . 114
show-methods . . . 115
summary-methods . . . 116
swissfranc . . . 117
unhideSheet-methods . . . 118
unmergeCells-methods . . . 119
with.workbook . . . 120
workbook-class . . . 121
writeNamedRegion-methods . . . 122
writeNamedRegionToFile . . . 124
writeWorksheet-methods . . . 126
writeWorksheetToFile . . . 127
XLC . . . 129
xlcDump . . . 131

xlcEdit 132
xlcFreeMemory 134
xlcMemoryReport 135
XLConnect-deprecated 135
xlcRestore 136
$-methods 137

Index **138**

`XLConnect-package` *Excel Connector for R*

Description

Provides comprehensive functionality to read, write and format Excel data.

Details

For an overview over the package please refer to the available demos:
`demo(package = "XLConnect")`

Author(s)

Mirai Solutions GmbH, <xlconnect@mirai-solutions.com>

References

Mirai Solutions GmbH: https://mirai-solutions.ch
XLConnect on GitHub: https://github.com/miraisolutions/xlconnect Mirai Solutions on GitHub: https://github.com/miraisolutions
Apache POI: https://poi.apache.org

Examples

```
## Not run:
# Load workbook; create if not existing
wb <- loadWorkbook("XLConnect.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "mtcars")

# Create a name reference
createName(wb, name = "mtcars", formula = "mtcars!$C$5")

# Write built-in data.frame 'mtcars' to the specified named region
writeNamedRegion(wb, mtcars, name = "mtcars")

# Save workbook
saveWorkbook(wb)
```

```
# clean up
file.remove("XLConnect.xlsx")

## End(Not run)
```

---

addImage-methods *Adding images to a worksheet*

Description

Adds an image to a worksheet using a named region.

Usage

```
 ## S4 method for signature 'workbook'
addImage(object, filename, name, originalSize, worksheetScope)
```

Arguments

object	The workbook to use
filename	Name of the image file. Supported are images of the following formats: JPG/JPEG, PNG, WMF, EMF, BMP, PICT.
name	Name of the named region that the image is set to
originalSize	If originalSize = TRUE, the image is inserted in the top left corner of the named region and not scaled. Otherwise, the image is scaled to fit the named region. The default value for originalSize is FALSE.
worksheetScope	Optional - the name of the worksheet in which the name is scoped; useful if different sheets have scoped regions with the same name.

Note

There is an known issue in Apache POI with adding images to xls workbooks. The result of adding images to workbooks that already contain shapes or images may be that previous images are removed or that existing images are replaced with newly added ones. It is therefore advised that you use the addImage functionality only with workbooks that have no existing shapes or images. Note that this only holds for xls workbooks (Excel 97-2003) and not for xlsx (Excel 2007+). There should be no issues with xlsx workbooks.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName

Examples

```
## Not run:
## Write an R plot to a specified named region
## This example makes use of the 'Tonga Trench Earthquakes' example

# Load workbook (create if not existing)
wb <- loadWorkbook("earthquake.xlsx", create = TRUE)

# Create a sheet named 'earthquake'
createSheet(wb, name = "earthquake")

# Create a named region called 'earthquake' referring to the sheet
# called 'earthquake'
createName(wb, name = "earthquake", formula = "earthquake!$B$2")

# Create R plot to a png device
require(lattice)
png(filename = "earthquake.png", width = 800, height = 600)
devAskNewPage(ask = FALSE)

Depth <- equal.count(quakes$depth, number=8, overlap=.1)
xyplot(lat ~ long | Depth, data = quakes)
update(trellis.last.object(),
       strip = strip.custom(strip.names = TRUE, strip.levels = TRUE),
       par.strip.text = list(cex = 0.75),
       aspect = "iso")

dev.off()

# Write image to the named region created above using the image's
# original size; i.e. the image's top left corner will match the
# specified cell's top left corner
addImage(wb, filename = "earthquake.png", name = "earthquake",
         originalSize = TRUE)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("earthquake.xlsx")
file.remove("earthquake.png")

## End(Not run)
```

---

appendNamedRegion-methods

Appending data to a named region

Description

Appends data to an existing named region.

Usage

```
## S4 method for signature 'workbook,ANY'
appendNamedRegion(object,data,name,header,overwriteFormulaCells,rownames,worksheetScope)
```

Arguments

object	The workbook to use
data	Data to write
name	Name of the (existing) named region to which to append the data
header	Specifies if the column names should be written. The default is FALSE.
overwriteFormulaCells	Specifies if existing formula cells in the workbook should be overwritten. The default is TRUE.
rownames	Name (character) of column to use for the row names of the provided data object. If specified, the row names of the data object (data.frame) will be included as an additional column with the specified name. If rownames = NULL (default), no row names will be included.
worksheetScope	Optional character vector with worksheet name(s) to target a name defined in the specified sheet(s) only. If not specified, the first matching named region is appended to. Use "" to specifically target a globally-scoped named region.

Details

Appends data to the existing named region specified by name. The data is appended at the bottom of the named region. See writeNamedRegion for further information on writing named regions.

Note

Named regions are automatically redefined to the area occupied by the previous and the newly appended data. This guarantees that the complete set of data can be re-read using readNamedRegion. Note however, that no checks are performed to see whether the appended data has the same shape/structure as the previous data.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, writeNamedRegion, readNamedRegion, writeWorksheet, appendWorksheet, readWorksheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")
```

```
# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Append mtcars data set to named region named 'mtcars'
appendNamedRegion(wb, mtcars, name = "mtcars")

## End(Not run)
```

---

appendWorksheet-methods

Appending data to worksheets

Description

Appends data to worksheets of a workbook.

Usage

```
## S4 method for signature 'workbook,ANY,character'
appendWorksheet(object,data,sheet,header,rownames)
## S4 method for signature 'workbook,ANY,numeric'
appendWorksheet(object,data,sheet,header,rownames)
```

Arguments

object	The workbook to write to
data	Data to append
sheet	The name or index of the sheet to append the data to
header	Specifies if the column names should be written. The default is TRUE.
rownames	Name (character) of column to use for the row names of the provided data object. If specified, the row names of the data object (data.frame) will be included as an additional column with the specified name. If rownames = NULL (default), no row names will be included.

Details

Appends data to the worksheet specified by sheet. Data will be appended at the bottom and left most column containing some data. If more complex "appending schemes" are required you may make direct use of writeWorksheet.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, writeWorksheet, readWorksheet, writeNamedRegion, appendNamedRegion, readNamedRegion

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Append mtcars data set to worksheet named 'mtcars'
appendWorksheet(wb, mtcars, sheet = "mtcars")

## End(Not run)
```

`aref` *Constructing Excel area references*

Description

Constructs an Excel area reference

Usage

```
aref(topLeft, dimension)
```

Arguments

`topLeft`	Top left corner. Either a `character` specifying a cell reference in the form "A1" or a `numeric` vector of length two specifying the corresponding coordinates.
`dimension`	Dimensions (`numeric`) of a 2-dimensional object (mostly a `data.frame` or a `matrix`)

Value

Returns the area reference (`character`) for the specified top left cell and dimension.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`aref2idx`, `idx2aref`, `idx2cref`, `col2idx`, `idx2col`

Examples

```
## Not run:
aref("A1", dim(mtcars))
aref(c(1, 1), dim(mtcars))

## End(Not run)
```

`aref2idx` *Converting Excel cell references to row and column based cell references*

Description

Converts Excel cell references to row and column based cell references

Usage

```
aref2idx(x)
```

Arguments

`x`	`Character` vector of Excel cell references (e.g. "A1:B6", "B6:C17", ...)

Value

Returns a numeric matrix with four columns and as many rows as cell references that have been provided. The first two columns represent the coordinates of the top left corner (row, column) and the third and fourth columns represent the bottom right corner of the referenced area.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`idx2aref`, `aref`, `cref2idx`, `idx2cref`, `col2idx`, `idx2col`

Examples

```
## Not run:
aref2idx(c("A1:B6", "B6:C17"))

## End(Not run)
```

cellstyle-class *Class "cellstyle"*

Description

This class represents a cell style in a Microsoft Excel workbook. S4 objects of this class and corresponding methods are used to manipulate cell styles. This includes setting data formats, borders, background- and foreground-colors, etc.

Objects from the Class

Cell styles are created by calling the createCellStyle method on a workbook object.

Slots

jobj: Object of class jobjRef (see package **rJava**) which represents a Java object reference that is used in the back-end to manipulate the underlying Excel cell style instance.

Note

XLConnect generally makes use of custom (named) cell styles. This allows users to more easily manage cell styles via Excel's cell style menu. For example, assuming you were using a specific custom cell style for your data table headers, you can change the header styling with a few clicks in Excel's cell style menu across all tables.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

Apply, create, or remove a cell style:
https://support.microsoft.com/en-us/office/apply-create-or-remove-a-cell-style-472213bf-66bd-40c8-
ocmsassetid=hp001216732&correlationid=5691ac73-b7a2-40c3-99aa-a06e806bb566&ui=en-us&
rs=en-us&ad=us

See Also

workbook, createCellStyle, setStyleAction, setCellStyle

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("cellstyles.xlsx", create = TRUE)

# We don't set a specific style action in this demo, so the
```

```
# default 'XLConnect' will be used (XLC$"STYLE_ACTION.XLCONNECT")

# Create a sheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' referring to the sheet
# called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$C$4")

# Write built-in data set 'mtcars' to the above defined named region.
# This will use the default style action 'XLConnect'.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Now let's color all weight cells of cars with a weight > 3.5 in red
# (mtcars$wt > 3.5)

# First, create a corresponding (named) cell style
heavyCar <- createCellStyle(wb, name = "HeavyCar")

# Specify the cell style to use a solid foreground color
setFillPattern(heavyCar, fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(heavyCar, color = XLC$"COLOR.RED")

# Which cars have a weight > 3.5 ?
rowIndex <- which(mtcars$wt > 3.5)

# NOTE: The mtcars data.frame has been written offset with top
# left cell C4 - and we have also written a header row!
# So, let's take that into account appropriately. Obviously,
# the two steps could be combined directly into one ...
rowIndex <- rowIndex + 4

# The same holds for the column index
colIndex <- which(names(mtcars) == "wt") + 2

# Set the 'HeavyCar' cell style for the corresponding cells.
# Note: the row and col arguments are vectorized!
setCellStyle(wb, sheet = "mtcars", row = rowIndex, col = colIndex,
             cellstyle = heavyCar)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("cellstyles.xlsx")

## End(Not run)
```

clearNamedRegion-methods

Clearing named regions in a workbook

Description

Clears named regions in a workbook.

Usage

```
  ## S4 method for signature 'workbook,character'
clearNamedRegion(object, name, worksheetScope)
```

Arguments

object	The workbook to use
name	The name of the named region to clear
worksheetScope	Optional - the name of the worksheet in which the region is scoped; useful if different sheets have scoped regions with the same name.

Details

Clearing a named region/range means to clear all the cells associated with that named region. Clearing named regions can be useful if (named) data sets in a worksheet need to be replaced, i.e. data is first read, modified in R and finally written back to the the same named region. Without clearing the named region first, (parts of) the original data may still be visible if they occupied a larger range in the worksheet.

If worksheetScope is unspecified, the first matching name found anywhere in the workbook will be cleared. Otherwise, only a name specifically scoped to the worksheet may be cleared. To only clear a name in global scope, pass "" as the value.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, clearSheet, clearRange, clearRangeFromReference, clearSheet

Examples

```
  ## Not run:
# mtcars xlsx file from demoFiles subfolder of
# package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx",
                             package = "XLConnect")

# Load workbook
```

```
wb <- loadWorkbook(demoExcelFile)

# Read named region 'mtcars'
data <- readNamedRegion(wb, name = "mtcars", header = TRUE)

# Only consider cars with a weight >= 5
data <- data[data$wt >= 5, ]

# Clear original named region
clearNamedRegion(wb, name = "mtcars")

# Write subsetted data back
# Note: this is covering a smaller area now -
# writeNamedRegion automatically redefines the named region
# to the size/area of the data
writeNamedRegion(wb, data = data, name = "mtcars",
                 header = TRUE)

## End(Not run)
```

`clearRange-methods` *Clearing cell ranges in a workbook*

Description

Clears cell ranges in a workbook.

Usage

```
  ## S4 method for signature 'workbook,numeric'
clearRange(object, sheet, coords)
  ## S4 method for signature 'workbook,character'
clearRange(object, sheet, coords)
```

Arguments

`object`	The workbook to use
`sheet`	The name or index of the worksheet in which to clear cell ranges
`coords`	Numeric vector of length 4 or numeric matrix with 4 columns where the elements of the vector or rows in the matrix refer to the coordinates of the top-left and bottom-right corners of the ranges to clear. I.e. a vector or each row specifies the coordinates {top row, left column, bottom row, right column}. You may use `aref2idx` to generate such a matrix.

Details

Clearing a cell range means to clear all the cells associated with that range.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, clearSheet, clearNamedRegion, clearRangeFromReference, clearSheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of
# package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx",
                             package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Clear range from top left corner (4,2) ^= B4 to
# bottom right corner (6,4) ^= D6
clearRange(wb, sheet = "mtcars", coords = c(4, 2, 6, 4))

# Clear two ranges in one go ...
mat = matrix(c(5, 1, 6, 4, 5, 7, 7, 9), ncol = 4,
             byrow = TRUE)
clearRange(wb, sheet = "mtcars", coords = mat)

# The above is equivalent to ...
clearRange(wb, sheet = "mtcars",
           coords = aref2idx(c("A5:D6", "G5:I7")))

# This in turn is the same as ...
clearRangeFromReference(wb, reference = c("mtcars!A5:D6",
                        "mtcars!G5:I7"))

## End(Not run)
```

---

clearRangeFromReference-methods

Clearing cell ranges in a workbook

Description

Clears cell ranges specified by area reference in a workbook.

Usage

```
  ## S4 method for signature 'workbook,character'
clearRangeFromReference(object, reference)
```

Arguments

object	The workbook to use
reference	character specifying an area reference in the form 'SheetX!A7:B19'

Details

Clearing a cell range means to clear all the cells associated with that range. This method is very similar to clearRange.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, clearSheet, clearNamedRegion, clearRange, clearSheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of
# package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx",
                             package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Clear ranges A5:D6 and G5:I7 on sheet mtcars
clearRangeFromReference(wb, reference = c("mtcars!A5:D6",
                        "mtcars!G5:I7"))

## End(Not run)
```

---

clearSheet-methods *Clearing worksheets in a workbook*

Description

Clears worksheets with specified names or indices in a workbook.

Usage

```
  ## S4 method for signature 'workbook,numeric'
clearSheet(object, sheet)
  ## S4 method for signature 'workbook,character'
clearSheet(object, sheet)
```

Arguments

object	The workbook to use
sheet	The name or the index of the worksheet to clear

Details

Clearing a worksheet means to clear all the cells in that worksheet. Consequently, the saved workbook should be smaller in size. Clearing a worksheet can be useful if data sets in a worksheet need to be replaced, i.e. data are first read, modified in R and finally written back to the worksheet. Without clearing the worksheet first, (parts of) the original data may still be visible if they occupied a larger range of the worksheet.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, clearNamedRegion, clearRange, clearRangeFromReference

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of
# package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx",
                             package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Clear worksheets named 'mtcars' and 'mtcars2'
clearSheet(wb, sheet = c("mtcars", "mtcars2"))

# Clear 3rd worksheet
clearSheet(wb, sheet = 3)

## End(Not run)
```

---

cloneSheet-methods *Cloning/copying worksheets*

Description

Clones (copies) a worksheet in a workbook.

Usage

```
## S4 method for signature 'workbook,numeric'
cloneSheet(object,sheet,name)
## S4 method for signature 'workbook,character'
cloneSheet(object,sheet,name)
```

Arguments

object	The workbook to use
sheet	The name or index of the worksheet to clone
name	The name to assign to the cloned worksheet. Throws an exception if the name to assign is the name of an already existing worksheet.

Details

If any worksheet-scoped named ranges are present on the original sheet, these named ranges will not be present on the cloned worksheet.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createSheet, removeSheet, renameSheet, getSheets, existsSheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Clone the 'mtcars' worksheet and assign it the name 'mtcars cloned'
cloneSheet(wb, sheet = "mtcars", name = "mtcars cloned")

## End(Not run)
```

col2idx *Converting Excel column names to indices*

Description

Converts Excel column names to indices.

Usage

```
col2idx(x)
```

Arguments

x	Character vector of Excel column names (e.g. "A", "AF", ...)

Value

Returns a vector of integers representing the corresponding column indices. Note that passing invalid column name references may result in an arbitrary number.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

idx2col, cref2idx, idx2cref, idx2aref, aref2idx, aref

Examples

```
## Not run:
col2idx(c("A", "BTG"))

## End(Not run)
```

configurePOI *Configuring Apache POI*

Description

Configures Apache POI and related components.

Usage

```
configurePOI(zip_max_files = 1000L, zip_min_inflate_ratio = 0.001,
  zip_max_entry_size = 0xFFFFFFFF, zip_max_text_size = 10*1024*1024,
  zip_entry_threshold_bytes = -1L, max_size_byte_array = -1L,
  java_io_tmpdir = tempdir())
```

Arguments

`zip_max_files`	Integer scalar specifying the maximum number of files allowed inside an *.xlsx file. Defaults to `1000`.
`zip_min_inflate_ratio`	Numeric scalar specifying the ratio between de- and inflated bytes to detect zip-bombs. If the compression ratio is better than the specified number an error will be thrown. Defaults to `0.001`.
`zip_max_entry_size`	Integer scalar specifying the maximum file size of a single zip entry in an *.xlsx file. Defaults to 4'294'967'295 bytes, which is 4GB.
`zip_max_text_size`	Integer scalar specifying the maximum number of characters of text that are extracted before an error is thrown. Defaults to 10'485'760.
`zip_entry_threshold_bytes`	Integer scalar specifying the number of bytes at which a zip entry is regarded as too large for holding in memory and the data is put in a temp file instead. Defaults to `-1L`, meaning temp files are not used and that zip entries with more than 2GB of data after decompressing will fail. `0L` means all zip entries are stored in temp files.
`max_size_byte_array`	Integer scalar specifying the maximum number of bytes that should be possible to be allocated in a single step. Increasing this limit can help if you are dealing with large Excel files, but note that this may demand a larger heap space (see option `java.parameters`; e.g. `options(java.parameters = "-Xmx8192m")`. Defaults to -1, which means that record-specific limits apply.)
`java_io_tmpdir`	Directory to hold POI temporary files and generally any temporary files produced by the Java subprocess. Defaults to the R session temporary directory `tempdir()`

Details

Many of the settings exposed here exist for security reasons to prevent excessive memory consumption and protect against security vulnerabilities when processing documents provided by untrusted sources.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

Apache POI configuration: https://poi.apache.org/components/configuration.html

Examples

```
## Not run:
configurePOI(zip_max_files = 5000L, max_size_byte_array = 250000000L)

## End(Not run)
```

---

createCellStyle-methods

Creating custom named and anonymous cell styles

Description

Creates a custom named or anonymous cellstyle.

Usage

```
## S4 method for signature 'workbook,character'
createCellStyle(object,name)
```

Arguments

object	The workbook to use
name	The name of the new cellstyle to create. Omit to create an anonymous cellstyle.

Details

Creates a named cellstyle with the specified name. Named cell styles may be used in conjunction with the *name prefix* style action (see setStyleAction) or may also be used directly with the method setCellStyle. Named cell styles can easily be changed from within Excel using the cell styles menu.

If name is missing, an anonymous cell style is created. Anonymous cell styles can be used in conjunction with the setCellStyle method.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, getOrCreateCellStyle, existsCellStyle, setStyleAction, setStyleNamePrefix, setCellStyle, setDataFormat, setBorder, setFillBackgroundColor, setFillForegroundColor, setFillPattern, setWrapText

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("createCellstyles.xlsx", create = TRUE)

# We don't set a specific style action in this demo, so the
# default 'XLConnect' will be used (XLC$"STYLE_ACTION.XLCONNECT")

# Create a sheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' referring to the sheet
# called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$C$4")

# Write built-in data set 'mtcars' to the above defined named region.
# This will use the default style action 'XLConnect'.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Now let's color all weight cells of cars with a weight > 3.5 in red
# (mtcars$wt > 3.5)

# First, create a corresponding (named) cell style
heavyCar <- createCellStyle(wb, name = "HeavyCar")

# Specify the cell style to use a solid foreground color
setFillPattern(heavyCar, fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(heavyCar, color = XLC$"COLOR.RED")

# Which cars have a weight > 3.5 ?
rowIndex <- which(mtcars$wt > 3.5)

# NOTE: The mtcars data.frame has been written offset with
# top left cell C4 - and we have also written a header row!
# So, let's take that into account appropriately. Obviously,
# the two steps could be combined directly into one ...
rowIndex <- rowIndex + 4

# The same holds for the column index
colIndex <- which(names(mtcars) == "wt") + 2

# Set the 'HeavyCar' cell style for the corresponding cells.
# Note: the row and col arguments are vectorized!
setCellStyle(wb, sheet = "mtcars", row = rowIndex, col = colIndex,
             cellstyle = heavyCar)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
```

```
file.remove("createCellstyles.xlsx")

## End(Not run)
```

---

createFreezePane-methods

Creating a freeze pane on a worksheet

Description

Creates a freeze pane on a specified worksheet.

Usage

```
## S4 method for signature 'workbook,character'
createFreezePane(object, sheet, colSplit, rowSplit, leftColumn, topRow)
## S4 method for signature 'workbook,numeric'
createFreezePane(object, sheet, colSplit, rowSplit, leftColumn, topRow)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet on which to create a freeze pane
colSplit	Horizontal position of freeze (as column index or name)
rowSplit	Vertical position of freeze (as number of rows)
leftColumn	Left column (as column index or name) visible in right pane. If not specified, the default is leftColumn=colSplit
topRow	Top row (as index) visible in bottom pane. If not specified, the default is topRow=rowSplit

Note

To keep an area of a worksheet visible while you scroll to another area of the worksheet, you can lock specific rows or columns in one area by freezing or splitting panes.

When you freeze panes, you keep specific rows or columns visible when you scroll in the worksheet. For example, you might want to keep row and column labels visible as you scroll.

When you split panes, you create separate worksheet areas that you can scroll within, while rows or columns in the non-scrolled area remain visible.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

References

How to create a freeze pane/split pane in Office 2007 https://support.microsoft.com/en-us/office/freeze-panes-to-lock-rows-and-columns-dab2ffc9-020d-4026-8121-67dd25f2508f?ocmsassetid=hp001217048&correlationid=b4f5baeb-b622-4487-a96f-514d2f00208a&ui=en-us&rs=en-us&ad=us

See Also

workbook createSplitPane removePane

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("freezePaneTest.xlsx", create = TRUE)

# Create a worksheet named 'Sheet1'
createSheet(wb, name = "Sheet1")

# Create a freeze pane on Sheet1, using as reference position the 5th column and the 5th row,
# showing the 10th column as the leftmost visible one in the right pane
# and the 10th row as the top visible one in the bottom pane.
createFreezePane(wb, "Sheet1", 5, 5, 10, 10)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("freezePaneTest.xlsx")

## End(Not run)
```

---

createName-methods *Creating names in a workbook*

Description

Creates a named range for a specified formula in a workbook.

Usage

```
## S4 method for signature 'workbook'
createName(object, name, formula, overwrite, worksheetScope)
```

Arguments

object	The workbook to use
name	The name of the range to be created
formula	Excel formula specifying the value / data the name refers to
overwrite	If a name with the same name already exists and overwrite = TRUE, then this name is removed first before the new one is created. If a name already exists and overwrite = FALSE, then an exception is thrown. The default value for overwrite is FALSE.
worksheetScope	Optional - specific worksheet the name should be scoped to. If unspecified the name will be scoped to the whole workbook.

Details

Creates a named range called name for the specified formula.

The formula should be specified as you would type it in Excel. Make sure that the worksheets, functions, ... exist that you are referring to in the formula.

The name, formula and overwrite arguments are vectorized such that multiple names can be created in one method call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

What are named regions/ranges?
https://web.archive.org/web/20240821110221/https://www.officearticles.com/excel/named_ranges_in_microsoft_excel.htm
How to create named regions/ranges?
https://www.youtube.com/watch?v=iAE9a0uRtpM

See Also

workbook, removeName, existsName, getDefinedNames, readNamedRegion, writeNamedRegion

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("createName.xlsx", create = TRUE)

# Create a worksheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' on the sheet called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$A$1")
```

```
# Write built-in data set 'mtcars' to the above defined named region
writeNamedRegion(wb, mtcars, name = "mtcars")

# Save workbook
saveWorkbook(wb)

# clean up
file.remove("createName.xlsx")

## End(Not run)
```

---

createSheet-methods *Creating worksheets in a workbook*

Description

Creates worksheets with specified names in a workbook.

Usage

```
  ## S4 method for signature 'workbook'
createSheet(object, name)
```

Arguments

object	The workbook to use
name	The name of the sheet to create

Details

Creates a worksheet with the specified name if it does not already exist. Note that the naming of worksheets needs to be in line with Excel's convention, otherwise an exception will be thrown. For example, worksheet names cannot be longer than 31 characters. Also note that the name argument is vectorized, so multiple worksheets can be created in one method call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, removeSheet, renameSheet, existsSheet, getSheets, cloneSheet

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("createSheet.xlsx", create = TRUE)

# Create a worksheet called 'CO2'
createSheet(wb, name = "CO2")

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("createSheet.xlsx")

## End(Not run)
```

`createSplitPane-methods`

Creating a split pane on a worksheet

Description

Creates a split pane on a specified worksheet.

Usage

```
## S4 method for signature 'workbook,character'
createSplitPane(object,sheet,xSplitPos,ySplitPos,leftColumn,topRow)
## S4 method for signature 'workbook,numeric'
createSplitPane(object,sheet,xSplitPos,ySplitPos,leftColumn,topRow)
```

Arguments

`object`	The workbook to use
`sheet`	The name or index of the sheet on which to create a split pane
`xSplitPos`	Horizontal position of split (in 1/20th of a point)
`ySplitPos`	Vertical position of split (in 1/20th of a point)
`leftColumn`	Left column (as index or column name) visible in right pane
`topRow`	Top row visible in bottom pane

Note

To keep an area of a worksheet visible while you scroll to another area of the worksheet, you can lock specific rows or columns in one area by freezing or splitting panes.

When you freeze panes, you keep specific rows or columns visible when you scroll in the worksheet. For example, you might want to keep row and column labels visible as you scroll.

When you split panes, you create separate worksheet areas that you can scroll within, while rows or columns in the non-scrolled area remain visible.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

References

How to create a freeze pane/split pane in Office 2007 https://support.microsoft.com/en-us/office/freeze-panes-to-lock-rows-and-columns-dab2ffc9-020d-4026-8121-67dd25f2508f?ocmsassetid=hp001217048&correlationid=b4f5baeb-b622-4487-a96f-514d2f00208a&ui=en-us&rs=en-us&ad=us

See Also

workbook createFreezePane removePane

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("splitPaneTest.xlsx", create = TRUE)

# Create a worksheet named 'Sheet1'
createSheet(wb, name = "Sheet1")

# Create a split pane on Sheet1, with coordinates (10000, 5000) expressed as 1/20th of a point,
# 10 (-> J) as left column visible in right pane and 10 as top row visible in bottom pane
createSplitPane(wb, "Sheet1", 10000, 5000, 10, 10)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("splitPaneTest.xlsx")

## End(Not run)
```

---

cref2idx *Converting Excel cell references to indices*

Description

Converts Excel cell references to row & column indices

Usage

```
cref2idx(x)
```

Arguments

x	Character vector of Excel cell references (e.g. "$A$20", "B18", ...)

Value

Returns a numeric matrix with two columns and as many rows as cell references that have been provided. The first column represents the row indices and the second column represents the column indices.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

idx2cref, col2idx, idx2col, idx2aref, aref2idx, aref

Examples

```
## Not run:
cref2idx(c("$A$20", "B18"))

## End(Not run)
```

---

existsCellStyle-methods

Retrieving named cell styles

Description

Checks whether a named cell style exists in a workbook.

Usage

```
## S4 method for signature 'workbook'
existsCellStyle(object,name)
```

Arguments

object	The workbook to use
name	The name of the cellstyle to check

Details

Checks whether the cellstyle with the specified name exists.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, createCellStyle, getOrCreateCellStyle

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("existsCellStyle.xlsx", create = TRUE)

# Cell style 'MyStyle' does not exist yet
stopifnot(!existsCellStyle(wb, "MyStyle"))

# Create the style "MyStyle"
createCellStyle(wb, "MyStyle")

# And now it is here
stopifnot(existsCellStyle(wb, "MyStyle"))

# clean up
file.remove("existsCellStyle.xlsx")

## End(Not run)
```

---

existsName-methods *Checking existence of named ranges in a workbook*

Description

Checks the existence of a named range in a workbook.

Usage

```
## S4 method for signature 'workbook'
existsName(object, name, worksheetScope)
```

Arguments

object	The workbook to use
name	The name to check for
worksheetScope	Optional - the specific worksheet to check

Details

Returns TRUE if the specified name exists and FALSE otherwise. Note that the name argument is vectorized and therefore multiple names can be checked for existence in one method call.

If worksheetScope is provided, TRUE will be returned only if a matching named range exists in the local scope of the specified sheet. To explicitly match only in the global scope, pass "" as the value.

If option XLConnect.setCustomAttributes is TRUE (default FALSE), the worksheet scope in which the name is defined is set as attribute worksheetScope on the result.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, removeName, getDefinedNames, readNamedRegion, writeNamedRegion

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Check if the name 'mtcars' exists
# (should return TRUE since the name is defined as 'mtcars!$A$1:$K$33')
existsName(wb, name = "mtcars")

# check if the worksheet-scoped name 'iris' exists
options(XLConnect.setCustomAttributes = TRUE)
wb <- loadWorkbook("demoFiles/iris.xlsx")

# should return TRUE with worksheet scope "iris"
res <- existsName(wb, name = "iris")
res
attributes(res)

## End(Not run)
```

---

existsSheet-methods *Checking for existence of worksheets in a workbook*

Description

Checks the existence of a worksheet in a workbook.

Usage

```
## S4 method for signature 'workbook'
existsSheet(object,name)
```

Arguments

object	The workbook to use
name	The sheet name to check for

Details

Checks if the specified worksheet exists. Returns TRUE if it exists, otherwise FALSE. The name argument is vectorized which allows to check for existence of multiple worksheets with one call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createSheet, removeSheet, renameSheet, getSheets, cloneSheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Check for existence of a worksheet called 'mtcars'
existsSheet(wb, "mtcars")

## End(Not run)
```

---

extraction-methods *Workbook data extraction & replacement operators*

Description

Operators that allow to extract/replace data from/on a workbook.

Arguments

x	The workbook object to use
i	Name of worksheet ([, [<-) or name of Excel name ([[, [[<-) to extract or replace
j	Only used with [[<-: Optional formula to define the Excel name if it does not yet exist on the workbook.
drop	Not used
value	Data object used for replacement
...	Arguments passed to the corresponding underlying function to read/write the data

Details

The workbook extraction operators are basically syntactic sugar for the common methods readWorksheet ([), writeWorksheet ([<-), readNamedRegion ([[), writeNamedRegion ([[<-).

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, readWorksheet, writeWorksheet, readNamedRegion, writeNamedRegion

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("extraction.xlsx", create = TRUE)

# Write mtcars data set on a worksheet named 'mtcars1'.
# Note: The 'mtcars1' sheet will be created automatically if it does
# not exist yet. Also, default values for other writeWorksheet arguments
# hold, i.e. the data set is written starting at the top left corner.
wb["mtcars1"] = mtcars

# Write mtcars data set on a worksheet named 'mtcars2'.
# Again, the 'mtcars2' worksheet is created automatically.
# Additionally specify arguments passed to the underlying method
# writeWorksheet.
wb["mtcars2", startRow = 6, startCol = 11, header = FALSE] = mtcars

# Read worksheets 'mtcars1' and 'mtcars2'.
# Note: The default arguments hold for the underlying method
# readWorksheet.
wb["mtcars1"]
wb["mtcars2"]
```

```
# Write mtcars data set to a named region named 'mtcars3'. Since
# it doesn't exist yet we also need to specify the formula to
# define it. Also note that the sheet 'mtcars3' referenced in the
# formula does not yet exist - it will be created automatically!
# Moreover, default values for other writeNamedRegion arguments hold.
wb[["mtcars3", "mtcars3!$B$7"]] = mtcars

# Redefine named region 'mtcars3'. Note that no formula specification
# is required since named region is already defined (see above example).
wb[["mtcars3"]] = mtcars

# Write mtcars data set to a named region 'mtcars4'. Since the named
# region does not yet exist a formula specification is required. Also,
# additional arguments are specified that are passed to the underlying
# method writeNamedRegion.
wb[["mtcars4", "mtcars4!$D$8", rownames = "Car"]] = mtcars

# Read the named regions 'mtcars3' and 'mtcars4'.
# Note: Default values hold for the underlying method readNamedRegion.
wb[["mtcars3"]]
wb[["mtcars4"]]

# clean up
file.remove("extraction.xlsx")

## End(Not run)
```

`extractSheetName` *Extracting the sheet name from a formula*

Description

Extracts the sheet name from a formula of the form <SHEET_NAME>!<CELL_ADDRESS>

Usage

```
extractSheetName(formula)
```

Arguments

`formula`	Formula string of the form <SHEET_NAME>!<CELL_ADDRESS>. Note that the validity of the formula won't be checked.

Value

Returns the name of the sheet referenced in the formula. For quoted sheet names (required if names contain e.g. whitespaces or exclamation marks (!)) in formulas the function returns the unquoted name.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

Examples

```
## Not run:
extractSheetName(c("MySheet!$A$1", "'My Sheet'!$A$1", "'My!Sheet'!$A$1"))

## End(Not run)
```

---

getActiveSheetIndex-methods

Querying the active worksheet index

Description

Queries the index of the active worksheet in a workbook.

Usage

```
## S4 method for signature 'workbook'
getActiveSheetIndex(object)
```

Arguments

object	The workbook to use

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getActiveSheetName

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query the active sheet index
activeSheet <- getActiveSheetIndex(wb)

## End(Not run)
```

getActiveSheetName-methods
Querying the active worksheet name

Description

Queries the name of the active worksheet in a workbook.

Usage

```
## S4 method for signature 'workbook'
getActiveSheetName(object)
```

Arguments

object	The workbook to use

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getActiveSheetIndex

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query the active sheet name
activeSheet <- getActiveSheetName(wb)

## End(Not run)
```

getBoundingBox-methods

Querying the coordinates of a worksheet bounding box

Description

This function queries the coordinates of a bounding box in an Excel worksheet. A bounding box is the rectangular region of minimum size containing all the non-empty cells in a sheet.

Usage

```
## S4 method for signature 'workbook,character'
getBoundingBox(object,sheet,startRow,startCol,endRow,endCol,autofitRow,autofitCol)
## S4 method for signature 'workbook,numeric'
getBoundingBox(object,sheet,startRow,startCol,endRow,endCol,autofitRow,autofitCol)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet from which to get the bounding box
startRow	Start reference row for the bounding box. Defaults to 0 meaning that the start row is determined automatically.
startCol	Start reference column for the bounding box. Defaults to 0 meaning that the start column is determined automatically.
endRow	End reference row for the bounding box. Defaults to 0 meaning that the end row is determined automatically.
endCol	End reference column for the bounding box. Defaults to 0 meaning that the end column is determined automatically.
autofitRow	logical specifying if leading and trailing empty rows should be skipped. Defaults to TRUE.
autofitCol	logical specifying if leading and trailing empty columns should be skipped. Defaults to TRUE.

Details

The result is a matrix containing the following coordinates:
[1,] top left row
[2,] top left column
[3,] bottom right row
[4,] bottom right column

In case more than one sheet is selected, the result matrix will contain a column for each sheet.

The bounding box resolution algorithm works as follows:
If `startRow <= 0` then the first available row in the sheet is assumed. If `endRow <= 0` then the last available row in the sheet is assumed. If `startCol <= 0` then the minimum column between `startRow` and `endRow` is assumed. If `endCol <= 0` then the maximum column between `startRow` and `endRow` is assumed. The arguments `autofitRow` and `autofitCol` (both defaulting to `TRUE`) can be used to skip leading and trailing empty rows even in case `startRow`, `endRow`, `startCol` and `endCol` are specified to values `> 0`. This can be useful if data is expected within certain given boundaries but the exact location is not available.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook

Examples

```
## Not run:
# multiregion xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/multiregion.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query bounding box for the second sheet
print(getBoundingBox(wb, sheet="SecondSheet"))

# Query bounding box for the first sheet, selecting the columns from 5 to 8
print(getBoundingBox(wb, sheet="FirstSheet", startCol=5, endCol=8))

## End(Not run)
```

---

getCellFormula-methods

Retrieving formula definitions from cells

Description

Retrieves a cell formula from a workbook.

Usage

```
## S4 method for signature 'workbook,character'
getCellFormula(object,sheet,row,col)
## S4 method for signature 'workbook,numeric'
getCellFormula(object,sheet,row,col)
```

Arguments

object	The workbook to use
sheet	The name or index of the worksheet containing the cell
row	The one-based row index of the cell to query
col	The one-based column index of the cell to query

Details

Retrieves the formula of the specified cell as a character, without the initial = character displayed in Excel. Raises an error if the specified cell is not a formula cell.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, setCellFormula

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("cellFormula.xlsx", create = TRUE)

createSheet(wb, "Formula")

# Assign a formula to A1
setCellFormula(wb, "Formula", 1, 1, "SUM($B$1:$B$29)")

# Returns the formula for Sheet1!A1
getCellFormula(wb, "Formula", 1, 1)
# The same with a numeric sheet index
getCellFormula(wb, 1, 1, 1)

# clean up
file.remove("cellFormula.xlsx")

## End(Not run)
```

---

getCellStyle-methods *Retrieving named cell styles*

Description

Retrieves a named cell style from a workbook.

Usage

```
## S4 method for signature 'workbook'
getCellStyle(object,name)
```

Arguments

object	The workbook to use
name	The name of the cellstyle to retrieve

Details

Retrieves the cellstyle with the specified name.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setStyleAction, createCellStyle, getOrCreateCellStyle, existsCellStyle, setStyleNamePrefix, setCellStyle, setDataFormat, setBorder, setFillBackgroundColor, setFillForegroundColor, setFillPattern, setWrapText

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("getCellstyles.xlsx", create = TRUE)

# You wouldn't usually ignore the return value here...
createCellStyle(wb, 'Header')

# ... but if you did it doesn't hurt.
cs <- getCellStyle(wb, 'Header')

# Specify the cell style to use a solid foreground color
setFillPattern(cs, fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(cs, color = XLC$"COLOR.RED")


# clean up
file.remove("getCellstyles.xlsx")

## End(Not run)
```

getCellStyleForType-methods
Querying the cell style per data type for the DATATYPE style action

Description

Queries the cell style for a specific data type as used by the DATATYPE style action.

Usage

```
## S4 method for signature 'workbook'
getCellStyleForType(object,type)
```

Arguments

object	The workbook to use
type	The data type for which to get the cellstyle.

Details

Based on the (cell) data type the DATATYPE style action (see setStyleAction) sets the cellstyle for the corresponding cells. The data type is normally specified via a corresponding data type constant from the XLC object.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, setCellStyleForType, setStyleAction

Examples

```
## Not run:
file.copy(system.file("demoFiles/template2.xlsx",
                      package = "XLConnect"),
          "datatype.xlsx", overwrite = TRUE)

# Load workbook
wb <- loadWorkbook("datatype.xlsx")

# Get current (existing) cell style for numerics
cs <- getCellStyleForType(wb, XLC$"DATA_TYPE.NUMERIC")
# Could also say cs <- getCellStyleForType(wb, "numeric")

# Change style
setBorder(cs, side = c("bottom", "right"), type = XLC$"BORDER.THICK",
```

```
            color = c(XLC$"COLOR.BLACK", XLC$"COLOR.RED"))

# Set style action to 'datatype'
setStyleAction(wb, XLC$"STYLE_ACTION.DATATYPE")

# Write built-in data set 'mtcars' to the named region
# 'mtcars' as defined by the Excel template.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Save workbook
saveWorkbook(wb)

# clean up
file.remove("datatype.xlsx")

## End(Not run)
```

`getDefinedNames-methods`

Retrieving defined names in a workbook

Description

Retrieves the defined names in a `workbook`.

Usage

```
## S4 method for signature 'workbook'
getDefinedNames(object, validOnly, worksheetScope)
```

Arguments

`object`	The `workbook` to use
`validOnly`	If `validOnly = TRUE` only names with valid references are returned. Valid references are ones not starting with #REF! or #NULL! - which could result e.g. due to a missing sheet reference. The default value for `validOnly` is `TRUE`.
`worksheetScope`	Optional - the name of the worksheet in which the names are scoped; to only query names in the global scope, use the value `""`

Details

If option `XLConnect.setCustomAttributes` is `TRUE` (default `FALSE`), a list of the worksheet scopes in which the names were found is set as attribute `worksheetScope` on the result.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, removeName, existsName, readNamedRegion, writeNamedRegion

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Retrieve defined names with valid references
getDefinedNames(wb)

## End(Not run)
```

---

getForceFormulaRecalculation-methods

Querying the coordinates of the range reference by an Excel name

---

Description

Queries the "force formula recalculation" flag on an Excel worksheet.

Usage

```
## S4 method for signature 'workbook,character'
getForceFormulaRecalculation(object,sheet)
## S4 method for signature 'workbook,numeric'
getForceFormulaRecalculation(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to query. This argument is vectorized such that multiple sheets can be queried with one method call. If sheet = "*", the flag is queried for all sheets in the workbook (in the order as returned by getSheets).

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getSheets, setForceFormulaRecalculation

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Ask whether Excel will automatically recalculate formulas on sheet mtcars
print(getForceFormulaRecalculation(wb, sheet = "mtcars"))

## End(Not run)
```

---

getLastColumn-methods *Querying the last (non-empty) column on a worksheet*

Description

Queries the last (non-empty) column on a worksheet.

Usage

```
## S4 method for signature 'workbook,character'
getLastColumn(object,sheet)
## S4 method for signature 'workbook,numeric'
getLastColumn(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet of which to query the last column

Details

Returns the (1-based) numeric index of the last non-empty column in the specified worksheet.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query the last row of the 'mtcars' worksheet
getLastColumn(wb, "mtcars")

# Query the last row of the 'mtcars2' worksheet
getLastColumn(wb, "mtcars2")

# Query the last row of the 'mtcars3' worksheet
getLastColumn(wb, "mtcars3")

## End(Not run)
```

---

getLastRow-methods *Querying the last (non-empty) row on a worksheet*

Description

Queries the last (non-empty) row on a worksheet.

Usage

```
## S4 method for signature 'workbook,character'
getLastRow(object,sheet)
## S4 method for signature 'workbook,numeric'
getLastRow(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet of which to query the last row

Details

Returns the numeric index of the last non-empty row in the specified worksheet.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query the last row of the 'mtcars' worksheet
getLastRow(wb, "mtcars")

# Query the last row of the 'mtcars2' worksheet
getLastRow(wb, "mtcars2")

# Query the last row of the 'mtcars3' worksheet
getLastRow(wb, "mtcars3")

## End(Not run)
```

---

getOrCreateCellStyle-methods

Retrieving or creating named cell styles

Description

Retrieves or creates cell styles in workbooks.

Usage

```
## S4 method for signature 'workbook,character'
getOrCreateCellStyle(object,name)
```

Arguments

object	The workbook to use
name	The name of the cellstyle to retrieve or to create

Details

Retrieves an existing cellstyle if it exists or creates a new one if it does not exist yet.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, createCellStyle, existsCellStyle

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("getOrCreateCellStyle.xlsx", create = TRUE)

# The first time, the style does not exist yet and gets created
myStyle <- getOrCreateCellStyle(wb, name = "MyStyle")

# The second time, we retrieve the already existing style
myStyle <- getOrCreateCellStyle(wb, name = "MyStyle")

# clean up
file.remove("getOrCreateCellStyle.xlsx")

## End(Not run)
```

---

getReferenceCoordinates-methods

Querying the coordinates of the range reference by an Excel name

---

Description

(DEPRECATED) Queries the coordinates of an Excel named range in a workbook.

Usage

```
## S4 method for signature 'workbook'
getReferenceCoordinates(object,name)
```

Arguments

object	The workbook to use
name	The name to query. This argument is vectorized such that multiple names can be queried with one method call.

Note

This function is deprecated. Use getReferenceCoordinatesForName instead.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, existsName, removeName, getReferenceFormula

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query reference coordinate for name 'mtcars'
print(getReferenceCoordinatesForName(wb, name = "mtcars"))

## End(Not run)
```

---

getReferenceCoordinatesForName-methods

Querying the coordinates of the range reference by an Excel name

---

Description

Queries the coordinates of an Excel named range in a workbook.

Usage

```
## S4 method for signature 'workbook'
getReferenceCoordinatesForName(object,name, worksheetScope)
```

Arguments

object	The workbook to use
name	The name to query. This argument is vectorized such that multiple names can be queried with one method call.
worksheetScope	Optional, the name of the worksheet to use for resolving the named region

Details

If worksheetScope is defined, only coordinates for a range scoped strictly to the specified worksheet are returned. To explicitly only query for named ranges in the global scope, pass "" as the value.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, existsName, removeName, getReferenceFormula, getReferenceCoordinatesForTable

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query reference coordinate for name 'mtcars'
print(getReferenceCoordinatesForName(wb, name = "mtcars"))

## End(Not run)
```

---

getReferenceCoordinatesForTable-methods

Querying the coordinates of the range of an Excel table

Description

Queries the coordinates of an Excel table (Office 2007+) in a workbook.

Usage

```
## S4 method for signature 'workbook,numeric'
getReferenceCoordinatesForTable(object,sheet,table)
## S4 method for signature 'workbook,character'
getReferenceCoordinatesForTable(object,sheet,table)
```

Arguments

object	The workbook to use
sheet	The index or name of the worksheet on which to look for the specified table
table	The name of the table to query. This argument is vectorized such that multiple tables can be queried with one method call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, existsName, removeName, getReferenceFormula, getReferenceCoordinatesForName

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query reference coordinates for table 'MtcarsTable' on sheet
# 'mtcars_table'
print(getReferenceCoordinatesForTable(wb, sheet = "mtcars_table",
                                      table = "MtcarsTable"))

## End(Not run)
```

`getReferenceFormula-methods`

Querying reference formulas of Excel names

Description

Queries the reference formula of an Excel named range in a `workbook`.

Usage

```
## S4 method for signature 'workbook'
getReferenceFormula(object,name, worksheetScope)
```

Arguments

`object`	The `workbook` to use
`name`	The named range to query. This argument is vectorized such that multiple names can be queried with one method call.
`worksheetScope`	Optional - the name of the worksheet in which the name is scoped; if undefined a matching name in any scope may be returned. To specify global scope only, use the value `""`

.

Details

If option `XLConnect.setCustomAttributes` is `TRUE` (default `FALSE`), the worksheet scope in which the queried name is defined is set as attribute `worksheetScope` on the result.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, existsName, removeName

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query reference formula for name 'mtcars'
print(getReferenceFormula(wb, name = "mtcars"))

## End(Not run)
```

---

getSheetPos-methods *Querying worksheet position*

Description

Queries the position of a worksheet in a workbook.

Usage

```
## S4 method for signature 'workbook,character'
getSheetPos(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name of the worksheet (character) to query. This argument is vectorized such that multiple worksheets can be queried with one method call.

Value

Returns the position index of the corresponding worksheet. Note that querying a non-existing worksheet results in a 0 index and does not throw an exception!

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, setSheetPos, getSheets

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query worksheet positions for the worksheets 'mtcars2', 'mtcars3',
# 'mtcars' and 'NotThere' (which actually does not exist)
print(getSheetPos(wb, sheet = c("mtcars2", "mtcars3", "mtcars", "NotThere")))

## End(Not run)
```

---

getSheets-methods *Querying available worksheets in a workbook*

Description

Returns all worksheet names in a workbook.

Usage

```
## S4 method for signature 'workbook'
getSheets(object)
```

Arguments

object	The workbook to use

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createSheet, removeSheet, renameSheet, getSheetPos, setSheetPos

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Query available worksheets
```

```
sheets <- getSheets(wb)

## End(Not run)
```

---

getTables-methods *Querying available Excel tables in a workbook*

Description

Queries the available Excel tables on the specified worksheet.

Usage

```
## S4 method for signature 'workbook,numeric'
getTables(object,sheet,simplify)
## S4 method for signature 'workbook,character'
getTables(object,sheet,simplify)
```

Arguments

object	The workbook to use
sheet	Index (integer) or name (character) of worksheet to query
simplify	logical specifying if the result should be simplified (defaults to TRUE). See details.

Details

Since this is a vectorized function (multiple sheets can be specified) the result is a named list (one component per sheet) if no simplification is applied. In cases where only one sheet is queried and simplify = TRUE (default) the result is simplified to a vector.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getSheets, readTable

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)
```

```
# Query available tables (table names) on sheet 'mtcars_table'
tables <- getTables(wb, sheet = "mtcars_table")

# ... or via sheet index
tables <- getTables(wb, sheet = 4)

## End(Not run)
```

`hideSheet-methods` *Hiding worksheets in a workbook*

Description

(Very) hides the specified worksheets in a `workbook`.

Usage

```
## S4 method for signature 'workbook,character'
hideSheet(object, sheet, veryHidden)
## S4 method for signature 'workbook,numeric'
hideSheet(object, sheet, veryHidden)
```

Arguments

`object`	The `workbook` to use
`sheet`	The name or index of the sheet to hide
`veryHidden`	If `veryHidden = TRUE`, the specified `sheet` is "very" hidden (see note), otherwise it is just hidden. Default is `FALSE`.

Details

The arguments `sheet` and `veryHidden` are vectorized such that multiple worksheets can be (very) hidden with one method call. An exception is thrown if the specified `sheet` does not exist.

Note

Note that hidden worksheets can be unhidden by users directly within Excel via standard functionality. Therefore Excel knows the concept of "very hidden" worksheets. These worksheets cannot be unhidden with standard Excel functionality but need programatic intervention to be made visible.

Also note that in case the specified worksheet to hide is the currently active worksheet, then `hideSheet` tries to set the active worksheet to the first non-hidden (not hidden and not very hidden) worksheet in the workbook. If there is no such worksheet, `hideSheet` will throw an exception.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, unhideSheet, isSheetHidden, isSheetVeryHidden, isSheetVisible

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("hiddenWorksheet.xlsx", create = TRUE)

# Write a couple of built-in data.frame's into sheets
# with corresponding name
for(obj in c("CO2", "airquality", "swiss")) {
  createSheet(wb, name = obj)
  writeWorksheet(wb, get(obj), sheet = obj)
}

# Hide sheet 'airquality';
# the sheet may be unhidden by a user from within Excel
# since veryHidden defaults to FALSE
hideSheet(wb, sheet = "airquality")

# Save workbook
saveWorkbook(wb)

# clean up
file.remove("hiddenWorksheet.xlsx")

## End(Not run)
```

---

idx2aref *Converting row and column based area references to Excel area references*

Description

Converts row & column based area references to Excel area references

Usage

```
idx2aref(x)
```

Arguments

x	Numeric (integer) matrix or vector of indices. If a matrix is provided it should have four columns with the first two columns representing the top left corner (row and column indices) and the third & fourth column representing the bottom right corner. If a vector is provided it will be converted to a matrix by filling the vector into a 4-column matrix by row.

Value

Returns a `character` vector of corresponding Excel area references.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`aref2idx`, `aref`, `idx2cref`, `cref2idx`, `idx2col`, `col2idx`

Examples

```
## Not run:
idx2aref(c(1, 1, 5, 4))

## End(Not run)
```

---

idx2col *Converting column indices to Excel column names*

Description

Converts column indices to Excel column names.

Usage

```
idx2col(x)
```

Arguments

x	Numeric (integer) vector of column indices

Value

Returns a character vector of corresponding Excel column names. Numbers <= `0` result in the empty string ("").

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

col2idx, idx2cref, cref2idx, idx2aref, aref2idx, aref

Examples

```
## Not run:
idx2col(c(1, 347))

## End(Not run)
```

---

idx2cref *Converting indices to Excel cell references*

Description

Converts row & column indices to Excel cell references

Usage

```
idx2cref(x, absRow = TRUE, absCol = TRUE)
```

Arguments

x	Numeric (integer) matrix or vector of indices. If a matrix is provided it should have two columns with the first column representing the row indices and the second column representing the column indices (i.e. each row represents a index-based cell reference). If a vector is provided it will be converted to a matrix by filling the vector into a 2-column matrix by row.
absRow	Boolean determining if the row index should be considered absolute. If TRUE (default), this will result in a '$'-prefixed row identifier.
absCol	Boolean determining if the column index should be considered absolute. If TRUE (default), this will result in a '$'-prefixed column identifier.

Value

Returns a character vector of corresponding Excel cell references.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

cref2idx, idx2col, col2idx, idx2aref, aref2idx, aref

Examples

```
## Not run:
idx2cref(c(5, 8, 14, 38))

## End(Not run)
```

---

isSheetHidden-methods *Checking if worksheets are hidden in a workbook*

Description

Checks if the specified worksheets are hidden (but not very hidden) in a workbook.

Usage

```
## S4 method for signature 'workbook,character'
isSheetHidden(object,sheet)
## S4 method for signature 'workbook,numeric'
isSheetHidden(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to check

Details

Returns TRUE if the specified sheet is hidden (not visible but also not very hidden), otherwise FALSE. sheet is vectorized such that multiple worksheets can be queried with one method call. An exception is thrown if the specified sheet does not exist.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, hideSheet, unhideSheet, isSheetVeryHidden, isSheetVisible

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("isSheetHidden.xlsx", create = TRUE)

# Write a couple of built-in data.frame's into sheets
# with corresponding name
for(obj in c("CO2", "airquality", "swiss")) {
  createSheet(wb, name = obj)
  writeWorksheet(wb, get(obj), sheet = obj)
}

# Hide sheet 'airquality'
hideSheet(wb, sheet = "airquality")

# Check if sheet 'airquality' is hidden;
# this should obviously return TRUE
isSheetHidden(wb, "airquality")

# Check if sheet 'swiss' is hidden;
# this should obviously return FALSE
isSheetHidden(wb, "swiss")


# clean up
file.remove("isSheetHidden.xlsx")

## End(Not run)
```

---

isSheetVeryHidden-methods

Checking if worksheets are very hidden in a workbook

Description

Checks if the specified worksheets are very hidden (but not just hidden) in a workbook.

Usage

```
## S4 method for signature 'workbook,character'
isSheetVeryHidden(object,sheet)
## S4 method for signature 'workbook,numeric'
isSheetVeryHidden(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to check

Details

Returns TRUE if the specified named sheet is very hidden (not visible but also not just hidden), otherwise FALSE. sheet is vectorized such that multiple worksheets can be queried with one method call. An exception is thrown if the specified sheet does not exist.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, hideSheet, unhideSheet, isSheetHidden, isSheetVisible

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("isSheetVeryHidden.xlsx", create = TRUE)

# Write a couple of built-in data.frame's into sheets
# with corresponding name
for(obj in c("CO2", "airquality", "swiss")) {
  createSheet(wb, name = obj)
  writeWorksheet(wb, get(obj), sheet = obj)
}

# Very hide sheet 'airquality'
hideSheet(wb, sheet = "airquality", veryHidden = TRUE)

# Hide sheet 'CO2'
hideSheet(wb, sheet = "CO2", veryHidden = FALSE)

# Check if sheet 'airquality' is very hidden;
# this should obviously return TRUE
isSheetVeryHidden(wb, "airquality")

# Check if sheet 'swiss' is very hidden;
# this should obviously return FALSE
isSheetVeryHidden(wb, "swiss")

# Check if sheet 'CO2' is very hidden;
# this should also return FALSE - the sheet
# is just hidden but not very hidden
isSheetVeryHidden(wb, "CO2")


# clean up
file.remove("isSheetVeryHidden.xlsx")

## End(Not run)
```

isSheetVisible-methods *Checking if worksheets are visible in a workbook*

Description

Checks if the specified worksheets are visible in a workbook.

Usage

```
## S4 method for signature 'workbook,character'
isSheetVisible(object,sheet)
## S4 method for signature 'workbook,numeric'
isSheetVisible(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to check

Details

Returns TRUE if the specified named sheet is visible (not hidden and not very hidden), otherwise FALSE. sheet is vectorized such that multiple worksheets can be queried with one method call. An exception is thrown if the specified sheet does not exist.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, hideSheet, unhideSheet, isSheetHidden, isSheetVeryHidden

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("isSheetVisible.xlsx", create = TRUE)

# Write a couple of built-in data.frame's into sheets
# with corresponding name
for(obj in c("CO2", "airquality", "swiss")) {
  createSheet(wb, name = obj)
  writeWorksheet(wb, get(obj), sheet = obj)
}
```

```
# Hide sheet 'CO2'
hideSheet(wb, sheet = "CO2", veryHidden = FALSE)

# Very hide sheet 'airquality'
hideSheet(wb, sheet = "airquality", veryHidden = TRUE)

# Check if sheet 'swiss' is visible;
# this should obviously return TRUE
isSheetVisible(wb, "swiss")

# Check if sheet 'CO2' is visible;
# this should obviously return FALSE
isSheetVisible(wb, "CO2")

# Check if sheet 'airquality' is visible;
# this should obviously return FALSE
isSheetVisible(wb, "airquality")


# clean up
file.remove("isSheetVisible.xlsx")

## End(Not run)
```

`loadWorkbook` *Loading Microsoft Excel workbooks*

Description

Loads or creates a Microsoft Excel `workbook` for further manipulation.

Usage

```
loadWorkbook(filename, create = FALSE, password = NULL)
```

Arguments

`filename`	Filename (absolute or relative) of Excel workbook to be loaded. Supported are Excel '97 (*.xls) and OOXML (Excel 2007+, *.xlsx) file formats. Paths are expanded using `path.expand`.
`create`	Specifies if the file should be created if it does not already exist (default is `FALSE`). Note that `create = TRUE` has no effect if the specified file exists, i.e. an existing file is loaded and not being recreated if `create = TRUE`.
`password`	Password to use when opening password protected files. The default `NULL` means no password is being used. This argument is ignored when creating new files using `create = TRUE`.

Value

Returns a `workbook` object for further manipulation.

Note

loadWorkbook is basically just a shortcut form of new("workbook", filename, create) with some additional error checking. As such it is the preferred way of creating workbook instances.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

Wikipedia: Office Open XML
https://en.wikipedia.org/wiki/Office_Open_XML

See Also

workbook, saveWorkbook

Examples

```
## Not run:
# Load existing demo Excel file 'mtcars.xlsx' from the XLConnect package
wb.mtcars <- loadWorkbook(system.file("demoFiles/mtcars.xlsx",
                          package = "XLConnect"))

# Create new workbook
wb.new <- loadWorkbook("myNewExcelFile.xlsx", create = TRUE)

# NOTE: The above statement does not write the file to disk!
# saveWorkbook(wb.new) would need to be called in order to write/save
# the file to disk!

# clean up
file.remove("myNewExcelFile.xlsx")

## End(Not run)
```

---

mergeCells-methods *Merging cells*

Description

Merges cells in a worksheet.

Usage

```
## S4 method for signature 'workbook,character'
mergeCells(object,sheet,reference)
## S4 method for signature 'workbook,numeric'
mergeCells(object,sheet,reference)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet on which to merge cells
reference	A cell range specification (character) in the form 'A1:B8'

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, unmergeCells, idx2cref

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("mergeCells.xlsx", create = TRUE)

# Create a worksheet called 'merge'
createSheet(wb, name = "merge")

# Merge the cells A1:B8 on the worksheet created above
mergeCells(wb, sheet = "merge", reference = "A1:B8")

# Save workbook
saveWorkbook(wb)

# clean up
file.remove("mergeCells.xlsx")

## End(Not run)
```

---

mirai *Mirai Solutions GmbH*

Description

Utility object to easily get to the Mirai Solutions GmbH web page. Just enter mirai in the R console.

Usage

```
mirai
```

References

Mirai Solutions GmbH https://mirai-solutions.ch

onErrorCell-methods *Behavior when error cells are detected*

Description

This function defines the behavior when reading data from a worksheet and error cells are detected.

Usage

```
## S4 method for signature 'workbook'
onErrorCell(object,behavior)
```

Arguments

object	The workbook to use
behavior	The behavior to follow when an error cell is detected. This is normally specified by a corresponding XLC error constant, i.e. either XLC$"ERROR.WARN" or XLC$"ERROR.STOP". XLC$"ERROR.WARN" means the error cell will be read as missing value (NA) and a corresponding warning will be generated (this is the default behavior). XLC$"ERROR.STOP" means that an exception will be thrown and further execution will be stopped immediately.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, readNamedRegion, readNamedRegionFromFile, readWorksheet, readWorksheetFromFile

Examples

```
## Not run:
# errorCell xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/errorCell.xlsx",
  package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Set error behavior to XLC$ERROR.WARN when detecting error cells
# Note: this is the default behavior
onErrorCell(wb, XLC$ERROR.WARN)
# Alternatively: wb$onErrorCell(XLC$ERROR.WARN)

# Read named region 'MyData' (with default header = TRUE)
data <- readNamedRegion(wb, name = "MyData")
```

```
# Now set error behavior to XLC$ERROR.STOP to immediately
# issue an exception and stop in case an error cell is
# detected
onErrorCell(wb, XLC$ERROR.STOP)
# Alternatively: wb$onErrorCell(XLC$ERROR.STOP)

# Read (again) named region 'MyData' (with default header = TRUE)
res <- try(readNamedRegion(wb, name = "MyData"))
# Did we get an error?
print(is(res, "try-error"))

## End(Not run)
```

`print-methods` *Print a workbook's filename*

Description

Prints the workbook's underlying filename.

Usage

```
## S4 method for signature 'workbook'
print(x,...)
```

Arguments

`x`	The workbook to print
`...`	Arguments passed on to standard `print`

Details

Prints the specified workbook's filename (see also the S4 `filename` slot of the workbook class).

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook

Examples

```
## Not run:
# Load existing demo Excel file 'mtcars.xlsx' from the XLConnect package
wb.mtcars <- loadWorkbook(system.file("demoFiles/mtcars.xlsx",
                          package = "XLConnect"))

# Print the workbook's underlying filename
print(wb.mtcars)

## End(Not run)
```

`readNamedRegion` *Reading named regions from a workbook*

Description

Reads named regions from a `workbook`.

Usage

```
## S4 method for signature 'workbook'
readNamedRegion(object, name, header, rownames, colTypes,
forceConversion, dateTimeFormat, check.names, useCachedValues, keep, drop, simplify,
readStrategy, worksheetScope)
```

Arguments

`object`	The `workbook` to use
`name`	The name of the named region to read
`header`	The argument `header` specifies if the first row should be interpreted as column names. The default value is `TRUE`.
`rownames`	Index (`numeric`) or name (`character`) of column that should be used as row names. The corresponding column will be removed from the data set. Defaults to `NULL` which means that no row names are applied. Row names must be either `integer` or `character`. Non-numeric columns will be coerced to `character`.
`colTypes`	Column types to use when reading in the data. Specified as a `character` vector of the corresponding type names (see `XLC`; `XLC$DATA_TYPE.<?>`). You may also use R class names such as `numeric`, `character`, `logical` and `POSIXt`. The types are applied in the given order to the columns - elements are recycled if necessary. Defaults to `character(0)` meaning that column types are determined automatically (see the Note section for more information). By default, type conversions are only applied if the specified column type is a more generic type (e.g. from Numeric to String) - otherwise `NA` is returned. The `forceConversion` flag can be set to force conversion into less generic types where possible.

forceConversion

logical specifying if conversions to less generic types should be forced. Defaults to FALSE meaning that if a column is specified to be of a certain type via the colTypes argument and a more generic type is detected in the column, then NA will be returned (example: column is specified to be DateTime but a more generic String is found). Specifying forceConversion = TRUE will try to enforce a conversion - if it succeeds the corresponding (converted) value will be returned, otherwise NA. See the Note section for some additional information.

dateTimeFormat Date/time format used when doing date/time conversions. Defaults to getOption("XLConnect.dateTimeFormat"). This should be a POSIX format specifier according to strptime although not all specifications have been implemented yet - the most important ones however are available.

check.names logical specifying if column names of the resulting data.frame should be checked to ensure that they are syntactically valid valid variable names and are not duplicated. See the check.names argument of data.frame. Defaults to TRUE.

useCachedValues

logical specifying whether to read cached formula results from the workbook instead of re-evaluating them. This is particularly helpful in cases for reading data produced by Excel features not supported in XLConnect like references to external workbooks. Defaults to FALSE, which means that formulas will be evaluated by XLConnect.

keep List of column names or indices to be kept in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments. Example: if name = c("NamedRegion1", "NamedRegion2", "NamedRegion3") and keep = c(1,2), keep will be internally converted into list(c(1,2)) and then replicated to match the number of named regions, i.e. keep = list(c(1,2), c(1,2), c(1,2)). The result is that the first two columns of each named region are kept. If keep = list(1,2) is specified, it will be replicated as list(1,2,1), i.e. respectively the first, second and first column of the named regions "NamedRegion1", "NamedRegion2", "NamedRegion3" will be kept.

drop List of column names or indices to be dropped in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments. Example: if name = c("NamedRegion1", "NamedRegion2", "NamedRegion3") and drop = c(1,2), drop will be internally converted into list(c(1,2)) and then replicated to match the number of named regions, i.e. drop = list(c(1,2), c(1,2), c(1,2)). The result is that the first two columns of each named region are dropped. If drop = list(1,2) is specified, it will be replicated as list(1,2,1), i.e. respectively the first, second and first column of the named regions "NamedRegion1", "NamedRegion2", "NamedRegion3" will be dropped.

simplify logical specifying if the result should be simplified, e.g. in case the data.frame would only have one row or one column (and data types match). Simplifying here is identical to calling unlist on the otherwise resulting data.frame (using use.names = FALSE). The default is FALSE.

readStrategy — character specifying the reading strategy to use. Currently supported strategies are:

- "default" (default): Can handle all supported data types incl. date/time values and can deal directly with missing value identifiers (see setMissingValue)
- "fast": Increased read performance. Date/time values are read as numeric (number of days since 1900-01-01; fractional days represent hours, minutes, and seconds) and only blank cells are recognized as missing (missing value identifiers as set in setMissingValue are ignored)

worksheetScope — Optional, the name of the worksheet to use for resolving the named region

Details

The arguments name, header, and worksheetScope are vectorized. As such, multiple named regions can be read with one method call. If only one single named region is read, the return value is a data.frame.If multiple named regions are specified, the return value is a (named) list of data.frame's returned in the order they have been specified with the argument name.
When reading dates, if your system uses a time zone that has / had daylight saving time, certain dates / timestamps will not be read exactly as they were written. See https://poi.apache.org/apidocs/dev/org/apache/poi/ss/usermodel/DateUtil.html#getJavaDate-double- If worksheetScope is unspecified, the contents of the name found anywhere in the workbook will be read. Otherwise, only a name specifically scoped to the specified sheet may be read. To read only names defined in the global scope, pass "" as the value. If option XLConnect.setCustomAttributes is TRUE (default FALSE), the worksheet scope in which the name was found is set as attribute worksheetScope on the result.

Note

If no specific column types (see argument colTypes) are specified, readNamedRegion tries to determine the resulting column types based on the read cell types. If different cell types are found in a specific column, the most general of those is used and mapped to the corresponding R data type. The order of data types from least to most general is Boolean (logical) < DateTime (POSIXct) < Numeric (numeric) < String (character). E.g. if a column is read that contains cells of type Boolean, Numeric and String then the resulting column in R would be character since character is the most general type.

Some additional information with respect to forcing data type conversion using forceConversion = TRUE:

- Forcing conversion from String to Boolean: TRUE is returned if and only if the target string is "true" (ignoring any capitalization). Any other string will return FALSE.
- Forcing conversion from Numeric to DateTime: since Excel understands Dates/Times as Numerics with some additional formatting, a conversion from a Numeric to a DateTime is actually possible. Numerics in this case represent the number of days since 1900-01-00 (yes, day 00! - see https://web.archive.org/web/20240821110422/http://www.cpearson.com/excel/datetime.htm). Note that in R 0 is represented as 1899-12-31 since there is no 1900-01-00. Fractional days represent hours, minutes, and seconds.

Author(s)

Martin Studer
Thomas Themel
Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

References

What are named regions/ranges?
https://web.archive.org/web/20240821110221/https://www.officearticles.com/excel/named_ranges_in_microsoft_excel.htm
How to create named regions/ranges?
https://www.youtube.com/watch?v=iAE9a0uRtpM

See Also

workbook, readWorksheet, writeNamedRegion,
writeWorksheet, readNamedRegionFromFile, readTable, onErrorCell

Examples

```
## Not run:
## Example 1:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read named region 'mtcars' (with default header = TRUE)
data <- readNamedRegion(wb, name = "mtcars")

## Example 2;
# conversion xlsx file from demoFiles subfolder of package XLConnect
excelFile <- system.file("demoFiles/conversion.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(excelFile)

# Read named region 'conversion' with pre-specified column types
# Note: in the worksheet all data was entered as strings!
# forceConversion = TRUE is used to force conversion from String
# into the less generic data types Numeric, DateTime & Boolean
df <- readNamedRegion(wb, name = "conversion", header = TRUE,
                      colTypes = c(XLC$DATA_TYPE.NUMERIC,
                                   XLC$DATA_TYPE.DATETIME,
                                   XLC$DATA_TYPE.BOOLEAN),
                      forceConversion = TRUE,
                      dateTimeFormat = "%Y-%m-%d %H:%M:%S")

## Example 3:
```

```
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read the columns 1, 3 and 5 of the named region 'mtcars' (with default header = TRUE)
data <- readNamedRegion(wb, name = "mtcars", keep=c(1,3,5))

# activate attributes (used by worksheet scope)
options(XLConnect.setCustomAttributes = TRUE)

# read the iris dataset from worksheet-scoped named region 'iris'
wb <- loadWorkbook("demoFiles/iris.xlsx")
data <- readNamedRegion(wb, name = "iris", worksheetScope = "iris")

# show worksheet scope attribute
attr(data, "worksheetScope")

## End(Not run)
```

`readNamedRegionFromFile` *Reading named regions from an Excel file (wrapper function)*

Description

Reads named regions from an Excel file.

Usage

```
readNamedRegionFromFile(file, ...)
```

Arguments

`file`	The file name of the workbook to read
`...`	Arguments passed to readNamedRegion

Details

This is a convenience wrapper to read named regions from a file without creating an intermediate `workbook` object. See readNamedRegion for more details.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

readNamedRegion, readWorksheetFromFile, writeNamedRegionToFile, writeWorksheetToFile, onErrorCell

Examples

```
## Not run:
# multiregion xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/multiregion.xlsx",
                             package = "XLConnect")

# Load a single named region into a single data.frame.
df <- readNamedRegionFromFile(demoExcelFile, name="Iris")

# Load multiple regions at once - returns a (named) list
# of data.frames.
df <- readNamedRegionFromFile(demoExcelFile,
                              name=c("Calendar", "Iris", "IQ"))

## End(Not run)
```

---

readTable *Reading Excel tables from a workbook*

Description

Reads Excel tables (Office 2007+) from a workbook.

Usage

```
## S4 method for signature 'workbook,numeric'
readTable(object, sheet, table, header, rownames, colTypes, forceConversion,
dateTimeFormat, check.names, useCachedValues, keep, drop, simplify, readStrategy)
## S4 method for signature 'workbook,character'
readTable(object, sheet, table, header, rownames, colTypes, forceConversion,
dateTimeFormat, check.names, useCachedValues, keep, drop, simplify, readStrategy)
```

Arguments

object	The workbook to use
sheet	The index or name of the worksheet on which to look for the specified table
table	The name of the table to read
header	The argument header specifies if the first row should be interpreted as column names. The default value is TRUE.
rownames	Index (numeric) or name (character) of column that should be used as row names. The corresponding column will be removed from the data set. Defaults to NULL which means that no row names are applied.

colTypes — Column types to use when reading in the data. Specified as a character vector of the corresponding type names (see XLC; XLC$DATA_TYPE.<?>). You may also use R class names such as numeric, character, logical and POSIXt. The types are applied in the given order to the columns - elements are recycled if necessary. Defaults to character(0) meaning that column types are determined automatically (see the Note section for more information).
By default, type conversions are only applied if the specified column type is a more generic type (e.g. from Numeric to String) - otherwise NA is returned. The forceConversion flag can be set to force conversion into less generic types where possible.

forceConversion — logical specifying if conversions to less generic types should be forced. Defaults to FALSE meaning that if a column is specified to be of a certain type via the colTypes argument and a more generic type is detected in the column, then NA will be returned (example: column is specified to be DateTime but a more generic String is found). Specifying forceConversion = TRUE will try to enforce a conversion - if it succeeds the corresponding (converted) value will be returned, otherwise NA. See the Note section for some additional information.

dateTimeFormat — Date/time format used when doing date/time conversions. Defaults to getOption("XLConnect.dateTimeFormat"). This should be a POSIX format specifier according to strptime although not all specifications have been implemented yet - the most important ones however are available.

check.names — logical specifying if column names of the resulting data.frame should be checked to ensure that they are syntactically valid valid variable names and are not duplicated. See the check.names argument of data.frame. Defaults to TRUE.

useCachedValues — logical specifying whether to read cached formula results from the workbook instead of re-evaluating them. This is particularly helpful in cases for reading data produced by Excel features not supported in XLConnect like references to external workbooks. Defaults to FALSE, which means that formulas will be evaluated by XLConnect.

keep — List of column names or indices to be kept in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments.

drop — List of column names or indices to be dropped in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments.

simplify — logical specifying if the result should be simplified, e.g. in case the data.frame would only have one row or one column (and data types match). Simplifying here is identical to calling unlist on the otherwise resulting data.frame (using use.names = FALSE). The default is FALSE.

readStrategy — character specifying the reading strategy to use. Currently supported strategies are:

- "default" (default): Can handle all supported data types incl. date/time values and can deal directly with missing value identifiers (see setMissingValue)
- "fast": Increased read performance. Date/time values are read as numeric (number of days since 1900-01-01; fractional days represent hours, minutes, and seconds) and only blank cells are recognized as missing (missing value identifiers as set in setMissingValue are ignored)

Note

If no specific column types (see argument colTypes) are specified, readNamedRegion tries to determine the resulting column types based on the read cell types. If different cell types are found in a specific column, the most general of those is used and mapped to the corresponding R data type. The order of data types from least to most general is Boolean (logical) < DateTime (POSIXct) < Numeric (numeric) < String (character). E.g. if a column is read that contains cells of type Boolean, Numeric and String then the resulting column in R would be character since character is the most general type.

Some additional information with respect to forcing data type conversion using forceConversion = TRUE:

- Forcing conversion from String to Boolean: TRUE is returned if and only if the target string is "true" (ignoring any capitalization). Any other string will return FALSE.
- Forcing conversion from Numeric to DateTime: since Excel understands Dates/Times as Numerics with some additional formatting, a conversion from a Numeric to a DateTime is actually possible. Numerics in this case represent the number of days since 1900-01-01. Fractional days represent hours, minutes, and seconds.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

Overview of Excel tables
https://support.microsoft.com/en-us/office/overview-of-excel-tables-7ab0bb7d-3a9e-4b56-a3c9-6c9433
ocmsassetid=ha010048546&correlationid=ecf0d51a-596f-42e5-9c05-8653648bb180&ui=en-us&
rs=en-us&ad=us

See Also

workbook, readNamedRegion, readWorksheet, writeNamedRegion, writeWorksheet, readNamedRegionFromFile, onErrorCell

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")
```

```
# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read table 'MtcarsTable' from sheet 'mtcars_table'
data <- readTable(wb, sheet = "mtcars_table", table = "MtcarsTable")

## End(Not run)
```

---

readWorksheet-methods *Reading data from worksheets*

Description

Reads data from worksheets of a workbook.

Usage

```
## S4 method for signature 'workbook,numeric'
readWorksheet(object,sheet,startRow,startCol,endRow,endCol,autofitRow,autofitCol,
region,header,rownames,colTypes,forceConversion,dateTimeFormat,check.names,
useCachedValues,keep,drop, simplify, readStrategy)
## S4 method for signature 'workbook,character'
readWorksheet(object,sheet,startRow,startCol,endRow,endCol,autofitRow,autofitCol,
region,header,rownames,colTypes,forceConversion,dateTimeFormat,check.names,
useCachedValues,keep,drop, simplify, readStrategy)
```

Arguments

object	The workbook to use
sheet	The name or index of the worksheet to read from
startRow	The index of the first row to read from. Defaults to 0 meaning that the start row is determined automatically.
startCol	The index of the first column to read from. Defaults to 0 meaning that the start column is determined automatically.
endRow	The index of the last row to read from. Defaults to 0 meaning that the end row is determined automatically.
endCol	The index of the last column to read from. Defaults to 0 meaning that the end column is determined automatically.
autofitRow	logical specifying if leading and trailing empty rows should be skipped. Defaults to TRUE.
autofitCol	logical specifying if leading and trailing empty columns should be skipped. Defaults to TRUE.
region	A range specifier in the form 'A10:B18'. This provides an alternative way to specify startRow, startCol, endRow and endCol. Range specifications take precedence over index specifications.

header — Interpret the first row of the specified area as column headers. The default is TRUE.

rownames — Index (numeric) or name (character) of column that should be used as row names. The corresponding column will be removed from the data set. Defaults to NULL which means that no row names are applied. Row names must be either integer or character. Non-numeric columns will be coerced to character.

colTypes — Column types to use when reading in the data. Specified as a character vector of the corresponding type names (see XLC; XLC$DATA_TYPE.<?>). You may also use R class names such as numeric, character, logical and POSIXt. The types are applied in the given order to the columns - elements are recycled if necessary. Defaults to character(0) meaning that column types are determined automatically (see the Note section for more information).
By default, type conversions are only applied if the specified column type is a more generic type (e.g. from Numeric to String) - otherwise NA is returned. The forceConversion flag can be set to force conversion into less generic types where possible.

forceConversion — logical specifying if conversions to less generic types should be forced. Defaults to FALSE meaning that if a column is specified to be of a certain type via the colTypes argument and a more generic type is detected in the column, then NA will be returned (example: column is specified to be DateTime but a more generic String is found). Specifying forceConversion = TRUE will try to enforce a conversion - if it succeeds the corresponding (converted) value will be returned, otherwise NA. See the Note section for some additional information.

dateTimeFormat — Date/time format used when doing date/time conversions. Defaults to getOption("XLConnect.dateTimeFormat"). This should be a POSIX format specifier according to strptime although not all specifications have been implemented yet - the most important ones however are available. When using the '%OS' specification for fractional seconds (without an additional integer) 3 digits will be used by default (getOption("digits.secs") is not considered).

check.names — logical specifying if column names of the resulting data.frame should be checked to ensure that they are syntactically valid variable names and are not duplicated. See the check.names argument of data.frame. Defaults to TRUE.

useCachedValues — logical specifying whether to read cached formula results from the workbook instead of re-evaluating them. This is particularly helpful in cases for reading data produced by Excel features not supported in XLConnect like references to external workbooks. Defaults to FALSE, which means that formulas will be evaluated by XLConnect.

keep — Vector of column names or indices to be kept in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments. Example: if sheet = c("Sheet1", "Sheet2", "Sheet3") and keep = c(1,2), keep will be internally converted into list(c(1,2)) and then replicated to match the number of sheets, i.e. keep = list(c(1,2), c(1,2), c(1,2)). The result is that the first two columns of each sheet are kept. If keep = list(1,2) is specified, it

will be replicated as `list(1,2,1)`, i.e. respectively the first, second and first column of the sheets "Sheet1", "Sheet2", "Sheet3" will be kept.

`drop` Vector of column names or indices to be dropped in the output data frame. It is possible to specify either `keep` or `drop`, but not both at the same time. Defaults to `NULL`. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments. Example: if `sheet = c("Sheet1", "Sheet2", "Sheet3")` and `drop = c(1,2)`, `drop` will be internally converted into `list(c(1,2))` and then replicated to match the number of sheets, i.e. `drop = list(c(1,2), c(1,2), c(1,2))`. The result is that the first two columns of each sheet are dropped. If `drop = list(1,2)` is specified, it will be replicated as `list(1,2,1)`, i.e. respectively the first, second and first column of the sheets "Sheet1", "Sheet2", "Sheet3" will be dropped.

`simplify` `logical` specifying if the result should be simplified, e.g. in case the `data.frame` would only have one row or one column (and data types match). Simplifying here is identical to calling `unlist` on the otherwise resulting `data.frame` (using `use.names = FALSE`). The default is `FALSE`.

`readStrategy` `character` specifying the reading strategy to use. Currently supported strategies are:

- `"default"` (default): Can handle all supported data types incl. date/time values and can deal directly with missing value identifiers (see `setMissingValue`)
- `"fast"`: Increased read performance. Date/time values are read as numeric (number of days since 1900-01-01; fractional days represent hours, minutes, and seconds) and only blank cells are recognized as missing (missing value identifiers as set in `setMissingValue` are ignored)

Details

Reads data from the worksheet specified by `sheet`. Data is read starting at the top left corner specified by `startRow` and `startCol` down to the bottom right corner specified by `endRow` and `endCol`. If `header = TRUE`, the first row is interpreted as column names of the resulting `data.frame`. If `startRow <= 0` then the first available row in the sheet is assumed. If `endRow = 0` then the last available row in the sheet is assumed. For `endRow = -n` with n > 0, the 'last row' - n rows is assumed. This is useful in cases where you want to skip the last n rows. If `startCol <= 0` then the minimum column between `startRow` and `endRow` is assumed. If `endCol = 0` then the maximum column between `startRow` and `endRow` is assumed. If `endCol = -n` with n > 0, the maximum column between `startRow` and `endRow` except for the last n columns is assumed.

In other words, if no boundaries are specified `readWorksheet` assumes the "bounding box" of the data as the corresponding boundaries.
The arguments `autofitRow` and `autofitCol` (both defaulting to `TRUE`) can be used to skip leading and trailing empty rows even in case `startRow`, `endRow`, `startCol` and `endCol` are specified to values > `0`. This can be useful if data is expected within certain given boundaries but the exact location is not available.

If all four coordinate arguments are missing this behaves as above with `startRow = 0`, `startCol = 0`, `endRow = 0` and `endCol = 0`. In this case `readWorksheet` assumes the "bounding box" of the data as the corresponding boundaries.

All arguments (except `object`) are vectorized. As such, multiple worksheets (and also multiple data regions from the same worksheet) can be read with one method call. If only one single data region is read, the return value is a `data.frame`. If multiple data regions are specified, the return value is a `list` of `data.frame`'s returned in the order they have been specified. If worksheets have been specified by name, the `list` will be a named `list` named by the corresponding worksheets.

Note

If no specific column types (see argument `colTypes`) are specified, `readWorksheet` tries to determine the resulting column types based on the read cell types. If different cell types are found in a specific column, the most general of those is used and mapped to the corresponding R data type. The order of data types from least to most general is Boolean (`logical`) < DateTime (`POSIXct`) < Numeric (`numeric`) < String (`character`). E.g. if a column is read that contains cells of type Boolean, Numeric and String then the resulting column in R would be `character` since `character` is the most general type.

Some additional information with respect to forcing data type conversion using `forceConversion = TRUE`:

- Forcing conversion from String to Boolean: `TRUE` is returned if and only if the target string is "true" (ignoring any capitalization). Any other string will return `FALSE`.
- Forcing conversion from Numeric to DateTime: since Excel understands Dates/Times as Numerics with some additional formatting, a conversion from a Numeric to a DateTime is actually possible. Numerics in this case represent the number of days since 1900-01-00 (yes, day 00! - see https://web.archive.org/web/20240821110422/http://www.cpearson.com/excel/datetime.htm). Note that in R 0 is represented as 1899-12-31 since there is no 1900-01-00. Fractional days represent hours, minutes, and seconds.

Author(s)

Martin Studer
Thomas Themel
Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `writeWorksheet`, `readNamedRegion`, `writeNamedRegion`, `readWorksheetFromFile`, `readTable`, `onErrorCell`

Examples

```
## Not run:
## Example 1:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
```

```
wb <- loadWorkbook(demoExcelFile)

# Read worksheet 'mtcars' (providing no specific area bounds;
# with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars")


## Example 2:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read worksheet 'mtcars' (providing area bounds; with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars", startRow = 1, startCol = 3,
                      endRow = 15, endCol = 8)


## Example 3:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read worksheet 'mtcars' (providing area bounds using the region argument;
# with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars", region = "C1:H15")


## Example 4:
# conversion xlsx file from demoFiles subfolder of package XLConnect
excelFile <- system.file("demoFiles/conversion.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(excelFile)

# Read worksheet 'Conversion' with pre-specified column types
# Note: in the worksheet all data was entered as strings!
# forceConversion = TRUE is used to force conversion from String
# into the less generic data types Numeric, DateTime & Boolean
df <- readWorksheet(wb, sheet = "Conversion", header = TRUE,
                    colTypes = c(XLC$DATA_TYPE.NUMERIC,
                                 XLC$DATA_TYPE.DATETIME,
                                 XLC$DATA_TYPE.BOOLEAN),
                    forceConversion = TRUE,
                    dateTimeFormat = "%Y-%m-%d %H:%M:%S")

## Example 5:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")
```

```
# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read the columns 1, 3 and 5 from the sheet 'mtcars' (with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars", keep=c(1,3,5))

## End(Not run)
```

`readWorksheetFromFile` *Reading data from worksheets in an Excel file (wrapper function)*

Description

Reads data from worksheets in an Excel file.

Usage

```
readWorksheetFromFile(file, ...)
```

Arguments

`file`	The path name of the file to read from.
`...`	Arguments passed to readWorksheet

Details

See readWorksheet for more information.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`readWorksheet`, `readNamedRegionFromFile`, `writeWorksheetToFile`, `writeNamedRegionToFile`, `onErrorCell`

Examples

```
## Not run:
# multiregion xlsx file from demoFiles subfolder of
# package XLConnect
demoExcelFile <- system.file("demoFiles/multiregion.xlsx",
                             package = "XLConnect")

# Read single area from first sheet of existing file,
# "B2:C3" in Excel speak
df.one <- readWorksheetFromFile(demoExcelFile, sheet = 1,
```

```
                                header = FALSE, startCol = 2,
                                startRow = 2, endCol = 3,
                                endRow = 3)

# Read three data sets in one from known positions
dflist <- readWorksheetFromFile(demoExcelFile,
                                sheet = c("FirstSheet",
                                          "FirstSheet",
                                          "SecondSheet"),
                                header = TRUE,
                                startRow = c(2,2,3),
                                startCol = c(2,5,2),
                                endCol = c(5,8,6),
                                endRow = c(9,15,153))

## End(Not run)
```

`removeName-methods` *Removing names from workbooks*

Description

Removes a named range reference from a `workbook`.

Usage

```
## S4 method for signature 'workbook'
removeName(object,name,worksheetScope)
```

Arguments

`object`	The `workbook` to use
`name`	The name to delete
`worksheetScope`	Optional - the name of the worksheet in which the name is scoped; useful if different sheets have locally-scoped named ranges with the same name.

Details

Removes the named range reference `name` from the specified workbook `object` if it does exist. Data in the referenced cells remains unchanged. Multiple names can be specified to be removed. Use `worksheetScope = ""` to only target names defined in the global scope.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createName, existsName,
getDefinedNames, readNamedRegion, writeNamedRegion

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Remove the named region called 'mtcars' from the above file
# (this named region is defined as 'mtcars!$A$1:$K$33')
removeName(wb, name = "mtcars")

## End(Not run)
```

---

removePane-methods *Removing panes from worksheet*

Description

Removes the split pane/freeze pane from the specified worksheet.

Usage

```
## S4 method for signature 'workbook,character'
removePane(object,sheet)
## S4 method for signature 'workbook,numeric'
removePane(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet from which to remove the split pane/freeze pane

Note

To keep an area of a worksheet visible while you scroll to another area of the worksheet, you can lock specific rows or columns in one area by freezing or splitting panes.

When you freeze panes, you keep specific rows or columns visible when you scroll in the worksheet. For example, you might want to keep row and column labels visible as you scroll.

When you split panes, you create separate worksheet areas that you can scroll within, while rows or columns in the non-scrolled area remain visible.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

References

How to create a freeze pane/split pane in Office 2007 https://support.microsoft.com/en-us/office/freeze-panes-to-lock-rows-and-columns-dab2ffc9-020d-4026-8121-67dd25f2508f?ocmsassetid=hp001217048&correlationid=b4f5baeb-b622-4487-a96f-514d2f00208a&ui=en-us&rs=en-us&ad=us

See Also

workbook createFreezePane createSplitPane

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("removePaneTest.xlsx", create = TRUE)

# Create a worksheet named 'Sheet1'
createSheet(wb, name = "Sheet1")

# Create a split pane on Sheet1, with coordinates (10000, 5000) expressed as 1/20th of a point,
# 10 (-> J) as left column visible in right pane and 10 as top row visible in bottom pane
createSplitPane(wb, "Sheet1", 10000, 5000, 10, 10)

# Remove the split pane from Sheet1
removePane(wb, "Sheet1")

# Save workbook (this actually writes the file to disk). Now the workbook has no split pane.
saveWorkbook(wb)

# clean up
file.remove("removePaneTest.xlsx")

## End(Not run)
```

---

removeSheet-methods *Removing worksheets from workbooks*

Description

Removes a worksheet from a workbook.

Usage

```
## S4 method for signature 'workbook,character'
removeSheet(object,sheet)
## S4 method for signature 'workbook,numeric'
removeSheet(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to remove

Note

When removing a worksheet that is the currently active sheet then **XLConnect** resets the active sheet to the first possible worksheet in the workbook.
Also note that deleting worksheets may result in invalid name references.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createSheet, existsSheet, getSheets, renameSheet, cloneSheet, setActiveSheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Remove the worksheet called 'mtcars' from the above file
removeSheet(wb, sheet = "mtcars")

## End(Not run)
```

---

renameSheet-methods *Renaming worksheets from workbooks*

Description

Renames a worksheet from a workbook.

Usage

```
## S4 method for signature 'workbook,character'
renameSheet(object,sheet,newName)
## S4 method for signature 'workbook,numeric'
renameSheet(object,sheet,newName)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to rename
newName	The new name of the sheet

Note

Note that renaming worksheets may result in invalid name references.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createSheet, existsSheet, getSheets, removeSheet, cloneSheet, setActiveSheet

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Rename the worksheet called 'mtcars' from the above file to 'MyCars'
renameSheet(wb, sheet = "mtcars", newName = "MyCars")

## End(Not run)
```

---

saveWorkbook-methods *Saving Microsoft Excel workbooks*

Description

Saves a workbook to the corresponding Excel file. This method actually writes the workbook object to disk.

Usage

```
## S4 method for signature 'workbook,missing'
saveWorkbook(object,file)
## S4 method for signature 'workbook,character'
saveWorkbook(object,file)
```

Arguments

object	The workbook to save
file	The file to which to save the workbook ("save as"). If not specified (missing), the workbook will be saved to the workbook's underlying file which is the file specified in loadWorkbook (also see the workbook class for more information). Note that due to currently missing functionality in Apache POI, workbooks can only be saved in the same file format - i.e. if the workbooks underlying file format is xls, then the file argument may only specify another xls file. Also note that when specifying the file argument the workbook's underlying filename changes to reflect the "save as" behavior. Paths are expanded using path.expand.

Details

Saves the specified workbook object to disk.

Note

As already mentioned in the documentation of the workbook class, a workbook's underlying Excel file is not saved (or being created in case the file did not exist and create = TRUE has been specified) unless the saveWorkbook method has been called on the object. This provides more flexibility to the user to decide when changes are saved and also provides better performance in that several changes can be written in one go (normally at the end, rather than after every operation causing the file to be rewritten again completely each time). This is due to the fact that workbooks are manipulated in-memory and are only written to disk with specifically calling saveWorkbook.

Further note that calling saveWorkbook more than once leads to an exception. This is due to a current issue in the underlying POI libraries. However, with **XLConnect** there should be no need to call saveWorkbook more than once so virtually this is no issue.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, loadWorkbook

Examples

```
## Not run:
# Create a new workbook 'saveMe.xlsx'
# (assuming the file to not exist already)
wb <- loadWorkbook("saveMe.xlsx", create = TRUE)

# Create a worksheet called 'mtcars'
createSheet(wb, name = "mtcars")

# Write built-in dataset 'mtcars' to sheet 'mtcars' created above
writeWorksheet(wb, mtcars, sheet = "mtcars")

# Save workbook - this actually writes the file 'saveMe.xlsx' to disk
saveWorkbook(wb)

# clean up
file.remove("saveMe.xlsx")

## End(Not run)
```

---

setActiveSheet-methods

Setting the active worksheet in a workbook

Description

Sets the active worksheet of a workbook.

Usage

```
## S4 method for signature 'workbook,character'
setActiveSheet(object,sheet)
## S4 method for signature 'workbook,numeric'
setActiveSheet(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to activate

Note

The active worksheet of a workbook is the worksheet that is displayed when the corresponding Excel file is opened.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, createSheet, removeSheet, renameSheet, existsSheet, getSheets

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Sets the active sheet to the sheet 'mtcars3'
setActiveSheet(wb, sheet = "mtcars3")

## End(Not run)
```

---

setAutoFilter-methods *Setting auto-filters on worksheets*

Description

Sets an auto-filter on a specified worksheet.

Usage

```
## S4 method for signature 'workbook,character'
setAutoFilter(object,sheet,reference)
## S4 method for signature 'workbook,numeric'
setAutoFilter(object,sheet,reference)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet on which to set the auto-filter
reference	A cell range specification (character) in the form 'A1:B8'

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("autofilter.xlsx", create = TRUE)

# Create a worksheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' on the sheet called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$A$1")

# Write built-in data set 'mtcars' to the above defined named region
# (using header = TRUE)
writeNamedRegion(wb, mtcars, name = "mtcars")

# Set an auto-filter for the named region written above
setAutoFilter(wb, sheet = "mtcars", reference = aref("A1", dim(mtcars)))

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("autofilter.xlsx")

## End(Not run)
```

---

setBorder-methods *Specifying borders for cell styles*

Description

Specifies borders for a cellstyle.

Usage

```
## S4 method for signature 'cellstyle'
setBorder(object,side,type,color)
```

Arguments

object	The cellstyle to edit
side	A vector with any combination of {"bottom", "left", "right", "top", "all"}
type	Specifies the border type to be used - it is normally specified by a corresponding XLC constant (see the XLC border constant, e.g. XLC$"BORDER.MEDIUM_DASHED")
color	Defines the border color and is normally also specified via an XLC constant.

Details

Specifies the border for a cellstyle. Note that the arguments type and color should be of the same length as side. In other words, for each specified side there should be a corresponding specification of type and color. If this is not the case the arguments will be automatically replicated to the length of side.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, setStyleAction, XLC

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setBorder.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "cellstyles")

# Create a custom anonymous cell style
cs <- createCellStyle(wb)

# Specify the border for the cell style created above
setBorder(cs, side = c("bottom", "right"), type = XLC$"BORDER.THICK",
          color = c(XLC$"COLOR.BLACK", XLC$"COLOR.RED"))

# Set the cell style created above for the top left cell (A1) in the
# 'cellstyles' worksheet
setCellStyle(wb, sheet = "cellstyles", row = 1, col = 1, cellstyle = cs)

# Save the workbook
saveWorkbook(wb)

# clean up
file.remove("setBorder.xlsx")

## End(Not run)
```

---

setCellFormula-methods

Setting cell formulas

Description

Sets cell formulas for specific cells in a workbook.

Usage

```
## S4 method for signature 'workbook,character'
setCellFormula(object,sheet,row,col,formula)
## S4 method for signature 'workbook,numeric'
setCellFormula(object,sheet,row,col,formula)
```

Arguments

object	The workbook to use
sheet	Name or index of the sheet the cell is on
row	Row index of the cell to edit
col	Column index of the cell to edit
formula	The formula to apply to the cell, without the initial = character used in Excel

Details

Note that the arguments are vectorized such that multiple cells can be set with one method call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getCellFormula,

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setCellFormula.xls", create = TRUE)

# Create a sheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' referring to the sheet
# called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$A$1")

# Write built-in data set 'mtcars' to the above defined named region.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Now, let us get Excel to calculate average weights.
# Where did we write the dataset?
corners <- getReferenceCoordinatesForName(wb, "mtcars")
# Put the average under the wt column
colIndex <- which(names(mtcars) == "wt")
rowIndex <- corners[2,1] + 1
```

```
# Construct the input range & formula
input <- paste(idx2cref(c(corners[1,1], colIndex,
                          corners[2,1], colIndex)), collapse=":")
formula <- paste("AVERAGE(", input, ")", sep="")

setCellFormula(wb, "mtcars", rowIndex, colIndex, formula)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("setCellFormula.xls")

## End(Not run)
```

`setCellStyle-methods` *Setting cell styles*

Description

Sets cell styles for specific cells in a workbook.

Usage

```
## S4 method for signature 'workbook,missing,character'
setCellStyle(object,formula,sheet,row,col,cellstyle)
## S4 method for signature 'workbook,missing,numeric'
setCellStyle(object,formula,sheet,row,col,cellstyle)
## S4 method for signature 'workbook,character,missing'
setCellStyle(object,formula,sheet,row,col,cellstyle)
```

Arguments

`object`	The workbook to use
`formula`	A formula specification in the form Sheet!B8:C17. Use either the argument `formula` or the combination of `sheet`, `row` and `col`.
`sheet`	Name or index of the sheet the cell is on. Use either the argument `formula` or the combination of `sheet`, `row` and `col`.
`row`	Row index of the cell to apply the cellstyle to.
`col`	Column index of the cell to apply the cellstyle to.
`cellstyle`	cellstyle to apply

Details

Sets the specified cellstyle for the specified cell (`row`, `col`) on the specified `sheet` or alternatively for the cells referred to by `formula`. Note that the arguments are vectorized such that multiple cells can be styled with one method call. Use either the argument `formula` or the combination of `sheet`, `row` and `col`.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, createCellStyle, setDataFormat, setBorder, setFillBackgroundColor, setFillForegroundColor, setFillPattern, setWrapText

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setCellStyle.xlsx", create = TRUE)

# We don't set a specific style action in this demo, so the default
# 'XLConnect' will be used (XLC$"STYLE_ACTION.XLCONNECT")

# Create a sheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' referring to the sheet
# called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$C$4")

# Write built-in data set 'mtcars' to the above defined named region.
# This will use the default style action 'XLConnect'.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Now let's color all weight cells of cars with a weight > 3.5 in red
# (mtcars$wt > 3.5)

# First, create a corresponding (named) cell style
heavyCar <- createCellStyle(wb, name = "HeavyCar")

# Specify the cell style to use a solid foreground color
setFillPattern(heavyCar, fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(heavyCar, color = XLC$"COLOR.RED")

# Which cars have a weight > 3.5 ?
rowIndex <- which(mtcars$wt > 3.5)

# NOTE: The mtcars data.frame has been written offset with
# top left cell C4 - and we have also written a header row!
# So, let's take that into account appropriately. Obviously,
# the two steps could be combined directly into one ...
rowIndex <- rowIndex + 4

# The same holds for the column index
```

```
colIndex <- which(names(mtcars) == "wt") + 2

# Set the 'HeavyCar' cell style for the corresponding cells.
# Note: the row and col arguments are vectorized!
setCellStyle(wb, sheet = "mtcars", row = rowIndex, col = colIndex,
             cellstyle = heavyCar)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("setCellStyle.xlsx")

## End(Not run)
```

`setCellStyleForType-methods` *Setting the cell style per data type for the DATATYPE style action*

Description

Sets the cell style for a specific data type as used by the DATATYPE style action.

Usage

```
## S4 method for signature 'workbook'
setCellStyleForType(object,type,style)
```

Arguments

`object`	The `workbook` to use
`type`	The data type for which to set the `style`
`style`	The `cellstyle` to set

Details

Based on the (cell) data type the DATATYPE style action (see `setStyleAction`) sets the `cellstyle` for the corresponding cells. The data `type` is normally specified via a corresponding data type constant from the `XLC` object.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `getCellStyleForType`, `setStyleAction`

Examples

```
## Not run:
file.copy(system.file("demoFiles/template2.xlsx",
                      package = "XLConnect"),
          "datatype.xlsx", overwrite = TRUE)

# Load workbook
wb <- loadWorkbook("datatype.xlsx")

# Create a new cell style to be used
cs <- createCellStyle(wb, name = "mystyle")

# Set data format (number format) as numbers with aligned fractions
setDataFormat(cs, format = "# ???/???")

# Define the above created cell style as style to be used for
# numerics
setCellStyleForType(wb, type = XLC$"DATA_TYPE.NUMERIC", style = cs)
# Could also say cs <- setCellStyleForType(wb, "numeric")

# Set style action to 'datatype'
setStyleAction(wb, XLC$"STYLE_ACTION.DATATYPE")

# Write built-in data set 'mtcars' to the named region
# 'mtcars' as defined by the Excel template.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Save workbook
saveWorkbook(wb)

# clean up
file.remove("datatype.xlsx")

## End(Not run)
```

---

setColumnWidth-methods

Setting the width of a column in a worksheet

Description

Sets the width of a column in a worksheet.

Usage

```
## S4 method for signature 'workbook,character'
setColumnWidth(object,sheet,column,width)
## S4 method for signature 'workbook,numeric'
setColumnWidth(object,sheet,column,width)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet
column	The index of the column to resize
width	The width of the specified column in units of 1/256th of a character width. If width = -1 (default), the column is auto-sized. If negative otherwise, the column will be sized to the sheet's default column width.

Details

Note that the arguments sheet, column and width are vectorized. As such the column width of multiple columns (potentially on different sheets) can be set with one method call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, setRowHeight

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Sets the column width of the 3rd column on sheet 'mtcars'
# to 4000/256th (= 15.625) character width
setColumnWidth(wb, sheet = "mtcars", column = 3, width = 4000)

## End(Not run)
```

---

setDataFormat-methods *Specifying custom data formats for cell styles*

Description

Specifies a custom data format for a cellstyle.

Usage

```
## S4 method for signature 'cellstyle'
setDataFormat(object,format)
```

Arguments

object	The cellstyle to use
format	A data format string

Details

Specifies the data format to be used by the corresponding cellstyle. Data formats are specified the standard Excel way. Refer to the Excel help or to the link below for more information.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, setStyleAction

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setDataFormat.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "cellstyles")

# Create a dummy data set with the current date/time (as POSIXct)
now <- data.frame(Now = Sys.time())

# Write the value to the 'cellstyles' worksheet in the top left
# corner (cell A1)
writeWorksheet(wb, now, sheet = "cellstyles", startRow = 1,
               startCol = 1, header = FALSE)

# Create a custom anonymous cell style
cs <- createCellStyle(wb)

# Specify a custom data format
setDataFormat(cs, format = "dddd d-m-yyyy h:mm AM/PM")

# Set the cell style created above for the top left cell (A1) in
# the 'cellstyles' worksheet
setCellStyle(wb, sheet = "cellstyles", row = 1, col = 1, cellstyle = cs)

# Set column width to display whole time/date string
setColumnWidth(wb, sheet = "cellstyles", column = 1, width = 6000)

# Save the workbook
saveWorkbook(wb)
```

```
# clean up
file.remove("setDataFormat.xlsx")

## End(Not run)
```

`setDataFormatForType-methods` *Setting the data format for the DATA_FORMAT_ONLY style action*

Description

Sets the data format for a specific data type as used by the DATA_FORMAT_ONLY style action.

Usage

```
## S4 method for signature 'workbook'
setDataFormatForType(object,type,format)
```

Arguments

`object`	The `workbook` to use
`type`	The data type for which to set the `format`.
`format`	A data format string

Details

Based on the (cell) data type the DATA_FORMAT_ONLY style action (see `setStyleAction`) sets the data format for the corresponding cells. The data `type` is normally specified via a corresponding data type constant from the `XLC` object. Data formats are specified the standard Excel way. Refer to the Excel help or to the link below for more information.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `setStyleAction`

Examples

```
## Not run:
# Copy existing Excel template to working directory
file.copy(system.file("demoFiles/template2.xlsx",
                      package = "XLConnect"),
          "dataformat.xlsx", overwrite = TRUE)
```

```
# Load workbook
wb <- loadWorkbook("dataformat.xlsx")

# Set the data format for numeric columns (cells)
# (keeping the defaults for all other data types)
setDataFormatForType(wb, type = XLC$"DATA_TYPE.NUMERIC",
                     format = "0.00")

# Set style action to 'data format only'
setStyleAction(wb, XLC$"STYLE_ACTION.DATA_FORMAT_ONLY")

# Write built-in data set 'mtcars' to the named region
# 'mtcars' as defined by the Excel template.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Save workbook
saveWorkbook(wb)

# clean up
file.remove("dataformat.xlsx")

## End(Not run)
```

`setFillBackgroundColor-methods` *Specifying the fill background color for cell styles*

Description

Specifies the fill background color for a `cellstyle`.

Usage

```
## S4 method for signature 'cellstyle,numeric'
setFillBackgroundColor(object,color)
```

Arguments

`object`	The `cellstyle` to manipulate
`color`	The fill background color to use for the `cellstyle`. The `color` is normally specified via a corresponding color constant from the `XLC` object.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `cellstyle`, `setCellStyle`, `setStyleAction`, `XLC`

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setFillBackgroundColor.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "cellstyles")

# Create a custom anonymous cell style
cs <- createCellStyle(wb)

# Specify the fill background color for the cell style created above
setFillBackgroundColor(cs, color = XLC$"COLOR.CORNFLOWER_BLUE")

# Specify the fill foreground color
setFillForegroundColor(cs, color = XLC$"COLOR.YELLOW")

# Specify the fill pattern
setFillPattern(cs, fill = XLC$"FILL.BIG_SPOTS")

# Set the cell style created above for the top left cell (A1) in the
# 'cellstyles' worksheet
setCellStyle(wb, sheet = "cellstyles", row = 1, col = 1, cellstyle = cs)

# Save the workbook
saveWorkbook(wb)

# clean up
file.remove("setFillBackgroundColor.xlsx")

## End(Not run)
```

---

setFillForegroundColor-methods

Specifying the fill foreground color for cell styles

Description

Specifies the fill foreground color for a cellstyle.

Usage

```
## S4 method for signature 'cellstyle,numeric'
setFillForegroundColor(object,color)
```

Arguments

object	The cellstyle to manipulate
color	The fill foreground color to use for the cellstyle. The color is normally specified via a corresponding color constant from the XLC object.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, setStyleAction, XLC

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setFillForegroundColor.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "cellstyles")

# Create a custom anonymous cell style
cs <- createCellStyle(wb)

# Specify the fill background color for the cell style created above
setFillBackgroundColor(cs, color = XLC$"COLOR.CORNFLOWER_BLUE")

# Specify the fill foreground color
setFillForegroundColor(cs, color = XLC$"COLOR.YELLOW")

# Specify the fill pattern
setFillPattern(cs, fill = XLC$"FILL.BIG_SPOTS")

# Set the cell style created above for the top left cell (A1) in the
# 'cellstyles' worksheet
setCellStyle(wb, sheet = "cellstyles", row = 1, col = 1, cellstyle = cs)

# Save the workbook
saveWorkbook(wb)

# clean up
file.remove("setFillForegroundColor.xlsx")

## End(Not run)
```

---

setFillPattern-methods

Specifying the fill pattern for cell styles

Description

Specifies the fill pattern for a cellstyle.

Usage

```
## S4 method for signature 'cellstyle'
setFillPattern(object,fill)
```

Arguments

object	The cellstyle to manipulate
fill	The fill pattern to use for the cellstyle. fill is normally specified via a corresponding fill constant from the XLC object.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, setStyleAction, XLC

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setFillPattern.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "cellstyles")

# Create a custom anonymous cell style
cs <- createCellStyle(wb)

# Specify the fill background color for the cell style created above
setFillBackgroundColor(cs, color = XLC$"COLOR.CORNFLOWER_BLUE")

# Specify the fill foreground color
setFillForegroundColor(cs, color = XLC$"COLOR.YELLOW")

# Specify the fill pattern
setFillPattern(cs, fill = XLC$"FILL.BIG_SPOTS")

# Set the cell style created above for the top left cell (A1) in the
# 'cellstyles' worksheet
setCellStyle(wb, sheet = "cellstyles", row = 1, col = 1, cellstyle = cs)

# Save the workbook
saveWorkbook(wb)

# clean up
file.remove("setFillPattern.xlsx")

## End(Not run)
```

setForceFormulaRecalculation-methods
Forcing Excel to recalculate formula values when opening a workbook

Description

This function controls a flag that forces Excel to recalculate formula values when a workbook is opened.

Usage

```
## S4 method for signature 'workbook,character'
setForceFormulaRecalculation(object,sheet,value)
## S4 method for signature 'workbook,numeric'
setForceFormulaRecalculation(object,sheet,value)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet for which to force formula recalculation. If sheet = "*", the flag is set for all sheets in the workbook.
value	logical specifying if formula recalculation should be forced or not

Details

The arguments sheet and value are vectorized such that multiple worksheets can be controlled with one method call.

Note

A typical use for this flag is forcing Excel into updating formulas that reference cells affected by writeWorksheet or writeNamedRegion. The exact behavior of Excel when the flag is set depends on version and file format.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getForceFormulaRecalculation

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Tell Excel to automatically recalculate formulas on sheet mtcars
setForceFormulaRecalculation(wb, sheet = "mtcars", TRUE)
# The same with a numerical sheet index
setForceFormulaRecalculation(wb, sheet = 1, TRUE)

## End(Not run)
```

---

setHyperlink-methods *Setting hyperlinks*

Description

Sets hyperlinks for specific cells in a workbook.

Usage

```
## S4 method for signature 'workbook,missing,character'
setHyperlink(object,formula,sheet,row,col,type,address)
## S4 method for signature 'workbook,missing,numeric'
setHyperlink(object,formula,sheet,row,col,type,address)
## S4 method for signature 'workbook,character,missing'
setHyperlink(object,formula,sheet,row,col,type,address)
```

Arguments

object	The workbook to use
formula	A formula specification in the form Sheet!B8:C17. Use either the argument formula or the combination of sheet, row and col.
sheet	Name or index of the sheet the cell is on. Use either the argument formula or the combination of sheet, row and col.
row	Row index of the cell to apply the cellstyle to.
col	Column index of the cell to apply the cellstyle to.
type	Hyperlink type. See the corresponding "HYPERLINK.*" constants from the XLC object.
address	Hyperlink address. This needs to be a valid URI including scheme. E.g. for email mailto:myself@me.org, for a URL https://www.somewhere.net or for a file file:///a/b/c.dat

Details

Sets a hyperlink for the specified cells. Note that cellstyles for hyperlinks can be defined independently using setCellStyle. The arguments are vectorized such that multiple hyperlinks can be set in one method call. Use either the argument formula or the combination of sheet, row and col.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, setCellStyle

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setHyperlink.xlsx", create = TRUE)

# Create a sheet named 'mtcars'
createSheet(wb, name = "mtcars")


# Write built-in data set 'mtcars' to the above defined worksheet
writeWorksheet(wb, mtcars, sheet = "mtcars", rownames = "Car")

# Set hyperlinks
links <- paste0("https://www.google.com?q=", gsub(" ", "+", rownames(mtcars)))
setHyperlink(wb, sheet = "mtcars", row = seq_len(nrow(mtcars)) + 1, col = 1,
  type = XLC$HYPERLINK.URL, address = links)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("setHyperlink.xlsx")

## End(Not run)
```

---

setMissingValue-methods

Setting missing value identifiers

Description

Defines the set of missing values (character or numeric) used when reading and writing data.

Usage

```
## S4 method for signature 'workbook,ANY'
setMissingValue(object,value)
```

Arguments

object	The workbook to use
value	vector or list of missing value identifiers (either character or numeric) that are recognized as missing (NA) when reading data. The first element of this vector will be used as missing value identifier when writing data. If value = NULL (default), missing values are represented by blank cells and only blank cells are recognized as missing.

Details

If there are no specific missing value identifiers defined the default behavior is to map missing values to blank (empty) cells. Otherwise, each string or numeric cell is checked if it matches one of the defined missing value identifiers. In addition, the first missing value identifier (i.e. the first element of the value argument) is used to represent missing values when writing data.
Note that the missing value identifiers have to be either character or numeric.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, writeNamedRegion, writeWorksheet

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("missingValue.xlsx", create = TRUE)

# Create a worksheet named 'airquality'
createSheet(wb, name = "airquality")

# Create a named region called 'airquality' on the sheet called
# 'airquality'
createName(wb, name = "airquality", formula = "airquality!$A$1")

# Set the missing value string to 'missing'
setMissingValue(wb, value = "missing")

# Write built-in data set 'airquality' to the above defined named region
writeNamedRegion(wb, airquality, name = "airquality")

# Save workbook
```

```
saveWorkbook(wb)

# clean up
file.remove("missingValue.xlsx")

## End(Not run)
```

`setRowHeight-methods` *Setting the height of a row in a worksheet*

Description

Sets the height of a row in a worksheet.

Usage

```
## S4 method for signature 'workbook,character'
setRowHeight(object,sheet,row,height)
## S4 method for signature 'workbook,numeric'
setRowHeight(object,sheet,row,height)
```

Arguments

`object`	The `workbook` to use
`sheet`	The name or index of the sheet to edit
`row`	The index of the row to resize
`height`	The height in points. If `height < 0` (default: -1), the row will be sized to the sheet's default row height.

Details

Note that the arguments `sheet`, `row` and `height` are vectorized. As such the row height of multiple rows (potentially on different worksheets) can be set with one method call.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `setColumnWidth`

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
mtcarsFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(mtcarsFile)

# Sets the row height of the 1st row on sheet 'mtcars'
# to 20 points
setRowHeight(wb, sheet = "mtcars", row = 1, height = 20)

## End(Not run)
```

---

setSheetColor-methods *Setting colors on worksheet tabs*

Description

Sets a color on a specified worksheet tab. This only works for xlsx files.

Usage

```
## S4 method for signature 'workbook,character'
setSheetColor(object,sheet,color)
## S4 method for signature 'workbook,numeric'
setSheetColor(object,sheet,color)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet on which to set the tab color
color	The color to use for the sheet tab. The color is normally specified via a corresponding color constant from the XLC object.

Author(s)

Nicola Lambiase
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, XLC

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("sheetcolor.xlsx", create = TRUE)

# Create a worksheet named 'Sheet1'
createSheet(wb, name = "Sheet1")

# Set the "Sheet1" tab color as red
setSheetColor(wb, "Sheet1", XLC$COLOR.RED)

# Create a worksheet named 'Sheet2'
createSheet(wb, name = "Sheet2")

# Set the tab color of the second workbook sheet as green
setSheetColor(wb, 2, XLC$COLOR.GREEN)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("sheetcolor.xlsx")

## End(Not run)
```

---

setSheetPos-methods *Setting worksheet position*

Description

Sets the position of a worksheets in a workbook.

Usage

```
## S4 method for signature 'workbook,character,numeric'
setSheetPos(object,sheet,pos)
```

Arguments

object	The workbook to use
sheet	The name of the worksheet (character) whose position to set. This argument is vectorized such that the positions of multiple worksheets can be set with one method call.
pos	The position index to set for the corresponding sheet. If missing, sheets will be positioned in the order they are specified in the argument sheet.

Details

It is important to note that the worksheet positions will be applied one after the other in the order they have been specified.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getSheetPos, getSheets

Examples

```
## Not run:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Move the 'mtcars3' worksheet to the front
setSheetPos(wb, sheet = "mtcars3", pos = 1)

## End(Not run)
```

---

setStyleAction-methods

Controlling application of cell styles when writing data to Excel

---

Description

Controls the application of cellstyles when writing data to Excel.

Usage

```
## S4 method for signature 'workbook'
setStyleAction(object,type)
```

Arguments

object	The workbook to use
type	Defines the style action to be used when writing data (writeNamedRegion, writeWorksheet) to the specified workbook object

Details

The following style actions are supported:

- `XLC$"STYLE_ACTION.XLCONNECT"`: This is the default. `data.frame` headers (if specified to be written) are colored in solid light grey (25 percent). character, numeric and logical vectors are written using Excel's "General" data format. Time/date vectors e.g. `Date` or `POSIXt`) are written with the "mm/dd/yyyy hh:mm:ss" data format. All cells are specified to wrap the text if necessary. The corresponding custom cell styles are called *XLConnect.Header*, *XLConnect.String*, *XLConnect.Numeric*, *XLConnect.Boolean* and *XLConnect.Date*.
- `XLC$"STYLE_ACTION.DATATYPE"`: This style action instructs **XLConnect** to apply `cellstyle`s per data type as set by the `setCellStyleForType` methods. In contrast to the `XLC$"STYLE_ACTION.DATA_FORMAT_ONLY` style action (see below) which only sets a data format to an existing cell style, this action actually sets a new `cellstyle`.
- `XLC$"STYLE_ACTION.NONE"`: This style action instructs **XLConnect** to apply no cell styles when writing data. Cell styles are kept as they are. This is useful in a scenario where all styling is predefined in an Excel template which is then only filled with data.
- `XLC$"STYLE_ACTION.PREDEFINED"`: This style action instructs **XLConnect** to use existing (predefined) `cellstyle`s when writing headers and columns. This is useful in a template-based approach where an Excel template with predefined `cellstyle`s for headers and columns is available. Normally, this would be used when the column dimensions (and potentially also the row dimensions) of the data tables are known up-front and as such a layout and corresponding cell styles can be pre-specified.
  If a `data.frame` is written including its header, it is assumed that the Excel file being written to has predefined `cellstyle`s in the header row. Furthermore, the first row of data is assumed to contain the cell styles to be replicated for any additional rows. As such, this style action may only be useful if the same column cell style should be applied across all rows. Please refer to the available demos for some examples.
- `XLC$"STYLE_ACTION.NAME_PREFIX"`: This style action instructs **XLConnect** to look for custom (named) `cellstyle`s with a specified prefix when writing columns and headers. This style name prefix can be set via the method `setStyleNamePrefix`.

  For column headers, it first checks if there is a cell style named
  <STYLE_NAME_PREFIX>.Header.<COLUMN_NAME>.
  If there is no such cell style, it checks for a cell style named
  <STYLE_NAME_PREFIX>.Header.<COLUMN_INDEX>.
  Again, if there is no such cell style, it checks for
  <STYLE_NAME_PREFIX>.Header
  (no specific column discrimination). As a final resort, it just takes the workbook default cell style.

  For columns, **XLConnect** first checks the availability of a cell style named
  <STYLE_NAME_PREFIX>.Column.<COLUMN_NAME>.
  If there is no such cell style, it checks for
  <STYLE_NAME_PREFIX>.Column.<COLUMN_INDEX>.
  If again there is no such cell style, it checks for
  <STYLE_NAME_PREFIX>.Column.<COLUMN_DATA_TYPE>

with *<COLUMN_DATA_TYPE>* being the corresponding data type from the set: *{Numeric, String, Boolean, DateTime}*. As a last resort, it would make use of the workbook's default cell style.

- `XLC$"STYLE_ACTION.DATA_FORMAT_ONLY"`: This style action instructs **XLConnect** to only set the data format for a cell but not to apply any other styling but rather keep the existing one. The data format to apply is determined by the data type of the cell (which is in turn determined by the corresponding R data type). The data format for a specific type can be set via the method `setDataFormatForType`. The default data format is "General" for the data types *Numeric*, *String* and *Boolean* and is "mm/dd/yyyy hh:mm:ss" for the data type *DateTime*.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `cellstyle`, `createCellStyle`, `writeNamedRegion`, `writeWorksheet`, `setStyleNamePrefix`, `setCellStyleForType`, `setDataFormatForType`

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("styleaction.xlsx", create = TRUE)

# Set style action to 'name prefix'
setStyleAction(wb, XLC$"STYLE_ACTION.NAME_PREFIX")
# Set the name prefix to 'MyPersonalStyle'
setStyleNamePrefix(wb, "MyPersonalStyle")

# We now create a named cell style to be used for the header
# (column names) of a data.frame
headerCellStyle <- createCellStyle(wb,
                            name = "MyPersonalStyle.Header")

# Specify the cell style to use a solid foreground color
setFillPattern(headerCellStyle,
               fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(headerCellStyle,
                    color = XLC$"COLOR.LIGHT_CORNFLOWER_BLUE")

# Specify a thick black bottom border
setBorder(headerCellStyle, side = "bottom",
          type = XLC$"BORDER.THICK",
          color = XLC$"COLOR.BLACK")

# We now create a named cell style to be used for
# the column named 'wt' (as you will see below, we will
```

```
# write the built-in data.frame 'mtcars')
wtColumnCellStyle <- createCellStyle(wb,
                           name = "MyPersonalStyle.Column.wt")

# Specify the cell style to use a solid foreground color
setFillPattern(wtColumnCellStyle,
               fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(wtColumnCellStyle,
                       color = XLC$"COLOR.LIGHT_ORANGE")

# We now create a named cell style to be used for
# the 3rd column in the data.frame
wtColumnCellStyle <- createCellStyle(wb,
                            name = "MyPersonalStyle.Column.3")

# Specify the cell style to use a solid foreground color
setFillPattern(wtColumnCellStyle,
               fill = XLC$"FILL.SOLID_FOREGROUND")

# Specify the foreground color to be used
setFillForegroundColor(wtColumnCellStyle,
                       color = XLC$"COLOR.LIME")

# Create a sheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' referring to
# the sheet called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$A$1")

# Write built-in data set 'mtcars' to the above defined named region.
# The style action 'name prefix' will be used when writing the data
# as defined above.
writeNamedRegion(wb, mtcars, name = "mtcars")

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("styleaction.xlsx")

## End(Not run)
```

---

setStyleNamePrefix-methods

Setting the style name prefix for the "name prefix" style action

---

Description

Sets the style name prefix for the "name prefix" style action.

Usage

```
## S4 method for signature 'workbook'
setStyleNamePrefix(object,prefix)
```

Arguments

object	The workbook to use
prefix	The name prefix

Details

Sets the prefix for the "name prefix" style action. See the method setStyleAction for more information.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setStyleAction, createCellStyle

---

setWrapText-methods *Specifying text wrapping behaviour*

Description

Specifies if text should be wrapped in a cell.

Usage

```
## S4 method for signature 'cellstyle'
setWrapText(object,wrap)
```

Arguments

object	The cellstyle to manipulate
wrap	If wrap = TRUE, the text is wrapped if it exceeds the width of the cell - otherwise not.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, cellstyle, setCellStyle, setStyleAction

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("setWrapText.xlsx", create = TRUE)

# Create a worksheet
createSheet(wb, name = "cellstyles")

# Create a dummy data set with some long text
text <- data.frame(
     Text = "Some very very very very very very very long text")

# Write the value to the 'cellstyles' worksheet in the
# top left corner (cell A1)
writeWorksheet(wb, text, sheet = "cellstyles", startRow = 1,
               startCol = 1, header = FALSE)

# Create a custom anonymous cell style
cs <- createCellStyle(wb)

# Specify to wrap the text
setWrapText(cs, wrap = TRUE)

# Set the cell style created above for the top left cell (A1)
# in the 'cellstyles' worksheet
setCellStyle(wb, sheet = "cellstyles", row = 1, col = 1,
             cellstyle = cs)

# Save the workbook
saveWorkbook(wb)

# clean up
file.remove("setWrapText.xlsx")

## End(Not run)
```

---

show-methods *Display a workbook object*

Description

Displays a workbook by printing it. This actually calls the workbook's print method.

Usage

```
## S4 method for signature 'workbook'
show(object)
```

Arguments

object	The workbook to display

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, print

Examples

```
## Not run:
# Load existing demo Excel file 'mtcars.xlsx' from the XLConnect package
wb.mtcars <- loadWorkbook(system.file("demoFiles/mtcars.xlsx",
                          package = "XLConnect"))

# Display the wb.mtcars object
wb.mtcars

# Alternatively, show can be called explicitely
show(wb.mtcars)

## End(Not run)
```

---

summary-methods *Summarizing workbook objects*

Description

Outputs a workbook summary including the underlying Excel filename, contained worksheets, hidden sheets, very hidden sheets, defined names and the active sheet name.

Usage

```
## S4 method for signature 'workbook'
summary(object)
```

Arguments

object	The workbook to summarize

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, getSheets, isSheetHidden, isSheetVeryHidden, getDefinedNames, getActiveSheetName

Examples

```
## Not run:
# Load existing demo Excel file 'mtcars.xlsx' from the XLConnect package
wb.mtcars <- loadWorkbook(system.file("demoFiles/mtcars.xlsx",
                          package = "XLConnect"))

# Print a workbook summary
summary(wb.mtcars)

## End(Not run)
```

---

swissfranc *Historical Exchange Rates: CHF vs EUR, USD and GBP*

Description

This data set provides historical exchange rates (CHF vs EUR, USD, GBP) in the time frame from January 1, 2014 to February 24, 2015. The exchange rates reflect bid prices with a 0% interbank rate.

Usage

```
swissfranc
```

Format

A data.frame with daily exchange rates in the mentioned time frame.

Source

retrieved via 'https://www.oanda.com/' - the retrieved time range is no longer available.

unhideSheet-methods *Unhiding worksheets in a workbook*

Description

Unhides the specified worksheets in a workbook.

Usage

```
## S4 method for signature 'workbook,character'
unhideSheet(object,sheet)
## S4 method for signature 'workbook,numeric'
unhideSheet(object,sheet)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet to unhide

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, hideSheet, isSheetHidden, isSheetVeryHidden, isSheetVisible

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("unhideWorksheet.xlsx", create = TRUE)

# Create sheet 'airquality'
createSheet(wb, name = "airquality")

# Write the built-in data set airquality to worksheet
# 'airquality'
writeWorksheet(wb, airquality, sheet = "airquality")

# Create sheet 'CO2'
createSheet(wb, name = "CO2")

# Write the built-in data set CO2 to worksheet 'C02'
writeWorksheet(wb, CO2, sheet = "CO2")

# Hide sheet 'airquality'
hideSheet(wb, sheet = "airquality")
```

```
# Unhide sheet 'airquality'
unhideSheet(wb, sheet = "airquality")

# clean up
file.remove("unhideWorksheet.xlsx")

## End(Not run)
```

---

unmergeCells-methods *Unmerging cells*

Description

Unmerges cells in a worksheet.

Usage

```
## S4 method for signature 'workbook,character'
unmergeCells(object,sheet,reference)
## S4 method for signature 'workbook,numeric'
unmergeCells(object,sheet,reference)
```

Arguments

object	The workbook to use
sheet	The name or index of the sheet on which to unmerge cells
reference	A cell range specification (character) in the form 'A1:B8'. Note that the specification must exactly correspond to the range of the merged cells.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook, mergeCells, idx2cref

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("unmergeCells.xlsx", create = TRUE)

# Create a worksheet called 'merge'
createSheet(wb, name = "merge")
```

```
# Merge the cells A1:B8 on the worksheet created above
mergeCells(wb, sheet = "merge", reference = "A1:B8")

# Unmerge the cells A1:B8
unmergeCells(wb, sheet = "merge", reference = "A1:B8")

# clean up
file.remove("unmergeCells.xlsx")

## End(Not run)
```

`with.workbook` *Evaluate an R expression in a workbook environment*

Description

Evaluate an R expression in an environment constructed from the named regions of an Excel workbook.

Usage

```
## S3 method for class 'workbook'
with(data, expr, ...)
```

Arguments

`data`	A workbook object, as returned by loadWorkbook.
`expr`	expression to evaluate
`...`	Additional arguments passed to readNamedRegion

Details

This method will read all named regions from the workbook when creating the environment. Names in the workbook will be processed through make.names to obtain the variable names.

Changes to the variables representing named regions will not affect the workbook contents and need to be saved explicitly using writeNamedRegion and saveWorkbook. If the workbook contains names that do not map to R identifiers,

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`getDefinedNames`, `readNamedRegion`,

Examples

```
## Not run:
# multiregion xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/multiregion.xlsx",
                             package = "XLConnect")

# load workbook
wb <- loadWorkbook(demoExcelFile)

# named regions: Calendar, IQ, Iris
print(getDefinedNames(wb))

# named regions as variables
with(wb, {
  print(Calendar)
  summary(IQ)
  summary(Iris)
})

## End(Not run)
```

`workbook-class` *Class "workbook"*

Description

This is **XLConnect**'s main entity representing a Microsoft Excel workbook. S4 objects of this class and corresponding methods are used to manipulate the underlying Excel workbook instances.

Objects from the Class

Objects can be created by calls of the form `loadWorkbook(filename, create)`. This is a shortcut form of `new("workbook", filename, create)` with some additional error checking.

Slots

`filename`**:** Object of class `character` which represents the filename of the underlying Microsoft Excel workbook.

`jobj`**:** Object of class `jobjRef` (see package **rJava**) which represents a Java object reference that is used in the back-end to manipulate the underlying Excel workbook instance.

Note: The `jobj` slot should not be accessed directly. `workbook` objects should only be manipulated via the corresponding methods.

Note

XLConnect supports both Excel 97-2003 (*.xls) and OOXML (Excel 2007+, *.xlsx) file formats.

A workbook's underlying Excel file is not saved (or being created in case the file did not exist and create = TRUE has been specified) unless the saveWorkbook method has been called on the object. This provides more flexibility to the user to decide when changes are saved and also provides better performance in that several changes can be written in one go (normally at the end, rather than after every operation causing the file to be rewritten again completely each time). This is due to the fact that workbooks are manipulated in-memory and are only written to disk with specifically calling saveWorkbook.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

Wikipedia: Office Open XML
https://en.wikipedia.org/wiki/Office_Open_XML

See Also

loadWorkbook, saveWorkbook

Examples

```
## Not run:
# Create a new workbook 'myWorkbook.xlsx'
# (assuming the file to not exist already)
wb <- loadWorkbook("myWorkbook.xlsx", create = TRUE)

# Create a worksheet called 'mtcars'
createSheet(wb, name = "mtcars")

# Write built-in dataset 'mtcars' to sheet 'mtcars' created above
writeWorksheet(wb, mtcars, sheet = "mtcars")

# Save workbook - this actually writes the file 'myWorkbook.xlsx' to disk
saveWorkbook(wb)

# clean up
file.remove("myWorkbook.xlsx")

## End(Not run)
```

---

writeNamedRegion-methods

Writing named regions to a workbook

Description

Writes data to the named regions defined in a workbook.

Usage

```
## S4 method for signature 'workbook,ANY'
writeNamedRegion(object, data, name, header,
  overwriteFormulaCells, rownames, worksheetScope)
```

Arguments

object	The workbook to use
data	Data to write
name	Name of the named region to write to
header	Specifies if the column names should be written. The default is TRUE.
overwriteFormulaCells	Specifies if existing formula cells in the workbook should be overwritten. The default is TRUE.
rownames	Name (character) of column to use for the row names of the provided data object. If specified, the row names of the data object (data.frame) will be included as an additional column with the specified name. If rownames = NULL (default), no row names will be included. May also be a list in case multiple data objects are written in one call (see below).
worksheetScope	Optional character vector with worksheet name(s) to limit the scope in which the name(s) to write to is/are expected to be found

.

Details

Writes data to the named region specified by name. Note that data is assumed to be a data.frame and is coerced to one if this is not already the case. The argument header specifies if the column names should be written. Note also that the arguments are vectorized and as such multiple named regions can be written with one call. In this case data is assumed to be a list of data objects (data.frame's).

Note

Named regions are automatically redefined to the area occupied by the written cells. This guarantees that the complete set of data can be re-read using readNamedRegion. Also, this allows the named region just to be defined as the top left cell to be written to. There is no need to know the exact size of the data in advance.

When writing data to Excel, writeNamedRegion further applies cell styles to the cells as defined by the workbook's "style action" (see setStyleAction).

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

References

What are named regions/ranges?
https://web.archive.org/web/20240821110221/https://www.officearticles.com/excel/named_ranges_in_microsoft_excel.htm
How to create named regions/ranges?
https://www.youtube.com/watch?v=iAE9a0uRtpM

See Also

workbook, writeWorksheet, appendNamedRegion, appendWorksheet, readNamedRegion, readWorksheet, writeNamedRegionToFile

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("writeNamedRegion.xlsx", create = TRUE)

# Create a worksheet named 'mtcars'
createSheet(wb, name = "mtcars")

# Create a named region called 'mtcars' on the sheet called 'mtcars'
createName(wb, name = "mtcars", formula = "mtcars!$A$1")

# Write built-in data set 'mtcars' to the above defined named region
# (using header = TRUE)
writeNamedRegion(wb, mtcars, name = "mtcars")

createSheet(wb, name="iris")
setActiveSheet(wb, "iris")

# Do the same with the iris data set, with a worksheet-scoped name
createName(wb, name = "iris", formula = "iris!$A$1", worksheetScope = "iris")
writeNamedRegion(wb, iris, name = "iris", worksheetScope="iris")

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("writeNamedRegion.xlsx")

## End(Not run)
```

---

writeNamedRegionToFile

Writing named regions to an Excel file (wrapper function)

Description

Writes named regions to an Excel file.

Usage

```
writeNamedRegionToFile(file, data, name, formula=NA, ..., worksheetScope = NULL,
styleAction = XLC$STYLE_ACTION.XLCONNECT, clearNamedRegions=FALSE)
```

Arguments

file	The path name of the file to write to
data	Data to write
name	Name of the named region to write to
formula	If formula is specified, each item defines the formula of the named region identified by the corresponding entry of name. Use this if you want to create the document from scratch instead of writing to a template!
worksheetScope	Optional character vector with worksheet name(s) to limit the scope in which the name(s) to write to is/are expected to be found. If not specified, the first matching named region is written to. Use "" to specifically target a globally-scoped named region.
...	Additional arguments passed to writeNamedRegion
styleAction	Style action to be used when writing the data. The default is XLC$STYLE_ACTION.XLCONNECT. See setStyleAction for more information.
clearNamedRegions	TRUE to clear content of existing named regions before writing data

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

writeNamedRegion, writeWorksheetToFile, readNamedRegionFromFile, readWorksheetFromFile

Examples

```
## Not run:
# multiregion xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/multiregion.xlsx",
                             package = "XLConnect")

template <- "template-ws.xlsx"
file.copy(demoExcelFile, template)

# Write single data.frame to a named region in an existing file
writeNamedRegionToFile(template, name = "Iris", iris)

# Write to a new file, defining the sheet and named region as we write.
# Format according to XLConnect defaults
```

```
writeNamedRegionToFile("iris.xlsx", name = "Iris", data = iris,
                       formula = "IrisData!$C$4",
                       styleAction = "XLCONNECT")
# clean up
file.remove("iris.xlsx")
file.remove("template-ws.xlsx")

## End(Not run)
```

`writeWorksheet-methods` *Writing data to worksheets*

Description

Writes data to worksheets of a workbook.

Usage

```
## S4 method for signature 'workbook,ANY,character'
writeWorksheet(object,data,sheet,startRow,startCol,header,overwriteFormulaCells,rownames)
## S4 method for signature 'workbook,ANY,numeric'
writeWorksheet(object,data,sheet,startRow,startCol,header,overwriteFormulaCells,rownames)
```

Arguments

`object`	The workbook to write to
`data`	Data to write
`sheet`	The name or index of the sheet to write to
`startRow`	Index of the first row to write to. The default is `startRow = 1`.
`startCol`	Index of the first column to write to. The default is `startCol = 1`.
`header`	Specifies if the column names should be written. The default is `TRUE`.
`overwriteFormulaCells`	Specifies if existing formula cells in the workbook should be overwritten. The default is `TRUE`.
`rownames`	Name (`character`) of column to use for the row names of the provided `data` object. If specified, the row names of the data object (`data.frame`) will be included as an additional column with the specified name. If `rownames = NULL` (default), no row names will be included. May also be a `list` in case multiple data objects are written in one call (see below).

Details

Writes `data` to the worksheet specified by `sheet`. `data` is assumed to be a `data.frame` and is coerced to one if this is not already the case. `startRow` and `startCol` define the top left corner of the data region to be written. Note that the arguments are vectorized and as such multiple data objects (`data.frame`'s) can be written to different worksheets in one call. In this case `data` is assumed to be a `list` of `data.frames`.

Note

When writing data to Excel, `writeWorksheet` further applies cell styles to the cells as defined by the `workbook`'s "style action" (see `setStyleAction`).

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`workbook`, `writeNamedRegion`, `appendWorksheet`, `appendNamedRegion`, `readWorksheet`, `readNamedRegion`, `writeWorksheetToFile`

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("writeWorksheet.xlsx", create = TRUE)

# Create a worksheet called 'CO2'
createSheet(wb, name = "CO2")

# Write built-in data set 'CO2' to the worksheet created above;
# offset from the top left corner and with default header = TRUE
writeWorksheet(wb, CO2, sheet = "CO2", startRow = 4, startCol = 2)

# Save workbook (this actually writes the file to disk)
saveWorkbook(wb)

# clean up
file.remove("writeWorksheet.xlsx")

## End(Not run)
```

---

writeWorksheetToFile *Writing data to worksheets in an Excel file (wrapper function)*

Description

Writes data to worksheets in an Excel file.

Usage

```
writeWorksheetToFile(file, data, sheet, ..., styleAction = XLC$STYLE_ACTION.XLCONNECT,
clearSheets = FALSE)
```

Arguments

file	The path name of the file to write to.
data	Data to write
sheet	The name or index of the sheet to write to
...	Additional arguments passed to writeWorksheet
styleAction	Style action to be used when writing the data - not vectorized! The default is XLC$STYLE_ACTION.XLCONNECT. See setStyleAction for more information.
clearSheets	TRUE to clear sheets before writing data.

Author(s)

Thomas Themel
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

writeWorksheet, writeNamedRegionToFile, readWorksheetFromFile, readNamedRegionFromFile

Examples

```
## Not run:
# multiregion xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/multiregion.xlsx",
                             package = "XLConnect")

# create a copy of the template
template <- "template-ws.xlsx"
file.copy(demoExcelFile, template)

# Write single data.frame to a specified location in an existing file
writeWorksheetToFile(template, data=iris, sheet="FirstSheet",
                     startRow=9, startCol = 9)

# create a copy of the template
template <- "template-multi-ws.xlsx"

# Write a few copies of the data.frame all over a new file
writeWorksheetToFile(template, data = list(i1 = iris, i2 = iris, i3 = iris),
                     sheet = c("FirstSheet", "SecondSheet", "FirstSheet"),
                     startRow = c(1,101,201), startCol = c(1,11,21))

# clean up
file.remove("template-multi-ws.xlsx")
file.remove("template-ws.xlsx")

## End(Not run)
```

XLC *XLConnect Constants*

Description

List structure defining several constants used across **XLConnect**.

Format

The format is:

```
List of 90
 $ ERROR.WARN : chr "WARN"
 $ ERROR.STOP : chr "STOP"
 $ DATA_TYPE.BOOLEAN            : chr "BOOLEAN"
 $ DATA_TYPE.NUMERIC            : chr "NUMERIC"
 $ DATA_TYPE.STRING             : chr "STRING"
 $ DATA_TYPE.DATETIME           : chr "DATETIME"
 $ STYLE_ACTION.XLCONNECT       : chr "XLCONNECT"
 $ STYLE_ACTION.NONE            : chr "NONE"
 $ STYLE_ACTION.PREDEFINED      : chr "PREDEFINED"
 $ STYLE_ACTION.NAME_PREFIX     : chr "STYLE_NAME_PREFIX"
 $ STYLE_ACTION.DATA_FORMAT_ONLY: chr "DATA_FORMAT_ONLY"
 $ BORDER.DASHED                : num 3
 $ BORDER.DASH_DOT              : num 9
 $ BORDER.DASH_DOT_DOT          : num 11
 $ BORDER.DOTTED                : num 7
 $ BORDER.DOUBLE                : num 6
 $ BORDER.HAIR                  : num 4
 $ BORDER.MEDIUM                : num 2
 $ BORDER.MEDIUM_DASHED         : num 8
 $ BORDER.MEDIUM_DASH_DOT       : num 10
 $ BORDER.MEDIUM_DASH_DOT_DOT   : num 12
 $ BORDER.NONE                  : num 0
 $ BORDER.SLANTED_DASH_DOT      : num 13
 $ BORDER.THICK                 : num 5
 $ BORDER.THIN                  : num 1
 $ COLOR.BLACK                  : num 8
 $ COLOR.WHITE                  : num 9
 $ COLOR.RED                    : num 10
 $ COLOR.BRIGHT_GREEN           : num 11
 $ COLOR.BLUE                   : num 12
 $ COLOR.YELLOW                 : num 13
 $ COLOR.PINK                   : num 14
 $ COLOR.TURQUOISE              : num 15
 $ COLOR.DARK_RED               : num 16
 $ COLOR.GREEN                  : num 17
```

```
$ COLOR.DARK_BLUE              : num 18
$ COLOR.DARK_YELLOW            : num 19
$ COLOR.VIOLET                 : num 20
$ COLOR.TEAL                   : num 21
$ COLOR.GREY_25_PERCENT        : num 22
$ COLOR.GREY_50_PERCENT        : num 23
$ COLOR.CORNFLOWER_BLUE        : num 24
$ COLOR.MAROON                 : num 25
$ COLOR.LEMON_CHIFFON          : num 26
$ COLOR.ORCHID                 : num 28
$ COLOR.CORAL                  : num 29
$ COLOR.ROYAL_BLUE             : num 30
$ COLOR.LIGHT_CORNFLOWER_BLUE  : num 31
$ COLOR.SKY_BLUE               : num 40
$ COLOR.LIGHT_TURQUOISE        : num 41
$ COLOR.LIGHT_GREEN            : num 42
$ COLOR.LIGHT_YELLOW           : num 43
$ COLOR.PALE_BLUE              : num 44
$ COLOR.ROSE                   : num 45
$ COLOR.LAVENDER               : num 46
$ COLOR.TAN                    : num 47
$ COLOR.LIGHT_BLUE             : num 48
$ COLOR.AQUA                   : num 49
$ COLOR.LIME                   : num 50
$ COLOR.GOLD                   : num 51
$ COLOR.LIGHT_ORANGE           : num 52
$ COLOR.ORANGE                 : num 53
$ COLOR.BLUE_GREY              : num 54
$ COLOR.GREY_40_PERCENT        : num 55
$ COLOR.DARK_TEAL              : num 56
$ COLOR.SEA_GREEN              : num 57
$ COLOR.DARK_GREEN             : num 58
$ COLOR.OLIVE_GREEN            : num 59
$ COLOR.BROWN                  : num 60
$ COLOR.PLUM                   : num 61
$ COLOR.INDIGO                 : num 62
$ COLOR.GREY_80_PERCENT        : num 63
$ COLOR.AUTOMATIC              : num 64
$ FILL.NO_FILL                 : num 0
$ FILL.SOLID_FOREGROUND        : num 1
$ FILL.FINE_DOTS               : num 2
$ FILL.ALT_BARS                : num 3
$ FILL.SPARSE_DOTS             : num 4
$ FILL.THICK_HORZ_BANDS        : num 5
$ FILL.THICK_VERT_BANDS        : num 6
$ FILL.THICK_BACKWARD_DIAG     : num 7
$ FILL.THICK_FORWARD_DIAG      : num 8
$ FILL.BIG_SPOTS               : num 9
```

```
$ FILL.BRICKS                : num 10
$ FILL.THIN_HORZ_BANDS       : num 11
$ FILL.THIN_VERT_BANDS       : num 12
$ FILL.THIN_BACKWARD_DIAG    : num 13
$ FILL.THIN_FORWARD_DIAG     : num 14
$ FILL.SQUARES               : num 15
$ FILL.DIAMONDS              : num 16
```

Details

The XLC list structure defines several constants used throughout **XLConnect**. The general convention for enumeration types is to address corresponding constants via *XLC$"<ENUM_TYPE>.<VALUE>"* where *<ENUM_TYPE>* specifies the name of the enumeration and *<VALUE>* specifies a corresponding enumeration value. An example is XLC$"COLOR.RED" where "COLOR" is the enumeration type and "RED" is the corresponding color enumeration value.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

---

xlcDump *Dumping data sets to Excel files*

Description

Dumps data sets to Excel files by writing each object to a separate worksheet.

Usage

```
xlcDump(list, ..., file = "dump.xlsx", pos = -1, overwrite = FALSE)
```

Arguments

list	character vector of names of objects inside environment pos to dump into an Excel file. Objects will be written using writeWorksheet - as such any object will be coerced to a data.frame. If missing, the list of objects will be determined via the function ls which takes any arguments specified via ....
...	Arguments that will be passed to the ls function for getting a list of object names in case the list argument is missing.
file	Excel file to which objects will be dumped. Can be an existing or a new file. Defaults to "dump.xlsx".
pos	Environment in which to look for objects. Can be specified either as an integer specifying the position in the search list, as a character naming an element in the search list or as an environment. Defaults to -1 which refers to the current environment.

`overwrite` `logical` specifying if data should be overwritten if objects with the same name have already been dumped to the Excel `file`.

Details

Each object is written to a separate worksheet named by the name of the object. Objects are written using the `writeWorksheet` method - as such any object will be coerced to `data.frame`.

Value

Named `logical` vector specifying if objects have been dumped or not. An object may not be dumped because there was an issue with the coercion to a `data.frame` or the object already existed (and `overwrite = FALSE`) in the workbook.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`xlcRestore`, `writeNamedRegion`, `writeWorksheet`, `writeNamedRegionToFile`, `writeWorksheetToFile`, `xlcEdit`

Examples

```
## Not run:
require(datasets)
xlcDump(c("airquality", "CO2", "iris", "PlantGrowth", "swiss"),
        file = "myDump.xlsx", pos = "package:datasets")
xlcRestore(file = "myDump.xlsx", overwrite = TRUE)
# clean up
file.remove("myDump.xlsx")

## End(Not run)
```

---

xlcEdit *Editing data sets in an Excel file editor*

Description

Provides the capability to edit an object/data.frame in an Excel file editor. After editing, the object is restored in the R session with the corresponding changes.

Usage

```
xlcEdit(obj, pos = globalenv(), ext = ".xlsx")
```

Arguments

obj	Object (data.frame) to edit.
pos	Where to look for the object specified by obj. See pos argument of get for more information.
ext	Extension to use for the Excel file being created. Defaults to ".xlsx".

Details

This function uses xlcDump and xlcRestore to dump objects to and restore objects from Excel files. An OS command is invoked to open the temporary Excel file in the default editor. Changes to the file have to be saved in order for them to take effect in the restored object.

Value

Invisibly returns the value of the xlcRestore operation.

Note

This function only works under Windows and MacOS with a corresponding Excel file editor, e.g. MS Excel or LibreOffice. Attempts to use this function under another OS will result in an error being thrown.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

xlcDump, xlcRestore, writeNamedRegion, writeWorksheet, writeNamedRegionToFile, writeWorksheetToFile

Examples

```
## Not run:
myObj = mtcars
xlcEdit(myObj)

## End(Not run)
```

`xlcFreeMemory` *Freeing Java Virtual Machine memory*

Description

Frees Java Virtual Machine (JVM) memory.

Usage

```
xlcFreeMemory(...)
```

Arguments

`...`	Further arguments to be passed to R's garbage collector (`gc`).

Details

This function uses Java's Runtime class to run the garbage collector.
Java memory is freed by first running R's garbage collector (`gc`) and then Java's garbage collector. This sequence is important as R's `gc` may release objects which in turn allows Java's garbage collector to release some objects.

Note, in general there should be no need to make active use of this with **XLConnect**. Both R and Java automatically perform garbage collection at times. However, this function might be useful to reclaim memory after removing a large data object that has been written/read to/from Excel.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

`xlcMemoryReport`, `gc`

Examples

```
## Not run:
xlcFreeMemory()

## End(Not run)
```

xlcMemoryReport *Reporting free Java Virtual Machine memory*

Description

Reports the amount of free memory in the Java Virtual Machine (JVM).

Usage

```
xlcMemoryReport()
```

Details

This function uses Java's Runtime class to query the free JVM memory.

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

xlcFreeMemory

Examples

```
## Not run:
xlcMemoryReport()

## End(Not run)
```

XLConnect-deprecated *Deprecated functions in package* **XLConnect**

Description

These functions are provided for compatibility with older versions of **XLConnect** only, and will be defunct in a later release.

Details

The following functions are deprecated and will be made defunct. Use the replacements as indicated.

- getReferenceCoordinates: getReferenceCoordinatesForName

xlcRestore *Restoring objects from Excel files*

Description

Restores objects from Excel files that have been dumped using xlcDump.

Usage

```
xlcRestore(file = "dump.xlsx", pos = -1, overwrite = FALSE)
```

Arguments

file	Excel file from which to restore objects. This is normally a file that has been produced with xlcDump. Defaults to "dump.xlsx".
pos	Environment into which to restore objects. Can be specified either as an integer specifying the position in the search list, as a character naming an element in the search list or as an environment. Defaults to -1 which refers to the current environment.
overwrite	logical specifying if data objects should be overwritten if they already exist inside the environment pos.

Value

Named logical vector specifying if objects have been restored or not. An object may not be restored because there was an issue with reading the data from the worksheet or the object already existed in the environment pos (and overwrite = FALSE).

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

xlcDump, readNamedRegion, readWorksheet, readNamedRegionFromFile, readWorksheetFromFile, xlcEdit

Examples

```
## Not run:
require(datasets)
xlcDump(c("airquality", "CO2", "iris", "PlantGrowth", "swiss"),
        file = "myDump.xlsx", pos = "package:datasets")
xlcRestore(file = "myDump.xlsx", overwrite = TRUE)
# clean up
file.remove("myDump.xlsx")

## End(Not run)
```

`$-methods` *Executing workbook methods in object$method(...) form*

Description

Allows to execute workbook methods in `workbook-object$method(...)` form.

Arguments

`x`	The object (workbook, cellstyle) to use

Details

`x$method(...)` (where `x` is a workbook-object) is equivalent to `method(x, ...)`

Note

The workbook `$`-operator allows to call workbook-methods in `workbook-object$method(...)` form. This form might be considered more convenient or readable for programmers coming from other object-oriented languages such as Java, C#, ...

Author(s)

Martin Studer
Mirai Solutions GmbH https://mirai-solutions.ch

See Also

workbook

Examples

```
## Not run:
# Load workbook (create if not existing)
wb <- loadWorkbook("dollar.xlsx", create = TRUE)

# Create a worksheet called 'CO2'
wb$createSheet(name = "CO2")

# Write built-in data set 'CO2' to the worksheet created above
wb$writeWorksheet(CO2, sheet = "CO2", startRow = 4, startCol = 2)

# Save workbook
wb$saveWorkbook()

# clean up
file.remove("dollar.xlsx")

## End(Not run)
```

Index

∗ **IO**
appendNamedRegion-methods, 6
appendWorksheet-methods, 8
configurePOI, 19
readNamedRegion, 67
readNamedRegionFromFile, 71
readTable, 72
readWorksheet-methods, 75
readWorksheetFromFile, 80
with.workbook, 120
writeNamedRegion-methods, 122
writeNamedRegionToFile, 124
writeWorksheet-methods, 126
writeWorksheetToFile, 127
∗ **classes**
cellstyle-class, 11
workbook-class, 121
∗ **datasets**
mirai, 64
swissfranc, 117
XLC, 129
∗ **error**
onErrorCell-methods, 65
∗ **file**
loadWorkbook, 62
saveWorkbook-methods, 85
xlcDump, 131
xlcRestore, 136
∗ **list**
XLC, 129
∗ **methods**
$-methods, 137
addImage-methods, 5
appendNamedRegion-methods, 6
appendWorksheet-methods, 8
clearNamedRegion-methods, 13
clearRange-methods, 14
clearRangeFromReference-methods, 15
clearSheet-methods, 16
cloneSheet-methods, 17
createCellStyle-methods, 21
createFreezePane-methods, 23
createName-methods, 24
createSheet-methods, 26
createSplitPane-methods, 27
existsCellStyle-methods, 29
existsName-methods, 30
existsSheet-methods, 31
extraction-methods, 32
getActiveSheetIndex-methods, 35
getActiveSheetName-methods, 36
getBoundingBox-methods, 37
getCellFormula-methods, 38
getCellStyle-methods, 39
getCellStyleForType-methods, 41
getDefinedNames-methods, 42
getForceFormulaRecalculation-methods, 43
getLastColumn-methods, 44
getLastRow-methods, 45
getOrCreateCellStyle-methods, 46
getReferenceCoordinates-methods, 47
getReferenceCoordinatesForName-methods, 48
getReferenceCoordinatesForTable-methods, 49
getReferenceFormula-methods, 50
getSheetPos-methods, 51
getSheets-methods, 52
getTables-methods, 53
hideSheet-methods, 54
isSheetHidden-methods, 58
isSheetVeryHidden-methods, 59
isSheetVisible-methods, 61
mergeCells-methods, 63
print-methods, 66

readNamedRegion, 67
readTable, 72
readWorksheet-methods, 75
removeName-methods, 81
removePane-methods, 82
removeSheet-methods, 83
renameSheet-methods, 84
saveWorkbook-methods, 85
setActiveSheet-methods, 87
setAutoFilter-methods, 88
setBorder-methods, 89
setCellFormula-methods, 90
setCellStyle-methods, 92
setCellStyleForType-methods, 94
setColumnWidth-methods, 95
setDataFormat-methods, 96
setDataFormatForType-methods, 98
setFillBackgroundColor-methods, 99
setFillForegroundColor-methods, 100
setFillPattern-methods, 101
setForceFormulaRecalculation-methods, 103
setHyperlink-methods, 104
setMissingValue-methods, 105
setRowHeight-methods, 107
setSheetColor-methods, 108
setSheetPos-methods, 109
setStyleAction-methods, 110
setStyleNamePrefix-methods, 113
setWrapText-methods, 114
show-methods, 115
summary-methods, 116
unhideSheet-methods, 118
unmergeCells-methods, 119
writeNamedRegion-methods, 122
writeWorksheet-methods, 126

∗ **misc**

xlcEdit, 132

∗ **package**

XLConnect-package, 4

∗ **print**

print-methods, 66
show-methods, 115
summary-methods, 116

∗ **utilities**

$-methods, 137
addImage-methods, 5
aref, 9
aref2idx, 10
cellstyle-class, 11
clearNamedRegion-methods, 13
clearRange-methods, 14
clearRangeFromReference-methods, 15
clearSheet-methods, 16
cloneSheet-methods, 17
col2idx, 19
createCellStyle-methods, 21
createFreezePane-methods, 23
createName-methods, 24
createSheet-methods, 26
createSplitPane-methods, 27
cref2idx, 28
existsCellStyle-methods, 29
existsName-methods, 30
existsSheet-methods, 31
extraction-methods, 32
extractSheetName, 34
getActiveSheetIndex-methods, 35
getActiveSheetName-methods, 36
getBoundingBox-methods, 37
getCellFormula-methods, 38
getCellStyle-methods, 39
getCellStyleForType-methods, 41
getDefinedNames-methods, 42
getForceFormulaRecalculation-methods, 43
getLastColumn-methods, 44
getLastRow-methods, 45
getOrCreateCellStyle-methods, 46
getReferenceCoordinates-methods, 47
getReferenceCoordinatesForName-methods, 48
getReferenceCoordinatesForTable-methods, 49
getReferenceFormula-methods, 50
getSheetPos-methods, 51
getSheets-methods, 52
getTables-methods, 53
hideSheet-methods, 54
idx2aref, 55
idx2col, 56
idx2cref, 57
isSheetHidden-methods, 58

isSheetVeryHidden-methods, 59
isSheetVisible-methods, 61
mergeCells-methods, 63
onErrorCell-methods, 65
removeName-methods, 81
removePane-methods, 82
removeSheet-methods, 83
renameSheet-methods, 84
setActiveSheet-methods, 87
setAutoFilter-methods, 88
setBorder-methods, 89
setCellFormula-methods, 90
setCellStyle-methods, 92
setCellStyleForType-methods, 94
setColumnWidth-methods, 95
setDataFormat-methods, 96
setDataFormatForType-methods, 98
setFillBackgroundColor-methods, 99
setFillForegroundColor-methods, 100
setFillPattern-methods, 101
setForceFormulaRecalculation-methods, 103
setHyperlink-methods, 104
setMissingValue-methods, 105
setRowHeight-methods, 107
setSheetColor-methods, 108
setSheetPos-methods, 109
setStyleAction-methods, 110
setStyleNamePrefix-methods, 113
setWrapText-methods, 114
summary-methods, 116
unhideSheet-methods, 118
unmergeCells-methods, 119
XLC, 129
xlcDump, 131
xlcEdit, 132
xlcFreeMemory, 134
xlcMemoryReport, 135
xlcRestore, 136
[*(extraction-methods)*, 32
[,workbook-method *(extraction-methods)*, 32
[-methods *(extraction-methods)*, 32
[<- *(extraction-methods)*, 32
[<-,workbook-method *(extraction-methods)*, 32
[<--methods *(extraction-methods)*, 32
[[*(extraction-methods)*, 32
[[,workbook-method *(extraction-methods)*, 32
[[-methods *(extraction-methods)*, 32
[[<- *(extraction-methods)*, 32
[[<-,workbook-method *(extraction-methods)*, 32
[[<--methods *(extraction-methods)*, 32
$ *($-methods)*, 137
$,cellstyle-method *($-methods)*, 137
$,workbook-method *($-methods)*, 137
$-methods, 137

addImage *(addImage-methods)*, 5
addImage,workbook-method *(addImage-methods)*, 5
addImage-methods, 5
appendNamedRegion, *8*, *124*, *127*
appendNamedRegion *(appendNamedRegion-methods)*, 6
appendNamedRegion,workbook,ANY-method *(appendNamedRegion-methods)*, 6
appendNamedRegion,workbook-method *(appendNamedRegion-methods)*, 6
appendNamedRegion-methods, 6
appendWorksheet, *7*, *124*, *127*
appendWorksheet *(appendWorksheet-methods)*, 8
appendWorksheet,workbook,ANY,character-method *(appendWorksheet-methods)*, 8
appendWorksheet,workbook,ANY,numeric-method *(appendWorksheet-methods)*, 8
appendWorksheet-methods, 8
aref, 9, *10*, *19*, *29*, *56–58*
aref2idx, *9*, 10, *14*, *19*, *29*, *56–58*

cellstyle, *21*, *29, 30*, *40, 41*, *46, 47*, *89, 90*, *92–94*, *96, 97*, *99–102*, *105*, *110–112*, *114, 115*, *137*
cellstyle-class, 11
clearNamedRegion, *15–17*
clearNamedRegion *(clearNamedRegion-methods)*, 13
clearNamedRegion,workbook,character-method *(clearNamedRegion-methods)*, 13
clearNamedRegion-methods, 12
clearRange, *13*, *16, 17*
clearRange *(clearRange-methods)*, 14

clearRange,workbook,character-method
(clearRange-methods), 14
clearRange,workbook,numeric-method
(clearRange-methods), 14
clearRange-methods, 14
clearRangeFromReference, *13*, *15*, *17*
clearRangeFromReference
(clearRangeFromReference-methods), 15
clearRangeFromReference,workbook,character-method
(clearRangeFromReference-methods), 15
clearRangeFromReference-methods, 15
clearSheet, *13*, *15*, *16*
clearSheet (clearSheet-methods), 16
clearSheet,workbook,character-method
(clearSheet-methods), 16
clearSheet,workbook,numeric-method
(clearSheet-methods), 16
clearSheet-methods, 16
cloneSheet, *26*, *32*, *84*, *85*
cloneSheet (cloneSheet-methods), 17
cloneSheet,workbook,character-method
(cloneSheet-methods), 17
cloneSheet,workbook,numeric-method
(cloneSheet-methods), 17
cloneSheet-methods, 17
col2idx, *9*, *10*, 19, *29*, *56–58*
configurePOI, 19
createCellStyle, *11*, *30*, *40*, *47*, *93*, *112*, *114*
createCellStyle
(createCellStyle-methods), 21
createCellStyle,workbook,character-method
(createCellStyle-methods), 21
createCellStyle,workbook,missing-method
(createCellStyle-methods), 21
createCellStyle-methods, 21
createFreezePane, *28*, *83*
createFreezePane
(createFreezePane-methods), 23
createFreezePane,workbook,character-method
(createFreezePane-methods), 23
createFreezePane,workbook,numeric-method
(createFreezePane-methods), 23
createFreezePane-methods, 23
createName, *5*, *31*, *43*, *48*, *49*, *51*, *82*
createName (createName-methods), 24
createName,workbook-method
(createName-methods), 24
createName-methods, 24
createSheet, *18*, *32*, *52*, *84*, *85*, *88*
createSheet (createSheet-methods), 26
createSheet,workbook-method
(createSheet-methods), 26
createSheet-methods, 26
createSplitPane, *24*, *83*
createSplitPane
(createSplitPane-methods), 27
createSplitPane,workbook,character-method
(createSplitPane-methods), 27
createSplitPane,workbook,numeric-method
(createSplitPane-methods), 27
createSplitPane-methods, 27
cref2idx, *10*, *19*, 28, *56–58*

data.frame, *68*, *73*, *76*

existsCellStyle, *21*, *40*, *47*
existsCellStyle
(existsCellStyle-methods), 29
existsCellStyle,workbook-method
(existsCellStyle-methods), 29
existsCellStyle-methods, 29
existsName, *25*, *43*, *48*, *49*, *51*, *82*
existsName (existsName-methods), 30
existsName,workbook-method
(existsName-methods), 30
existsName-methods, 30
existsSheet, *18*, *26*, *84*, *85*, *88*
existsSheet (existsSheet-methods), 31
existsSheet,workbook-method
(existsSheet-methods), 31
existsSheet-methods, 31
extraction-methods, 32
extractSheetName, 34

gc, *134*
get, *133*
getActiveSheetIndex, *36*
getActiveSheetIndex
(getActiveSheetIndex-methods), 35
getActiveSheetIndex,workbook-method
(getActiveSheetIndex-methods), 35
getActiveSheetIndex-methods, 35
getActiveSheetName, *35*, *117*

getActiveSheetName
(getActiveSheetName-methods), 36
getActiveSheetName,workbook-method
(getActiveSheetName-methods), 36
getActiveSheetName-methods, 36
getBoundingBox
(getBoundingBox-methods), 37
getBoundingBox,workbook,character-method
(getBoundingBox-methods), 37
getBoundingBox,workbook,numeric-method
(getBoundingBox-methods), 37
getBoundingBox-methods, 37
getCellFormula, *91*
getCellFormula
(getCellFormula-methods), 38
getCellFormula,workbook,character-method
(getCellFormula-methods), 38
getCellFormula,workbook,numeric-method
(getCellFormula-methods), 38
getCellFormula-methods, 38
getCellStyle (getCellStyle-methods), 39
getCellStyle,workbook-method
(getCellStyle-methods), 39
getCellStyle-methods, 39
getCellStyleForType, *94*
getCellStyleForType
(getCellStyleForType-methods), 41
getCellStyleForType,workbook-method
(getCellStyleForType-methods), 41
getCellStyleForType-methods, 41
getDefinedNames, *25*, *31*, *82*, *117*, *120*
getDefinedNames
(getDefinedNames-methods), 42
getDefinedNames,workbook-method
(getDefinedNames-methods), 42
getDefinedNames-methods, 42
getForceFormulaRecalculation, *103*
getForceFormulaRecalculation
(getForceFormulaRecalculation-methods), 43
getForceFormulaRecalculation,workbook,character-method
(getForceFormulaRecalculation-methods), 43
getForceFormulaRecalculation,workbook,numeric-method
(getForceFormulaRecalculation-methods), 43
getForceFormulaRecalculation-methods, 43
getLastColumn (getLastColumn-methods), 44
getLastColumn,workbook,character-method
(getLastColumn-methods), 44
getLastColumn,workbook,numeric-method
(getLastColumn-methods), 44
getLastColumn-methods, 44
getLastRow (getLastRow-methods), 45
getLastRow,workbook,character-method
(getLastRow-methods), 45
getLastRow,workbook,numeric-method
(getLastRow-methods), 45
getLastRow-methods, 45
getOrCreateCellStyle, *21*, *30*, *40*
getOrCreateCellStyle
(getOrCreateCellStyle-methods), 46
getOrCreateCellStyle,workbook,character-method
(getOrCreateCellStyle-methods), 46
getOrCreateCellStyle-methods, 46
getReferenceCoordinates
(getReferenceCoordinates-methods), 47
getReferenceCoordinates,workbook-method
(getReferenceCoordinates-methods), 47
getReferenceCoordinates-methods, 47
getReferenceCoordinatesForName, *47*, *49*, *135*
getReferenceCoordinatesForName
(getReferenceCoordinatesForName-methods), 48
getReferenceCoordinatesForName,workbook-method
(getReferenceCoordinatesForName-methods), 48
getReferenceCoordinatesForName-methods, 48
getReferenceCoordinatesForTable, *49*
getReferenceCoordinatesForTable
(getReferenceCoordinatesForTable-methods), 49
getReferenceCoordinatesForTable,workbook,character-method
(getReferenceCoordinatesForTable-methods),

49
getReferenceCoordinatesForTable,workbook,numeric-method (getReferenceCoordinatesForTable-methods), 49
getReferenceCoordinatesForTable-methods, 49
getReferenceFormula, *48, 49*
getReferenceFormula (getReferenceFormula-methods), 50
getReferenceFormula,workbook-method (getReferenceFormula-methods), 50
getReferenceFormula-methods, 50
getSheetPos, *52*, *110*
getSheetPos (getSheetPos-methods), 51
getSheetPos,workbook,character-method (getSheetPos-methods), 51
getSheetPos-methods, 51
getSheets, *18*, *26*, *32*, *43*, *51*, *53*, *84, 85*, *88*, *110*, *117*
getSheets (getSheets-methods), 52
getSheets,workbook-method (getSheets-methods), 52
getSheets-methods, 52
getTables (getTables-methods), 53
getTables,workbook,character-method (getTables-methods), 53
getTables,workbook,numeric-method (getTables-methods), 53
getTables-methods, 53

hideSheet, *58*, *60, 61*, *118*
hideSheet (hideSheet-methods), 54
hideSheet,workbook,character-method (hideSheet-methods), 54
hideSheet,workbook,numeric-method (hideSheet-methods), 54
hideSheet-methods, 54

idx2aref, *9, 10*, *19*, *29*, 55, *57, 58*
idx2col, *9, 10*, *19*, *29*, *56*, 56, *58*
idx2cref, *9, 10*, *19*, *29*, *56, 57*, 57, *64*, *119*
isSheetHidden, *55*, *60, 61*, *117, 118*
isSheetHidden (isSheetHidden-methods), 58
isSheetHidden,workbook,character-method (isSheetHidden-methods), 58
isSheetHidden,workbook,numeric-method (isSheetHidden-methods), 58
isSheetHidden-methods, 58
isSheetVeryHidden, *55*, *58*, *61*, *117, 118*
isSheetVeryHidden (isSheetVeryHidden-methods), 59
isSheetVeryHidden,workbook,character-method (isSheetVeryHidden-methods), 59
isSheetVeryHidden,workbook,numeric-method (isSheetVeryHidden-methods), 59
isSheetVeryHidden-methods, 59
isSheetVisible, *55*, *58*, *60*, *118*
isSheetVisible (isSheetVisible-methods), 61
isSheetVisible,workbook,character-method (isSheetVisible-methods), 61
isSheetVisible,workbook,numeric-method (isSheetVisible-methods), 61
isSheetVisible-methods, 61

loadWorkbook, 62, *86*, *120–122*

make.names, *120*
mergeCells, *119*
mergeCells (mergeCells-methods), 63
mergeCells,workbook,character-method (mergeCells-methods), 63
mergeCells,workbook,numeric-method (mergeCells-methods), 63
mergeCells-methods, 63
mirai, 64

onErrorCell, *70*, *72*, *74*, *78*, *80*
onErrorCell (onErrorCell-methods), 65
onErrorCell,workbook-method (onErrorCell-methods), 65
onErrorCell-methods, 65

print, *116*
print (print-methods), 66
print,workbook-method (print-methods), 66
print-methods, 66

readNamedRegion, *7, 8*, *25*, *31*, *33*, *43*, *65*, 67, *71, 72*, *74*, *78*, *82*, *120*, *123, 124*, *127*, *136*
readNamedRegion,workbook-method (readNamedRegion), 67

readNamedRegion-methods
(readNamedRegion), 67
readNamedRegionFromFile, *65*, *70*, 71, *74*, *80*, *125*, *128*, *136*
readTable, *53*, *70*, 72, *78*
readTable,workbook,character-method
(readTable), 72
readTable,workbook,numeric-method
(readTable), 72
readTable-methods (readTable), 72
readWorksheet, *7*, *8*, *33*, *65*, *70*, *74*, *80*, *124*, *127*, *136*
readWorksheet (readWorksheet-methods), 75
readWorksheet,workbook,character-method
(readWorksheet-methods), 75
readWorksheet,workbook,numeric-method
(readWorksheet-methods), 75
readWorksheet-methods, 75
readWorksheetFromFile, *65*, *72*, *78*, 80, *125*, *128*, *136*
removeName, *25*, *31*, *43*, *48*, *49*, *51*
removeName (removeName-methods), 81
removeName,workbook-method
(removeName-methods), 81
removeName-methods, 81
removePane, *24*, *28*
removePane (removePane-methods), 82
removePane,workbook,character-method
(removePane-methods), 82
removePane,workbook,numeric-method
(removePane-methods), 82
removePane-methods, 82
removeSheet, *18*, *26*, *32*, *52*, *85*, *88*
removeSheet (removeSheet-methods), 83
removeSheet,workbook,character-method
(removeSheet-methods), 83
removeSheet,workbook,numeric-method
(removeSheet-methods), 83
removeSheet-methods, 83
renameSheet, *18*, *26*, *32*, *52*, *84*, *88*
renameSheet (renameSheet-methods), 84
renameSheet,workbook,character-method
(renameSheet-methods), 84
renameSheet,workbook,numeric-method
(renameSheet-methods), 84
renameSheet-methods, 84

saveWorkbook, *63*, *120*, *122*
saveWorkbook (saveWorkbook-methods), 85
saveWorkbook,workbook,character-method
(saveWorkbook-methods), 85
saveWorkbook,workbook,missing-method
(saveWorkbook-methods), 85
saveWorkbook-methods, 85
setActiveSheet, *84*, *85*
setActiveSheet
(setActiveSheet-methods), 87
setActiveSheet,workbook,character-method
(setActiveSheet-methods), 87
setActiveSheet,workbook,numeric-method
(setActiveSheet-methods), 87
setActiveSheet-methods, 87
setAutoFilter (setAutoFilter-methods), 88
setAutoFilter,workbook,character-method
(setAutoFilter-methods), 88
setAutoFilter,workbook,numeric-method
(setAutoFilter-methods), 88
setAutoFilter-methods, 88
setBorder, *21*, *40*, *93*
setBorder (setBorder-methods), 89
setBorder,cellstyle-method
(setBorder-methods), 89
setBorder-methods, 89
setCellFormula, *39*
setCellFormula
(setCellFormula-methods), 90
setCellFormula,workbook,character-method
(setCellFormula-methods), 90
setCellFormula,workbook,numeric-method
(setCellFormula-methods), 90
setCellFormula-methods, 90
setCellStyle, *11*, *21*, *30*, *40*, *47*, *90*, *97*, *99*, *101*, *102*, *105*, *115*
setCellStyle (setCellStyle-methods), 92
setCellStyle,workbook,character,missing-method
(setCellStyle-methods), 92
setCellStyle,workbook,missing,character-method
(setCellStyle-methods), 92
setCellStyle,workbook,missing,numeric-method
(setCellStyle-methods), 92
setCellStyle-methods, 92
setCellStyleForType, *41*, *111*, *112*
setCellStyleForType
(setCellStyleForType-methods), 94

setCellStyleForType,workbook-method
(setCellStyleForType-methods), 94
setCellStyleForType-methods, 94
setColumnWidth, *107*
setColumnWidth
(setColumnWidth-methods), 95
setColumnWidth,workbook,character-method
(setColumnWidth-methods), 95
setColumnWidth,workbook,numeric-method
(setColumnWidth-methods), 95
setColumnWidth-methods, 95
setDataFormat, *21*, *40*, *93*
setDataFormat (setDataFormat-methods), 96
setDataFormat,cellstyle-method
(setDataFormat-methods), 96
setDataFormat-methods, 96
setDataFormatForType, *112*
setDataFormatForType
(setDataFormatForType-methods), 98
setDataFormatForType,workbook-method
(setDataFormatForType-methods), 98
setDataFormatForType-methods, 98
setFillBackgroundColor, *21*, *40*, *93*
setFillBackgroundColor
(setFillBackgroundColor-methods), 99
setFillBackgroundColor,cellstyle,numeric-method
(setFillBackgroundColor-methods), 99
setFillBackgroundColor-methods, 99
setFillForegroundColor, *21*, *40*, *93*
setFillForegroundColor
(setFillForegroundColor-methods), 100
setFillForegroundColor,cellstyle,numeric-method
(setFillForegroundColor-methods), 100
setFillForegroundColor-methods, 100
setFillPattern, *21*, *40*, *93*
setFillPattern
(setFillPattern-methods), 101
setFillPattern,cellstyle-method
(setFillPattern-methods), 101
setFillPattern-methods, 101
setForceFormulaRecalculation, *43*
setForceFormulaRecalculation
(setForceFormulaRecalculation-methods), 103
setForceFormulaRecalculation,workbook,character-method
(setForceFormulaRecalculation-methods), 103
setForceFormulaRecalculation,workbook,numeric-method
(setForceFormulaRecalculation-methods), 103
setForceFormulaRecalculation-methods, 103
setHyperlink (setHyperlink-methods), 104
setHyperlink,workbook,character,missing-method
(setHyperlink-methods), 104
setHyperlink,workbook,missing,character-method
(setHyperlink-methods), 104
setHyperlink,workbook,missing,numeric-method
(setHyperlink-methods), 104
setHyperlink-methods, 104
setMissingValue, *69*, *74*, *77*
setMissingValue
(setMissingValue-methods), 105
setMissingValue,workbook,ANY-method
(setMissingValue-methods), 105
setMissingValue,workbook-method
(setMissingValue-methods), 105
setMissingValue-methods, 105
setRowHeight, *96*
setRowHeight (setRowHeight-methods), 107
setRowHeight,workbook,character-method
(setRowHeight-methods), 107
setRowHeight,workbook,numeric-method
(setRowHeight-methods), 107
setRowHeight-methods, 107
setSheetColor (setSheetColor-methods), 108
setSheetColor,workbook,character-method
(setSheetColor-methods), 108
setSheetColor,workbook,numeric-method
(setSheetColor-methods), 108
setSheetColor-methods, 108
setSheetPos, *51*, *52*
setSheetPos (setSheetPos-methods), 109
setSheetPos,workbook,character,missing-method
(setSheetPos-methods), 109
setSheetPos,workbook,character,numeric-method
(setSheetPos-methods), 109

setSheetPos-methods, 109
setStyleAction, *11*, *21*, *40, 41*, *90*, *94*, *97–99*, *101, 102*, *114, 115*, *123*, *125*, *127, 128*
setStyleAction (setStyleAction-methods), 110
setStyleAction,workbook-method (setStyleAction-methods), 110
setStyleAction-methods, 110
setStyleNamePrefix, *21*, *40*, *111, 112*
setStyleNamePrefix (setStyleNamePrefix-methods), 113
setStyleNamePrefix,workbook-method (setStyleNamePrefix-methods), 113
setStyleNamePrefix-methods, 113
setWrapText, *21*, *40*, *93*
setWrapText (setWrapText-methods), 114
setWrapText,cellstyle-method (setWrapText-methods), 114
setWrapText-methods, 114
show (show-methods), 115
show,workbook-method (show-methods), 115
show-methods, 115
strptime, *68*, *73*, *76*
summary (summary-methods), 116
summary,workbook-method (summary-methods), 116
summary-methods, 116
swissfranc, 117

unhideSheet, *55*, *58*, *60, 61*
unhideSheet (unhideSheet-methods), 118
unhideSheet,workbook,character-method (unhideSheet-methods), 118
unhideSheet,workbook,numeric-method (unhideSheet-methods), 118
unhideSheet-methods, 118
unmergeCells, *64*
unmergeCells (unmergeCells-methods), 119
unmergeCells,workbook,character-method (unmergeCells-methods), 119
unmergeCells,workbook,numeric-method (unmergeCells-methods), 119
unmergeCells-methods, 119

with.workbook, 120

workbook, *5*, *7, 8*, *11*, *13–18*, *21*, *23–33*, *35–55*, *58–67*, *70–72*, *74, 75*, *78*, *81–88*, *90–94*, *96–99*, *101–110*, *112*, *114–119*, *122–124*, *126, 127*, *137*
workbook-class, 121
writeNamedRegion, *7, 8*, *25*, *31*, *33*, *43*, *70*, *74*, *78*, *82*, *103*, *106*, *110*, *112*, *120*, *125*, *127*, *132, 133*
writeNamedRegion (writeNamedRegion-methods), 122
writeNamedRegion,workbook,ANY-method (writeNamedRegion-methods), 122
writeNamedRegion,workbook-method (writeNamedRegion-methods), 122
writeNamedRegion-methods, 122
writeNamedRegionToFile, *72*, *80*, *124*, 124, *128*, *132, 133*
writeWorksheet, *7, 8*, *33*, *70*, *74*, *78*, *103*, *106*, *110*, *112*, *124*, *128*, *131–133*
writeWorksheet (writeWorksheet-methods), 126
writeWorksheet,workbook,ANY,character-method (writeWorksheet-methods), 126
writeWorksheet,workbook,ANY,numeric-method (writeWorksheet-methods), 126
writeWorksheet-methods, 126
writeWorksheetToFile, *72*, *80*, *125*, *127*, 127, *132, 133*

XLC, *41*, *65*, *67*, *73*, *76*, *89, 90*, *94*, *98–102*, *104*, *108*, 129
xlcDump, 131, *133*, *136*
xlcEdit, *132*, 132, *136*
xlcFreeMemory, 134, *135*
xlcMemoryReport, *134*, 135
XLConnect (XLConnect-package), 4
XLConnect-deprecated, 135
XLConnect-package, 4
xlcRestore, *132, 133*, 136