Package 'excel.link'

January 17, 2024

Type Package

Title Convenient Data Exchange with Microsoft Excel

Version 0.9.12

Author Gregory Demin <excel.link.feedback@gmail.com>. To comply CRAN policy includes source code from 'RDCOMClient' (http://www.omegahat.net/RDCOMClient/) by Duncan Temple Lang <duncan@wald.ucdavis.edu>.

Maintainer Gregory Demin <excel.link.feedback@gmail.com>

Depends methods, grDevices, utils

Suggests knitr, rmarkdown

VignetteBuilder knitr

OS_type windows

Description Allows access to data in running instance of Microsoft Excel (e. g. 'xl[a1] = xl[b2]*3' and so on). Graphics can be transferred with 'xl[a1] = current.graphics()'. Additionally there are function for reading/writing 'Excel' files - 'xl.read.file'/'xl.save.file'. They are not fast but able to read/write '*.xlsb'-files and password-protected files. There is an Excel workbook with examples of calling R from Excel in the 'doc' folder. It tries to keep things as simple as possible - there are no needs in any additional installations besides R, only 'VBA' code in the Excel workbook. Microsoft Excel is required for this package.

License GPL (>= 2)

URL https://github.com/gdemin/excel.link

BugReports https://github.com/gdemin/excel.link/issues

LazyLoad yes

ByteCompile TRUE

NeedsCompilation yes

RoxygenNote 7.3.0

Repository CRAN

Date/Publication 2024-01-17 19:30:02 UTC

R topics documented:

`excel.link-package` *excel.link: convenient data exchange with Microsoft Excel*

Description

The excel.link package mainly consists of two rather independent parts: one is for transferring data/graphics to running instance of Excel, another part - work with data table in Excel in similar way as with usual data.frame.

Transferring data

Package provided family of objects: `xl`, `xlc`, `xlr` and `xlrc`. You don't need to initialize these objects or to do any other preliminary actions. Just after execution `library(excel.link)` you can transfer data to Excel active sheet by simple assignment, for example: `xlrc[a1] = iris`. In this notation 'iris' dataset will be written with column and row names. If you doesn't need column/row names just remove 'r'/'c' letters (`xlc[a1] = iris` - with column names but without row names). To read Excel data just type something like this: `xl[a1:b5]`. You will get data.frame with values from range a1:a5 without column and row names. It is possible to use named ranges (e. g. `xl[MyNamedRange]`). To transfer graphics use `xl[a1] = current.graphics()`. You can make active binding to Excel range:

```
 xl.workbook.add()
xl_iris %=crc% a1 # bind variable to current region around cell A1 on Excel active sheet
 xl_iris = iris # put iris data set
 identical(xl_iris$Sepal.Width, iris$Sepal.Width)
 xl_iris$test = "Hello, world!" # add new column on Excel sheet
 xl_iris = within(xl_iris, {
    new_col = Sepal.Width * Sepal.Length # add new column on Excel sheet
    })
```

Live connection

For example we put iris datasset to Excel sheet: `xlc[a1] = iris`. After that we connect Excel range with R object: `xl_iris = xl.connect.table("a1",row.names = FALSE, col.names = TRUE)`. So we can:

- get data from this Excel range: `xl_iris$Species`
- add new data to this Excel range: `xl_iris$new_column = 42`
- sort this range: `sort(xl_iris,column = "Sepal.Length")`
- and more...

Live connection is faster than active binding to range but is less universal (for example, you can't use `within` statement with it).

See Also

`xl`, `current.graphics`, `xl.connect.table`

`COMStop` *Functions from RDCOMClient package*

Description

For details about these functions see help for RDCOMClient package by Duncan Temple Lang `<duncan@wald.ucdavis.edu>`.

Usage

```
COMStop(msg, status, class = "COMError")

createCOMReference(ref, className)

DispatchMethods

.COMInit(status = TRUE)

COMCreate(name, ..., existing = TRUE)

getCOMInstance(guid, force = TRUE, silent = FALSE)

getCLSID(appName)

## S4 replacement method for signature 'COMIDispatch,ANY'
x$name <- value

## S4 method for signature 'COMIDispatch'
x$name
```

```
## S4 method for signature 'COMIDispatch,numeric'
x[[i, j, ...]]

## S4 method for signature 'COMIDispatch,ANY'
x[[i, j, ...]]

## S4 replacement method for signature 'COMIDispatch,character,missing'
x[[i, j, ...]] <- value

## S4 replacement method for signature 'COMIDispatch,numeric,ANY'
x[[i, j, ...]] <- value

.COM(
  obj,
  name,
  ...,
  .dispatch = as.integer(3),
  .return = TRUE,
  .ids = numeric(0),
  .suppliedArgs
)

asCOMArray(obj)

isValidCOMObject(obj)

COMList(obj, class = "COMList")

## S4 method for signature 'COMList'
length(x)

## S4 method for signature 'COMList,numeric'
x[[i, j, ...]]

## S4 replacement method for signature 'COMList,numeric,ANY'
x[[i, j, ...]] <- value

## S4 method for signature 'COMList'
length(x)

## S4 method for signature 'COMList'
lapply(X, FUN, ...)

## S4 method for signature 'COMList'
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

## S4 method for signature 'COMIDispatch'
```

```
lapply(X, FUN, ...)

## S4 method for signature 'COMIDispatch'
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

getItemClassName(x)

## S4 method for signature 'COMTypedList'
getItemClassName(x)

## S4 method for signature 'COMTypedList,numeric'
x[[i, j, ...]]

## S4 method for signature 'COMTypedParameterizedNamedList'
names(x)

## S4 method for signature 'COMTypedList,character'
x[[i, j, ...]]

## S4 method for signature 'COMTypedNamedList'
getItemClassName(x)

## S4 method for signature 'COMTypedNamedList'
names(x)

## S4 method for signature 'CompiledCOMCoClass'
getItemClassName(x)

## S4 method for signature 'CompiledCOMCoClass,character'
x[[i, j, ...]]

## S4 replacement method for signature 'CompiledCOMCoClass,character,ANY'
x[[i, j, ...]] <- value

## S4 method for signature 'CompiledCOMCoClass'
x$name

## S4 replacement method for signature 'CompiledCOMCoClass,character'
x$name <- value

COMNames(x)

## S4 method for signature 'CompiledCOMIDispatch'
names(x)

## S4 method for signature 'CompiledCOMIDispatch'
x$name
```

```
## S4 method for signature 'CompiledCOMIDispatch,character'
x[[i, j, ...]]

## S4 method for signature 'CompiledCOMIDispatch,numeric'
x[[i, j, ...]]

## S4 replacement method for signature 'CompiledCOMIDispatch,character'
x$name <- value

## S4 replacement method for signature 'CompiledCOMIDispatch,character,ANY'
x[[i, j, ...]] <- value

## S4 method for signature 'COMList,numeric,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'COMTypedNamedList,numeric,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'COMTypedNamedList,character,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'COMTypedNamedList,character'
x[[i, j, ..., exact = NA]]

## S4 method for signature 'EnumValue'
show(object)

EnumValue(id, value, obj = new("EnumValue"))

## S4 method for signature 'character,numeric,EnumValue'
EnumValue(id, value, obj = new("EnumValue"))

## S4 method for signature 'character,EnumValue,ANY'
EnumValue(id, value, obj = new("EnumValue"))

## S4 method for signature 'numeric,EnumValue,ANY'
EnumValue(id, value, obj = new("EnumValue"))

## S4 method for signature 'character,missing,EnumValue'
EnumValue(id, value, obj = new("EnumValue"))

## S4 method for signature 'numeric,missing,EnumValue'
EnumValue(id, value, obj = new("EnumValue"))

createTypeVarName(className, var, quote = TRUE)

## S4 method for signature 'COMIDispatch'
createTypeVarName(className, var, quote = TRUE)
```

```
## S4 method for signature 'CompiledCOMCoClass'
createTypeVarName(className, var, quote = TRUE)

## S4 method for signature 'character'
createTypeVarName(className, var, quote = TRUE)

getCOMElements(type, env = NA, namesOnly = FALSE)
```

Arguments

msg	See RDCOMClient documentation.
status	See RDCOMClient documentation.
class	See RDCOMClient documentation.
ref	See RDCOMClient documentation.
className	See RDCOMClient documentation.
name	See RDCOMClient documentation.
...	See RDCOMClient documentation.
existing	See RDCOMClient documentation.
guid	See RDCOMClient documentation.
force	See RDCOMClient documentation.
silent	See RDCOMClient documentation.
appName	See RDCOMClient documentation.
x	See RDCOMClient documentation.
value	See RDCOMClient documentation.
i	See RDCOMClient documentation.
j	See RDCOMClient documentation.
obj	See RDCOMClient documentation.
.dispatch	See RDCOMClient documentation.
.return	See RDCOMClient documentation.
.ids	See RDCOMClient documentation.
.suppliedArgs	See RDCOMClient documentation.
X	See RDCOMClient documentation.
FUN	See RDCOMClient documentation.
simplify	See RDCOMClient documentation.
USE.NAMES	See RDCOMClient documentation.
drop	See RDCOMClient documentation.
exact	See RDCOMClient documentation.
object	See RDCOMClient documentation.
id	See RDCOMClient documentation.

`var`	See RDCOMClient documentation.
`quote`	See RDCOMClient documentation.
`type`	See RDCOMClient documentation.
`env`	See RDCOMClient documentation.
`namesOnly`	See RDCOMClient documentation.

Format

An object of class `integer` of length 3.

Author(s)

Duncan Temple Lang `<duncan@wald.ucdavis.edu>`

`current.graphics` *Auxiliary function for export graphics to Microsoft Excel*

Description

Auxiliary function for export graphics to Microsoft Excel

Usage

```
current.graphics(
  type = c("png", "jpeg", "bmp", "tiff"),
  filename = NULL,
  picname = NULL,
  ...
)
```

Arguments

`type`	file type. Ignored if argument 'filename' provided.
`filename`	character. filename (or full path) of file with graphics.
`picname`	character. Picture name in Excel.
`...`	arguments for internally used `dev.copy` function

Details

If argument `type` provided this function will save graphics from windows plotting device to temporary file and return path to this file. Argument `filename` is intended to transfer plots to Excel from file-based graphics devices (see Examples) or just insert into Excel file with graphics. If argument `filename` is provided argument `type` will be ignored and returned value is path to file `filename` with class attribute 'current.graphics'. So it could be used with expressions such `xl[a1] = current.graphics(filename="plot.png")`. If `picname` is provided then picture will be inserted in Excel with this name. If picture `picname` already exists in Excel it will be deleted. This argument is useful when we need to change old picture in Excel instead of adding new picture. `picname` will be automatically prepended by "_" to avoid conflicts with Excel range names.

Value

Path to file with saved graphics with class attribute 'current.graphics'. If used with argument `type` than result has attribute `temp.file = TRUE`.

Examples

```
## Not run:
xl.workbook.add()
plot(sin)
xl[a1] = current.graphics()
plot(cos)
cos.plot = current.graphics()
xl.sheet.add()
xl[a1] = list("Cosine plotting",cos.plot,"End of cosine plotting")

# the same thing without graphic windows
png("1.png")
plot(sin)
dev.off()
sin.plot = current.graphics(filename = "1.png")
png("2.png")
plot(cos)
dev.off()
cos.plot = current.graphics(filename = "2.png")
output = list("Cosine plotting",cos.plot,"Sine plotting",sin.plot)
xl.workbook.add()
xl[a1] = output

## End(Not run)
```

`xl` *Data exchange with running Microsoft Excel instance.*

Description

`xl`, `xlc`, `xlr`, `xlrc` objects are already defined in the package. It doesn't need to create or init them. Just after attaching package one can write something like this: `xl[a1] = "Hello, world!"` and this text should appears in `A1` cell on active sheet of active Excel workbook. `xl(*)n` family of functions creates new worksheet for output. You can provide sheet name and position via `xl.sheet.name/before`.

Usage

```
## S3 method for class 'xl'
x[str.rng, drop = !(has.rownames(x) | has.colnames(x)), na = "", ...]

## S3 method for class 'xl'
```

```
x[[str.rng, drop = !(has.rownames(x) | has.colnames(x)), na = "", ...]]

## S3 method for class 'xl'
x$str.rng

## S3 replacement method for class 'xl'
x[[str.rng, na = "", ...]] <- value

## S3 replacement method for class 'xl'
x$str.rng <- value

## S3 replacement method for class 'xl'
x[str.rng, na = "", ...] <- value

## S3 replacement method for class 'xln'
x[[str.rng, na = "", xl.sheet.name = NULL, before = NULL, ...]] <- value

## S3 replacement method for class 'xln'
x$str.rng <- value

## S3 replacement method for class 'xln'
x[str.rng, na = "", xl.sheet.name = NULL, before = NULL, ...] <- value

xl.selection(drop = TRUE, na = "", row.names = FALSE, col.names = FALSE, ...)
```

Arguments

`x`	One of `xl`, `xlc`, `xlr`, `xlrc` objects. `xl` - read/write with/without column and row names, "r" - with rownames, "c" - with colnames
`str.rng`	character Excel range. For single bracket operations it can be without quotes in almost all cases.
`drop`	logical. If TRUE the result is coerced to the lowest possible dimension. By default dimensions will be dropped if there are no columns and rows names.
`na`	character. NA representation in Excel. By default it is empty string.
`...`	additional parameters. Not yet used.
`value`	suitable replacement value. It will be recycled to fill excel range only if it is object of length 1. In other cases size of excel range is ignored - all data will be placed in Excel sheet starting from top-left cell of submitted range.
`xl.sheet.name`	character. sheet name in active workbook (for `xl(*)n`).
`before`	character/numeric. sheet name or sheet number in active workbook before which new sheet will be added (for `xl(*)n`).
`row.names`	logical value indicating whether the Excel range contains the row names as its first column.
`col.names`	logical value indicating whether the Excel range contains the column names as its first row.

Details

xl object represents Microsoft Excel application. For convenient interactive usage arguments can be given without quotes in most cases (e. g. xl[a1] = 5 or xl[u2:u85] = "Hi" or xl[MyNamedRange] = 42, but xl["Sheet1!A1"] = 42). When it used in your own functions or you need to use variable as argument it is recommended apply double brackets notation: xl[["a1"]] = 5 or xl[["u2:u85"]] = "Hi" or xl[["MyNamedRange"]] = 42. Difference between xl, xlc, xlrc and xlr is xl ignore row and column names, xlc suppose read and write to Excel with column names, xlrc - with column and row names and so on. There is argument drop which is TRUE by default for xl and FALSE by default for other options. xl.selection returns data.frame with data from current selection in Excel. All these functions never coerce characters to factors

Value

Returns appropriate dataset from Excel.

See Also

cr, xl.current.region,

Examples

```
## Not run:
data(iris)
rownames(iris) = as.character(rownames(iris))
iris$Species = as.character(iris$Species)
xl.workbook.add()
xlrc$a1 = iris
xl.iris = xl.current.region("a1",row.names=TRUE,col.names=TRUE)
identical(xl.iris,iris)

xl.sheet.add("Datasets examples")
data.sets = list("Iris dataset",iris,"Cars dataset",cars,"Titanic dataset",as.data.frame(Titanic))
xlrc[a1] = data.sets


## End(Not run)
```

---

xl.bind.range *Active bindings to Excel ranges*

Description

xl.bind.range and xl.bind.current.region create sym in environment env so that getting the value of sym return bound Excel range, and assigning to sym will write the value to be assigned to Excel range. In case of xl.bind.range range will be updated after each assignment accordingly to the size of the assigned value. xl.bind.current.region always returns data from current region (Ctrl+Shift+* in Excel) of bound range. %=xl% etc are shortcuts for xl.bind.range and `xl.bind.current.region`. "r" means with row names, "c" means with column names. Range in most cases can be provided without quotes: `a1 %=xl% a1:b100`. Functions with '=' and with '<-' in the names do the same things - they are just for those who prefer '=' assignment and for those who prefer '<-' assignment. Assignment and reading may be slow because these functions always read/write entire dataset.

Usage

```
xl.bind.range(
  sym,
  str.range,
  drop = TRUE,
  na = "",
  row.names = FALSE,
  col.names = FALSE,
  env = parent.frame()
)

xl.bind.current.region(
  sym,
  str.range,
  drop = TRUE,
  na = "",
  row.names = FALSE,
  col.names = FALSE,
  env = parent.frame()
)

sym %=xl% value

sym %=xlr% value

sym %=xlc% value

sym %=xlrc% value

sym %=cr% value

sym %=crr% value

sym %=crc% value

sym %=crrc% value

sym %<xl-% value

sym %<xlr-% value

sym %<xlc-% value
sym %<xlrc-% value

sym %<cr-% value

sym %<crr-% value

sym %<crc-% value

sym %<crrc-% value

xl.binding.address(sym)
```

Arguments

`sym`	character/active binding.
`str.range`	character Excel range.
`drop`	logical. If TRUE the result is coerced to the lowest possible dimension. By default dimensions will be dropped if there are no columns and rows names.
`na`	character. NA representation in Excel. By default it is empty string.
`row.names`	logical value indicating whether the Excel range contains the row names as its first column.
`col.names`	logical value indicating whether the Excel range contains the column names as its first row.
`env`	an environment.
`value`	character Excel range address. It can be without quotes in many cases.

Value

`xl.binding.address` returns list with three components about bound Excel range: `address`, `rows` - number of rows, `columns` - number of columns. All other functions don't return anything but create active binding to Excel range in the environment.

Author(s)

Idea by Stefan Fritsch (`https://github.com/gdemin/excel.link/issues/1`)

See Also

`xl`, `xlr`, `xlc`, `xlrc`

Examples

```
## Not run:
 xl.workbook.add()
 range_a1 %=xl% a1 # binding range_a1 to cell A1 on active sheet
 range_a1 # should be NA
 range_a1 = 42 # value in Excel should be changed
 identical(range_a1, 42)
 cr_a1 %=cr% a1 # binding cr_a1 to current region around cell A1 on active sheet
 identical(cr_a1, range_a1)
 # difference between 'cr' and 'xl':
 xl[a2] = 43
 range_a1 # 42
 xl.binding.address(range_a1)
 xl.binding.address(cr_a1)
 cr_a1 # identical to 42:43
 # make cr and xl identical:
 range_a1 = 42:43
 identical(cr_a1, range_a1)

 xl_iris %=crc% a1 # bind current region A1 on active sheet with column names
 xl_iris = iris # put iris dataset to Excel sheet
 identical(xl_iris$Sepal.Width, iris$Sepal.Width) # should be TRUE

 xl_iris$new_col = xl_iris$Sepal.Width*xl_iris$Sepal.Length # add new column on Excel sheet


## End(Not run)
```

`xl.connect.table` *Live connection with data on Microsoft Excel sheet*

Description

`xl.connect.table` returns object that can be operated as usual data.frame object and this operations (e. g. subsetting, assignment) will be immediately reflected on connected Excel range. See examples. Connected range is 'current region', e. g. selection which can be obtained by pressing `Ctrl+Shift+*` when selected `str.rng` (or top-left cell of this range is active).

Usage

```
xl.connect.table(str.rng = "A1", row.names = TRUE, col.names = TRUE, na = "")

## S3 method for class 'excel.range'
sort(x, decreasing = FALSE, column, ...)
```

Arguments

`str.rng`	string which represents Excel range
`row.names`	a logical value indicating whether the Excel range contains row names as its first column
`col.names`	a logical value indicating whether the Excel range contains column names as its first row
`na`	character. NA representation in Excel. By default it is empty string
`x`	object of class `excel.range`

`decreasing`	logical. Should the sort be increasing or decreasing?
`column`	numeric or character. Column by which we will sort. There is special value - 'rownames'. In this case 'x' will be sorted by row names if it has it.
`...`	arguments to be passed to or from methods or (for the default methods and objects without a class)

Details

Subsetting. Indices in subsetting operations are numeric/character/logical vectors or empty (missing). Numeric values are coerced to integer as by 'as.integer' (and hence truncated towards zero). Character vectors will be matched to the 'colnames' of the object (or Excel column names if `has.colnames = FALSE`). For extraction form if column name doesn't exist error will be generated. For replacement form new column will be created. If indices are logical vectors they indicate elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. Indices can also be negative integers, indicating elements/slices to leave out of the selection.

Value

- `xl.connect.table` returns object of `excel.range` class which represent data on Excel sheet. This object can be treated similar to data.frame. So you can assign values, delete columns/rows and so on. For more information see examples.
- `sort` sorts Excel range by single column (multiple columns currently not supported) and invisibly return NULL.

Examples

```
## Not run:
### session example

library(excel.link)
xl.workbook.add()
xl.sheet.add("Iris dataset", before = 1)
xlrc[a1] = iris
xl.iris = xl.connect.table("a1", row.names = TRUE, col.names = TRUE)
dists = dist(xl.iris[, 1:4])
clusters = hclust(dists, method = "ward.D")
xl.iris$clusters = cutree(clusters, 3)
plot(clusters)
pl.clus = current.graphics()
cross = table(xl.iris$Species, xl.iris$clusters)
plot(cross)
pl.cross = current.graphics()
xl.sheet.add("Results", before = 2)
xlrc$a1 = list("Crosstabulation", cross,pl.cross, "Dendrogram", pl.clus)

### completely senseless actions
### to demonstrate various operations and
### compare them with operations on usual data.frame
```

```
# preliminary operations
data(iris)
rownames(iris) = as.character(rownames(iris))
iris$Species = as.character(iris$Species)
xl.workbook.add()

# drop dataset to Excel and connect it
xlrc[a1] = iris
xl.iris = xl.connect.table("a1", row.names = TRUE, col.names = TRUE)
identical(xl.iris[], iris)

# dim/colnames/rownames
identical(dim(xl.iris),dim(iris))
identical(colnames(xl.iris),colnames(iris))
identical(rownames(xl.iris),rownames(iris))

# sort datasets
iris = iris[order(iris$Sepal.Length), ]
sort(xl.iris, column = "Sepal.Length")
identical(xl.iris[], iris)

# sort datasets by rownames
sort(xl.iris, column = "rownames")
iris = iris[order(rownames(iris)), ]
identical(xl.iris[], iris)

# different kinds of subsetting
identical(xl.iris[,1:3], iris[,1:3])
identical(xl.iris[,3], iris[,3])
identical(xl.iris[26,1:3], iris[26,1:3])
identical(xl.iris[-26,1:3], iris[-26,1:3])
identical(xl.iris[50,], iris[50,])
identical(xl.iris$Species, iris$Species)
identical(xl.iris[,'Species', drop = FALSE], iris[,'Species', drop = FALSE])
identical(xl.iris[c(TRUE,FALSE), 'Sepal.Length'],
             iris[c(TRUE,FALSE), 'Sepal.Length'])

# column creation and assignment
xl.iris[,'group'] = xl.iris$Sepal.Length > mean(xl.iris$Sepal.Length)
iris[,'group'] = iris$Sepal.Length > mean(iris$Sepal.Length)
identical(xl.iris[], iris)

# value recycling
xl.iris$temp = c('aa','bb')
iris$temp = c('aa','bb')
identical(xl.iris[], iris)

# delete column
xl.iris[,"temp"] = NULL
iris[,"temp"] = NULL
identical(xl.iris[], iris)
```

```
## End(Not run)
```

`xl.current.region` *Read/write from/to Excel current region.*

Description

Current region is a region that will be selected by pressing `Ctrl+Shift+*` in Excel. The current region is a range bounded by any combination of blank rows and blank columns. `cr`, `crc`, `crr`, `crrc` objects are already defined in the package. It doesn't need to create or init them.

Usage

```
xl.current.region(
  str.rng,
  drop = TRUE,
  na = "",
  row.names = FALSE,
  col.names = FALSE,
  ...
)

## S3 method for class 'cr'
x[[str.rng, drop = !(has.rownames(x) | has.colnames(x)), na = "", ...]]

## S3 replacement method for class 'cr'
x[[str.rng, na = "", ...]] <- value
```

Arguments

`str.rng`	character Excel range. For single bracket operations it can be without quotes in almost all cases.
`drop`	logical. If TRUE the result is coerced to the lowest possible dimension. By default dimensions will be dropped if there are no columns and rows names.
`na`	character. NA representation in Excel. By default it is empty string.
`row.names`	logical value indicating whether the Excel range contains the row names as its first column.
`col.names`	logical value indicating whether the Excel range contains the column names as its first row.
`...`	additional parameters. Not yet used.
`x`	One of `cr`, `crc`, `crr`, `crrc` objects. `cr` - read/write with/without column and row names, "r" - with rownames, "c" - with colnames
`value`	suitable replacement value. All data will be placed in Excel sheet starting from top-left cell of current region. Current region will be cleared before writing.

Details

cr object represents Microsoft Excel application. For convenient interactive usage arguments can be given without quotes in most cases (e. g. cr[a1] = 5 or cr[u2:u85] = "Hi" or cr[MyNamedRange] = 42, but cr["Sheet1!A1"] = 42). When it used in your own functions or you need to use variable as argument it is recommended apply double brackets notation: cr[["a1"]] = 5 or cr[["u2:u85"]] = "Hi" or cr[["MyNamedRange"]] = 42. Difference between cr, crc, crrc and crr is cr ignore row and column names, crc suppose read and write to Excel with column names, crrc - with column and row names and so on. There is argument drop which is TRUE by default for cr and FALSE by default for other options. All these functions never coerce characters to factors

Value

Returns appropriate dataset from Excel.

See Also

xl

Examples

```
## Not run:
data(iris)
data(mtcars)
xl.workbook.add()
xlc$a1 = iris
identical(crc[a1],xlc[a1:e151]) # should be TRUE
identical(crc$a1,xlc[a1:e151]) # should be TRUE
identical(crc$a1,xlc[a1]) # should be FALSE

# current region will be cleared before writing - no parts of iris dataset
crrc$a1 = mtcars
identical(crrc$a1,xlrc[a1:l33]) # should be TRUE


## End(Not run)
```

---

xl.get.excel *Returns reference to Excel application.*

Description

Returns reference to Microsoft Excel application. If there is no running instance exists it will create a new instance.

Usage

```
xl.get.excel()
```

Value

object of class 'COMIDispatch' (as returned by `COMCreate` from RDCOMClient package).

Examples

```
## Not run:
xls = xl.get.excel()

## End(Not run)
```

`xl.index2address` *Converts Excel address to indexes and vice versa.*

Description

Converts Excel address to indexes and vice versa.

Usage

```
xl.index2address(top, left, bottom = NULL, right = NULL)

xl.address2index(str.range)
```

Arguments

`top`	integer top index of top-left cell
`left`	integer left index of top-left cell
`bottom`	integer bottom index of bottom-right cell
`right`	integer right index of bottom-right cell
`str.range`	character Excel range address

Value

xl.index2address returns character address (e. g. A1:B150), xl.address2index returns vector with four components: top, left, bottom, right.

Examples

```
xl.address2index("A1:D150")
xl.index2address(top=1, left=1)

## Not run:
a1 %=xl% a1
a1 = iris
addr = xl.binding.address(a1)$address
```

```
xl.address2index(addr)


## End(Not run)
```

---

xl.property *Excel constants and helper function for setting Excel range properties.*

Description

`xl.constants` is a list with (surprise!) Excel named constants.

Usage

```
xl.property(...)

xl.constants
```

Arguments

`...`	names of arguments are properties as in Excel VBA, values are properties values.

Format

An object of class `list` of length 2024.

Value

list of class `xl.property`.

Examples

```
## Not run:
# create random matrix
rand_mat = matrix(runif(16), ncol = 4)

# put it on the new worksheet
xln[a1] = rand_mat

# set bold font, format numbers as percent and align it
cr[a1] = xl.property(Font.Bold = TRUE,
                     NumberFormat = "0.00%",
                     HorizontalAlignment = xl.constants$xlCenter
                     )

## End(Not run)
```

xl.read.file — *Functions for saving and reading data to/from Excel file.*

Description

Functions for saving and reading data to/from Excel file.

Usage

```
xl.read.file(
  filename,
  header = TRUE,
  row.names = NULL,
  col.names = NULL,
  xl.sheet = NULL,
  top.left.cell = "A1",
  na = "",
  password = NULL,
  write.res.password = NULL,
  excel.visible = FALSE
)

xl.save.file(
  r.obj,
  filename,
  row.names = TRUE,
  col.names = TRUE,
  xl.sheet = NULL,
  top.left.cell = "A1",
  na = "",
  password = NULL,
  write.res.password = NULL,
  excel.visible = FALSE,
  file.format = xl.constants$xlOpenXMLWorkbook
)
```

Arguments

`filename`	a character
`header`	a logical value indicating whether the file contains the names of the variables as its first line. If TRUE and top-left corner is empty cell, first column is considered as row names. Ignored if row.names or col.names is not NULL.
`row.names`	a logical value indicating whether the row names of r.obj are to be read/saved along with r.obj
`col.names`	a logical value indicating whether the column names of r.obj are to be read/saved along with r.obj

`xl.sheet`	character. Name of Excel sheet where data is located/will be saved. By default it is NULL and data will be read/saved from/to active sheet.
`top.left.cell`	character. Top-left corner of data in Excel sheet. By default is 'A1'.
`na`	character. NA representation in Excel. By default it is empty string.
`password`	character. Password for password-protected workbook.
`write.res.password`	character. Second password for editing workbook.
`excel.visible`	a logical value indicating will Excel visible during this operations. FALSE by default.
`r.obj`	R object.
`file.format`	integer. Excel file format. By default it is `xl.constants$xlOpenXMLWorkbook`. You can use `xl.constants$xlOpenXMLWorkbookMacroEnabled` for workbooks with macros (*.xlsm) or `xl.constants$xlExcel12` for binary workbook (.xlsb).

Details

`xl.read.file` reads only rectangular data set. It is highly recommended to have all column names and ids in data set. Orphaned rows/columns located apart from the main data will be ignored. `xl.save.file` can save all objects for which `xl.write` method exists - see examples.

Value

`xl.read.file` always returns data.frame. `xl.save.file` invisibly returns NULL.

See Also

`xl.write`, `xl.workbook.save`, `xl.workbook.open`, `current.graphics`

Examples

```
## Not run:
data(iris)
xl.save.file(iris,"iris.xlsx")
xl.iris = xl.read.file("iris.xlsx")
all(iris == xl.iris) # Shoud be TRUE
unlink("iris.xlsx")

# Save to file list with different data types
dists = dist(iris[,1:4])
clusters = hclust(dists,method="ward.D")
iris$clusters = cutree(clusters,3)
png("1.png")
plot(clusters)
dev.off()
pl.clus = current.graphics(filename="1.png")
cross = table(iris$Species,iris$clusters)
png("2.png")
```

```
plot(cross)
dev.off()
pl.cross = current.graphics(filename="2.png")
output = list("Iris",pl.clus,cross,pl.cross,"Data:","",iris)
xl.save.file(output,"output.xls")
xl.workbook.open("output.xls")
# xl.workbook.close() # close workbook
# unlink("output.xls") # delete file

# password-protected file
data(iris)
xl.save.file(iris,"iris.xlsx", password = "pass")
xl.iris = xl.read.file("iris.xlsx", password = "pass")
all(iris == xl.iris) # Shoud be TRUE
unlink("iris.xlsx")


## End(Not run)
```

`xl.sheet.add` *Basic operations with worksheets.*

Description

Basic operations with worksheets.

Usage

```
xl.sheet.add(xl.sheet.name = NULL, before = NULL)

xl.sheet.duplicate(before = NULL)

xl.sheet.name(xl.sheet.name = NULL)

xl.sheet.visible(xl.sheet)

xl.sheet.hide(xl.sheet = NULL)

xl.sheet.show(xl.sheet)

xl.sheets()

xl.sheet.activate(xl.sheet)

xl.sheet.delete(xl.sheet = NULL)
```

Arguments

`xl.sheet.name`	character. sheet name/new sheet name
`before`	character/numeric. sheet name or sheet number in active workbook before which new sheet will be added
`xl.sheet`	character/numeric. sheet name or sheet number in active workbook

Details

- `xl.sheet.add` adds new sheet with given name and invisibly returns name of this newly added sheet. Added sheet become active. If `xl.sheet.name` is missing default name will be used. If `before` argument is missing, sheet will be added at the last position. If sheet with given name already exists error will be generated.
- `xl.sheet.name` rename active sheet. If its argument is missing then it just return active sheet name.
- `xl.sheet.hide`/`xl.sheet.show` hide and show sheet by its name. `xl.sheet.visible` returns current visibility status of the sheet.
- `xl.sheet.activate` activates sheet with given name/number. If sheet with this name doesn't exist error will be generated.
- `xl.sheet.delete` deletes sheet with given name/number. If name doesn't submitted it delete active sheet.

Value

- `xl.sheet.add`/`xl.sheet.activate`/`xl.sheet.duplicate` invisibly return name of created/activated/duplicated sheet.
- `xl.sheets` returns vector of sheet names in active workbook.
- `xl.sheet.delete` invisibly returns NULL.

See Also

`xl.workbooks`

Examples

```
## Not run:
xl.workbook.add()
sheets = xl.sheets()
xl.sheet.add("Second")
xl.sheet.add("First", before="Second")
for (sheet in sheets) xl.sheet.delete(sheet) # only 'First' and 'Second' exist in workbook now
xl.sheet.activate("Second") #last sheet activated
xl.sheet.duplicate() # duplicate second sheet
xl.sheet.name() # "Second (2)"
xl.sheet.name("Third") # "Third"


## End(Not run)
```

`xl.workbook.add` *Basic operations with Excel workbooks*

Description

Basic operations with Excel workbooks

Usage

```
xl.workbook.add(filename = NULL)

xl.workbook.open(filename, password = NULL, write.res.password = NULL)

xl.workbook.activate(xl.workbook.name)

xl.workbooks(full.names = FALSE)

xl.workbook.save(
  filename,
  password = NULL,
  write.res.password = NULL,
  file.format = xl.constants$xlOpenXMLWorkbook
)

xl.workbook.close(xl.workbook.name = NULL)
```

Arguments

`filename`	character. Excel workbook filename.
`password`	character. Password for password-protected workbook.
`write.res.password`	character. Second password for editing workbook.
`xl.workbook.name`	character. Excel workbook name.
`full.names`	logical. Should we return full path to the workbook? FALSE, by default.
`file.format`	integer. Excel file format. By default it is `xl.constants$xlOpenXMLWorkbook`. You can use `xl.constants$xlOpenXMLWorkbookMacroEnabled` for workbooks with macros (*.xlsm) or `xl.constants$xlExcel12` for binary workbook (.xlsb).

Details

- `xl.workbook.add` adds new workbook and invisibly returns name of this newly created workbook. Added workbook become active. If `filename` argument is provided then Excel workbook `filename` will be used as template.
- `xl.workbook.activate` activates workbook with given name. If workbook with this name doesn't exist error will be generated.

- `xl.workbook.save` saves active workbook. If only `filename` submitted it saves in the working directory. If name of workbook is omitted than new workbook is saved under its default name in the current working directory. It doesn't prompt about overwriting if file already exists.
- `xl.workbook.close` closes workbook with given name. If name isn't submitted it closed active workbook. It doesn't prompt about saving so if you don't save changes before closing all changes will be lost.

Value

- `xl.workbook.add`/`xl.workbook.open`/`xl.workbook.activate` invisibly return name of created/open/activated workbook.
- `xl.workbooks` returns character vector of open workbooks.
- `xl.workbook.save` invisibly returns path to the saved workbook.
- `xl.workbook.close` invisibly returns NULL.

See Also

`xl.sheets`, `xl.read.file`, `xl.save.file`

Examples

```
## Not run:
## senseless actions
data(iris)
data(cars)
xl.workbook.add()
xlrc[a1] = iris
xl.workbook.save("iris.xlsx")
xl.workbook.add()
xlrc[a1] = cars
xl.workbook.save("cars.xlsx")
xl.workbook.activate("iris")
xl.workbook.close("cars")
xl.workbook.open("cars.xlsx")
xl.workbooks()
for (wb in xl.workbooks()) xl.workbook.close(wb)
unlink("iris.xlsx")
unlink("cars.xlsx")

# password-protected workbook
data(iris)
xl.workbook.add()
xlrc[a1] = iris
xl.workbook.save("iris.xlsx", password = "my_password")
xl.workbook.close()
xl.workbook.open("iris.xlsx", password = "my_password")
xl.workbook.close()
unlink("iris.xlsx")

## End(Not run)
```

xl.write *Methods for writing data to Excel sheet*

Description

Methods for writing data to Excel sheet

Usage

```
xl.write(r.obj, xl.rng, na = "", ...)

## S3 method for class 'current.graphics'
xl.write(r.obj, xl.rng, na = "", delete.file = FALSE, ...)

## S3 method for class 'list'
xl.write(r.obj, xl.rng, na = "", ...)

## S3 method for class 'matrix'
xl.write(r.obj, xl.rng, na = "", row.names = TRUE, col.names = TRUE, ...)

## S3 method for class 'data.frame'
xl.write(r.obj, xl.rng, na = "", row.names = TRUE, col.names = TRUE, ...)

## Default S3 method:
xl.write(r.obj, xl.rng, na = "", row.names = TRUE, ...)
```

Arguments

`r.obj`	R object
`xl.rng`	An object of class `COMIDispatch` (as used in RDCOMClient package) - reference to Excel range
`na`	character. NA representation in Excel. By default it is empty string
`...`	arguments for further processing
`delete.file`	a logical value indicating whether delete file with graphics after insertion in Excel
`row.names`	a logical value indicating whether the row names/vector names of r.obj should to be written along with r.obj
`col.names`	a logical value indicating whether the column names of r.obj should to be written along with r.obj

Details

`xl.rng` should be COM-reference to Excel range, not string. Method invisibly returns number of columns and rows occupied by `r.obj` on Excel sheet. It's useful for multiple objects writing to prevent their overlapping. It is more convenient to use `xl` object. `xl.write` aimed mostly for programming purposes, not for interactive usage.

Value

c(rows,columns) Invisibly returns rows and columns number ocuppied by `r.obj` on Excel sheet.

See Also

`xl`, `xlr`, `xlc`, `xlrc`, `current.graphics`

Examples

```
## Not run:
xls = xl.get.excel()
xl.workbook.add()
rng = xls[["Activesheet"]]$Cells(1,1)
nxt = xl.write(iris,rng,row.names = TRUE,col.names = TRUE)
rng = rng$Offset(nxt[1] + 1,0)
nxt = xl.write(cars,rng,row.names = TRUE,col.names = TRUE)
rng = rng$Offset(nxt[1] + 1,0)
nxt = xl.write(as.data.frame(Titanic),rng,row.names = TRUE,col.names = TRUE)

data(iris)
data(cars)
data(Titanic)
xl.sheet.add()
rng = xls[["Activesheet"]]$Cells(1,1)
data.sets = list("Iris dataset",iris,
     "Cars dataset",cars,
     "Titanic dataset",as.data.frame(Titanic))
xl.write(data.sets,rng,row.names = TRUE,col.names = TRUE)


## End(Not run)
```

Index