Package 'rsyncrosim'

May 6, 2025

Type Package

Title The R Interface to 'SyncroSim'

Version 2.1.2

Description 'SyncroSim' is a generalized framework for managing scenario-based datasets (<https://syncrosim.com/>). 'rsyncrosim' provides an interface to 'SyncroSim'. Simulation models can be added to 'SyncroSim' in order to transform these datasets, taking advantage of general features such as defining scenarios of model inputs, running Monte Carlo simulations, and summarizing model outputs. 'rsyncrosim' requires 'SyncroSim' 2.3.5 or higher (API documentation: <https://docs.syncrosim.com/>).

License MIT + file LICENSE

Encoding UTF-8

Imports methods, DBI, RSQLite, terra, gtools, processx, stats, utils

Suggests knitr, testthat (>= 3.0.0), ggplot2, Rcpp, rmarkdown

SystemRequirements SyncroSim (>=3.1.0)

Collate 'AAAClassDefinitions.R' 'addPackage.R' 'addRow.R' 'backup.R' 'chart.R' 'chartCriteria.R' 'chartData.R' 'chartDisagg.R' 'chartErrorBar.R' 'chartId.R' 'chartInclude.R' 'chartOptionsFont.R' 'chartOptionsFormat.R' 'chartOptionsLegend.R' 'chartOptionsXAxis.R' 'chartOptionsYAxis.R' 'command.R' 'condaFilepath.R' 'createCondaEnv.R' 'datasheet.R' 'datasheetSpatRaster.R' 'dateModified.R' 'delete.R' 'deleteLibrary.R' 'dependency.R' 'description.R' 'filepath.R' 'folder.R' 'folderId.R' 'ignoreDependencies.R' 'info.R' 'installConda.R' 'installPackage.R' 'internalHelpers.R' 'viewProfile.R' 'packages.R' 'name.R' 'scenarioId.R' 'projectId.R' 'sqlStatement.R' 'scenario.R' 'project.R' 'ssimLibrary.R' 'session.R' 'internalWrappers.R' 'mergeDependencies.R' 'owner.R' 'parentId.R' 'print.R' 'printCmd.R' 'readOnly.R' 'removePackage.R' 'rsyncrosim.R' 'run.R' 'runLog.R' 'saveDatasheet.R' 'signin.R' 'signout.R' 'silent.R' 'ssimEnvironment.R' 'uninstallPackage.R' 'useConda.R' 'version.R'

RoxygenNote 7.3.2

URL https://syncrosim.github.io/rsyncrosim/

BugReports https://github.com/syncrosim/rsyncrosim/issues/

Config/testthat/edition 3

NeedsCompilation no

Author Colin Daniel [aut],
Josie Hughes [aut],
Valentin Lucet [aut],
Alex Embrey [aut],
Katie Birchard [aut, cre],
Leonardo Frid [aut],
Tabitha Kennedy [aut],
Shreeram Senthivasan [aut],
ApexRMS [cph]

Maintainer Katie Birchard <katie.birchard@apexrms.com>

Repository CRAN

Date/Publication 2025-05-06 20:50:02 UTC

Contents

addPackage *Add SyncroSim package(s)*

Description

Adds package(s) to a SsimLibrary.

Usage

```
addPackage(ssimLibrary, packages, versions = NULL, forceUpdate = FALSE)

## S4 method for signature 'character'
addPackage(ssimLibrary, packages, versions = NULL, forceUpdate = FALSE)

## S4 method for signature 'SsimLibrary'
addPackage(ssimLibrary, packages, versions = NULL, forceUpdate = FALSE)
```

Arguments

ssimLibrary	SsimLibrary object
packages	character string or vector of package name(s)
versions	character string or vector of package version(s). If NULL then uses the latest installed version of the package
forceUpdate	logical. If FALSE (default) user will be prompted to approve any required updates. If TRUE, required updates will be applied silently.

Value

Invisibly returns TRUE upon success (i.e.successful addition of the package) or FALSE upon failure.

See Also

packages

Examples

```
## Not run:
# Install "stsim" and "stsimecodep" SyncroSim packages
installPackage(packages = c("stsim", "stsim"),
               versions = c("4.0.1", "4.3.5"))
installPackage("stsimecodep")

# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
```

```
# Add package
addPackage(myLibrary, packages = "stsim", versions = "4.0.1")
addPackage(myLibrary, packages = "stsimecodep")
packages(myLibrary)

# Change package version
addPackage(myLibrary, packages = "stsim", versions = "4.3.5")

# Remove package
removePackage(myLibrary, packages = c("stsim", "stsimecodep"))
packages(myLibrary)

## End(Not run)
```

---

addRow *Add row(s) to a data.frame*

Description

This function is mostly used internally to add rows to data.frames associated with SyncroSim Datasheets retrieved from the command line.

Usage

```
addRow(targetDataframe, value)

## S4 method for signature 'data.frame'
addRow(targetDataframe, value)
```

Arguments

targetDataframe	data.frame
value	data.frame, character string, vector, or list. Columns or elements in value should be a subset of columns in targetDataframe

Details

Preserves the types and factor levels of the targetDataframe. Fills missing values if possible using factor levels. If value is a named vector or list, it will be converted to a single row data.frame. If value is an unnamed vector or list, the number of elements should equal the number of columns in the targetDataframe; elements are assumed to be in same order as data.frame columns.

Value

A dataframe with new rows.

Examples

```
# Create an example data.frame
oldDataframe <- as.data.frame(mtcars)

# Add a single row to the example data.frame
newDataframe <- addRow(oldDataframe, list(mpg = 100, wt = 10))

# Create an example data.frame with more than one row of data
multipleRows <- data.frame(mpg = c(40, 50, 75), wt = c(4, 7, 6))

# Add the old example data.frame to the new example data.frame
newDataframe <- addRow(oldDataframe, multipleRows)
```

---

backup *Backup a SsimLibrary*

Description

Backup a SsimLibrary. The backup folder can be defined in the SyncroSim User Interface, but is by default at the same level as the SsimLibrary file, and is called libraryName.backup.

Usage

```
backup(ssimObject)

## S4 method for signature 'character'
backup(ssimObject)

## S4 method for signature 'SsimObject'
backup(ssimObject)
```

Arguments

ssimObject	SsimLibrary, Project or Scenario object

Value

Invisibly returns TRUE upon success (i.e.successful backup) and FALSE upon failure.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
```

```
# Back up data from the SsimLibrary
backup(myLibrary)

## End(Not run)
```

---

chart *Create or open a chart*

Description

Create or open a Chart from a SyncroSim Project.

Usage

```
chart(ssimObject = NULL, chart = NULL, create = FALSE, summary = FALSE)

## S4 method for signature 'character'
chart(ssimObject = NULL, chart = NULL, create = FALSE, summary = FALSE)

## S4 method for signature 'SsimObject'
chart(ssimObject = NULL, chart = NULL, create = FALSE, summary = FALSE)
```

Arguments

ssimObject	Project or Scenario object
chart	character or integer. If character, then will either open an existing chart if create=FALSE, or will create a new chart with the given name if the chart does not exist yet or create=TRUE. If integer, will open the existing chart with the given chart ID (if the ID exists). If no value is provided and create=TRUE, a new chart will be created with the default naming convention (e.g. "_Chart1", "_Chart2")
create	logical. Whether to create a new chart if the chart name given already exists in the SyncroSim library. If FALSE (Default), then will return the existing chart with the given name. If TRUE, then will return a new chart with the same name as an existing chart (but different chart ID)
summary	logical. If TRUE, returns a summary of chart information as an R data.frame. If FALSE (Default), then returns a SyncroSim Chart object

Value

A Chart object representing a SyncroSim chart

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession,
                         packages = "stsim")
myProject <- project(myLibrary, project = "My Project")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Create a new chart
myChart <- chart(myProject, chart = "New Chart")

## End(Not run)
```

`Chart-class` *SyncroSim Chart class*

Description

`Chart` object representing a SyncroSim Chart object. A Chart object is used to create line or column charts from tabular output data in the and can be viewed using the SyncroSim User Interface.

Slots

`session` `Session` object. The Session associated with the Chart's SsimLibrary

`filepath` character string. The path to the Chart's SsimLibrary on disk

`chartId` integer. The Chart id

`projectId` integer. The Project id

See Also

See `chart` for options when creating or loading a SyncroSim Chart

chartCriteria *Retrieves chart variables*

Description

Retrieves the available variables for charting, or the variables that are set for an existing chart.

Usage

```
chartCriteria(ssimObject, chart = NULL, variable = NULL, filter = NULL)

## S4 method for signature 'SsimObject'
chartCriteria(ssimObject, chart = NULL, variable = NULL, filter = NULL)
```

Arguments

ssimObject	Project or Chart object
chart	character or integer. Either the name or ID of an existing chart. If NULL and a Project is provided as the first argument, then will return the available variables for charting.
variable	character. The name of a charting variable. If provided, then will return a list of the available filter columns for that variable. Default is NULL.
filter	character. The name of a filter column for a specified variable. If provided, then will return a list of values that pertain to the specified filter. If the filter column is used to disaggregate the chart data (using the chartDisagg function), one panel will be created for each of these values. If you would like to omit values from the chart, you can also add or remove values by the specified filter column using the chartInclude function. Default is NULL.

Details

Example arguments:

- If ssimObject is SyncroSim Project and chart is NULL: Returns a data.frame of available variables for creating a new chart.
- If ssimObject is SyncroSIm Chart or chart is not NULL: Returns a data.frame of variables in use by the specified chart.
- If variable is not NULL: Returns a list of filter columns that belong to the given variable.
- If variable and filter are not NULL: Returns a dataframe of value IDs and names that belong to the given variable and filter.

Value

A data.frame or list of variables, filter columns, and filter values.

Examples

```
## Not run:
# Create a chart object
myChart <- chart(myProject, chart = "New Chart")

# Retrieve variables that can be used to create new charts
chartCriteria(myProject)

# Retrieve variables being used by existing chart
chartCriteria(myChart)

## End(Not run)
```

---

chartData *Sets the* Chart *type and axes*

Description

Sets the Chart type and adds the variables to plot in the line chart.

Usage

```
chartData(
  chart,
  type = "Line",
  addX = NULL,
  addY = NULL,
  removeX = NULL,
  removeY = NULL,
  timesteps = NULL,
  iterationType = "Mean",
  iteration = 1
)

## S4 method for signature 'Chart'
chartData(
  chart,
  type = "Line",
  addX = NULL,
  addY = NULL,
  removeX = NULL,
  removeY = NULL,
  timesteps = NULL,
  iterationType = "Mean",
  iteration = 1
)
```

Arguments

chart	Chart object
type	character. Chart type. Can be "Line" (Default) or "Column".
addX	character or character vector. X variable(s) to add to the chart. If NULL (Default), does not add any X variables. If no X variables specified in chart, then will default to plotting timesteps on the X axis.
addY	character or character vector. Y variable(s) to add to the chart. If NULL (Default), does not add any Y variables.
removeX	character or character vector. X variable(s) to remove from plot. If NULL (Default), then does not remove any X variables.
removeY	character or character vector. Y variable(s) to remove from plot. If NULL (Default), then does not remove any Y variables.
timesteps	integer vector. The range of timesteps to plot against If NULL, then uses SyncroSim defaults.
iterationType	character. How to display multiple iterations in the chart. Can be "Mean" (Default), "Single", or "All".
iteration	integer. If the iterationType is set to "Single", this argument determines which iteration to display. Default is 1.

Value

A Chart object representing a SyncroSim chart

Examples

```
## Not run:
# Create a chart object
myChart <- chart(myProject, chart = "New Chart")

# Set the chart type and data
myChart <- chartData(myChart, y = c("variable1", "variable2"),
timesteps = c(0,10), iterationType = "single", iteration = 1)


## End(Not run)
```

---

chartDisagg *Disaggregates the Chart by a Y variable*

Description

Disaggregates the Chart by given filter column(s) in a Y variable.

Usage

```
chartDisagg(chart, variable, addFilter = NULL, removeFilter = NULL)

## S4 method for signature 'Chart'
chartDisagg(chart, variable, addFilter = NULL, removeFilter = NULL)
```

Arguments

chart	Chart object
variable	character. The variable to disaggregate the Y axis by.
addFilter	character or character vector. Adds Y variable column(s) to disaggregate the chart by.
removeFilter	character or character vector. Removes Y variable column(s) from disaggregating the chart.

Value

A Chart object representing a SyncroSim chart

Examples

```
## Not run:
# Create a chart object
myChart <- chart(myProject, chart = "New Chart")

# Set the chart type and data
myChart <- chartData(myChart, y = c("variable1", "variable2"),
timesteps = c(0,10), iterationType = "single", iteration = 1)

# Disaggregate the chart by a filter column
myChart <- chartDisagg(myChart, variable = "variable1",
addFilter=c("col1", "col2"))

# Remove a filter column from the chart disaggregation
myChart <- chartDisagg(myChart, variable = "variable1",
removeFilter="col1")

## End(Not run)
```

---

chartErrorBar *Modify the error bars of a Chart*

Description

Set the type and properties of the error bars of a Chart.

Usage

```
chartErrorBar(chart, type = NULL, lower = NULL, upper = NULL)

## S4 method for signature 'Chart'
chartErrorBar(chart, type = NULL, lower = NULL, upper = NULL)
```

Arguments

chart	Chart object
type	character. Type of error bar. Values can be "percentile", "minmax", or "none". Default is NULL.
lower	float. If the error bar type is set to "percentile", then sets the minimum percentile for the lower range of the error bar. Default is NULL.
upper	float. If the error bar type is set to "percentile", then sets the maximum percentile for the upper range of the error bar. Default is NULL.

Value

A Chart object representing a SyncroSim chart or, if no arguments other than the chart are provided, a data.frame of the current chart error bar settings.

Examples

```
## Not run:
# Open a chart object
myChart <- chart(myProject, chart = "My Chart")

# Set the chart error bars to display the minimum/maximum of the data
myChart <- chartErrorBar(myChart, type = "minmax")

# Disable the chart error bars
myChart <- chartErrorBar(myChart, type = "none")

# Set the chart error bars to display the 95th percentile error bars
myChart <- chartErrorBar(myChart, type = "percentile", lower = 2.5,
                         upper = 97.5)

## End(Not run)
```

---

chartId *Retrieves chartId of SyncroSim Chart*

Description

Retrieves the Chart Id of a SyncroSim Chart.

Usage

```
chartId(ssimObject)

## S4 method for signature 'character'
chartId(ssimObject)

## S4 method for signature 'Chart'
chartId(ssimObject)
```

Arguments

ssimObject	Chart object

Value

An integer: chart id.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set the SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession,
                         packages = "stsim",
                         overwrite = TRUE)
myProject <- project(myLibrary, project = "Definitions")

# Get the chart object corresponding to the chart called "My Chart"
myChart <- chart(myProject, chart = "My Chart")

# Get Chart ID for SyncroSim Chart
chartId(myChart)

## End(Not run)
```

---

chartInclude *Add or remove values by column in a Chart*

Description

Add or remove values by a specified column in the X or Y axis of a Chart.

Usage

```
chartInclude(
  chart,
  variable,
  filter,
  axis = "Y",
  addValue = NULL,
  removeValue = NULL
)

## S4 method for signature 'Chart'
chartInclude(
  chart,
  variable,
  filter,
  axis = "Y",
  addValue = NULL,
  removeValue = NULL
)
```

Arguments

chart	Chart object
variable	character. A variable belonging to the X or Y axis.
filter	character or character vector. A filter column belonging to the X or Y variable.
axis	character. Either "X" or "Y" corresponding to the X or Y axis of the chart. Default is "Y".
addValue	character or character vector. Adds value(s) from the specified filter column and X or Y variable to be included in the chart.
removeValue	character or character vector. Removes value(s) from the specified filter column and X or Y variable from being included in the chart.

Value

A Chart object representing a SyncroSim chart

Examples

```
## Not run:
# Create a chart object
myChart <- chart(myProject, chart = "New Chart")

# Set the chart type and data
myChart <- chartData(myChart, y = c("variable1", "variable2"),
timesteps = c(0,10), iterationType = "single", iteration = 1)

# Include specific values in the chart
myChart <- chartInclude(myChart, variable = "variable1",
```

```
filter="col1", addValue=c("val1", "val2", "val3"))

# Remove specific values from the chart
myChart <- chartInclude(myChart, variable = "variable1",
filter="col1", removeValue="val3")

## End(Not run)
```

---

chartOptionsFont *Modifies the font settings for a* Chart

Description

Modifies the font style and size of various Chart components.

Usage

```
chartOptionsFont(
  chart,
  titleFont = NULL,
  titleStyle = NULL,
  titleSize = NULL,
  panelFont = NULL,
  panelStyle = NULL,
  panelSize = NULL,
  axisFont = NULL,
  axisStyle = NULL,
  axisSize = NULL,
  legendFont = NULL,
  legendStyle = NULL,
  legendSize = NULL
)

## S4 method for signature 'Chart'
chartOptionsFont(
  chart,
  titleFont = NULL,
  titleStyle = NULL,
  titleSize = NULL,
  panelFont = NULL,
  panelStyle = NULL,
  panelSize = NULL,
  axisFont = NULL,
  axisStyle = NULL,
  axisSize = NULL,
  legendFont = NULL,
  legendStyle = NULL,
```

```
  legendSize = NULL
)
```

Arguments

chart	Chart object
titleFont	character. Sets the font for the title of the chart axes (e.g., "Microsoft Sans Serif, "Times New Roman", "Arial Narrow"). Default is NULL.
titleStyle	character. Sets the font style for the title. Values can be "standard", "italic", "bold", or "bold/italic". Default is NULL.
titleSize	integer. Sets the font size for the title of the chart axes. Default is NULL.
panelFont	character. Sets the font for the title of the chart panels (e.g., "Microsoft Sans Serif, "Times New Roman", "Arial"). Default is NULL.
panelStyle	character. Sets the font style for the chart panels. Values can be "standard", "italic", "bold", or "bold/italic". Default is NULL.
panelSize	integer. Sets the font size for the chart panels. Default is NULL.
axisFont	character. Sets the font for the title of the chart panel axes (e.g., "Microsoft Sans Serif, "Times New Roman", "Arial"). Default is NULL.
axisStyle	character. Sets the font style for the chart panel axes. Values can be "standard", "italic", "bold", or "bold/italic". Default is NULL.
axisSize	integer. Sets the font size for the chart panel axes. Default is NULL.
legendFont	character. Sets the font for the title of the chart legend (e.g., "Microsoft Sans Serif, "Times New Roman", "Arial"). Default is NULL.
legendStyle	character. Sets the font style for the chart legend. Values can be "standard", "italic", "bold", or "bold/italic". Default is NULL.
legendSize	integer. Sets the font size for the chart legend. Default is NULL.

Value

A Chart object representing a SyncroSim chart or, if no arguments other than the chart are provided, a data.frame of the current chart font settings.

Examples

```
## Not run:
# Open a chart object
myChart <- chart(myProject, chart = "My Chart")

# Set the font for the chart panels
myChart <- chartOptionsFont(myChart, panelFont = "Microsoft Sans Serif",
                            panelStyle = "bold/italic", panelSize = 8)

# Return a dataframe of the current font settings
myChart <- chartOptionsFont(myChart)

## End(Not run)
```

chartOptionsFormat *Modifies the font settings for a* Chart

Description

Modifies the font style and size of various Chart components.

Usage

```
chartOptionsFormat(
  chart,
  noDataAsZero = NULL,
  showDataPoints = NULL,
  showDataPointsOnly = NULL,
  showPanelTitles = NULL,
  showToolTips = NULL,
  showNoDataPanels = NULL,
  lineWidth = NULL
)

## S4 method for signature 'Chart'
chartOptionsFormat(
  chart,
  noDataAsZero = NULL,
  showDataPoints = NULL,
  showDataPointsOnly = NULL,
  showPanelTitles = NULL,
  showToolTips = NULL,
  showNoDataPanels = NULL,
  lineWidth = NULL
)
```

Arguments

chart	Chart object
noDataAsZero	logical. Determines whether NA, Null and No Data values should be charted as zero. Default is NULL.
showDataPoints	logical. Determines whether each data point should be displayed with a point (i.e., circle). Default is NULL.
showDataPointsOnly	logical. Determines whether only points should be displayed (i.e., no line in the line charts). Default is NULL.
showPanelTitles	logical. Determines whether to show a title above each panel. Default is NULL.
showToolTips	logical. Determines whether to show the tool tip when hovering the cursor over a data point. Default is NULL.

showNoDataPanels
: logical. Determines whether to show chart panels with no data. Default is NULL.

lineWidth
: integer. Sets the charts' line thickness. Default is NULL.

Value

A Chart object representing a SyncroSim chart or, if no arguments other than the chart are provided, a data.frame of the current chart format settings.

Examples

```
## Not run:
# Open a chart object
myChart <- chart(myProject, chart = "My Chart")

# Set the format for the chart panels
myChart <- chartOptionsFormat(myChart, noDataAsZero = TRUE,
                              showDataPoints = FALSE,
                              showDataPointsOnly = FALSE,
                              showPanelTitles = TRUE,
                              showToolTips = TRUE,
                              showNoDataPanels = FALSE,
                              lineWidth = 1)

# Return a dataframe of the current font settings
myChart <- chartOptionsFormat(myChart)

## End(Not run)
```

---

chartOptionsLegend *Modifies the legend settings for a* Chart

---

Description

Modifies the legend settings for a Chart.

Usage

```
chartOptionsLegend(
  chart,
  show = NULL,
  showScenarioName = NULL,
  showScenarioId = NULL,
  showStageName = NULL,
  showTimestamp = NULL
)

## S4 method for signature 'Chart'
```

```
chartOptionsLegend(
  chart,
  show = NULL,
  showScenarioName = NULL,
  showScenarioId = NULL,
  showStageName = NULL,
  showTimestamp = NULL
)
```

Arguments

chart	Chart object
show	logical. Whether to show the chart legend. Default is NULL.
showScenarioName	logical. Whether to show the scenario name in the legend. Default is NULL.
showScenarioId	logical. Whether to show the scenario ID in the legend. Default is NULL.
showStageName	logical. Determines whether to show the stage name (i.e., transformer name) in the legend. Default is NULL.
showTimestamp	logical. Whether to show the timestamp of the scenario run in the legend. Default is NULL. Default is NULL.

Value

A Chart object representing a SyncroSim chart or, if no arguments other than the chart are provided, a data.frame of the current chart legend settings.

Examples

```
## Not run:
# Open a chart object
myChart <- chart(myProject, chart = "My Chart")

# Remove the scenario ID and the timestamp from the chart
myChart <- chartOptionsLegend(myChart, showScenarioId = FALSE,
                              showTimestamp = FALSE)

# Hide the chart legend
myChart <- chartOptionsLegend(myChart, show = FALSE)

## End(Not run)
```

chartOptionsXAxis *Modify the X Axis of a* Chart

Description

Set the title and style of the X Axis of a Chart.

Usage

```
chartOptionsXAxis(
  chart,
  title = NULL,
  numberStyle = NULL,
  decimals = NULL,
  thousandsSeparator = NULL
)

## S4 method for signature 'Chart'
chartOptionsXAxis(
  chart,
  title = NULL,
  numberStyle = NULL,
  decimals = NULL,
  thousandsSeparator = NULL
)
```

Arguments

chart	Chart object
title	character. Title of the X Axis. Default is NULL.
numberStyle	character. Sets the style for the axes labels. Options include "number", scientific", or "currency". Default is NULL.
decimals	float. Sets the number of decimal places to be displayed in the axes labels. Values can be between 0 and 8. Default is NULL.
thousandsSeparator	logical. Whether to use a thousand separator (i.e., 1,000,000). Default is NULL.

Value

A Chart object representing a SyncroSim chart or, if no arguments other than the chart are provided, a data.frame of the current chart X Axis settings.

Examples

```
## Not run:
# Open a chart object
myChart <- chart(myProject, chart = "My Chart")
```

```
# Set the chart X Axis title
myChart <- chartOptionsXAxis(myChart, title = "Year")

# Return a dataframe of the current X Axis settings
myChart <- chartOptionsXAxis(myChart)

## End(Not run)
```

`chartOptionsYAxis` *Modify the Y axis of a* `Chart`

Description

Set the title and style of the Y axis of a `Chart`.

Usage

```
chartOptionsYAxis(
  chart,
  title = NULL,
  numberStyle = NULL,
  decimals = NULL,
  thousandsSeparator = NULL,
  minZero = NULL,
  sameScale = NULL,
  fixedIntervals = NULL
)

## S4 method for signature 'Chart'
chartOptionsYAxis(
  chart,
  title = NULL,
  numberStyle = NULL,
  decimals = NULL,
  thousandsSeparator = NULL,
  minZero = NULL,
  sameScale = NULL,
  fixedIntervals = NULL
)
```

Arguments

`chart`	`Chart` object
`title`	character. Title of the Y axis. Default is `NULL`.
`numberStyle`	character. Sets the style for the axes labels. Options include "number", scientific", or "currency". Default is `NULL`.

decimals	float. Sets the number of decimal places to be displayed in the axes labels. Values can be between 0 and 8. Default is NULL.
thousandsSeparator	logical. Whether to use a thousand separator (i.e., 1,000,000). Default is NULL.
minZero	logical. Whether the minimum value displayed in the Y axis should be zero.
sameScale	logical. Whether the Y axis scale should be consistent across chart panels. Default is NULL.
fixedIntervals	logical. Whether the interval between Y axis labels should be consistent across chart panels. Default is NULL.

Value

A Chart object representing a SyncroSim chart or, if no arguments other than the chart are provided, a data.frame of the current chart Y axis settings.

Examples

```
## Not run:
# Open a chart object
myChart <- chart(myProject, chart = "My Chart")

# Set the chart Y axis title
myChart <- chartOptionsYAxis(myChart, title = "Year")

# Return a dataframe of the current Y axis settings
myChart <- chartOptionsYAxis(myChart)

## End(Not run)
```

---

command *SyncroSim console command*

Description

This function issues a command to the SyncroSim console, and is mostly used internally by other functions.

Usage

```
command(
  args,
  session = NULL,
  program = "SyncroSim.Console.exe",
  wait = TRUE,
  progName = NULL
)
```

Arguments

args	character string, named list, named vector, unnamed list, or unnamed vector. Arguments for the SyncroSim console. See 'details' for more information about this argument
session	Session object. If NULL(default), the default session will be used
program	character. The name of the target SyncroSim executable. Options include "SyncroSim.Console.exe" (default), "SyncroSim.VizConsole.exe", "SyncroSim.PackageManager.exe" and "SyncroSim.Multiband.exe"
wait	logical. If TRUE(default) R will wait for the command to finish before proceeding. Note that silent(session) is ignored if wait=FALSE
progName	character. Internal argument for setting path to SyncroSim installation folder.

Details

Example args, and the resulting character string passed to the SyncroSim console:

- Character string e.g. "–create –help": "–create –help"
- Named list or named vector e.g. list(name1=NULL,name2=value2): "–name1 –name2=value2"
- Unnamed list or unnamed vector e.g. c("create","help"): "–create –help"

Value

Character string: output from the SyncroSim program.

Examples

```
## Not run:
# Install "stsim" if not already installed
installPackage("stsim")

# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib.ssim")

# Specify the command line arguments for creating a new stsim SsimLibrary
args <- list(create = NULL, library = NULL,
             name = myLibraryName,
             package = "stsim")

# Use a default session to create a new SsimLibrary in the current working directory
output <- command(args, session = session(printCmd = TRUE))
output

# Provide arguments to the command line using an unnamed vector
command(c("create", "help"))

# Provide arguments to the command line using a character string
command("--create --help")

# Provide arguments to the command line using a named list
```

```
command(list(create = NULL, help = NULL))

# Call on a different program to find all installed packages
command(list(installed = NULL), program = "SyncroSim.PackageManager.exe")

## End(Not run)
```

`condaFilepath` *Path to Conda installation folder*

Description

Gets or sets the path to the Conda installation folder. Can be used to direct SyncroSim to a custom Conda installation.

Usage

```
condaFilepath(session)

## S4 method for signature 'Session'
condaFilepath(session)

## S4 method for signature 'missingOrNULLOrChar'
condaFilepath(session)

condaFilepath(session) <- value

## S4 replacement method for signature 'character'
condaFilepath(session) <- value

## S4 replacement method for signature 'Session'
condaFilepath(session) <- value
```

Arguments

`session`	`Session` object or character (i.e. filepath to a session). If `NULL`, `session()` will be used
`value`	character. If empty, then returns the current Conda installation path

Value

A character: the currently set filepath of the Conda installation folder.

Examples

```
## Not run:
# Set up a SyncroSim Session
mySession <- session()

# Retrieve Conda installation path for the SyncroSim Session
condaFilepath(mySession)

# Set the Conda installation path for the SyncroSim Session
condaFilepath(mySession) <- "C:/miniconda3"

## End(Not run)
```

`createCondaEnv` *Create SyncroSim package conda environments*

Description

Creates the conda environment for the specified SyncroSim package(s).

Usage

```
createCondaEnv(pkgs, session = NULL)

## S4 method for signature 'ANY,character'
createCondaEnv(pkgs, session = NULL)

## S4 method for signature 'ANY,missingOrNULL'
createCondaEnv(pkgs, session = NULL)

## S4 method for signature 'ANY,Session'
createCondaEnv(pkgs, session = NULL)
```

Arguments

`pkgs`	character or list of characters.
`session`	`Session` object or character (i.e. filepath to a session). If `NULL`, `session()` will be used

Value

Invisibly returns `TRUE` upon success (i.e.successful creation of the conda environment(s)) or `FALSE` upon failure.

Examples

```
## Not run:
# Set up a SyncroSim Session
mySession <- session()

# Create the conda environment for helloworldConda package
condaFilepath(pkgs = "helloworldConda", mySession)

## End(Not run)
```

`datasheet` *Retrieve a SyncroSim Datasheet*

Description

This function retrieves a SyncroSim Datasheet, either by calling the SyncroSim console, or by directly querying the `SsimLibrary` database.

Usage

```
datasheet(
  ssimObject,
  name = NULL,
  project = NULL,
  scenario = NULL,
  summary = NULL,
  optional = FALSE,
  empty = FALSE,
  filterColumn = NULL,
  filterValue = NULL,
  lookupsAsFactors = TRUE,
  sqlStatement = list(select = "SELECT *", groupBy = ""),
  includeKey = FALSE,
  forceElements = FALSE,
  fastQuery = FALSE,
  returnScenarioInfo = FALSE,
  returnInvisible = FALSE,
  rawValues = FALSE,
  verbose = TRUE
)

## S4 method for signature 'list'
datasheet(
  ssimObject,
  name = NULL,
  project = NULL,
```

```
  scenario = NULL,
  summary = NULL,
  optional = FALSE,
  empty = FALSE,
  filterColumn = NULL,
  filterValue = NULL,
  lookupsAsFactors = TRUE,
  sqlStatement = list(select = "SELECT *", groupBy = ""),
  includeKey = FALSE,
  forceElements = FALSE,
  fastQuery = FALSE,
  returnScenarioInfo = FALSE,
  returnInvisible = FALSE,
  rawValues = FALSE,
  verbose = TRUE
)

## S4 method for signature 'character'
datasheet(
  ssimObject,
  name,
  project,
  scenario,
  summary,
  optional,
  empty,
  filterColumn,
  filterValue,
  lookupsAsFactors,
  sqlStatement,
  includeKey,
  fastQuery,
  returnScenarioInfo,
  returnInvisible,
  rawValues,
  verbose
)

## S4 method for signature 'SsimObject'
datasheet(
  ssimObject,
  name = NULL,
  project = NULL,
  scenario = NULL,
  summary = NULL,
  optional = FALSE,
  empty = FALSE,
  filterColumn = NULL,
```

```
  filterValue = NULL,
  lookupsAsFactors = TRUE,
  sqlStatement = list(select = "SELECT *", groupBy = ""),
  includeKey = FALSE,
  forceElements = FALSE,
  fastQuery = FALSE,
  returnScenarioInfo = FALSE,
  returnInvisible = FALSE,
  rawValues = FALSE,
  verbose = TRUE
)
```

Arguments

ssimObject	SsimLibrary, Project, or Scenario object or list of objects. Note that all objects in a list must be of the same type, and belong to the same SsimLibrary
name	character or character vector. Sheet name(s). If NULL (default), all datasheets in the ssimObject will be returned. Note that setting summary=FALSE and name=NULL pulls all Datasheets, which is time consuming and not generally recommended
project	numeric or numeric vector. One or more Project ids
scenario	numeric or numeric vector. One or more Scenario ids
summary	logical or character. If TRUE (default) returns a data.frame of sheet names and other info including built-in core SyncroSim Datasheets. If FALSE returns data.frame or list of data.frames.
optional	logical. If summary=TRUE and optional=TRUE returns only scope, name and displayName. If summary=FALSE and optional=TRUE returns all of the Datasheet's columns, including the optional columns. If summary=TRUE, optional=FALSE (default), returns only those columns that are mandatory and contain data (if empty=FALSE). Ignored if summary=FALSE, empty=FALSE and lookupsAsFactors=FALSE
empty	logical. If TRUE returns empty data.frames for each Datasheet. Ignored if summary=TRUE Default is FALSE
filterColumn	character string. The column to filter a Datasheet by. (e.g. "TransitionGroupId"). Note that to use the filterColumn argument, you must also specify the filterValue argument. Default is NULL
filterValue	character string or integer. The value to filter the filterColumn by. To use the filterValue argument, you must also specify the filterColumn argument. Default is NULL
lookupsAsFactors	logical. If TRUE (default) dependencies returned as factors with allowed values (levels). Set FALSE to speed calculations. Ignored if summary=TRUE
sqlStatement	list returned by sqlStatement. SELECT and GROUP BY SQL statements passed to SQLite database. Ignored if summary=TRUE (optional)
includeKey	logical. If TRUE include primary key in table. Default is FALSE
forceElements	logical. If FALSE (default) and name has a single element returns a data.frame; otherwise returns a list of data.frames. Ignored if summary=TRUE

`fastQuery` logical. If `TRUE`, the request is optimized for performance. Ignored if combined with summary, empty, or `sqlStatement` flags. Default is `FALSE`

`returnScenarioInfo` logical. If `TRUE`, returns the Scenario Id, Scenario Name, Parent Id, and Parent Name columns with the Scenario-scoped Datasheet. Does nothing if the Datasheet exists at the Library or Project level. Default is `FALSE`

`returnInvisible` logical. If `TRUE`, returns columns that are invisible in the User Interface (i.e., are only used and populated internally by SyncroSim or the SyncroSim Package). Default is `FALSE`

`rawValues` logical. If `TRUE`, returns the raw ID values rather than automatically translating the values to strings. Default is `FALSE`.

`verbose` logical. If set to `FALSE`, will not print notes about datasheet validation. Default is `TRUE`.

Details

If `summary=TRUE` or `summary=NULL` and `name=NULL` a data.frame describing the Datasheets is returned. If `optional=TRUE`, columns include: `scope`, `packages`, `name`, `displayName`, `isSingle`, `data`. data only displayed for a SyncroSim `Scenario`. `dataInherited` and `dataSource` columns added if a Scenario has dependencies. If `optional=FALSE`, columns include: `scope`, `name`, `displayName`. All other arguments are ignored.

Otherwise, for each element in name a Datasheet is returned as follows:

- If `lookupsAsFactors=TRUE` (default): Each column is given the correct data type, and dependencies returned as factors with allowed values (levels). A warning is issued if the lookup has not yet been set.
- If `empty=TRUE`: Each column is given the correct data type. Fast (1 less console command).
- If `empty=FALSE` and `lookupsAsFactors=FALSE`: Column types are not checked, and the optional argument is ignored. Fast (1 less console command).
- If SsimObject is a list of `Scenario` or `Project` objects (output from `run`, `Scenario` or `Project`): Adds ScenarioId/ProjectId column if appropriate.
- If Scenario/Project is a vector: Adds ScenarioId/ProjectId column as necessary.
- If requested Datasheet has Scenario scope and contains info from more than one Scenario: ScenarioId/ScenarioName/ScenarioParent columns identify the Scenario by `name`, `id`, and `parent` (if a result Scenario).
- If requested Datasheet has Project scope and contains info from more than one Project: ProjectId/ProjectName columns identify the Project by `name` and `id`

Value

If `summary=TRUE` returns a data.frame of Datasheet names and other information, otherwise returns a data.frame or list of these.

Examples

```
## Not run:
# Install helloworldSpatial package from package server
installPackage("helloworldSpatial")

# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib_datasheet")

# Set the SyncroSim Session
mySession <- session()

# Create a new SsimLibrary with the example template from helloworldSpatial
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession,
                         packages = "helloworldSpatial")

# Set the Project and Scenario
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Get all Datasheet info for the Scenario
myDatasheets <- datasheet(myScenario)


# Return a list of data.frames (1 for each Datasheet)
myDatasheetList <- datasheet(myScenario, summary = FALSE)

# Get a specific Datasheet
myDatasheet <- datasheet(myScenario, name = "helloworldSpatial_RunControl")

# Include primary key when retrieving a Datasheet
myDatasheet <- datasheet(myScenario, name = "helloworldSpatial_RunControl",
                         includeKey = TRUE)

# Return all columns, including optional ones
myDatasheet <- datasheet(myScenario, name = "helloworldSpatial_RunControl",
                         summary = TRUE, optional = TRUE)

# Return Datasheet as an element
myDatasheet <- datasheet(myScenario, name = "helloworldSpatial_RunControl",
                         forceElements = TRUE)
myDatasheet$helloworldSpatial_RunControl

# Get a Datasheet without pre-specified values
myDatasheetEmpty <- datasheet(myScenario,
                              name = "helloworldSpatial_RunControl",
                              empty = TRUE)

# If Datasheet is empty, do not return dependencies as factors
myDatasheetEmpty <- datasheet(myScenario,
                              name = "helloworldSpatial_RunControl",
                              empty = TRUE,
```

```
                              lookupsAsFactors = FALSE)

# Optimize query
myDatasheet <- datasheet(myScenario, name = "helloworldSpatial_RunControl",
                         fastQuery = TRUE)

# Get specific SsimLibrary core Datasheet
myDatasheet <- datasheet(myLibrary, name = "core_Backup")

# Use an SQL statement to query a Datasheet
mySQL <- sqlStatement(
  groupBy = c("ScenarioId"),
  aggregate = c("MinimumTimestep"),
  where = list(MinimumTimestep = c(1))
)
myAggregatedDatasheet <- datasheet(myScenario,
                                   name = "helloworldSpatial_RunControl",
                                   sqlStatement = mySQL)

## End(Not run)
```

---

datasheetSpatRaster *Retrieve spatial data from a SyncroSim Datasheet*

Description

This function retrieves spatial columns from one or more SyncroSim Scenario Datasheets.

Usage

```
datasheetSpatRaster(
  ssimObject,
  datasheet,
  column = NULL,
  scenario = NULL,
  iteration = NULL,
  timestep = NULL,
  filterColumn = NULL,
  filterValue = NULL,
  subset = NULL,
  forceElements = FALSE,
  pathOnly = FALSE
)

## S4 method for signature 'character'
datasheetSpatRaster(
  ssimObject,
  datasheet,
```

```
  column = NULL,
  scenario = NULL,
  iteration = NULL,
  timestep = NULL,
  filterColumn = NULL,
  filterValue = NULL,
  subset = NULL,
  forceElements = FALSE,
  pathOnly = FALSE
)

## S4 method for signature 'list'
datasheetSpatRaster(
  ssimObject,
  datasheet,
  column = NULL,
  scenario = NULL,
  iteration = NULL,
  timestep = NULL,
  filterColumn = NULL,
  filterValue = NULL,
  subset = NULL,
  forceElements = FALSE,
  pathOnly = FALSE
)

## S4 method for signature 'SsimObject'
datasheetSpatRaster(
  ssimObject,
  datasheet,
  column = NULL,
  scenario = NULL,
  iteration = NULL,
  timestep = NULL,
  filterColumn = NULL,
  filterValue = NULL,
  subset = NULL,
  forceElements = FALSE,
  pathOnly = FALSE
)

## S4 method for signature 'Scenario'
datasheetSpatRaster(
  ssimObject,
  datasheet,
  column = NULL,
  scenario = NULL,
  iteration = NULL,
```

```
  timestep = NULL,
  filterColumn = NULL,
  filterValue = NULL,
  subset = NULL,
  forceElements = FALSE,
  pathOnly = FALSE
)
```

Arguments

ssimObject	SsimLibrary/Project/Scenario object or list of Scenario objects. If SsimLibrary/Project, then scenario argument is required
datasheet	character string. The name of the Datasheet containing the raster data
column	character string. The name of the column in the datasheet containing the file names for raster data. If NULL (default) then use the first column that contains raster file names
scenario	character string, integer, or vector of these. The Scenarios to include. Required if SsimObject is an SsimLibrary/Project, ignored if SsimObject is a list of Scenarios (optional)
iteration	integer, character string, or vector of integer/character strings. Iteration(s) to include. If NULL (default) then all iterations are included. If no Iteration column is in the Datasheet, then ignored
timestep	integer, character string, or vector of integer/character string. Timestep(s) to include. If NULL (default) then all timesteps are included. If no Timestep column is in the Datasheet, then ignored
filterColumn	character string. The column to filter a Datasheet by. (e.g. "TransitionGroupID"). Note that to use the filterColumn argument, you must also specify a filterValue. Default is NULL
filterValue	character string or integer. The value of the filterColumn to filter the Datasheet by. To use the filterValue argument, you must also specify a filterColumn. Default is NULL
subset	logical expression indicating Datasheet rows to return. e.g. expression(grepl("Ts0001", Filename, fixed=T)). See subset() for details (optional)
forceElements	logical. If TRUE then returns a single raster as a RasterStack; otherwise returns a single raster as a RasterLayer directly. Default is FALSE
pathOnly	logical. If TRUE then returns a list of filepaths to the raster files on disk. Default is FALSE

Details

The names of the returned SpatRaster contain metadata. For Datasheets without Filename this is:

paste0(<datasheet name>,".Scn",<scenario id>,".",<tif name>).

For Datasheets containing Filename this is:

paste0(<datasheet name>,".Scn",<scenario id>,".It",<iteration>,".Ts",<timestep>).

Value

A SpatRaster object, or List. See terra package documentation for details.

Examples

```
## Not run:
# Extract specific Datasheet rasters by iteration and timestep
resultRaster <- datasheetSpatRaster(resultScenario,
                  datasheet = "helloworldSpatial_IntermediateDatasheet",
                  column = "OutputRasterFile",
                  iteration = 3,
                  timestep = 2
)

# Extract specific Datasheet SpatRasters using pattern matching
resultDatasheet <- datasheet(resultScenario,
                             name = "helloworldSpatial_IntermediateDatasheet")
outputRasterPaths <- resultDatasheet$OutputRasterFile
resultRaster <- datasheetSpatRaster(resultScenario,
                  datasheet = "helloworldSpatial_IntermediateDatasheet",
                  column = "OutputRasterFile",
                  subset = expression(grepl("ts20",
                                            outputRasterPaths,
                                            fixed = TRUE))
)

# Return the raster Datasheets as a SpatRaster list
resultRaster <- datasheetSpatRaster(resultScenario,
                 datasheet = "helloworldSpatial_IntermediateDatasheet",
                 column = "OutputRasterFile",
                 forceElements = TRUE)

# Filter for only rasters that fit specific criteria (ST-Sim example)
resultRaster <- datasheetSpatRaster(resultScenario,
                 datasheet = "stsim_OutputSpatialTransition",
                 timestep = 5,
                 iteration = 5,
                 filterColumn = "TransitionTypeID",
                 filterValue = "Fire")

## End(Not run)
```

`dateModified` *Last date a SsimLibrary, Project, Scenario, or Folder was modified*

Description

The most recent modification date of a `SsimLibrary`, `Project`, `Scenario` or `Folder`.

Usage

```
dateModified(ssimObject)

## S4 method for signature 'character'
dateModified(ssimObject)

## S4 method for signature 'SsimLibrary'
dateModified(ssimObject)

## S4 method for signature 'Project'
dateModified(ssimObject)

## S4 method for signature 'Scenario'
dateModified(ssimObject)

## S4 method for signature 'Folder'
dateModified(ssimObject)
```

Arguments

ssimObject	SsimLibrary, Project, Scenario, or Folder object

Value

A character string: date and time of the most recent modification to the SsimObject provided as input.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session and SsimLibrary
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# Check the last date of modification of the SsimLibrary
dateModified(myLibrary)

## End(Not run)
```

---

delete *Delete Project, Scenario, Folder, Chart or Datasheet*

Description

Delete Project, Scenario, Folder, Chart or Datasheet

Usage

```
delete(
  ssimObject,
  project = NULL,
  scenario = NULL,
  folder = NULL,
  chart = NULL,
  datasheet = NULL,
  force = FALSE,
  session = NULL
)

## S4 method for signature 'SsimObject'
delete(
  ssimObject,
  project = NULL,
  scenario = NULL,
  folder = NULL,
  chart = NULL,
  datasheet = NULL,
  force = FALSE,
  session = NULL
)
```

Arguments

ssimObject	SsimLibrary, Project, Scenario, Folder, or Chart object
project	character string, numeric, or vector of these. One or more Project names or ids. Note that project argument is ignored if ssimObject is a list. Note that integer ids are slightly faster (optional)
scenario	character string, numeric, or vector of these. One or more Scenario names or ids. Note that scenario argument is ignored if ssimObject is a list. Note that integer ids are slightly faster (optional)
folder	character string, numeric, or vector of these. One or more Folder names or ids. Note that folder argument is ignored if ssimObject is a list. Note that integer ids are slightly faster (optional)
chart	character string, numeric, or vector of these. One or more Chart names or ids. Note that chart argument is ignored if SsimObject is a list. Note that integer ids are slightly faster (optional)
datasheet	character string or vector of these. One or more datasheet names (optional)
force	logical. If FALSE (default), user will be prompted to approve removal of each item
session	Session object. If NULL (default), session() will be used. Only applicable when ssimObject argument is a character

Details

Deletes one or more items. Note that this is irreversible. To delete a library, you must use the deleteLibrary function instead.

Value

Invisibly returns a list of boolean values corresponding to each input: TRUE upon success (i.e.successful deletion) and FALSE upon failure.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "a project")

# Check the Projects associated with this SsimLibrary
project(myLibrary)

# Delete Project
delete(myLibrary, project = "a project", force = TRUE)

# Check that Project was successfully deleted from SsimLibrary
project(myLibrary)

## End(Not run)
```

---

deleteLibrary *Delete Library*

Description

Deletes a SyncroSim library. Note this is irreversable.

Usage

```
deleteLibrary(
  ssimLibrary,
  force = FALSE,
  removeBackup = FALSE,
  removePublish = FALSE,
  removeCustom = FALSE,
  session = NULL
```

```
)

## S4 method for signature 'SsimLibrary'
deleteLibrary(ssimLibrary, force, removeBackup, removePublish, removeCustom)

## S4 method for signature 'character'
deleteLibrary(
  ssimLibrary,
  force = FALSE,
  removeBackup = FALSE,
  removePublish = FALSE,
  removeCustom = FALSE,
  session = NULL
)
```

Arguments

ssimLibrary	SsimLibrary or path to a library
force	Logical. If FALSE (default) prompt to confirm that the library should be deleted. This is irreversable.
removeBackup	logical. If TRUE, will remove the backup folder when deleting a library. Default is FALSE.
removePublish	logical. If TRUE, will remove the publish folder when deleting a library. Default is FALSE.
removeCustom	logical. If TRUE and custom folders have been configured for a library, then will remove the custom publish and/or backup folders when deleting a library. Note that the removePublish and removeBackup arguments must also be set to TRUE to remove the respective custom folders. Default is FALSE.
session	Session

Value

"saved" or failure message.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")
myLibraryName2 <- file.path(tempdir(), "testlib2")

# Set up a SyncroSim Session and create SsimLibrary
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# Delete library from SsimObject
deleteLibrary(myLibrary, force = TRUE, removeBackup = TRUE)

# Create another library
```

```
myLibrary <- ssimLibrary(name = myLibraryName2, session = mySession)

# Delete library from path
deleteLibrary(myLibraryName2)

## End(Not run)
```

---

dependency *Get, set or remove Scenario dependencies*

Description

List dependencies, set dependencies, or remove dependencies from a SyncroSim Scenario. Setting dependencies is a way of linking together Scenario Datafeeds, such that a change in the Scenario that is the source dependency will update the dependent Scenario as well.

Usage

```
dependency(ssimObject)

## S4 method for signature 'character'
dependency(ssimObject)

## S4 method for signature 'Scenario'
dependency(ssimObject)

dependency(ssimObject) <- value

## S4 replacement method for signature 'Scenario'
dependency(ssimObject) <- value
```

Arguments

ssimObject	Scenario object, character string, integer, or vector of these. The Scenario object, name, or ID to which a dependency is to be added (or has already been added if remove=TRUE). Note that integer ids are slightly faster.
value	Scenario object, character string, integer, or vector of these. The Scenario object, name, or ID to be used as the dependency. If an empty vector is provided, all dependencies are removed. If multiple elements are provided, elements should be ordered from highest to lowest precedence.

Details

If dependency==NULL, other arguments are ignored, and set of existing dependencies is returned in order of precedence (from highest to lowest precedence). Otherwise, returns list of saved or error messages for each dependency of each scenario.

Note that pre-existing dependencies will be removed when adding new dependencies unless those elements are included in the vector of new dependencies.

Value

A data.frame: all dependencies for a given Scenario

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, Project, and 2 Scenarios
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")
myNewScenario <- scenario(myProject,
                          scenario = "my New Scenario")

# Set myScenario as a dependency of myNewScenario
dependency(myNewScenario) <- myScenario

# Get all dependencies info
dependency(myNewScenario)

# Remove all dependencies
dependency(myNewScenario) <- c()

## End(Not run)
```

---

description *Description of SsimLibrary, Project or Scenario*

Description

Get or set the description of a SsimLibrary, Project, or Scenario.

Usage

```
description(ssimObject)

description(ssimObject) <- value

## S4 method for signature 'character'
description(ssimObject)

## S4 method for signature 'SsimObject'
description(ssimObject)

## S4 replacement method for signature 'character'
```

```
description(ssimObject) <- value

## S4 replacement method for signature 'SsimObject'
description(ssimObject) <- value
```

Arguments

ssimObject	SsimLibrary, Project, Scenario, or Folder object
value	character string specifying the new description

Value

A character string: the description of the SsimObject

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")

# Retrieve the description of the SyncroSim Project
mydescription <- description(myProject)

# Set the description of the SyncroSim Project
description(myProject) <- "my description"

## End(Not run)
```

---

filepath *Retrieves the path to a SyncroSim object on disk*

Description

Retrieves the path to a SyncroSim Session, SsimLibrary, Project, Scenario, of Folder on disk.

Usage

```
filepath(ssimObject)

## S4 method for signature 'character'
filepath(ssimObject)

## S4 method for signature 'Session'
```

```
filepath(ssimObject)

## S4 method for signature 'SsimObject'
filepath(ssimObject)

## S4 method for signature 'Folder'
filepath(ssimObject)
```

Arguments

ssimObject	Session, Project, SsimLibrary, or Folder object

Value

A character string: the path to a SyncroSim object on disk.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session and SsimLibrary
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# Get the file path
myFilePath <- filepath(myLibrary)

## End(Not run)
```

---

folder *Create or open a Folder*

Description

Create or open a Folder from a SyncroSim Project.

Usage

```
folder(
  ssimObject = NULL,
  folder = NULL,
  parentFolder = NULL,
  summary = FALSE,
  create = FALSE
)
```

Arguments

ssimObject	SsimLibrary or Project object.
folder	character or integer. If character, then will either open an existing folder if create=FALSE, or will create a new folder with the given name if the folder does not exist yet or create=TRUE (Default). If integer, will open the existing folder with the given folder ID (if the ID exists).
parentFolder	character, integer, or SyncroSim Folder object. If not NULL (Default), the new folder will be created inside of the specified parent folder
summary	logical. If FALSE, then returns a folder object. If TRUE, then returns a dataframe of information about the specified folder
create	logical. Whether to create a new folder if the folder name given already exists in the SyncroSim library. If FALSE (Default), then will return the existing folder with the given name. If TRUE, then will return a new folder with the same name as an existing folder (but different folder ID)

Value

A Folder object representing a SyncroSim folder.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "My Project")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Create a new folder
myFolder <- folder(myProject, folder = "New Folder")

# Create a nested folder within "New Folder"
myNestedFolder <- folder(myProject, folder = "New Nested Folder",
                         parentFolder = myFolder)

# Retrieve a dataframe of all folders in a project
folder(myProject)

## End(Not run)
```

Folder-class *SyncroSim Folder class*

Description

`Folder` object representing a SyncroSim Folder. A Folder is used to organize SyncroSim Scenarios within a `Project`, and can be nested within other Folders at the project-level. These are used mostly in the SyncroSim User Interface.

Slots

`session` `Session` object. The Session associated with the Folder's SsimLibrary

`filepath` character string. The path to the Folder's SsimLibrary on disk

`folderId` integer. The Folder id

`parentId` integer. The parent Folder id (if the folder is nested)

`projectId` integer. The Project id

See Also

See `folder` for options when creating or loading a SyncroSim Folder

folderId *Retrieves folderId of SyncroSim Folder or Scenario*

Description

Retrieves the Folder Id of a SyncroSim `Folder` or `Scenario`. Can also use to set the Folder Id for a `Scenario` - this will move the `Scenario` into the desired folder in the SyncroSim User Interface.

Usage

```
folderId(ssimObject)

## S4 method for signature 'character'
folderId(ssimObject)

## S4 method for signature 'Folder'
folderId(ssimObject)

## S4 method for signature 'Scenario'
folderId(ssimObject)

folderId(ssimObject) <- value

## S4 replacement method for signature 'Scenario'
folderId(ssimObject) <- value
```

Arguments

ssimObject	Folder or Scenario object
value	integer of the folder ID to move the Scenario to. Only applicable if the ssimObject provided is a Scenario.

Value

An integer: folder id.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession,
                         overwrite = TRUE)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")
myFolder <- folder(myProject, "New Folder")

# Get Folder ID for SyncroSim Folder and Scenario
folderId(myFolder)
folderId(myScenario)

# Move the Scenario into the newly created folder
folderId(myScenario) <- folderId(myFolder)
folderId(myScenario)

## End(Not run)
```

---

ignoreDependencies *Ignore dependencies for a Scenario*

Description

Retrieves or sets the Datafeeds to ignore for a Scenario.

Usage

```
ignoreDependencies(ssimObject)

## S4 method for signature 'character'
ignoreDependencies(ssimObject)
```

```
## S4 method for signature 'Scenario'
ignoreDependencies(ssimObject)

ignoreDependencies(ssimObject) <- value

## S4 replacement method for signature 'character'
ignoreDependencies(ssimObject) <- value

## S4 replacement method for signature 'Scenario'
ignoreDependencies(ssimObject) <- value
```

Arguments

ssimObject	Scenario object
value	character string of Datafeed names to be ignored, separated by commas (optional)

Value

A character string: Scenario Datafeeds that will be ignored.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# List the Datafeeds to ignore
ignoreDependencies(myScenario)

# Set Scenario Datafeeds to ignore
ignoreDependencies(myScenario) <- "stsim_RunControl,stsim_TransitionTarget"

## End(Not run)
```

---

info *Retrieves information about a library*

Description

Retrieves some basic metadata about a SsimLibrary: Name, Owner, Last Modified, Size, Read Only, Data files, Publish files, Temporary files, Backup files, and Use conda.

Usage

```
info(ssimLibrary)

## S4 method for signature 'SsimLibrary'
info(ssimLibrary)
```

Arguments

ssimLibrary	SsimLibrary object

Value

Returns a data.frame with information on the properties of the SsimLibrary object.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session and SsimLibrary
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# Get information about SsimLibrary
info(myLibrary)

## End(Not run)
```

---

installConda *Installs Miniforge or Miniconda*

Description

This function installs the Miniforge or Miniconda package manager software to the default installation path within the SyncroSim installation folder. If you already have conda installed in the non-default location, you can point SyncroSim towards that installation using the condaFilepath function.

Usage

```
installConda(session, software = "miniforge")

## S4 method for signature 'character'
installConda(session, software = "miniforge")

## S4 method for signature 'missingOrNULL'
```

```
installConda(session, software = "miniforge")

## S4 method for signature 'Session'
installConda(session, software = "miniforge")
```

Arguments

session	Session object. If NULL (default), session() will be used
software	character. Whether to install the latest release of "miniforge" (Default) or "miniconda".

Value

Invisibly returns TRUE upon success (i.e.successful install) and FALSE upon failure.

Examples

```
## Not run:
# Install miniforge for the default SyncroSim session
installConda()

# Install miniconda for the default SyncroSim session
installConda(software = "miniconda")

## End(Not run)
```

---

installPackage *Adds package to SyncroSim Installation*

Description

This function installs a package to the SyncroSim Session. If only the package name is provided as input, the function queries the SyncroSim package server for the specified package. If a file path is provided as input, the function installs a package to SyncroSim from a local package file (ends in ".ssimpkg"). The list of SyncroSim packages can be found here.

Usage

```
installPackage(packages, versions = NULL, session = NULL)

## S4 method for signature 'ANY,ANY,character'
installPackage(packages, versions = NULL, session = NULL)

## S4 method for signature 'ANY,ANY,missingOrNULL'
installPackage(packages, versions = NULL, session = NULL)

## S4 method for signature 'ANY,ANY,Session'
installPackage(packages, versions = NULL, session = NULL)
```

Arguments

packages	character string. The name or file path of the package to install
versions	character string. The packages version(s) to install if installing a package from the server. If NULL then installs the latest version
session	Session object. If NULL (default), session() will be used

Value

Invisibly returns TRUE upon success (i.e.successful install) and FALSE upon failure.

Examples

```
## Not run:
# Create a new SyncroSim Session
mySession <- session()

# Install package from the package server
installPackage(packages="stsim", versions="4.0.1", session = mySession)

# Install package using a local file path
installPackage("c:/path/to/stsim.ssimpkg")

## End(Not run)
```

---

mergeDependencies *Merge dependencies for a Scenario*

Description

Retrieves or sets whether or not a Scenario is configured to merge dependencies at run time.

Usage

```
mergeDependencies(ssimObject)

## S4 method for signature 'character'
mergeDependencies(ssimObject)

## S4 method for signature 'Scenario'
mergeDependencies(ssimObject)

mergeDependencies(ssimObject) <- value

## S4 replacement method for signature 'character'
mergeDependencies(ssimObject) <- value

## S4 replacement method for signature 'Scenario'
mergeDependencies(ssimObject) <- value
```

Arguments

ssimObject	Scenario object
value	logical. If TRUE the Scenario will be set to merge dependencies at run time. Default is FALSE

Value

A logical: TRUE if the scenario is configured to merge dependencies at run time, and FALSE otherwise.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set up a SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Retrieve whether or not dependencies will be merged for a Scenario
mergeDependencies(myScenario)

# Set whether or not dependencies will be merged for a Scenario
mergeDependencies(myScenario) <- TRUE

## End(Not run)
```

---

name *Name of a SsimLibrary, Project, Scenario, Folder, or Chart*

Description

Retrieves or sets the name of a SsimLibrary, Project, Scenario, or Folder.

Usage

```
name(ssimObject)

## S4 method for signature 'character'
name(ssimObject)

## S4 method for signature 'SsimLibrary'
name(ssimObject)
```

```
## S4 method for signature 'Scenario'
name(ssimObject)

## S4 method for signature 'Project'
name(ssimObject)

## S4 method for signature 'Folder'
name(ssimObject)

## S4 method for signature 'Chart'
name(ssimObject)

name(ssimObject) <- value

## S4 replacement method for signature 'character'
name(ssimObject) <- value

## S4 replacement method for signature 'SsimLibrary'
name(ssimObject) <- value

## S4 replacement method for signature 'Project'
name(ssimObject) <- value

## S4 replacement method for signature 'Scenario'
name(ssimObject) <- value

## S4 replacement method for signature 'Folder'
name(ssimObject) <- value

## S4 replacement method for signature 'Chart'
name(ssimObject) <- value
```

Arguments

ssimObject	Scenario, Project, SsimLibrary, Folder or Chart object
value	character string of the new name

Value

A character string: the name of the SsimObject.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
```

```
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession,
                         packages = "stsim")
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")
myFolder <- folder(myProject, folder = "New Folder")
myChart <- chart(myProject, chart = "New Chart")

# Retrieve names of the SsimObjects
name(myLibrary)
name(myProject)
name(myScenario)
name(myFolder)
name(myChart)

# Set the name of the SyncroSim Scenario
name(myScenario) <- "My Scenario Name"

## End(Not run)
```

---

owner *Owner of a SsimLibrary, Project, or Scenario*

Description

Retrieves or sets the owner of a SsimLibrary, Project, or Scenario.

Usage

```
owner(ssimObject)

owner(ssimObject) <- value

## S4 method for signature 'character'
owner(ssimObject)

## S4 method for signature 'SsimLibrary'
owner(ssimObject)

## S4 method for signature 'Project'
owner(ssimObject)

## S4 method for signature 'Scenario'
owner(ssimObject)

## S4 replacement method for signature 'character'
owner(ssimObject) <- value
```

```
## S4 replacement method for signature 'SsimObject'
owner(ssimObject) <- value
```

Arguments

ssimObject	Session, Project, or SsimLibrary object
value	character string of the new owner

Value

A character string: the owner of the SsimObject.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Retrieve the owner of an SsimObject
owner(myLibrary)
owner(myProject)
owner(myScenario)

# Set the owner of a SyncroSim Scenario
owner(myScenario) <- "Apex RMS"

## End(Not run)
```

---

packages *Installed or available packages*

Description

Retrieves the packages installed or available in the current session if called on a Session object, or the packages added to a SyncroSim Library if called on a SsimLibrary object.

Usage

```
packages(ssimObject = NULL, installed = TRUE)

## S4 method for signature 'character'
packages(ssimObject = NULL, installed = TRUE)
```

```
## S4 method for signature 'missingOrNULL'
packages(ssimObject = NULL, installed = TRUE)

## S4 method for signature 'Session'
packages(ssimObject = NULL, installed = TRUE)

## S4 method for signature 'SsimLibrary'
packages(ssimObject)
```

Arguments

ssimObject	Session or SsimLibrary object. If NULL (default), session() will be used
installed	logical or character. TRUE (default) to list installed packages or FALSE to list available packages on the server

Value

Returns a data.frame of packages installed or templates available for a specified package.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session and SsimLibrary
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# List all installed packages
packages(mySession)

# List all available packages on the server (including currently installed)
packages(installed = FALSE)

# Check the package(s) in your SsimLibrary
packages(myLibrary)

## End(Not run)
```

---

parentId *Retrieves the parent Scenario id or parent Folder id*

Description

Retrieves the id of the parent of a SyncroSim results Scenario or a SyncroSim Folder.

Usage

```
parentId(child)

## S4 method for signature 'character'
parentId(child)

## S4 method for signature 'Scenario'
parentId(child)

## S4 method for signature 'Folder'
parentId(child)
```

Arguments

child	Scenario or Folder object

Value

An integer id of the parent Scenario if input is a Scenario, or an integer id of the parent Folder if input is a Folder. If the input Scenario or Folder does not have a parent, the function returns NA

Examples

```
## Not run:
# Set the file path and name of an existing SsimLibrary
myLibraryName <- "MyLibrary.ssim"

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Run Scenario to generate results
resultScenario <- run(myScenario)

# Find the parent ID of the Scenario
parentId(resultScenario)

## End(Not run)
```

---

printCmd *Retrieves printCmd setting of a Session*

Description

Retrieves a printCmd setting of a Session object. The printCmd setting configures a Session for printing commands sent to the console.

Usage

```
printCmd(session = NULL)

## S4 method for signature 'Session'
printCmd(session = NULL)

## S4 method for signature 'missingOrNULLOrChar'
printCmd(session = NULL)
```

Arguments

session	Session object or character. The Session or path to a Session where the printCmd settings are retrieved from. If NULL (default), session() will be used

Value

A logical : TRUE if the session is configured to print commands and FALSE if it is not.

Examples

```
## Not run:
# Set SyncroSim Session
mySession <- session()

# Retrieve printCmd settings for given Session
printCmd(mySession)

## End(Not run)
```

---

progressBar *Sets the progress bar in the SyncroSim User Interface*

Description

This function is designed to facilitate the development of R-based Syncrosim Packages, such as beginning, stepping, ending, and reporting the progress for a SyncroSim simulation.

Usage

```
progressBar(
  type = "step",
  iteration = NULL,
  timestep = NULL,
  totalSteps = NULL,
  message
)
```

Arguments

type	character. Update to apply to progress bar. Options include "begin", "end", "step", "report", and "message" (Default is "step")
iteration	integer. The current iteration. Only used if type = "report"
timestep	integer. The current timestep. Only used if type = "report"
totalSteps	integer. The total number of steps in the simulation. Only used if type = "begin"
message	character. An arbitrary messsage to be printed to the status bar. Only used if type = "message".

Value

No returned value, used for side effects

Examples

```
## Not run:
# Begin the progress bar for a simulation
progressBar(type = "begin", totalSteps = numIterations * numTimesteps)

# Increase the progress bar by one step for a simulation
progressBar(type = "step")

# Report progress for a simulation
progressBar(type = "report", iteration = iter, timestep = ts)

# Report arbitrary progress message
progressBar(type = "message", message = msg)

# End the progress bar for a simulation
progressBar(type = "end")

## End(Not run)
```

`project` *Create or open Project(s)*

Description

Creates or retrieves a `Project` or multiple Projects from a SsimLibrary.

Usage

```
project(
  ssimObject = NULL,
  project = NULL,
  sourceProject = NULL,
  summary = NULL,
  forceElements = FALSE,
  overwrite = FALSE
)
```

Arguments

`ssimObject`	`Scenario`, `SsimLibrary`, or `Chart` object, or a character string (i.e. a filepath)
`project`	`Project` object, character, integer, or vector of these. Names or ids of one or more Projects. Note that integer ids are slightly faster (optional)
`sourceProject`	`Project` object, character, or integer. If not `NULL` (default), new Projects will be copies of the sourceProject
`summary`	logical. If `TRUE` then return the Project(s) in a data.frame with the projectId, name, description, owner, dateModified, readOnly. Default is `TRUE` if `project=NULL` and SsimObject is not Scenario/Project, `FALSE` otherwise
`forceElements`	logical. If `TRUE` then returns a single Project as a named list; otherwise returns a single project as a `Project` object. Applies only when `summary=FALSE` Default is `FALSE`
`overwrite`	logical. If `TRUE` an existing Project will be overwritten. Default is `FALSE`

Details

For each element of project:

- If element identifies an existing Project: Returns the existing Project.
- If element identifies more than one Project: Error.
- If element does not identify an existing Project: Creates a new Project named element. Note that SyncroSim automatically assigns an id to a new Project.

Value

Returns a `Project` object representing a SyncroSim Project. If summary is `TRUE`, returns a data.frame of Project names and descriptions.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib_project")

# Set the SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(ssimObject = myLibrary, project = "My project name")
myproject2 <- project(ssimObject = myLibrary, project = "My new project name")

# Get a named list of existing Projects
# Each element in the list is named by a character version of the Project ID
myProjects <- project(myLibrary, summary = FALSE)
names(myProjects)

# Get an existing Project.
myProject <- myProjects[[1]]
myProject <- project(myLibrary, project = "My new project name")

# Get/set the Project properties
name(myProject)
name(myProject) <- "New project name"

# Create a new Project from a copy of an existing Project
myNewProject <- project(myLibrary, project = "My copied project",
                        sourceProject = 1)

# Overwrite an existing Project
myNewProject <- project(myLibrary, project = "My copied project",
                        overwrite = TRUE)

## End(Not run)
```

`Project-class` *SyncroSim Project class*

Description

`Project` object representing a SyncroSim Project. A Project is the intermediate level of organization in the SyncroSim workflow, between the `ssimLibrary` and the `scenario`. It contains information relevant to a group of Scenarios.

Slots

`session` `Session` object. The Session associated with the Project's SsimLibrary

`filepath` character string. The path to the Project's SsimLibrary on disk

`datasheetNames` Names and scopes of datasheets in the Project's Library

`projectId` integer. The Project id

See Also

See project for options when creating or loading a SyncroSim Project.

---

projectId *Retrieves projectId of SyncroSim Project, Scenario, Folder, or Chart*

Description

Retrieves the projectId of a SyncroSim Project, Scenario, Folder or Chart.

Usage

```
projectId(ssimObject)

## S4 method for signature 'character'
projectId(ssimObject)

## S4 method for signature 'Project'
projectId(ssimObject)

## S4 method for signature 'Scenario'
projectId(ssimObject)

## S4 method for signature 'Folder'
projectId(ssimObject)

## S4 method for signature 'Chart'
projectId(ssimObject)
```

Arguments

ssimObject	Scenario, Project, Folder, or Chart object

Value

An integer: project id.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")
```

```
# Get Project ID for SyncroSim Project and Scenario
projectId(myProject)
projectId(myScenario)

## End(Not run)
```

---

readOnly *Read-only status of a SsimLibrary, Project, Scenario, Folder or Chart*

Description

Retrieves or sets whether or not a SsimLibrary, Project, Scenario, Folder, or Chart is read-only.

Usage

```
readOnly(ssimObject)

## S4 method for signature 'character'
readOnly(ssimObject)

## S4 method for signature 'SsimLibrary'
readOnly(ssimObject)

## S4 method for signature 'Project'
readOnly(ssimObject)

## S4 method for signature 'Scenario'
readOnly(ssimObject)

## S4 method for signature 'Folder'
readOnly(ssimObject)

## S4 method for signature 'Chart'
readOnly(ssimObject)

readOnly(ssimObject) <- value

## S4 replacement method for signature 'character'
readOnly(ssimObject) <- value

## S4 replacement method for signature 'SsimObject'
readOnly(ssimObject) <- value

## S4 replacement method for signature 'Folder'
```

```
readOnly(ssimObject) <- value

## S4 replacement method for signature 'Chart'
readOnly(ssimObject) <- value
```

Arguments

ssimObject	Scenario, Project, SsimLibrary, or Folder object
value	logical. If TRUE the SsimObject will be read-only. Default is FALSE

Value

A logical: TRUE if the SsimObject is read-only and FALSE otherwise.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, Project, Scenario, and Folder
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")
myFolder <- folder(myProject, "My Folder")

# Retrieve the read-only status of a SsimObject
readOnly(myLibrary)
readOnly(myProject)
readOnly(myScenario)
readOnly(myFolder)

# Set the read-only status of a SsimObject
readOnly(myScenario) <- TRUE

## End(Not run)
```

---

removePackage *Removes SyncroSim package(s)*

Description

Removes package(s) from a SsimLibrary.

Usage

```
removePackage(ssimLibrary, packages)

## S4 method for signature 'character'
removePackage(ssimLibrary, packages)

## S4 method for signature 'SsimLibrary'
removePackage(ssimLibrary, packages)
```

Arguments

ssimLibrary	SsimLibrary object
packages	character string or vector of package name(s)

Value

This function invisibly returns TRUE upon success (i.e.successful removal of the package) or FALSE upon failure.

See Also

packages

Examples

```
## Not run:
# Install "stsim" and "stsimecodep" SyncroSim packages
installPackage(packages = c("stsim", "stsim"),
               versions = c("4.0.1", "4.3.5"))
installPackage("stsimecodep")

# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# Add package
addPackage(myLibrary, packages = "stsim", versions = "4.0.1")
addPackage(myLibrary, packages = "stsimecodep")
packages(myLibrary)

# Remove package
removePackage(myLibrary, packages = c("stsim", "stsimecodep"))
packages(myLibrary)

## End(Not run)
```

rsyncrosim *rsyncrosim: The R interface to SyncroSim: https://syncrosim.com/*

Description

rsyncrosim provides an interface to SyncroSim, a generalized framework for running and managing scenario-based stochastic simulations over space and time. Different kinds of simulation models can "plug-in" to SyncroSim as packages and take advantage of general features common to many kinds of simulation models, such as defining scenarios of inputs, running Monte Carlo simulations, and viewing charts and maps of outputs.

Details

To learn more about rsyncrosim, start with the vignette tutorial: `browseVignettes("rsyncrosim")`.

Author(s)

Maintainer: Katie Birchard <katie.birchard@apexrms.com>

Authors:

- Colin Daniel <colin.daniel@apexrms.com>
- Josie Hughes <josie.hughes@canada.ca>
- Valentin Lucet
- Alex Embrey
- Leonardo Frid
- Tabitha Kennedy
- Shreeram Senthivasan

Other contributors:

- ApexRMS [copyright holder]

See Also

Useful links:

- https://syncrosim.github.io/rsyncrosim/
- Report bugs at https://github.com/syncrosim/rsyncrosim/issues/

run *Run scenarios*

Description

Run one or more SyncroSim Scenario(s).

Usage

```
run(ssimObject, scenario = NULL, summary = FALSE, transformerName = NULL)

## S4 method for signature 'character'
run(ssimObject, scenario = NULL, summary = FALSE, transformerName = NULL)

## S4 method for signature 'list'
run(ssimObject, scenario = NULL, summary = FALSE, transformerName = NULL)

## S4 method for signature 'SsimObject'
run(ssimObject, scenario = NULL, summary = FALSE, transformerName = NULL)
```

Arguments

ssimObject	SsimLibrary, Project, or Scenario object, or a list of Scenarios, or character (i.e. path to a SsimLibrary on disk)
scenario	character, integer, or vector of these. Scenario names or ids. If NULL (default), then runs all Scenarios associated with the SsimObject. Note that integer ids are slightly faster
summary	logical. If FALSE (default) result Scenario objects are returned. If TRUE (faster) result Scenario ids are returned
transformerName	character. The name of the transformer to run (optional)

Details

Note that breakpoints are ignored unless the SsimObject is a single Scenario.

Value

If summary = FALSE, returns a result Scenario object or a named list of result Scenarios. The name is the parent Scenario for each result. If summary = TRUE, returns summary info for result Scenarios.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- "testlib"
```

```
# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
myLibrary <- ssimLibrary(name = myLibraryName,
                         packages = "helloworldSpatial")
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")
myScenario2 <- scenario(myProject, scenario = "My Scenario 2")

# Run with default parameters
resultScenario <- run(myScenario)

# Only return summary information
resultScenarioSummary <- run(myScenario, summary = TRUE)

# Run 2 scenarios at once
resultScenarios <- run(c(myScenario, myScenario2))

## End(Not run)
```

---

runLog *Retrieves run log of result Scenario*

Description

Retrieves the run log of a result Scenario.

Usage

```
runLog(scenario)

## S4 method for signature 'character'
runLog(scenario)

## S4 method for signature 'Scenario'
runLog(scenario)
```

Arguments

scenario	Scenario object.

Value

A character string: the run log for a result scenario.

Examples

```
## Not run:
# Set the file path and name of an existing SsimLibrary
myLibraryName <- file.path("MyLibrary.ssim")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Run Scenario
resultScenario <- run(myScenario)

# Retrieve the run log of the result Scenario
runLog(resultScenario)

## End(Not run)
```

`runtimeDataFolder` *SyncroSim Data Folder*

Description

This function is part of a set of functions designed to facilitate the development of R-based Syncrosim Packages. This function creates and returns a SyncroSim Data Folder.

Usage

```
runtimeDataFolder(scenario, datasheetName)
```

Arguments

`scenario`	`Scenario` object. A SyncroSim result Scenario
`datasheetName`	character. The datasheet name

Value

Returns a data folder name for the specified datasheet.

Examples

```
## Not run:
dataFolder <- runtimeDataFolder()

## End(Not run)
```

runtimeTempFolder *SyncroSim Temporary Folder*

Description

This function is part of a set of functions designed to facilitate the development of R-based Syncrosim Packages. This function creates and returns a SyncroSim Temporary Folder.

Usage

```
runtimeTempFolder(folderName)
```

Arguments

folderName	character. The folder name

Value

Returns a temporary folder name.

Examples

```
## Not run:
tempFolder <- runtimeTempFolder()

## End(Not run)
```

saveDatasheet *Save Datasheet(s)*

Description

Saves Datasheets to a SsimLibrary, Project, or Scenario.

Usage

```
saveDatasheet(
  ssimObject,
  data,
  name = NULL,
  fileData = NULL,
  append = NULL,
  forceElements = FALSE,
  force = FALSE,
  breakpoint = FALSE,
```

```
  import = TRUE,
  path = NULL
)

## S4 method for signature 'character'
saveDatasheet(
  ssimObject,
  data,
  name = NULL,
  fileData = NULL,
  append = NULL,
  forceElements = FALSE,
  force = FALSE,
  breakpoint = FALSE,
  import = TRUE,
  path = NULL
)

## S4 method for signature 'SsimObject'
saveDatasheet(
  ssimObject,
  data,
  name = NULL,
  fileData = NULL,
  append = NULL,
  forceElements = FALSE,
  force = FALSE,
  breakpoint = FALSE,
  import = TRUE,
  path = NULL
)
```

Arguments

ssimObject	SsimLibrary, Project, or Scenario object
data	data.frame, named vector, or list of these. One or more Datasheets to load
name	character or vector of these. The name(s) of the Datasheet(s) to be saved. If a vector of names is provided, then a list must be provided for the data argument. Names provided here will override those provided with data argument's list
fileData	named list or SpatRaster object. Names are file names (without paths), corresponding to entries in data The elements are objects containing the data associated with each name. Currently supports terra SpatRaster objects as elements, (support for Raster objects is deprecated)
append	logical. If TRUE, the incoming data will be appended to the Datasheet if possible. Default is TRUE for Project/SsimLibrary-scope Datasheets, and FALSE for Scenario-scope Datasheets. See 'details' for more information about this argument

forceElements	logical. If FALSE (default) a single return message will be returned as a character string. Otherwise it will be returned in a list
force	logical. If Datasheet scope is Project/SsimLibrary, and append=FALSE, Datasheet will be deleted before loading the new data. This can also delete other definitions and results, so if force=FALSE (default) user will be prompted for approval
breakpoint	logical. Set to TRUE when modifying Datasheets in a breakpoint function. Default is FALSE
import	logical. Set to TRUE to import the data after saving. Default is FALSE
path	character. output path (optional)

Details

SsimObject/Project/Scenario should identify a single SsimObject.

If fileData != NULL, each element of names(fileData) should correspond uniquely to at most one entry in data. If a name is not found in data the element will be ignored with a warning. If names(fileData) are full filepaths, rsyncrosim will write each object to the corresponding path for subsequent loading by SyncroSim. Note this is generally more time-consuming because the files must be written twice. If names(fileData) are not filepaths (faster, recommended), rsyncrosim will write each element directly to the appropriate SyncroSim input/output folders. rsyncrosim will write each element of fileData directly to the appropriate SyncroSim input/output folders. If fileData != NULL, data should be a data.frame, vector, or list of length 1, not a list of length >1.

About the 'append' argument:

- A Datasheet is a VALIDATION SOURCE if its data can be used to validate column values in a different Datasheet.
- The append argument will be ignored if the Datasheet is a validation source and has a Project scope. In this case the data will be MERGED.

Value

Invisibly returns a vector or list of logical values for each input: TRUE upon success (i.e.successful save) and FALSE upon failure.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession,
                         packages = "helloworldSpatial")
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Get all Datasheet info
```

```
myDatasheets <- datasheet(myScenario)

# Get a specific Datasheet
myDatasheet <- datasheet(myScenario, name = "helloworldSpatial_RunControl")

# Modify Datasheet
myDatasheet$MaximumTimestep <- 10

# Save Datasheet
saveDatasheet(ssimObject = myScenario,
              data = myDatasheet,
              name = "helloworldSpatial_RunControl")

# Import data after saving
saveDatasheet(ssimObject = myScenario,
              data = myDatasheet,
              name = "helloworldSpatial_RunControl",
              import = TRUE)

# Save the new Datasheet to a specified output path
saveDatasheet(ssimObject = myScenario,
              data = myDatasheet,
              name = "helloworldSpatial_RunControl",
              path = tempdir())

# Save a raster stack using fileData
# Create a raster stack - add as many raster files as you want here
map1 <- datasheetSpatRaster(myScenario,
                            datasheet = "helloworldSpatial_InputDatasheet",
                            column = "InterceptRasterFile")
inRasters <- terra::rast(map1)

# Change the name of the rasters in the input Datasheets to match the stack
inSheet <- datasheet(myScenario, name = "helloworldSpatial_InputDatasheet")
inSheet[1,"InterceptRasterFile"] <- names(inRasters)[1]

# Save the raster stack to the input Datasheet
saveDatasheet(myScenario, data = inSheet,
              name = "helloworldSpatial_InputDatasheet",
              fileData = inRasters)

## End(Not run)
```

---

scenario *Create or open Scenario(s)*

Description

Create or open one or more Scenarios from a SsimLibrary.

Usage

```
scenario(
  ssimObject = NULL,
  scenario = NULL,
  sourceScenario = NULL,
  folder = NULL,
  summary = NULL,
  results = FALSE,
  forceElements = FALSE,
  overwrite = FALSE
)
```

Arguments

ssimObject	SsimLibrary or Project object, or character (i.e. a filepath)
scenario	character, integer, or vector of these. Names or ids of one or more Scenarios. Note integer ids are slightly faster, but can only be used to open existing Scenarios
sourceScenario	character or integer. If not NULL (Default), new Scenarios will be copies of the sourceScenario
folder	Folder object, character, or integer. The Folder object, name (must be unique), or Folder ID. If not NULL (Default), new Scenarios will be moved into the specified folder
summary	logical. If TRUE then loads and returns the Scenario(s) in a named vector/dataframe with the scenarioId, name, description, owner, dateModified, readOnly, parentID. Default is TRUE if scenario=NULL, FALSE otherwise
results	logical. If TRUE only return result Scenarios. Default is FALSE
forceElements	logical. If TRUE then returns a single Scenario as a named list; if FALSE (default), returns a single Scenario as a Scenario object. Applies only when summary=FALSE
overwrite	logical. If TRUE an existing Scenario will be overwritten. Default is FALSE

Details

For each element of Scenario:

- If element/Project/SsimObject uniquely identifies an existing Scenario: Returns the existing Scenario.
- If element/Project/SsimObject uniquely identifies more than one existing Scenario: Error.
- If element/Project/SsimObject do not identify an existing Scenario or Project: Error.
- If element/Project/SsimObject do not identify an existing Scenario and element is numeric: Error - a name is required for new Scenarios. SyncroSim will automatically assign an id when a Scenario is created.
- If element/Project/SsimObject do not identify an existing Scenario and do identify a Project, and element is a character string: Creates a new Scenario named element in the Project. SyncroSim automatically assigns an id. If sourceScenario is not NULL the new Scenario will be a copy of sourceScenario.

Value

A Scenario object representing a SyncroSim scenario, a list of Scenario objects, or a data frame of Scenario names and descriptions. If summary = FALSE, returns one or more Scenario objects representing SyncroSim Scenarios. If summary = TRUE, returns Scenario summary info.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "My Project")

# Create a new Scenario
myScenario <- scenario(myProject, scenario = "My Scenario")


# Create a new Scenario from an existing Scenario
myScenarioCopy <- scenario(myProject, scenario = "My Scenario Copy",
                           sourceScenario = myScenario)

# Find all the Scenarios in a SsimLibrary
scenario(myLibrary)

# Only return the results Scenarios for a SsimLibrary
scenario(myLibrary, results = TRUE)

# Overwrite an existing Scenario
myNewScenario <- scenario(myProject, scenario = "My New Scenario",
                          overwrite = TRUE)

## End(Not run)
```

Scenario-class *SyncroSim Scenario class*

Description

Scenario object representing a SyncroSim Scenario. A Scenario is the lowest level of organization in the SyncroSim workflow, and is often used to isolate information on a single Datasheet.

Slots

session Session object. The Session associated with the Scenario

filepath character string. The path to the Scenario's SsimLibrary on disk

`datasheetNames` character string. Names and scope of all Datasheets in Scenario's SsimLibrary

`projectId` integer. The Project id

`scenarioId` integer. The Scenario id

`parentId` integer. For a result Scenario, this is the id of the parent Scenario. 0 indicates this is not a result Scenario

`folderId` integer. The folder in which the Scenario exists. If the Scenario exists at the root of the project, then this value is NULL.

See Also

See `scenario` for options when creating or loading a SyncroSim Scenario.

`scenarioId` *Retrieves scenarioId of Scenario*

Description

Retrieves the scenarioId of a `Scenario`.

Usage

```
scenarioId(scenario)

## S4 method for signature 'character'
scenarioId(scenario)

## S4 method for signature 'Scenario'
scenarioId(scenario)
```

Arguments

`scenario` `Scenario` object

Value

Integer id of the input Scenario.

Examples

```
## Not run:
# Set the file path and name of the new SsimLibrary
myLibraryName <- file.path(tempdir(),"testlib")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")
```

```
myScenario <- scenario(myProject, scenario = "My Scenario")

# Get Scenario ID of Scenario
scenarioId(myScenario)

## End(Not run)
```

`session` *Create or return SyncroSim Session*

Description

Methods to create or return a SyncroSim `Session`.

Usage

```
session(x = NULL, silent = TRUE, printCmd = FALSE)

## S4 method for signature 'missingOrNULLOrChar'
session(x = NULL, silent = TRUE, printCmd = FALSE)

## S4 method for signature 'SsimObject'
session(x = NULL, silent = TRUE, printCmd = FALSE)

## S4 method for signature 'Folder'
session(x = NULL, silent = TRUE, printCmd = FALSE)

session(ssimObject) <- value

## S4 replacement method for signature 'NULLOrChar'
session(ssimObject) <- value

## S4 replacement method for signature 'SsimObject'
session(ssimObject) <- value
```

Arguments

`x`	character or SsimObject. Path to SyncroSim installation. If `NULL` (default), then default path is used
`silent`	logical. Applies only if x is a path or `NULL` If `TRUE`, warnings from the console are ignored. Otherwise they are printed. Default is `FALSE`
`printCmd`	logical. Applies only if x is a path or `NULL` If `TRUE`, arguments passed to the SyncroSim console are also printed. Helpful for debugging. Default is `FALSE`
`ssimObject`	`Project` or `Scenario` object
`value`	`Session` object

Details

In order to avoid problems with SyncroSim version compatibility and SsimLibrary updating, the new Session must have the same filepath as the Session of the SsimObject e.g. `filepath(value)==filepath(session(ssim`
Therefore, the only time when you will need to set a new SyncroSim Session is if you have updated the SyncroSim software and want to update an existing SsimObject to use the new software.

Value

A SyncroSim `Session` object.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session, SsimLibrary, and Project
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)
myProject <- project(myLibrary, project = "Definitions")

# Lists the folder location of SyncroSim Session
filepath(mySession)

# Lists the version of SyncroSim Session
version(mySession)

# Data frame of the packages installed with this version of SyncroSim
packages(mySession)

# Set a new SyncroSim Session for the SyncroSim Project
session(myProject) <- session(x = filepath(session(myProject)))

## End(Not run)
```

Session-class *SyncroSim Session class*

Description

A SyncroSim Session object contains a link to a SyncroSim installation. `SsimLibrary`, `Project` and `Scenario` objects contain a `Session` used to query and modify the object.

Slots

`filepath` The path to the SyncroSim installation

`silent` If `FALSE`, all SyncroSim output with non-zero exit status is printed. Helpful for debugging. Default is `TRUE`

printCmd If TRUE, arguments passed to the SyncroSim console are also printed. Helpful for debugging. Default is FALSE

condaFilepath The path to the Conda installation. Default is "default"

See Also

See session for options when creating a Session.

---

signIn *Signs in to SyncroSim*

Description

Signs in to syncrosim.com to authenticate user credentials.

Usage

```
signIn(session = NULL)
```

Arguments

session Session object. If NULL(default), the default session will be used

Value

Character string: whether sign in was successful or not.

Examples

```
## Not run:
# Sign in to SyncroSim session
mySession <- session()
signIn(mySession)

## End(Not run)
```

signOut *Signs out of SyncroSim*

Description

Signs out of syncrosim.com.

Usage

```
signOut(session = NULL)
```

Arguments

session	Session object. If NULL(default), the default session will be used

Value

Character string: whether sign out was successful or not.

Examples

```
## Not run:
# Sign out of SyncroSim session
mySession <- session()
signOut(mySession)

## End(Not run)
```

silent *Silent status of SyncroSim Session*

Description

Checks or sets whether a SyncroSim Session is silent or not. In a silent session, warnings from the console are ignored.

Usage

```
silent(session)

## S4 method for signature 'Session'
silent(session)

## S4 method for signature 'missingOrNULLOrChar'
silent(session)
```

```
silent(session) <- value

## S4 replacement method for signature 'character'
silent(session) <- value

## S4 replacement method for signature 'Session'
silent(session) <- value
```

Arguments

session	Session object or character (i.e. filepath to a session). If NULL, session() will be used
value	logical. If TRUE (default), the SyncroSim Session will be silent

Value

A logical: TRUE if the session is silent and FALSE otherwise.

Examples

```
## Not run:
# Set up a SyncroSim Session
mySession <- session()

# Check the silent status of a SyncroSim Session
silent(mySession)

# Set the silent status of a SyncroSim Session
silent(mySession) <- FALSE

## End(Not run)
```

---

sqlStatement *Construct an SQLite query*

Description

Creates SELECT, GROUP BY and WHERE SQL statements. The resulting list of SQL statements will be converted to an SQLite database query by the datasheet function.

Usage

```
sqlStatement(
  groupBy = NULL,
  aggregate = NULL,
  aggregateFunction = "SUM",
  where = NULL
)
```

Arguments

groupBy	character string or vector of these. Vector of variables (column names) to GROUP BY (optional)
aggregate	character string of vector of these. Vector of variables (column names) to aggregate using aggregateFunction (optional)
aggregateFunction	character string. An SQL aggregate function (e.g. SUM, COUNT). Default is SUM
where	named list. A list of subset variables. Names are column names, and elements are the values to be selected from each column (optional)

Details

Variables are column names of the Datasheet. See column names using datasheet(,empty=TRUE) Variables not included in groupBy, aggregate or where will be dropped from the table. Note that it is not possible to construct a complete SQL query at this stage, because the datasheet function may add ScenarioId and/or ProjectId to the query.

Value

Returns a list of SELECT, GROUP BY and WHERE SQL statements used by the datasheet function to construct an SQLite database query.

Examples

```
## Not run:
# Query total Amount for each combination of ScenarioId, Iteration, Timestep and StateLabelXID,
# including only Timesteps 0,1 and 2, and Iterations 3 and 4.
mySQL <- sqlStatement(
  groupBy = c("ScenarioId", "Iteration", "Timestep"),
  aggregate = c("yCum"),
  aggregateFunction = "SUM",
  where = list(Timestep = c(0, 1, 2), Iteration = c(3, 4))
)
mySQL

## End(Not run)
## Not run:
# The SQL statement can then be used in the datasheet function

# Set the file path and name of an existing SsimLibrary
myLibraryName <- file.path("MyLibrary.ssim")

# Set the SyncroSim Session, SsimLibrary, Project, and Scenario
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName,
                         session = mySession)
myProject <- project(myLibrary, project = "Definitions")
myScenario <- scenario(myProject, scenario = "My Scenario")

# Run Scenario to generate results
```

```
resultScenario <- run(myScenario)

# Use the SQL statement when loading the Datasheet
myAggregatedDataFrame <- datasheet(resultScenario,
                                   name = "helloworldSpatial_OutputDatasheet",
                                   sqlStatement = mySQL)

# View aggregated DataFrame
myAggregatedDataFrame

## End(Not run)
```

`ssimEnvironment` *SyncroSim Environment*

Description

This function is part of a set of functions designed to facilitate the development of R-based Syncrosim Packages. `ssimEnvironment` retrieves specific environment variables.

Usage

```
ssimEnvironment()
```

Value

Returns a single-row data.frame of SyncroSim specific environment variables.

Examples

```
## Not run:
# Get the whole set of variables
e <- ssimEnvironment()

# Get the path to transfer directory, for instance
transferdir <- e$TransferDirectory

## End(Not run)
```

ssimLibrary *Create or open a SsimLibrary*

Description

Creates or opens a SsimLibrary object. If summary = TRUE, returns SsimLibrary summary info. If summary = NULL, returns SsimLibrary summary info if ssimObject is a SsimLibrary, SsimLibrary object otherwise.

Usage

```
ssimLibrary(
  name = NULL,
  summary = NULL,
  packages = NULL,
  session = NULL,
  forceUpdate = FALSE,
  overwrite = FALSE,
  useConda = NULL
)

## S4 method for signature 'SsimObject'
ssimLibrary(
  name = NULL,
  summary = NULL,
  packages = NULL,
  session = NULL,
  forceUpdate = FALSE,
  overwrite = FALSE,
  useConda = NULL
)

## S4 method for signature 'missingOrNULLOrChar'
ssimLibrary(
  name = NULL,
  summary = NULL,
  packages = NULL,
  session = NULL,
  forceUpdate = FALSE,
  overwrite = FALSE,
  useConda = NULL
)
```

Arguments

name	SsimLibrary, Project or Scenario object, or character string (i.e. path to a SsimLibrary or SsimObject)

summary	logical. Default is TRUE
packages	character or character vector. The SyncroSim Package(s) to add to the Library if creating a new Library (optional)
session	Session object. If NULL (default), session() will be used
forceUpdate	logical. If FALSE (default) user will be prompted to approve any required updates. If TRUE, required updates will be applied silently.
overwrite	logical. If TRUE an existing SsimLibrary will be overwritten
useConda	logical. If set to TRUE, then all packages associated with the Library will have their Conda environments created and Conda environments will be used during runtime.If set to FALSE, then no packages will have their Conda environments created and Conda environments will not be used during runtime. Default is NULL

Details

Example arguments:

- If name is SyncroSim Project or Scenario: Returns the SsimLibrary associated with the Project or Scenario.
- If name is NULL: Create/open a SsimLibrary in the current working directory with the filename SsimLibrary.ssim.
- If name is a string: If string is not a valid path treat as filename in working directory. If no file suffix provided in string then add .ssim. Attempts to open a SsimLibrary of that name. If SsimLibrary does not exist creates a SsimLibrary of type package in the current working directory.
- If given a name and a package: Create/open a SsimLibrary called name.ssim. Returns an error if the SsimLibrary already exists but is a different type of package.

Value

Returns a SsimLibrary object.

Examples

```
## Not run:
# Make sure packages are installed
installPackage("stsim")

# Create or open a SsimLibrary using the default Session
myLibrary <- ssimLibrary(name = file.path(tempdir(), "mylib"))

# Create SsimLibrary using a specific Session
mySession <- session()

myLibrary <- ssimLibrary(name = file.path(tempdir(), "mylib"),
                         session = mySession)

# Retrieve SsimLibrary properties
```

```
session(myLibrary)

# Create SsimLibrary from template
installPackage("helloworldSpatial")
mySession <- session()
myLibrary <- ssimLibrary(name = file.path(tempdir(), "mylib"),
                         session = mySession,
                         forceUpdate = TRUE,
                         packages = "helloworldSpatial",
                         overwrite = TRUE)


## End(Not run)
```

`SsimLibrary-class` *SyncroSim Library class*

Description

`SsimLibrary` object representing a SyncroSim Library. A SsimLibrary is the highest level of organization in the SyncroSim workflow and contains at least one `Project`.

Slots

`session` `Session` object

`filepath` character string. The path to the SsimLibrary on disk

`datasheetNames` character string. The name and scope of all Datasheets in the SsimLibrary.

See Also

See `ssimLibrary` for options when creating or loading a SyncroSim SsimLibrary.

`tempfilepath` *Retrieves the temporary file path to a SyncroSim object on disk*

Description

Retrieves the temporary file path to a SyncroSim `Session`, `SsimLibrary`, `Project` or `Scenario` on disk.

Usage

```
tempfilepath(ssimObject)

## S4 method for signature 'character'
tempfilepath(ssimObject)

## S4 method for signature 'Session'
tempfilepath(ssimObject)

## S4 method for signature 'SsimObject'
tempfilepath(ssimObject)
```

Arguments

ssimObject	Session, Project, or SsimLibrary object

Value

A character string: the temporary file path to a SyncroSim object on disk.

Examples

```
## Not run:
# Specify file path and name of new SsimLibrary
myLibraryName <- file.path(tempdir(), "testlib")

# Set up a SyncroSim Session and SsimLibrary
mySession <- session()
myLibrary <- ssimLibrary(name = myLibraryName, session = mySession)

# Get the temporary file path
myFilePath <- tempfilepath(myLibrary)

## End(Not run)
```

---

uninstallPackage *Removes a package from SyncroSim installation*

Description

Removes a package from SyncroSim installation

Usage

```
uninstallPackage(packages, versions = NULL, session = NULL)

## S4 method for signature 'ANY,ANY,character'
uninstallPackage(packages, versions = NULL, session = NULL)

## S4 method for signature 'ANY,ANY,missingOrNULL'
uninstallPackage(packages, versions = NULL, session = NULL)

## S4 method for signature 'ANY,ANY,Session'
uninstallPackage(packages, versions = NULL, session = NULL)
```

Arguments

packages	character or character vector. The name(s) of the package(s) to uninstall
versions	character or character vector. The version(s) of the package(s) to uninstall. If NULL then will uninstall all versions of the package(s).
session	Session object. If NULL (default), session() will be used

Value

Invisibly returns TRUE upon success (i.e.successful removal) and FALSE upon failure.

Examples

```
## Not run:
# Set SyncroSim session
mySession <- session()

# Install packages to SyncroSim session
installPackages(packages = c("stsim", "stsim"),
                versions = c("4.0.1", "4.3.5"))

# Uninstalls specific version of package from SyncroSim session
uninstallPackage(packages = "stsim", versions = "4.0.1", session = mySession)

# Uninstalls all instances ofa package from SyncroSim session
uninstallPackage(packages = "stsim", session = mySession)

## End(Not run)
```

---

updateRunLog *Function to write to the SyncroSim run log*

Description

This function is designed to facilitate the development of R-based Syncrosim Packages by allowing developers to send messages to the run log.

Usage

```
updateRunLog(..., sep = "", type = "status")
```

Arguments

...	One or more objects which can be coerced to character which are pasted together using sep.
sep	character. Used to separate terms. Not NA_character_
type	character. Type of message to add to run log. One of "status", (default) "info", or "warning".

Value

No returned value, used for side effects

Examples

```
## Not run:
# Write a message to run log
updateRunLog(msg)

# Construct and write a message to run log
updateRunLog(msg, additionalMsg, sep = " ")

## End(Not run)
```

---

useConda *Conda configuration of a SsimLibrary*

Description

Retrieves or sets the Conda configuration of a SsimLibrary. Note that in order to use conda environments, you will first need to ensure that the conda environment has been created for a given package. You can create the conda environment for a package using the createCondaEnv function.

Usage

```
useConda(ssimObject)

## S4 method for signature 'character'
useConda(ssimObject)

## S4 method for signature 'SsimLibrary'
useConda(ssimObject)

useConda(ssimObject) <- value
```

```
## S4 replacement method for signature 'logical'
useConda(ssimObject) <- value

## S4 replacement method for signature 'SsimLibrary'
useConda(ssimObject) <- value
```

Arguments

ssimObject	SsimLibrary object
value	logical for whether to use Conda environments for the given SyncroSim Library. If set to TRUE, then Conda environments will be used. If set to FALSE, then Conda environments will not be used during runtime.

Value

Logical: whether Conda environments will be used during runtime for the given SsimLibrary

Examples

```
## Not run:
# Set up a SyncroSim Session, SsimLibrary
mySession <- session()

# Retrieve Conda configuration status of the SsimLibrary
useConda(myLibrary)

# Set the Conda configuration of the SyncroSim Library
useConda(myLibrary) <- TRUE

# Only use Conda with the specified SyncroSim packages
useConda(myLibrary) <- "helloworld"

# Only use Conda with multiple specified SyncroSim packages
useConda(myLibrary) <- c("helloworld", "stsim")

## End(Not run)
```

---

version *Retrieves SyncroSim version*

Description

Retrieves the version of a SyncroSim Session.

Usage

```
version(session = NULL)

## S4 method for signature 'character'
version(session = NULL)

## S4 method for signature 'missingOrNULL'
version(session = NULL)

## S4 method for signature 'Session'
version(session = NULL)
```

Arguments

session	Session object

Value

A character string e.g. "2.2.13".

Examples

```
## Not run:
# Set SyncroSim Session
mySession <- session()

# Retrieve version of SyncroSim associated with Session
version(mySession)

## End(Not run)
```

---

viewProfile *Display SyncroSim profile*

Description

Displays the currently signed in SyncroSim profile information. To sign in to SyncroSim use the signIn function.

Usage

```
viewProfile(session = NULL, ...)
```

Arguments

session	Session object. If NULL(default), the default session will be used
...	other internal parameters

Examples

```
## Not run:
# Retrieve profile information for a SyncroSim session
mySession <- session()
viewProfile(mySession)

## End(Not run)
```

Index