Package 'SpaDES.core'

August 30, 2017

Type Package

Title Core Utilities for Developing and Running Spatially Explicit Discrete Event Simulation Models

Description Provide the core discrete event simulation (DES) framework for implementing spatially explicit simulation models. The core DES components facilitate modularity, and easily enable the user to include additional functionality by running user-built simulation modules.

URL https://predictiveecology.github.io/SpaDES.core/, https://github.com/PredictiveEcology/SpaDES.core

Date 2017-08-28

Version 0.1.0

Depends R (>= 3.3.2), quickPlot, reproducible

Imports chron (>= 2.3-50), CircStats (>= 0.2-4), data.table (>= 1.10.4), DEoptim (>= 2.2-4), DiagrammeR (>= 0.8.2), digest (>= 0.6.12), dplyr (>= 0.5.0), fastdigest, fpCompare (>= 0.2.1), grDevices, httr (>= 1.2.1), igraph (>= 1.0.1), lazyeval (>= 0.2.0), lubridate (>= 1.3.3), methods, parallel, R.utils (>= 2.5.0), RandomFields (>= 3.1.24), raster (>= 2.5-8), stats, sp (>= 1.2-4), stringi (>= 1.1.3), tcltk, utils

Suggests covr, hunspell, knitr, Matrix, png, RColorBrewer (>= 1.1-2), rgdal, rgenoud, rmarkdown, SpaDES.tools, testthat (>= 1.0.2), tkrplot

License GPL-3

VignetteBuilder knitr, rmarkdown

BugReports https://github.com/PredictiveEcology/SpaDES.core/issues

ByteCompile yes

Collate 'environment.R' 'priority.R' 'module-dependencies-class.R' 'misc-methods.R' 'helpers.R' 'simList-class.R' 'POM.R' 'cache.R' 'check.R' 'checkpoint.R' 'copy.R' 'experiment.R' 'simulation-parseModule.R' 'simulation-simInit.R' 'load.R' 'module-define.R' 'module-dependencies-methods.R'

'moduleMetadata.R' 'module-repository.R' 'module-template.R'
'moduleCoverage.R' 'times.R' 'simList-accessors.R'
'plotting-diagrams.R' 'plotting.R' 'probability.R' 'progress.R'
'save.R' 'simulation-spades.R' 'spades-classes.R'
'spades-core-package.R' 'zzz.R'

RoxygenNote 6.0.1

NeedsCompilation no

Author Alex M Chubaty [aut, cre],
Eliot J B McIntire [aut],
Yong Luo [ctb],
Steve Cumming [ctb],
Her Majesty the Queen in Right of Canada, as represented by the
Minister of Natural Resources Canada [cph]

Maintainer Alex M Chubaty <alexander.chubaty@canada.ca>

Repository CRAN

Date/Publication 2017-08-30 14:18:43 UTC

R topics documented:

`SpaDES.core-package` *Categorized overview of the* `SpaDES.core` *package*

Description

This package allows implementation a variety of simulation-type models, with a focus on spatially explicit models. The core simulation components are built upon a discrete event simulation framework that facilitates modularity, and easily enables the user to include additional functionality by running user-built simulation modules. Included are numerous tools to visualize various spatial data formats, as well as non-spatial data. Much work has been done to speed up the core of the DES, with current benchmarking as low as 700 microseconds overhead for each event (including queuing, sorting, spawning event etc.).

Bug reports: `https://github.com/PredictiveEcology/SpaDES.core/issues`

Module repository: `https://github.com/PredictiveEcology/SpaDES-modules`

Wiki: `https://github.com/PredictiveEcology/SpaDES/wiki`

---

1 Spatial discrete event simulation (`SpaDES`)

A collection of top-level functions for doing spatial discrete event simulation.

1.1 Simulations: There are two workhorse functions that initialize and run a simulation, and third function for doing multiple spades runs:

`simInit`	Initialize a new simulation
`spades`	Run a discrete event simulation
`experiment`	Run multiple `spades` calls

1.2 Events: Within a module, important simulation functions include:

`scheduleEvent`	Schedule a simulation event
`removeEvent`	Remove an event from the simulation queue (not yet implemented)

2 The simList object class

The principle exported object class is the simList. All SpaDES simulations operate on this object class.

simList	The 'simList' class

---

3 simList methods

Collections of commonly used functions to retrieve or set slots (and their elements) of a simList object are summarized further below.

3.1 Simulation parameters:

globals	List of global simulation parameters.
params	Nested list of all simulation parameter.
P	Namespaced version of params (i.e., do not have to specify module name).

3.2 loading from disk, saving to disk:

inputs	List of loaded objects used in simulation. (advanced)
outputs	List of objects to save during simulation. (advanced)

3.3 objects in the simList:

ls, objects	Names of objects referenced by the simulation environment.
ls.str	List the structure of the simList objects.
objs	List of objects referenced by the simulation environment.

3.4 Simulation paths:

Accessor functions for the paths slot and its elements.

cachePath	Global simulation cache path.
modulePath	Global simulation module path.
inputPath	Global simulation input path.
outputPath	Global simulation output path.
paths	Global simulation paths (cache, modules, inputs, outputs).

3.5 Simulation times:

Accessor functions for the simtimes slot and its elements.

time	Current simulation time, in units of longest module.
start	Simulation start time, in units of longest module.

`end`	Simulation end time, in units of longest module.
`times`	List of all simulation times (current, start, end), in units of longest module..

3.6 Simulation event queues: Accessor functions for the `events` and `completed` slots. By default, the event lists are shown when the `simList` object is printed, thus most users will not require direct use of these methods.

`events`	Scheduled simulation events (the event queue). (advanced)
`current`	Currently executing event. (advanced)
`completed`	Completed simulation events. (advanced)

3.7 Modules and dependencies: Accessor functions for the `depends`, `modules`, and `.loadOrder` slots. These are included for advanced users.

`depends`	List of simulation module dependencies. (advanced)
`modules`	List of simulation modules to be loaded. (advanced)

3.8 `simList` environment: The `simList` has a slot called `.envir` which is an environment. All objects in the `simList` are actually in this environment, i.e., the `simList` is not a `list`. In R, environments use pass-by-reference semantics, which means that copying a `simList` object using normal R assignment operation (e.g., `sim2 <- sim1`), will not copy the objects contained within the `.envir` slot. The two objects (`sim1` and `sim2`) will share identical objects within that slot. Sometimes, this not desired, and a true copy is required.

`envir`	Access the environment of the simList directly (advanced)
`copy`	Deep copy of a simList. (advanced)

3.9 Checkpointing:

Accessor method	Module	Description
`checkpointFile`	`.checkpoint`	Name of the checkpoint file. (advanced)
`checkpointInterval`	`.checkpoint`	The simulation checkpoint interval. (advanced)

3.10 Progress Bar:

`progressType`	`.progress`	Type of graphical progress bar used. (advanced)
`progressInterval`	`.progress`	Interval for the progress bar. (advanced)

4 Module operations

4.1 Creating, distributing, and downloading modules: Modules are the basic unit of SpaDES. These are generally created and stored locally, or are downloaded from remote repositories, including our SpaDES-modules repository on GitHub.

checksums	Verify (and optionally write) checksums for a module's data files.
downloadModule	Open all modules nested within a base directory.
getModuleVersion	Get the latest module version # from module repository.
newModule	Create new module from template.
newModuleDocumentation	Create empty documentation for a new module.
openModules	Open all modules nested within a base directory.
moduleMetadata	Shows the module metadata.
zipModule	Zip a module and its associated files.

4.2 Module metadata: Each module requires several items to be defined. These comprise the metadata for that module (including default parameter specifications, inputs and outputs), and are currently written at the top of the module's .R file.

defineModule	Define the module metadata
defineParameter	Specify a parameter's name, value and set a default
expectsInput	Specify an input object's name, class, description, sourceURL and other specifications
createsOutput	Specify an output object's name, class, description and other specifications

4.3 Module dependencies: Once a set of modules have been chosen, the dependency information is automatically calculated once simInit is run. There are several functions to assist with dependency information:

depsEdgeList	Build edge list for module dependency graph
depsGraph	Build a module dependency graph using igraph

---

5 Module functions

A collection of functions that help with making modules can be found in the suggested SpaDES.tools *package, and are summarized below.*

5.1 Spatial spreading/distances methods: Spatial contagion is a key phenomenon for spatially explicit simulation models. Contagion can be modelled using discrete approaches or continuous approaches. Several SpaDES.tools functions assist with these:

adj	An optimized (i.e., faster) version of adjacent
cir	Identify pixels in a circle around a SpatialPoints* object
directionFromEachPoint	Fast calculation of direction and distance surfaces
distanceFromEachPoint	Fast calculation of distance surfaces
rings	Identify rings around focal cells (e.g., buffers and donuts)

spokes	Identify outward radiating spokes from initial points
spread	Contagious cellular automata
wrap	Create a torus from a grid

5.2 Spatial agent methods: Agents have several methods and functions specific to them:

crw	Simple correlated random walk function
heading	Determines the heading between SpatialPoints*
makeLines	Makes SpatialLines object for, e.g., drawing arrows
move	A meta function that can currently only take "crw"
specificNumPerPatch	Initiate a specific number of agents per patch

5.3 GIS operations: In addition to the vast amount of GIS operations available in R (mostly from contributed packages such as sp, raster, maps, maptools and many others), we provide the following GIS-related functions:

equalExtent	Assess whether a list of extents are all equal

5.4 'Map-reduce'–type operations: These functions convert between reduced and mapped representations of the same data. This allows compact representation of, e.g., rasters that have many individual pixels that share identical information.

rasterizeReduced	Convert reduced representation to full raster.

5.5 Colors in Raster* objects: We likely will not want the default colours for every map. Here are several helper functions to add to, set and get colors of Raster* objects:

setColors	Set colours for plotting Raster* objects
getColors	Get colours in a Raster* objects
divergentColors	Create a color palette with diverging colors around a middle

5.6 Random Map Generation: It is often useful to build dummy maps with which to build simulation models before all data are available. These dummy maps can later be replaced with actual data maps.

gaussMap	Creates a random map using Gaussian random fields
randomPolygons	Creates a random polygon with specified number of classes

5.7 Checking for the existence of objects: SpaDES modules will often require the existence of objects in the simList. These are helpers for assessing this:

checkObject	Check for a existence of an object within a simList
checkPath	Checks the specified filepath for formatting consistencies

5.8 SELES-type approach to simulation: These functions are essentially skeletons and are not fully implemented. They are intended to make translations from SELES. You must know how to use SELES for these to be useful:

1 `agentLocation`	Agent location
`initiateAgents`	Initiate agents into a SpatialPointsDataFrame
`numAgents`	Number of agents
`probInit`	Probability of initiating an agent or event
`transitions`	Transition probability

5.9 Miscellaneous: Functions that may be useful within a `SpaDES` context:

`inRange`	Test whether a number lies within range [a,b]
`layerNames`	Get layer names for numerous object classes
`loadPackages`	Simple wrapper for loading packages
`numLayers`	Return number of layers
`paddedFloatToChar`	Wrapper for padding (e.g., zeros) floating numbers to character
`updateList`	Update values in a named list

---

6 Caching simulations and simulation components

Simulation caching uses the `reproducible` *package.*

Caching can be done in a variety of ways, most of which are up to the module developer. However, the one most common usage would be to cache a simulation run. This might be useful if a simulation is very long, has been run once, and the goal is just to retrieve final results. This would be an alternative to manually saving the outputs.

See example in `spades`, achieved by using `cache = TRUE` argument.

`Cache`	Caches a function, but often accessed as arg in `spades`
`cache`	deprecated. Please use `Cache`
`showCache`	Shows information about the objects in the cache
`clearCache`	Removes objects from the cache
`keepCache`	Keeps only the objects described
`clearStubArtifacts`	Removes any erroneous items in a cache repository

A module developer can build caching into their module by creating cached versions of their functions.

---

7 Plotting

Much of the underlying plotting functionality is provided by the `quickPlot` *package.*

There are several user-accessible plotting functions that are optimized for modularity and speed of plotting:

Commonly used:

`Plot`	The workhorse plotting function

Simulation diagrams:

`eventDiagram`	Gantt chart representing the events in a completed simulation.
`moduleDiagram`	Network diagram of simplified module (object) dependencies.
`objectDiagram`	Sequence diagram of detailed object dependencies.

Other useful plotting functions:

`clearPlot`	Helpful for resolving many errors
`clickValues`	Extract values from a raster object at the mouse click location(s)
`clickExtent`	Zoom into a raster or polygon map that was plotted with `Plot`
`clickCoordinates`	Get the coordinates, in map units, under mouse click
`dev`	Specify which device to plot on, making a non-RStudio one as default
`newPlot`	Open a new default plotting device
`rePlot`	Replots all elements of device for refreshing or moving plot

---

8 File operations

In addition to R's file operations, we have added several here to aid in bulk loading and saving of files for simulation purposes:

`loadFiles`	Load simulation objects according to a filelist
`rasterToMemory`	Read a raster from file to RAM
`saveFiles`	Save simulation objects according to outputs and params

---

9 Sample modules included in package

Several dummy modules are included for testing of functionality. These can be found with `file.path(find.package("SpaD`

`randomLandscapes`	Imports, updates, and plots several raster map layers
`caribouMovement`	A simple agent-based (a.k.a., individual-based) model
`fireSpread`	A simple model of a spatial spread process

10 Package options

SpaDES packages use the following options to configure behaviour:

- spades.cachePath: The default local directory in which to cache simulation outputs. Default is a temporary directory (typically /tmp/RtmpXXX/SpaDES/cache).
- spades.inputPath: The default local directory in which to look for simulation inputs. Default is a temporary directory (typically /tmp/RtmpXXX/SpaDES/inputs).
- spades.lowMemory: If true, some functions will use more memory efficient (but slower) algorithms. Default FALSE.
- spades.modulePath: The default local directory where modules and data will be downloaded and stored. Default is a temporary directory (typically /tmp/RtmpXXX/SpaDES/modules).
- spades.moduleRepo: The default GitHub repository to use when downloading modules. Default "PredictiveEcology/SpaDES-modules".
- spades.nCompleted: The maximum number of completed events to retain in the completed event queue. Default 1000L.
- spades.outputPath: The default local directory in which to save simulation outputs. Default is a temporary directory (typically /tmp/RtmpXXX/SpaDES/outputs).
- spades.tolerance: The default tolerance value used for floating point number comparisons. Default .Machine$double.eps^0.5.
- spades.useragent: The default user agent to use for downloading modules from GitHub.com. Default "http://github.com/PredictiveEcology/SpaDES".

Author(s)

Maintainer: Alex M Chubaty <alexander.chubaty@canada.ca>

Authors:

- Eliot J B McIntire <eliot.mcintire@canada.ca>

Other contributors:

- Yong Luo <yluo1@lakeheadu.ca> [contributor]
- Steve Cumming <Steve.Cumming@sbf.ulaval.ca> [contributor]
- Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [copyright holder]

See Also

Useful links:

- https://predictiveecology.github.io/SpaDES.core/
- https://github.com/PredictiveEcology/SpaDES.core
- Report bugs at https://github.com/PredictiveEcology/SpaDES.core/issues

---

`.addTagsToOutput,simList-method`
addTagsToOutput for simList class objects

Description

See `.addTagsToOutput`.

Usage

```
## S4 method for signature 'simList'
.addTagsToOutput(object, outputObjects, FUN,
  preDigestByClass)
```

Arguments

`object`	Any R object.
`outputObjects`	Optional character vector indicating which objects to return. This is only relevant for `simList` objects
`FUN`	A function
`preDigestByClass`	A list, usually from `.preDigestByClass`

Author(s)

Eliot McIntire

See Also

`.addTagsToOutput`

`.cacheMessage,simList-method`
cacheMessage for simList class objects

Description

See `.cacheMessage`.

Usage

```
## S4 method for signature 'simList'
.cacheMessage(object, functionName)
```

Arguments

object	Any R object.
functionName	A character string indicating the function name

See Also

.cacheMessage

.checkCacheRepo,list-method *checkCacheRepo for simList class objects*

Description

See .checkCacheRepo.

Usage

```
## S4 method for signature 'list'
.checkCacheRepo(object, create = FALSE)
```

Arguments

object	An R object
create	Logical. If TRUE, then it will create the path for cache.

See Also

.checkCacheRepo

.fileExtensions *File extensions map*

Description

How to load various types of files in R.

This function has two roles: 1) to proceed with the loading of files that are in a simList or 2) as a short cut to simInit(inputs = filelist). Generally not to be used by a user.

A data.frame with information on how to load various types of files in R, containing the columns:

- exts: the file extension;
- fun: the function to use for files with this file extension;
- package: the package from which to load fun.

Because of the environment slot, this is not quite as straightforward as just saving the object. This also has option for file-backed Rasters.

Usage

```
.fileExtensions()

loadFiles(sim, filelist, ...)

## S4 method for signature 'simList,missing'
loadFiles(sim, filelist, ...)

## S4 method for signature 'missing,ANY'
loadFiles(sim, filelist, ...)

## S4 method for signature 'missing,missing'
loadFiles(sim, filelist, ...)

.saveFileExtensions()

saveSimList(sim, filename, keepFileBackedAsIs, envir = parent.frame())
```

Arguments

sim	simList object.
filelist	list or data.frame to call loadFiles directly from the filelist as described in Details
...	Additional arguments.
filename	Character string with the path for saving simList
keepFileBackedAsIs	Logical. If there are file-backed Raster objects, should they be kept in their file-backed format, or loaded into RAM and saved within the .RData file. If TRUE (default), then the files will be copied to file.path(dirname(filename), "rasters").
envir	environment to search for objects to be saved.

Value

A saved .RData file in filename location.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

inputs

Examples

```
## Not run:

# Load random maps included with package
```

```
filelist <- data.frame(
    files = dir(system.file("maps", package = "quickPlot"),
            full.names = TRUE, pattern = "tif"),
    functions = "rasterToMemory", package = "quickPlot"
)
sim1 <- loadFiles(filelist = filelist)
clearPlot()
if (interactive()) Plot(sim1$DEM)

# Second, more sophisticated. All maps loaded at time = 0, and the last one is reloaded
#  at time = 10 and 20 (via "intervals").
# Also, pass the single argument as a list to all functions...
#  specifically, when add "native = TRUE" as an argument to the raster function
files = dir(system.file("maps", package = "quickPlot"),
            full.names = TRUE, pattern = "tif")
arguments = I(rep(list(native = TRUE), length(files)))
filelist = data.frame(
   files = files,
   functions = "raster::raster",
   objectName = NA,
   arguments = arguments,
   loadTime = 0,
   intervals = c(rep(NA, length(files)-1), 10)
)

sim2 <- loadFiles(filelist = filelist)

# if we extend the end time and continue running, it will load an object scheduled
#  at time = 10, and it will also schedule a new object loading at 20 because
#  interval = 10
end(sim2) <- 20
sim2 <- spades(sim2) # loads the percentPine map 2 more times, once at 10, once at 20

## End(Not run)
```

`.objSizeInclEnviros,simList-method`

objSizeInclEnviros for simList class objects

Description

See `.objSizeInclEnviros`.

Usage

```
## S4 method for signature 'simList'
.objSizeInclEnviros(object)
```

Arguments

`object`	Any R object.

See Also

.objSizeInclEnviros

.parseElems,simList-method
.parseElems for simList class objects

Description

See .parseElems.

Usage

```
## S4 method for signature 'simList'
.parseElems(tmp, elems, envir)
```

Arguments

tmp	A evaluated object
elems	A character string to be parsed
envir	An environment

See Also

.parseElems

.preDigestByClass,simList-method
Pre-digesting method for simList

Description

Takes a snapshot of simList objects.

Usage

```
## S4 method for signature 'simList'
.preDigestByClass(object)
```

Arguments

object	Any R object.

Details

See .preDigestByClass.

Author(s)

Eliot McIntire

See Also

.preDigestByClass

.prepareOutput,simList-method

prepareOutput for simList class objects

Description

See .prepareOutput.

Usage

```
## S4 method for signature 'simList'
.prepareOutput(object, cacheRepo, ...)
```

Arguments

object	Any R object
cacheRepo	A repository used for storing cached objects. This is optional if Cache is used inside a SpaDES module.
...	Arguments of FUN function .

See Also

.prepareOutput

`.robustDigest,simList-method`
`.robustDigest` *for* `simList` *class objects*

Description

This is intended to be used within the `Cache` function, but can be used to evaluate what a `simList` would look like once it is converted to a repeatably digestible object.

Usage

```
## S4 method for signature 'simList'
.robustDigest(object, objects,
  compareRasterFileLength = 1e+06, algo = "xxhash64",
  digestPathContent = FALSE, classOptions = list())
```

Arguments

`object`	an object to digest.
`objects`	Optional character vector indicating which objects are to be considered while making digestible. This is only relevant if the object being passed is an environment or list or the like.
`compareRasterFileLength`	Numeric. Optional. When there are Rasters, that have file-backed storage, this is passed to the length arg in `digest` when determining if the Raster file is already in the database. Note: uses `digest` for file-backed Raster. Default 1e6. Passed to `.prepareFileBackedRaster`.
`algo`	The algorithms to be used; currently available choices are `md5`, which is also the default, `sha1`, `crc32`, `sha256`, `sha512`, `xxhash32`, `xxhash64` and `murmur32`.
`digestPathContent`	Logical. Should arguments that are of class `Path` (see examples below) have their name digested (`FALSE`; default), or their file contents (`TRUE`).
`classOptions`	Optional list. This will pass into `.robustDigest` for specific classes. Should be options that the `.robustDigest` knows what to do with.

Details

See `.robustDigest`. This method strips out stuff from a simList class object that would make it otherwise not reproducibly digestible between sessions, operating systems, or machines. This will likely still not allow identical digest results across R versions.

Author(s)

Eliot Mcintire

See Also

.robustDigest

.simList-class *The* simList *class*

Description

Contains the minimum components of a SpaDES simulation. Various slot accessor methods (i.e., get and set functions) are provided (see 'Accessor Methods' below).

Details

Based on code from chapter 7.8.3 of Matloff (2011): "Discrete event simulation". Here, we implement a discrete event simulation in a more modular fashion so it's easier to add simulation components (i.e., "simulation modules"). We use S4 classes and methods, and use data.table instead of data.frame to implement the event queue (because it is much more efficient).

Slots

modules List of character names specifying which modules to load.

params Named list of potentially other lists specifying simulation parameters.

events The list of scheduled events (i.e., event queue), as a data.table. See 'Event Lists' for more information.

current The current event, as a data.table. See 'Event Lists' for more information..

completed The list of completed events, as a list. See 'Event Lists' for more information. It is kept as a list of individual events for speed. The completed method converts it to a sorted data.table.

depends A .simDeps list of .moduleDeps objects containing module object dependency information.

simtimes List of numerical values describing the simulation start and end times; as well as the current simulation time.

inputs A list of length 2, containing: 1) a data.frame or data.table of files and metadata, and 2) a list of optional arguments to pass to an import function.

outputs A list of length 2 containing: 1) a data.frame or data.table of files and metadata, and 2) a list of optional arguments to pass to an export function.

paths Named list of modulePath, inputPath, and outputPath paths. Partial matching is performed.

.envir Environment referencing the objects used in the simulation. Several "shortcuts" to accessing objects referenced by this environment are provided, and can be used on the simList object directly instead of specifying the .envir slot: $, [[, ls, ls.str, objs. See examples.

Accessor Methods

Several slot (and sub-slot) accessor methods are provided for use, and categorized into separate help pages:

simList-accessors-envir	Simulation enviroment.
simList-accessors-events	Scheduled and completed events.
simList-accessors-inout	Passing data in to / out of simulations.
simList-accessors-modules	Modules loaded and used; module dependencies.
simList-accessors-objects	Accessing objects used in the simulation.
simList-accessors-params	Global and module-specific parameters.
simList-accessors-paths	File paths for modules, inputs, and outputs.
simList-accessors-times	Simulation times.

Event Lists

The main event list is a sorted data.table (keyed) on eventTime, and eventPriority. The completed event list is an ordered list in the exact order that the events were executed. Each event is represented by a data.table row consisting of:

eventTime	The time the event is to occur.
moduleName	The module from which the event is taken.
eventType	A character string for the programmer-defined event type.
eventPriority	The priority given to the event.

Note

The simList class extends the .simList superclass by adding a slot .envir to store the simulation environment containing references to simulation objects. The simList_ class extends the .simList superclass, by adding a slot .list containing the simulation objects. Thus, simList is identical to simList_, except that the former uses an environment for objects and the latter uses a list. The class simList_ is only used internally.

Author(s)

Alex Chubaty and Eliot McIntire

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

---

.tagsByClass,simList-method

tagsByClass for simList class objects

Description

See .tagsByClass. Adds current moduleName, eventType, eventTime, and function:spades as userTags

Usage

```
## S4 method for signature 'simList'
.tagsByClass(object)
```

Arguments

object	Any R object.

Author(s)

Eliot McIntire

See Also

.tagsByClass

---

append_attr *Add a module to a* moduleList

Description

Ordinary base lists and vectors do not retain their attributes when subsetted or appended. This function appends items to a list while preserving the attributes of items in the list (but not of the list itself).

Usage

```
append_attr(x, y)

## S4 method for signature 'list,list'
append_attr(x, y)
```

Arguments

x	A list of items with optional attributes.
y	See x.

Details

Similar to updateList but does not require named lists.

Value

An updated list with attributes.

Author(s)

Alex Chubaty and Eliot McIntire

Examples

```
library(igraph) # igraph exports magrittr's pipe operator
tmp1 <- list("apple", "banana") %>% lapply(., `attributes<-`, list(type = "fruit"))
tmp2 <- list("carrot") %>% lapply(., `attributes<-`, list(type = "vegetable"))
append_attr(tmp1, tmp2)
rm(tmp1, tmp2)
```

---

checkModule *Check for the existence of a remote module*

Description

Looks in the remote `repo` for a module named `name`.

Usage

```
checkModule(name, repo)

## S4 method for signature 'character,character'
checkModule(name, repo)

## S4 method for signature 'character,missing'
checkModule(name)
```

Arguments

`name`	Character string giving the module name.
`repo`	GitHub repository name. Default is `"PredictiveEcology/SpaDES-modules"`, which is specified by the global option `spades.moduleRepo`.

Author(s)

Eliot McIntire and Alex Chubaty

---

checkModuleLocal *Check for the existence of a module locally*

Description

Looks the module path for a module named `name`, and checks for existence of all essential module files listed below.

Usage

```
checkModuleLocal(name, path, version)

## S4 method for signature 'character,character,character'
checkModuleLocal(name, path, version)

## S4 method for signature 'character,ANY,ANY'
checkModuleLocal(name, path, version)
```

Arguments

name	Character string giving the module name.
path	Local path to modules directory. Default is specified by the global option spades.modulePath.
version	Character specifying the desired module version.

Details

- 'data/CHECKSUMS.txt'
- 'name.R'

Value

Logical indicating presence of the module (invisibly).

Author(s)

Alex Chubaty

---

checkObject *Check for existence of object(s) referenced by a* objects *slot of a* simList *object*

Description

Check that a named object exists in the provide simList environment slot, and optionally has desired attributes.

Usage

```
checkObject(sim, name, object, layer, ...)

## S4 method for signature 'simList,missing,Raster,character'
checkObject(sim, name, object,
  layer, ...)

## S4 method for signature 'simList,missing,ANY,missing'
```

```
checkObject(sim, name, object, layer,
  ...)

## S4 method for signature 'simList,character,missing,missing'
checkObject(sim, name, object,
  layer, ...)

## S4 method for signature 'simList,character,missing,character'
checkObject(sim, name, object,
  layer, ...)

## S4 method for signature 'missing,ANY,missing,ANY'
checkObject(sim, name, object, layer, ...)
```

Arguments

sim	A simList object.
name	A character string specifying the name of an object to be checked.
object	An object. This is mostly used internally, or with layer, because it will fail if the object does not exist.
layer	Character string, specifying a layer name in a Raster, if the name is a Raster* object.
...	Additional arguments. Not implemented.

Value

Invisibly return TRUE indicating object exists; FALSE if not.

Author(s)

Alex Chubaty and Eliot McIntire

See Also

library.

---

checkParams *Check use and existence of params passed to simulation.*

Description

Checks that all parameters passed are used in a module, and that all parameters used in a module are passed.

Usage

```
checkParams(sim, coreModules, coreParams, path, ...)

## S4 method for signature 'simList,list,list,character'
checkParams(sim, coreModules,
  coreParams, path, ...)
```

Arguments

sim	A simList simulation object.
coreModules	List of core modules.
coreParams	List of default core parameters.
path	The location of the modules' source files.
...	Additional arguments. Not implemented.

Value

Invisibly return TRUE indicating object exists; FALSE if not. Sensible messages are be produced identifying missing parameters.

Author(s)

Alex Chubaty

---

checksums *Calculate checksums for a module's data files*

Description

Verify (and optionally write) checksums for data files in a module's data/ subdirectory. The file data/CHECKSUMS.txt contains the expected checksums for each data file. Checksums are computed using SpaDES.core:::.digest, which is simply a wrapper around digest::digest.

Usage

```
checksums(module, path, write, ...)

## S4 method for signature 'character,character,logical'
checksums(module, path, write, ...)

## S4 method for signature 'character,character,missing'
checksums(module, path, write, ...)
```

Arguments

module	Character string giving the name of the module.
path	Character string giving the path to the module directory.
write	Logical indicating whether to overwrite CHECKSUMS.txt. Default is FALSE, as users should not change this file. Module developers should write this file prior to distributing their module code, and update accordingly when the data change.
...	Passed to digest, notably algo, so the digest algorithm can be specified.

Details

Modules may require data that for various reasons cannot be distributed with the module source code. In these cases, the module developer should ensure that the module downloads and extracts the data required. It is useful to not only check that the data files exist locally but that their checksums match those expected. See also downloadData.

Value

A data.frame with columns: result, expectedFile, actualFile, and checksum.

Note

In version 1.2.0 and earlier, two checksums per file were required because of differences in the checksum hash values on Windows and Unix-like platforms. Recent versions use a different (faster) algorithm and only require one checksum value per file. To update your 'CHECKSUMS.txt' files using the new algorithm, see https://github.com/PredictiveEcology/SpaDES/issues/295#issuecomment-246513405.

Author(s)

Alex Chubaty

Examples

```
## Not run:
moduleName <- "my_module"
modulePath <- file.path("path", "to", "modules")

## verify checksums of all data files
checksums(moduleName, modulePath)

## write new CHECKSUMS.txt file

# 1. verify that all data files are present (and no extra files are present)
list.files(file.path(modulePath, moduleName, "data"))

# 2. calculate file checksums and write to file (this will overwrite CHECKSUMS.txt)
checksums(moduleName, modulePath, write = TRUE)

## End(Not run)
```

classFilter *Filter objects by class*

Description

Based on http://stackoverflow.com/a/5158978/1380598.

Usage

```
classFilter(x, include, exclude, envir)

## S4 method for signature 'character,character,character,environment'
classFilter(x, include,
  exclude, envir)

## S4 method for signature 'character,character,character,missing'
classFilter(x, include,
  exclude)

## S4 method for signature 'character,character,missing,environment'
classFilter(x, include,
  envir)

## S4 method for signature 'character,character,missing,missing'
classFilter(x, include)
```

Arguments

x	Character vector of object names to filter, possibly from ls.
include	Class(es) to include, as a character vector.
exclude	Optional class(es) to exclude, as a character vector.
envir	The environment ins which to search for objects. Default is the calling environment.

Value

Vector of object names matching the class filter.

Note

inherits is used internally to check the object class, which can, in some cases, return results inconsistent with is. See http://stackoverflow.com/a/27923346/1380598. These (known) cases are checked manually and corrected.

Author(s)

Alex Chubaty

Examples

```
## Not run:
  ## from global environment
  a <- list(1:10)     # class `list`
  b <- letters        # class `character`
  d <- stats::runif(10)      # class `numeric`
  f <- sample(1L:10L) # class `numeric`, `integer`
  g <- lm( jitter(d) ~ d ) # class `lm`
  h <- glm( jitter(d) ~ d ) # class `lm`, `glm`
  classFilter(ls(), include=c("character", "list"))
  classFilter(ls(), include = "numeric")
  classFilter(ls(), include = "numeric", exclude = "integer")
  classFilter(ls(), include = "lm")
  classFilter(ls(), include = "lm", exclude = "glm")
  rm(a, b, d, f, g, h)

## End(Not run)

## from local (e.g., function) environment
local({
  e <- environment()
  a <- list(1:10)     # class `list`
  b <- letters        # class `character`
  d <- stats::runif(10)      # class `numeric`
  f <- sample(1L:10L) # class `numeric`, `integer`
  g <- lm( jitter(d) ~ d ) # class `lm`
  h <- glm( jitter(d) ~ d ) # class `lm`, `glm`
  classFilter(ls(), include=c("character", "list"), envir = e)
  classFilter(ls(), include = "numeric", envir = e)
  classFilter(ls(), include = "numeric", exclude = "integer", envir = e)
  classFilter(ls(), include = "lm", envir = e)
  classFilter(ls(), include = "lm", exclude = "glm", envir = e)
  rm(a, b, d, e, f, g, h)
})

## from another environment
e = new.env(parent = emptyenv())
e$a <- list(1:10)     # class `list`
e$b <- letters        # class `character`
e$d <- stats::runif(10)      # class `numeric`
e$f <- sample(1L:10L) # class `numeric`, `integer`
e$g <- lm( jitter(e$d) ~ e$d ) # class `lm`
e$h <- glm( jitter(e$d) ~ e$d ) # class `lm`, `glm`
classFilter(ls(e), include=c("character", "list"), envir = e)
classFilter(ls(e), include = "numeric", envir = e)
classFilter(ls(e), include = "numeric", exclude = "integer", envir = e)
classFilter(ls(e), include = "lm", envir = e)
classFilter(ls(e), include = "lm", exclude = "glm", envir = e)
rm(a, b, d, f, g, h, envir = e)
rm(e)
```

Copy,simList-method *Copy for simList class objects*

Description

Because a simList works with an environment to hold all objects, all objects within that slot are pass-by-reference. That means it is not possible to simply copy an object with an assignment operator: the two objects will share the same objects. As one simList object changes so will the other. when this is not the desired behaviour, use this function. NOTE: use capital C, to limit confusion with `data.table::copy()` See `Copy`.

Usage

```
## S4 method for signature 'simList'
Copy(object, objects, queues)
```

Arguments

`object`	An R object (likely containing environments) or an environment.
`objects`	Whether the objects contained within the simList environment should be copied. Default `TRUE`, which may be slow.
`queues`	Logical. Should the events queues (`events`, `current`, `completed`) be deep copied via `data.table::copy`

Author(s)

Eliot McIntire

See Also

`Copy`

copyModule *Create a copy of an existing module*

Description

Create a copy of an existing module

Usage

```
copyModule(from, to, path, ...)

## S4 method for signature 'character,character,character'
copyModule(from, to, path, ...)

## S4 method for signature 'character,character,missing'
copyModule(from, to, path, ...)
```

Arguments

from	The name of the module to copy.
to	The name of the copy.
path	The path to a local module directory. Defaults to the path set by the spades.modulePath option. See setPaths.
...	Additional arguments to file.copy, e.g., overwrite = TRUE.

Value

Invisible logical indicating success (TRUE) or failure (FALSE).

Author(s)

Alex Chubaty

Examples

```
## Not run: copyModule(from, to)
```

---

createsOutput *Define an output object of a module*

Description

Used to specify an output object's name, class, description and other specifications.

Usage

```
createsOutput(objectName, objectClass, desc, ...)

## S4 method for signature 'ANY,ANY,ANY'
createsOutput(objectName, objectClass, desc, ...)

## S4 method for signature 'character,character,character'
createsOutput(objectName, objectClass,
  desc, ...)
```

Arguments

objectName	Character string to define the output object's name.
objectClass	Character string to specify the output object's class.
desc	Text string providing a brief description of the output object.
...	Other specifications of the output object.

Value

A data.frame suitable to be passed to outputObjects in a module's metadata.

Author(s)

Yong Luo

Examples

```
outputObjects <- dplyr::bind_rows(
  createsOutput(objectName = "outputObject1", objectClass = "character",
                desc = "this is for example"),
  createsOutput(objectName = "outputObject2", objectClass = "numeric",
                desc = "this is for example",
                otherInformation = "I am the second output object")
)
```

---

defineModule *Define a new module.*

Description

Specify a new module's metadata as well as object and package dependencies. Packages are loaded during this call.

Usage

```
defineModule(sim, x)

## S4 method for signature '.simList,list'
defineModule(sim, x)
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
x	A list with a number of named elements, referred to as the metadata. See details.

Value

Updated `simList` object.

Required metadata elements

`name`	Module name. Must match the filename (without the `.R` extension). This is currently not parsed by SpaDES
`description`	Brief description of the module. This is currently not parsed by SpaDES; it is for human readers only.
`keywords`	Author-supplied keywords. This is currently not parsed by SpaDES; it is for human readers only.
`childModules`	If this contains any character vector, then it will be treated as a parent module. If this is a parent module, th
`authors`	Module author information (as a vector of `person` objects. This is currently not parsed by SpaDES; it is for
`version`	Module version number (will be coerced to `numeric_version` if a character or numeric are supplied). The
`spatialExtent`	The spatial extent of the module supplied via `raster::extent`. This is currently unimplemented. Once im
`timeframe`	Vector (length 2) of POSIXt dates specifying the temporal extent of the module. Currently unimplemented.
`timeunit`	Time scale of the module (e.g., "day", "year"). This MUST be specified. It indicates what '1' unit of time m
`citation`	List of character strings specifying module citation information. Alternatively, a list of filenames of `.bib` o
`documentation`	List of filenames referring to module documentation sources. This is currently not parsed by SpaDES; it is
`reqdPkgs`	List of R package names required by the module. These packages will be loaded when `simInit` is called.
`parameters`	A data.frame specifying the parameters used in the module. Usually produced by `rbind`-ing the outputs of
`inputObjects`	A `data.frame` specifying the data objects expected as inputs to the module, with columns `objectName` (cla
`outputObjects`	A `data.frame` specifying the data objects output by the module, with columns identical to those in `inputO`

Author(s)

Alex Chubaty

Examples

```
## Not run:
  ## a default version of the defineModule is created with a call to newModule
  newModule("test", path = tempdir())

  ## view the resulting module file
  if (interactive()) file.edit(file.path(tempdir(), "test", "test.R"))

 # The default defineModule created by newModule is currently (SpaDES version 1.3.1.9044):
  defineModule(sim, list(
    name = "test",
    description = "insert module description here",
    keywords = c("insert key words here"),
    authors = c(person(c("First", "Middle"), "Last",
                       email = "email@example.com", role = c("aut", "cre"))),
    childModules = character(0),
    version = list(SpaDES = "1.3.1.9044", test = "0.0.1"),
    spatialExtent = raster::extent(rep(NA_real_, 4)),
    timeframe = as.POSIXlt(c(NA, NA)),
```

```
    timeunit = NA_character_, # e.g., "year",
    citation = list("citation.bib"),
    documentation = list("README.txt", "test.Rmd"),
    reqdPkgs = list(),
    parameters = rbind(
      #defineParameter("paramName", "paramClass", value, min, max,
      # "parameter description")),
      defineParameter(".plotInitialTime", "numeric", NA, NA, NA,
      "This describes the simulation time at which the first plot event should occur"),
      defineParameter(".plotInterval", "numeric", NA, NA, NA,
      "This describes the simulation time at which the first plot event should occur"),
      defineParameter(".saveInitialTime", "numeric", NA, NA, NA,
      "This describes the simulation time at which the first save event should occur"),
      defineParameter(".saveInterval", "numeric", NA, NA, NA,
      "This describes the simulation time at which the first save event should occur")
    ),
    inputObjects = bind_rows(
      expectsInput(objectName = NA_character_, objectClass = NA_character_,
        sourceURL = NA_character_, desc = NA_character_, other = NA_character_)
    ),
    outputObjects = bind_rows(
      createsOutput(objectName = NA_character_, objectClass = NA_character_,
        desc = NA_character_, other = NA_character_)
    )
  ))


## End(Not run)
```

`defineParameter` *Define a parameter used in a module*

Description

Used to specify a parameter's name, value, and set a default.

Usage

```
defineParameter(name, class, default, min, max, desc)

## S4 method for signature 'character,character,ANY,ANY,ANY,character'
defineParameter(name,
  class, default, min, max, desc)


  ## S4 method for signature 'character,character,ANY,missing,missing,character'
defineParameter(name,
  class, default, desc)
```

```
## S4 method for signature 'missing,missing,missing,missing,missing,missing'
defineParameter()
```

Arguments

name	Character string giving the parameter name.
class	Character string giving the parameter class.
default	The default value to use when none is specified by the user. Non-standard evaluation is used for the expression.
min	With max, used to define a suitable range of values. Non-standard evaluation is used for the expression.
max	With min, used to define a suitable range of values. Non-standard evaluation is used for the expression.
desc	Text string providing a brief description of the parameter.

Value

data.frame

Author(s)

Alex Chubaty

Examples

```
parameters = rbind(
  defineParameter("lambda", "numeric", 1.23, desc = "intrinsic rate of increase"),
  defineParameter("P", "numeric", 0.2, 0, 1, "probability of attack")
)
```

---

depsEdgeList *Build edge list for module dependency graph*

Description

Build edge list for module dependency graph

Usage

```
depsEdgeList(sim, plot)

## S4 method for signature 'simList,logical'
depsEdgeList(sim, plot)

## S4 method for signature 'simList,missing'
depsEdgeList(sim, plot)
```

Arguments

sim	A simList object.
plot	Logical indicating whether the edgelist (and subsequent graph) will be used for plotting. If TRUE, duplicated rows (i.e., multiple object dependencies between modules) are removed so that only a single arrow is drawn connecting the modules. Default is FALSE.

Value

A data.table whose first two columns give a list of edges and remaining columns the attributes of the dependency objects (object name, class, etc.).

Author(s)

Alex Chubaty

---

depsGraph *Build a module dependency graph*

Description

Build a module dependency graph

Usage

```
depsGraph(sim, plot)

## S4 method for signature 'simList,logical'
depsGraph(sim, plot)

## S4 method for signature 'simList,missing'
depsGraph(sim)
```

Arguments

sim	A simList object.
plot	Logical indicating whether the edgelist (and subsequent graph) will be used for plotting. If TRUE, duplicated rows (i.e., multiple object dependencies between modules) are removed so that only a single arrow is drawn connecting the modules. Default is FALSE.

Value

An igraph object.

Author(s)

Alex Chubaty

`doEvent.checkpoint` *Simulation checkpoints.*

Description

Save and reload the current state of the simulation, including the state of the random number generator, by scheduling checkpoint events.

Usage

```
doEvent.checkpoint(sim, eventTime, eventType, debug = FALSE)

checkpointLoad(file)

.checkpointSave(sim, file)

checkpointFile(sim)

## S4 method for signature '.simList'
checkpointFile(sim)

checkpointFile(sim) <- value

## S4 replacement method for signature '.simList'
checkpointFile(sim) <- value

checkpointInterval(sim)

## S4 method for signature '.simList'
checkpointInterval(sim)

checkpointInterval(sim) <- value

## S4 replacement method for signature '.simList'
checkpointInterval(sim) <- value
```

Arguments

`sim`	A `simList` simulation object.
`eventTime`	A numeric specifying the time of the next event.
`eventType`	A character string specifying the type of event: one of either `"init"`, `"load"`, or `"save"`.
`debug`	Optional logical flag determines whether sim debug info will be printed (default `debug = FALSE`).
`file`	The checkpoint file.
`value`	The object to be stored at the slot.

Details

RNG save code adapted from: http://www.cookbook-r.com/Numbers/Saving_the_state_of_the_random_number_generator/ and https://stackoverflow.com/questions/13997444/

Value

Returns the modified simList object.

Author(s)

Alex Chubaty

See Also

.Random.seed.

Other functions to access elements of a simList object: .addDepends, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

downloadData *Download module data*

Description

Download external data for a module if not already present in the module directory or if there is a checksum mismatch indicating that the file is not the correct one.

Usage

```
downloadData(module, path, quiet)

## S4 method for signature 'character,character,logical'
downloadData(module, path, quiet)

## S4 method for signature 'character,missing,missing'
downloadData(module)

## S4 method for signature 'character,missing,logical'
downloadData(module, quiet)

## S4 method for signature 'character,character,missing'
downloadData(module, path)
```

Arguments

module	Character string giving the name of the module.
path	Character string giving the path to the module directory.
quiet	Logical. This is passed to download.file. Default is FALSE.

Value

Invisibly, a list of downloaded files.

Author(s)

Alex Chubaty

`downloadModule` *Download a module from a SpaDES module GitHub repository*

Description

Download a .zip file of the module and extract (unzip) it to a user-specified location.

Usage

```
downloadModule(name, path, version, repo, data, quiet)


  ## S4 method for signature
## 'character,character,character,character,logical,logical'
downloadModule(name,
  path, version, repo, data, quiet)


  ## S4 method for signature 'character,missing,missing,missing,missing,missing'
downloadModule(name)

## S4 method for signature 'character,ANY,ANY,ANY,ANY,ANY'
downloadModule(name, path, version,
  repo, data, quiet)
```

Arguments

`name`	Character string giving the module name.
`path`	Character string giving the location in which to save the downloaded module.
`version`	The module version to download. (If not specified, or `NA`, the most recent version will be retrieved.)
`repo`	GitHub repository name. Default is `"PredictiveEcology/SpaDES-modules"`, which is specified by the global option `spades.moduleRepo`.
`data`	Logical. If TRUE, then the data that is identified in the module metadata will be downloaded, if possible. Default if FALSE.
`quiet`	Logical. This is passed to `download.file`. Default is FALSE.

Details

Currently only works with a public GitHub repository, where modules are in a `modules` directory in the root tree on the `master` branch.

Value

A list of length 2. The first element is a character vector containing a character vector of extracted files for the module. The second element is a tbl with details about the data that is relevant for the function, including whether it was downloaded or not, whether it was renamed (because there was a local copy that had the wrong file name).

Note

`downloadModule` uses the `GITHUB_PAT` environment variable if a value is set. This alleviates 403 errors caused by too-frequent downloads. Generate a GitHub personal access token at https://github.com/settings/tokens.

The default is to overwrite any existing files in the case of a conflict.

Author(s)

Alex Chubaty

---

dwrpnorm2 *Vectorized wrapped normal density function*

Description

This is a modified version of `dwrpnorm` found in `CircStats` to allow for multiple angles at once (i.e., vectorized).

Usage

```
dwrpnorm2(theta, mu, rho, sd = 1, acc = 1e-05, tol = acc)
```

Arguments

`theta`	value at which to evaluate the density function, measured in radians.
`mu`	mean direction of distribution, measured in radians.
`rho`	mean resultant length of distribution.
`sd`	different way of select `rho`, see details below.
`acc`	parameter defining the accuracy of the estimation of the density. Terms are added to the infinite summation that defines the density function until successive estimates are within `acc` of each other.
`tol`	the same as `acc`.

Author(s)

Eliot McIntire

Examples

```
# Values for which to evaluate density
theta <- c(1:500) * 2 * pi / 500
# Compute wrapped normal density function
density <- c(1:500)
for(i in 1:500) density[i] <- dwrpnorm2(theta[i], pi, .75)
if (interactive()) plot(theta, density)
# Approximate area under density curve
sum(density * 2 * pi / 500)
```

---

dyears *SpaDES time units*

Description

SpaDES modules commonly use approximate durations that divide with no remainder among themselves. For example, models that simulate based on a "week" timestep, will likely want to fall in lock step with a second module that is a "year" timestep. Since, weeks, months, years don't really have this behaviour because of: leap years, leap seconds, not quite 52 weeks in a year, months that are of different duration, etc. We have generated a set of units that work well together that are based on the astronomical or "Julian" year. In an astronomical year, leap years are added within each year with an extra 1/4 day, (i.e., 1 year == 365.25 days); months are defined as year/12, and weeks as year/52.

Usage

```
dyears(x)

## S4 method for signature 'numeric'
dyears(x)

dmonths(x)

## S4 method for signature 'numeric'
dmonths(x)

dweeks(x)

## S4 method for signature 'numeric'
dweeks(x)

dweek(x)
```

```
dmonth(x)

dyear(x)

dsecond(x)

dday(x)

dhour(x)

dNA(x)

## S4 method for signature 'ANY'
dNA(x)
```

Arguments

x	numeric. Number of the desired units

Details

When these units are not correct, a module developer can create their own time unit using, and create a function to calculate the number of seconds in that unit using the "d" prefix (for duration), following the `lubridate` package standard: `dfortnight <- function(x) lubridate::duration(dday(14))`. Then the module developer can use "fortnight" as the module's time unit.

Value

Number of seconds within each unit

Author(s)

Eliot McIntire

---

envir *Simulation environment*

Description

Accessor functions for the `.envir` slot in a `simList` object. These are included for advanced users.

Usage

```
envir(sim)

## S4 method for signature 'simList'
envir(sim)

envir(sim) <- value

## S4 replacement method for signature 'simList'
envir(sim) <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
value	The object to be stored at the slot.

Details

Currently, only get and set methods are defined. Subset methods are not.

Value

Returns or sets the value of the slot from the simList object.

Author(s)

Alex Chubaty

See Also

SpaDES.core-package, specifically the section 1.2.8 on simList environment.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

eventDiagram *Simulation event diagram*

Description

Create a Gantt Chart representing the events in a completed simulation. This event diagram is constructed using the completed event list To change the number of events shown, provide an n argument.

Usage

```
eventDiagram(sim, n, startDate, ...)

## S4 method for signature 'simList,numeric,character'
eventDiagram(sim, n, startDate, ...)

## S4 method for signature 'simList,missing,character'
eventDiagram(sim, n, startDate, ...)

## S4 method for signature 'simList,missing,missing'
eventDiagram(sim, n, startDate, ...)
```

Arguments

sim	A simList object (typically corresponding to a completed simulation).
n	The number of most recently completed events to plot.
startDate	A character representation of date in YYYY-MM-DD format.
...	Additional arguments passed to mermaid. Useful for specifying height and width.

Details

Simulation time is presented on the x-axis, starting at date 'startDate'. Each module appears in a color-coded row, within which each event for that module is displayed corresponding to the sequence of events for that module. Note that only the start time of the event is meaningful is these figures: the width of the bar associated with a particular module's event DOES NOT correspond to an event's "duration".

Based on this StackOverflow answer: http://stackoverflow.com/a/29999300/1380598.

Value

Plots an event diagram as Gantt Chart, invisibly returning a mermaid object.

Note

A red vertical line corresponding to the current date may appear on the figure. This is useful for Gantt Charts generally but can be considered a 'bug' here.

Author(s)

Alex Chubaty

See Also

mermaid.

events *Simulation event lists*

Description

Accessor functions for the events and completed slots of a simList object. By default, the event lists are shown when the simList object is printed, thus most users will not require direct use of these methods.

events	Scheduled simulation events (the event queue).
completed	Completed simulation events.

Usage

```
events(sim, unit)

## S4 method for signature '.simList,character'
events(sim, unit)

## S4 method for signature '.simList,missing'
events(sim, unit)

events(sim) <- value

## S4 replacement method for signature '.simList'
events(sim) <- value

current(sim, unit)

## S4 method for signature '.simList,character'
current(sim, unit)

## S4 method for signature '.simList,missing'
current(sim, unit)

current(sim) <- value

## S4 replacement method for signature '.simList'
current(sim) <- value

completed(sim, unit)

## S4 method for signature '.simList,character'
completed(sim, unit)

## S4 method for signature '.simList,missing'
```

```
completed(sim, unit)

completed(sim) <- value

## S4 replacement method for signature '.simList'
completed(sim) <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
unit	Character. One of the time units used in SpaDES.
value	The object to be stored at the slot.

Details

Currently, only get and set methods are defined. Subset methods are not.

Value

Returns or sets the value of the slot from the simList object.

Note

Each event is represented by a data.table row consisting of:

- eventTime: The time the event is to occur.
- moduleName: The module from which the event is taken.
- eventType: A character string for the programmer-defined event type.

See Also

SpaDES.core-package, specifically the section 1.2.6 on Simulation event queues.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

expectsInput *Define an input object that the module expects.*

Description

Used to specify an input object's name, class, description, source url and other specifications.

Usage

```
expectsInput(objectName, objectClass, desc, sourceURL, ...)

## S4 method for signature 'ANY,ANY,ANY,ANY'
expectsInput(objectName, objectClass, desc,
  sourceURL, ...)

## S4 method for signature 'character,character,character,character'
expectsInput(objectName,
  objectClass, desc, sourceURL, ...)

## S4 method for signature 'character,character,character,missing'
expectsInput(objectName,
  objectClass, desc, sourceURL, ...)
```

Arguments

objectName	Character string to define the input object's name.
objectClass	Character string to specify the input object's class.
desc	Text string providing a brief description of the input object.
sourceURL	Character string to specify an URL to reach the input object, default is NA.
...	Other specifications of the input object.

Value

A data.frame suitable to be passed to inputObjects in a module's metadata.

Author(s)

Yong Luo

Examples

```
inputObjects <- dplyr::bind_rows(
  expectsInput(objectName = "inputObject1", objectClass = "character",
               desc = "this is for example", sourceURL = "not available"),
  expectsInput(objectName = "inputObject2", objectClass = "numeric",
               desc = "this is for example", sourceURL = "not available",
               otherInformation = "I am the second input object")
)
```

experiment *Run an experiment using* spades

Description

This is essentially a wrapper around the spades call that allows for multiple calls to spades. This function will use a single processor, or multiple processors if beginCluster has been run first or a cluster object is passed in the cl argument (gives more control to user).

Usage

```
experiment(sim, replicates = 1, params, modules, objects = list(), inputs,
  dirPrefix = "simNum", substrLength = 3, saveExperiment = TRUE,
  experimentFile = "experiment.RData", clearSimEnv = FALSE, notOlderThan,
  cl, ...)

## S4 method for signature 'simList'
experiment(sim, replicates = 1, params, modules,
  objects = list(), inputs, dirPrefix = "simNum", substrLength = 3,
  saveExperiment = TRUE, experimentFile = "experiment.RData",
  clearSimEnv = FALSE, notOlderThan, cl, ...)
```

Arguments

sim	A simList simulation object, generally produced by simInit.
replicates	The number of replicates to run of the same simList. See details and examples.
params	Like for simInit, but for each parameter, provide a list of alternative values. See details and examples.
modules	Like for simInit, but a list of module names (as strings). See details and examples.
objects	Like for simInit, but a list of named lists of named objects. See details and examples.
inputs	Like for simInit, but a list of inputs data.frames. See details and examples.
dirPrefix	String vector. This will be concatenated as a prefix on the directory names. See details and examples.
substrLength	Numeric. While making outputPath for each spades call, this is the number of characters kept from each factor level. See details and examples.
saveExperiment	Logical. Should params, modules, inputs, sim, and resulting experimental design be saved to a file. If TRUE are saved to a single list called experiment. Default TRUE.
experimentFile	String. Filename if saveExperiment is TRUE; saved to outputPath(sim) in .RData format. See Details.
clearSimEnv	Logical. If TRUE, then the envir(sim) of each simList in the return list is emptied. This is to reduce RAM load of large return object. Default FALSE.

notOlderThan	Date or time. Passed to reproducible::Cache to update the cache. Default is NULL, meaning don't update the cache. If Sys.time() is provided, then it will force a recache, i.e., remove old value and replace with new value. Ignored if cache is FALSE.
cl	A cluster object. Optional. This would generally be created using parallel::makeCluster or equivalent. This is an alternative way, instead of beginCluster(), to use parallelism for this function, allowing for more control over cluster use.
...	Passed to spades. Specifically, debug, .plotInitialTime, .saveInitialTime, cache and/or notOlderThan. Caching is still experimental. It is tested to work under some conditions, but not all. See details.

Details

Generally, there are 2 reasons to do this: replication and varying simulation inputs to accomplish some sort of simulation experiment. This function deals with both of these cases. In the case of varying inputs, this function will attempt to create a fully factorial experiment among all levels of the variables passed into the function. If all combinations do not make sense, e.g., if parameters and modules are varied, and some of the parameters don't exist in all combinations of modules, then the function will do an "all meaningful combinations" factorial experiment. Likewise, fully factorial combinations of parameters and inputs may not be the desired behaviour. The function requires a simList object, acting as the basis for the experiment, plus optional inputs and/or objects and/or params and/or modules and/or replications.

This function requires a complete simList: this simList will form the basis of the modifications as passed by params, modules, inputs, and objects. All params, modules, inputs or objects passed into this function will override the corresponding params, modules, inputs, or identically named objects that are in the sim argument.

This function is parallel aware, using the same mechanism as used in the raster package. Specifically, if you start a cluster using beginCluster, then this experiment function will automatically use that cluster. It is always a good idea to stop the cluster when finished, using endCluster.

Here are generic examples of how params, modules, objects, and inputs should be structured.

```
params = list(moduleName = list(paramName = list(val1, val2))).

modules = list(c("module1","module2"), c("module1","module3"))

objects = list(objName = list(object1=object1, object2=object2))

inputs = list(       data.frame(file = pathToFile1, loadTime = 0, objectName = "landscape",
)
```

Output directories are changed using this function: this is one of the dominant side effects of this function. If there are only replications, then a set of subdirectories will be created, one for each replicate. If there are varying parameters and or modules, outputPath is updated to include a subdirectory for each level of the experiment. These are not nested, i.e., even if there are nested factors, all subdirectories due to the experimental setup will be at the same level. Replicates will be one level below this. The subdirectory names will include the module(s), parameter names, the parameter values, and input index number (i.e., which row of the inputs data.frame). The default rule for naming is a concatenation of:

1. The experiment level (arbitrarily starting at 1). This is padded with zeros if there are many experiment levels.

2. The module, parameter name and parameter experiment level (not the parameter value, as values could be complex), for each parameter that is varying.

3. The module set.

4. The input index number

5. Individual identifiers are separated by a dash.

6. Module - Parameter - Parameter index triplets are separated by underscore.

e.g., a folder called: `01-fir_spr_1-car_N_1-inp_1` would be the first experiment level (01), the first parameter value for the `spr*` parameter of the `fir*` module, the first parameter value of the N parameter of the `car*` module, and the first input dataset provided.

This subdirectory name could be long if there are many dimensions to the experiment. The parameter `substrLength` determines the level of truncation of the parameter, module and input names for these subdirectories. For example, the resulting directory name for changes to the `spreadprob` parameter in the `fireSpread` module and the `N` parameter in the `caribouMovement` module would be: `1_fir_spr_1-car_N_1` if `substrLength` is 3, the default.

Replication is treated slightly differently. `outputPath` is always 1 level below the experiment level for a replicate. If the call to `experiment` is not a factorial experiment (i.e., it is just replication), then the default is to put the replicate subdirectories at the top level of `outputPath`. To force this one level down, `dirPrefix` can be used or a manual change to `outputPath` before the call to experiment.

`dirPrefix` can be used to give custom names to directories for outputs. There is a special value, `"simNum"`, that is used as default, which is an arbitrary number associated with the experiment. This corresponds to the row number in the `attr(sims, "experiment")`. This `"simNum"` can be used with other strings, such as `dirPrefix = c("expt", "simNum")`.

The experiment structure is kept in two places: the return object has an attribute, and a file named `experiment.RData` (see argument `experimentFile`) located in `outputPath(sim)`.

`substrLength`, if `0`, will eliminate the subdirectory naming convention and use only `dirPrefix`.

If `cache = TRUE` is passed, then this will pass this to `spades`, with the additional argument `replicate = x`, where x is the replicate number. That means that if a user runs `experiment` with `replicate = 4` and `cache = TRUE`, then SpaDES will run 4 replicates, caching the results, including replicate = 1, replicate = 2, replicate = 3, and replicate = 4. Thus, if a second call to experiment with the exact same simList is passed, and `replicates = 6`, the first 4 will be taken from the cached copies, and replicate 5 and 6 will be run (and cached) as normal. If `notOlderThan` used with a time that is more recent than the cached copy, then a new spades will be done, and the cached copy will be deleted from the cache repository, so there will only ever be one copy of a particular replicate for a particular simList. NOTE: caching may not work as desired on a Windows machine because the sqlite database can only be written to one at a time, so there may be collisions.

Value

Invisibly returns a list of the resulting `simList` objects from the fully factorial experiment. This list has an attribute, which a list with 2 elements: the experimental design provided in a wide data.frame and the experiment values in a long data.frame. There is also a file saved with these two data.frames. It is named whatever is passed into `experimentFile`. Since returned list of `simList` objects may be large, the user is not obliged to return this object (as it is returned invisibly). Clearly, there may be objects saved during simulations. This would be determined as per a normal spades call, using `outputs` like, say, `outputs(sims[[1]])`.

Author(s)

Eliot McIntire

See Also

simInit

Examples

```
if (interactive()) {
  library(igraph) # use %>% in a few examples
  library(raster)

  tmpdir <- file.path(tempdir(), "examples")

  # Create a default simList object for use through these examples
  mySim <- simInit(
    times = list(start = 0.0, end = 2.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
      # Turn off interactive plotting
      fireSpread = list(.plotInitialTime = NA),
      caribouMovement = list(.plotInitialTime = NA),
      randomLandscapes = list(.plotInitialTime = NA)
    ),
    modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
    paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
                 outputPath = tmpdir),
    # Save final state of landscape and caribou
   outputs = data.frame(objectName = c("landscape", "caribou"), stringsAsFactors = FALSE)
  )

  # Example 1 - test alternative parameter values
  # Create an experiment - here, 2 x 2 x 2 (2 levels of 2 params in fireSpread,
  #    and 2 levels of 1 param in caribouMovement)

  # Here is a list of alternative values for each parameter. They are length one
  #   numerics here -- e.g., list(0.2, 0.23) for spreadprob in fireSpread module,
  #   but they can be anything, as long as it is a list.
  experimentParams <- list(fireSpread = list(spreadprob = list(0.2, 0.23),
                                             nFires = list(20, 10)),
                           caribouMovement = list(N = list(100, 1000)))

  sims <- experiment(mySim, params = experimentParams)

  # see experiment:
  attr(sims, "experiment")

  # Read in outputs from sims object
  FireMaps <- do.call(stack, lapply(1:NROW(attr(sims, "experiment")$expDesign),
                                    function(x) sims[[x]]$landscape$Fires))
  if (interactive()) Plot(FireMaps, new = TRUE)
```

```
# Or reload objects from files, useful if sim objects too large to store in RAM
caribouMaps <- lapply(sims, function(sim) {
  caribou <- readRDS(outputs(sim)$file[outputs(sim)$objectName == "caribou"])
})
names(caribouMaps) <- paste0("caribou", 1:8)
# Plot whole named list
if (interactive()) Plot(caribouMaps, size = 0.1)

# Example 2 - test alternative modules
# Example of changing modules, i.e., caribou with and without fires
# Create an experiment - here, 2 x 2 x 2 (2 levels of 2 params in fireSpread,
#    and 2 levels of 1 param in caribouMovement)
experimentModules <- list(
  c("randomLandscapes", "fireSpread", "caribouMovement"),
  c("randomLandscapes", "caribouMovement"))
sims <- experiment(mySim, modules = experimentModules)
attr(sims, "experiment")$expVals # shows 2 alternative experiment levels

# Example 3 - test alternative parameter values and modules
# Note, this isn't fully factorial because all parameters are not
#   defined inside smaller module list
sims <- experiment(mySim, modules = experimentModules, params = experimentParams)
attr(sims, "experiment")$expVals # shows 10 alternative experiment levels

# Example 4 - manipulate manipulate directory names -
#  "simNum" is special value for dirPrefix, it is converted to 1, 2, ...
sims <- experiment(mySim, params = experimentParams, dirPrefix = c("expt", "simNum"))
attr(sims, "experiment")$expVals # shows 8 alternative experiment levels, 24 unique
                                 #   parameter values

# Example 5 - doing replicate runs -
sims <- experiment(mySim, replicates = 2)
attr(sims, "experiment")$expDesign # shows 2 replicates of same experiment

# Example 6 - doing replicate runs, but within a sub-directory
sims <- experiment(mySim, replicates = 2, dirPrefix = c("expt"))
lapply(sims, outputPath) # shows 2 replicates of same experiment, within a sub directory

# Example 7 - doing replicate runs, of a complex, non factorial experiment.
# Here we do replication, parameter variation, and module variation all together.
# This creates 20 combinations.
# The experiment function tries to make fully factorial, but won't
# if all the levels don't make sense. Here, changing parameter values
# in the fireSpread module won't affect the simulation when the fireSpread
# module is not loaded:

# library(raster)
# beginCluster(20) # if you have multiple clusters available, use them here to save time
sims <- experiment(mySim, replicates = 2, params = experimentParams,
                   modules = experimentModules,
                   dirPrefix = c("expt", "simNum"))
# endCluster() # end the clusters
```

```
attr(sims, "experiment")

# Example 8 - Use replication to build a probability map.
# For this to be meaningful, we need to provide a fixed input landscape,
#   not a randomLandscape for each experiment level. So requires 2 steps.
# Step 1 - run randomLandscapes module twice to get 2 randomly
#  generated landscape maps. We will use 1 right away, and we will
#  use the two further below
mySimRL <- simInit(
  times = list(start = 0.0, end = 0.1, timeunit = "year"),
  params = list(
    .globals = list(stackName = "landscape"),
    # Turn off interactive plotting
    randomLandscapes = list(.plotInitialTime = NA)
  ),
  modules = list("randomLandscapes"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
               outputPath = file.path(tmpdir, "landscapeMaps1")),
 outputs = data.frame(objectName = "landscape", saveTime = 0, stringsAsFactors = FALSE)
)
# Run it twice to get two copies of the randomly generated landscape
mySimRLOut <- experiment(mySimRL, replicate = 2)

# extract one of the random landscapes, which will be passed into next as an object
landscape <- mySimRLOut[[1]]$landscape

# here we don't run the randomLandscapes module; instead we pass in a landscape
#  as an object, i.e., a fixed input
mySimNoRL <- simInit(
  times = list(start = 0.0, end = 1, timeunit = "year"), # only 1 year to save time
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
    # Turn off interactive plotting
    fireSpread = list(.plotInitialTime = NA),
    caribouMovement = list(.plotInitialTime = NA)
  ),
  modules = list("fireSpread", "caribouMovement"), # No randomLandscapes modules
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
               outputPath = tmpdir),
  objects = c("landscape"), # Pass in the object here
 # Save final state (the default if saveTime is not specified) of landscape and caribou
 outputs = data.frame(objectName = c("landscape", "caribou"), stringsAsFactors = FALSE)
)

# Put outputs into a specific folder to keep them easy to find
outputPath(mySimNoRL) <- file.path(tmpdir, "example8")
sims <- experiment(mySimNoRL, replicates = 8) # Run experiment
attr(sims, "experiment") # shows the experiment, which in this case is just replicates

# list all files that were saved called 'landscape'
landscapeFiles <- dir(outputPath(mySimNoRL), recursive = TRUE, pattern = "landscape",
                      full.names = TRUE)
```

```
# Can read in Fires layers from disk since they were saved, or from the sims
#  object
# Fires <- lapply(sims, function(x) x$landscape$Fires) %>% stack
Fires <- lapply(landscapeFiles, function(x) readRDS(x)$Fires) %>% stack()
Fires[Fires > 0] <- 1 # convert to 1s and 0s
fireProb <- sum(Fires) / nlayers(Fires) # sum them and convert to probability
if (interactive()) Plot(fireProb, new = TRUE)

# Example 9 - Pass in inputs, i.e., input data objects taken from disk
#  Here, we, again, don't provide randomLandscapes module, so we need to
#  provide an input stack called lanscape. We point to the 2 that we have
#  saved to disk in Example 8
mySimInputs <- simInit(
  times = list(start = 0.0, end = 2.0, timeunit = "year"),
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
    # Turn off interactive plotting
    fireSpread = list(.plotInitialTime = NA),
    caribouMovement = list(.plotInitialTime = NA)
  ),
  modules = list("fireSpread", "caribouMovement"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
               outputPath = tmpdir),
  # Save final state of landscape and caribou
 outputs = data.frame(objectName = c("landscape", "caribou"), stringsAsFactors = FALSE)
)
landscapeFiles <- dir(tmpdir, pattern = "landscape_year0", recursive = TRUE, full.names = TRUE)

# Varying inputs files - This could be combined with params, modules, replicates also
outputPath(mySimInputs) <- file.path(tmpdir, "example9")
sims <- experiment(mySimInputs,
                   inputs = lapply(landscapeFiles, function(filenames) {
                     data.frame(file = filenames, loadTime = 0,
                                objectName = "landscape",
                                stringsAsFactors = FALSE)
                   })
 )

# load in experimental design object
experiment <- load(file = file.path(tmpdir, "example9", "experiment.RData")) %>% get()
print(experiment) # shows input files and details

# Example 10 - Use a very simple output dir name using substrLength = 0,
#   i.e., just the simNum is used for outputPath of each spades call
outputPath(mySim) <- file.path(tmpdir, "example10")
sims <- experiment(mySim, modules = experimentModules, replicates = 2,
                   substrLength = 0)
lapply(sims, outputPath) # shows that the path is just the simNum
experiment <- load(file = file.path(tmpdir, "example10", "experiment.RData")) %>% get()
print(experiment) # shows input files and details

# Example 11 - use clearSimEnv = TRUE to remove objects from simList
# This will shrink size of return object, which may be useful because the
```

```
  #  return from experiment function may be a large object (it is a list of
  # simLists). To see size of a simList, you have to look at the objects
  #  contained in the  envir(simList).  These can be obtained via objs(sim)
  sapply(sims, function(x) object.size(objs(x))) %>% sum + object.size(sims)
  # around 3 MB
  # rerun with clearSimEnv = TRUE
  sims <- experiment(mySim, modules = experimentModules, replicates = 2,
                     substrLength = 0, clearSimEnv = TRUE)
  sapply(sims, function(x) object.size(objs(x))) %>% sum + object.size(sims)
  # around 250 kB, i.e., all the simList contents except the objects.

  # Example 12 - pass in objects
  experimentObj <- list(landscape = lapply(landscapeFiles, readRDS) %>%
                                    setNames(paste0("landscape", 1:2)))
  # Pass in this list of landscape objects
  sims <- experiment(mySimNoRL, objects = experimentObj)

  # Remove all temp files
  unlink(tmpdir, recursive = TRUE)
}
```

`F` *`F` allows the user to access functions from their module. It should be unnecessary to use this as namespacing should take care of it. But, if the user wants to specify a function this way, it will work too. To access a function from within a module, you can use* `F(sim)$functionName`.

Description

`F` allows the user to access functions from their module. It should be unnecessary to use this as namespacing should take care of it. But, if the user wants to specify a function this way, it will work too. To access a function from within a module, you can use `F(sim)$functionName`.

Usage

```
F(sim, module = NULL, functionName = NULL)

## S4 method for signature '.simList'
F(sim, module = NULL, functionName = NULL)
```

Arguments

`sim`	A `simList` object from which to extract element(s) or in which to replace element(s).
`module`	Optional character string indicating which module params should come from.
`functionName`	Optional character string indicating which function is desired.

fileExt *Extract the file extension of a file*

Description

Extract the file extension of a file

Usage

```
fileExt(x)
```

Arguments

x	List or character vector of file names.

Value

A character vector of file extensions.

Author(s)

Eliot McIntire and Alex Chubaty

fileName *Extract filename (without extension) of a file*

Description

Extract filename (without extension) of a file

Usage

```
fileName(x)
```

Arguments

x	List or character vector

Value

A character vector.

Author(s)

Eliot McIntire

getModuleVersion *Find the latest module version from a SpaDES module repository*

Description

Modified from http://stackoverflow.com/a/25485782/1380598.

Usage

```
getModuleVersion(name, repo)

## S4 method for signature 'character,character'
getModuleVersion(name, repo)

## S4 method for signature 'character,missing'
getModuleVersion(name)
```

Arguments

name	Character string giving the module name.
repo	GitHub repository name. Default is "PredictiveEcology/SpaDES-modules", which is specified by the global option spades.moduleRepo.

Author(s)

Alex Chubaty

globals *Get and set simulation globals.*

Description

globals, and the alias G, accesses or sets the "globals" in the simList. This currently is not an explicit slot in the simList, but it is a .globals element in the params slot of the simList.

Usage

```
globals(sim)

## S4 method for signature '.simList'
globals(sim)

globals(sim) <- value

## S4 replacement method for signature '.simList'
```

```
globals(sim) <- value

G(sim)

## S4 method for signature '.simList'
G(sim)

G(sim) <- value

## S4 replacement method for signature '.simList'
G(sim) <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
value	The object to be stored at the slot.

See Also

SpaDES.core-package, specifically the section 1.2.1 on Simulation Parameters.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

initialize,simList-method

Generate a simList *object*

Description

Given the name or the definition of a class, plus optionally data to be included in the object, new returns an object from that class.

Usage

```
## S4 method for signature 'simList'
initialize(.Object)
```

Arguments

.Object	A simList object.

inputs *Inputs and outputs*

Description

These functions are one of three mechanisms to add the information about which input files to load in a spades call and the information about which output files to save. 1) As arguments to a simInit call. Specifically, inputs or outputs. See ?simInit. 2) With the inputs(simList) or outputs(simList) function call. 3) By adding a function called .inputObjects inside a module, which will be executed during the simInit call. This last way is the most "modular" way to create default data sets for your model. See below for more details.

inputArgs and outputArgs are ways to specify any arguments that are needed for file loading and file saving. This is still somewhat experimental.

Usage

```
inputs(sim)

## S4 method for signature '.simList'
inputs(sim)

inputs(sim) <- value

## S4 replacement method for signature '.simList'
inputs(sim) <- value

outputs(sim)

## S4 method for signature '.simList'
outputs(sim)

outputs(sim) <- value

## S4 replacement method for signature '.simList'
outputs(sim) <- value

inputArgs(sim)

## S4 method for signature '.simList'
inputArgs(sim)

inputArgs(sim) <- value

## S4 replacement method for signature '.simList'
inputArgs(sim) <- value
```

```
outputArgs(sim)

## S4 method for signature '.simList'
outputArgs(sim)

outputArgs(sim) <- value

## S4 replacement method for signature '.simList'
outputArgs(sim) <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
value	The object to be stored at the slot. See Details.

Details

Accessor functions for the inputs and outputs slots in a simList object.

Value

Returns or sets the value(s) of the input or output slots in the simList object.

inputs function or argument in simInit

inputs accepts a data.frame, with up to 7 columns. Columns are:

file	required, a character string indicating the file path. There is no default.
objectName	optional, character string indicating the name of the object that the loaded file will be assigned to in the simLis
fun	optional, a character string indicating the function to use to load that file. Defaults to the known extensions in S
package	optional character string indicating the package in which to find the fun);
loadTime	optional numeric, indicating when in simulation time the file should be loaded. The default is the highest priori
interval	optional numeric, indicating at what interval should this same exact file be reloaded from disk, e.g,. 10 would r
arguments	is a list of lists of named arguments, one list for each fun. For example, if fun="raster", arguments = list

Currently, only file is required. All others will be filled with defaults if not specified.

See the modules vignette for more details (browseVignettes("SpaDES.core")).

.inputObjects function placed inside module

Any code placed inside a function called .inputObjects will be run during the simInit for the purpose of creating any objects required by this module, i.e., objects identified in the inputObjects element of defineModule. This is useful if there is something required before simulation to produce the module object dependencies, including such things as downloading default datasets, e.g., downloadData('LCC2005', modulePath(sim)). Nothing should be created here that does not create an named object in inputObjects. Any other initiation procedures should be put in the "init" eventType of the doEvent function. Note: the module developer can use 'sim$.userSuppliedObjNames'

inside the function to selectively skip unnecessary steps because the user has provided those inputObjects in the simInit call. e.g., the following code would look to see if the user had passed `defaultColor` into during `simInit`. If the user had done this, then this function would not override that value with 'red'. If the user has not passed in a value for `defaultColor`, then the module will get it here:

```
if (!('defaultColor' %in% sim$.userSuppliedObjNames)) {  sim$defaultColor <- 'red'
}
```

outputs function or argument in `simInit`

`outputs` accepts a data.frame similar to the `inputs` data.frame, but with up to 6 columns.

`objectName`	required, character string indicating the name of the object in the `simList` that will be saved to disk (without th
`file`	optional, a character string indicating the file path to save to. The default is to concatenate `objectName` with the
`fun`	optional, a character string indicating the function to use to save that file. The default is `saveRDS`
`package`	optional character string indicating the package in which to find the `fun`);
`saveTime`	optional numeric, indicating when in simulation time the file should be saved. The default is the lowest priority
`arguments`	is a list of lists of named arguments, one list for each `fun`. For example, if `fun = "write.csv"`, `arguments =`

See the modules vignette for more details (`browseVignettes("SpaDES.core")`).

Note

The automatic file type handling only adds the correct extension from a given `fun` and `package`. It does not do the inverse, from a given extension find the correct `fun` and `package`.

See Also

`SpaDES.core-package`, specifically the section 1.2.2 on loading and saving.

Other functions to access elements of a `simList` object: `.addDepends`, `doEvent.checkpoint`, `envir`, `events`, `globals`, `ls.simList`, `ls.str.simList`, `modules`, `objs`, `packages`, `params`, `paths`, `progressInterval`, `times`

Examples

```
#######################
# inputs
#######################

# Start with a basic empty simList
sim <- simInit()

test <- 1:10
library(igraph) # for %>%
library(reproducible) # for checkPath
tmpdir <- file.path(tempdir(), "inputs") %>% checkPath(create = TRUE)
tmpFile <- file.path(tmpdir, "test.rds")
saveRDS(test, file = tmpFile)
inputs(sim) <- data.frame(file = tmpFile) # using only required column, "file"
```

```
inputs(sim) # see that it is not yet loaded, but when it is scheduled to be loaded
simOut <- spades(sim)
inputs(simOut) # confirm it was loaded
simOut$test

# can put data.frame for inputs directly inside simInit call
allTifs <- dir(system.file("maps", package = "quickPlot"),
               full.names = TRUE, pattern = "tif")

# next: .objectNames are taken from the filenames (without the extension)
# This will load all 5 tifs in the SpaDES sample directory, using
#   the raster fuction in the raster package, all at time = 0
if (require("rgdal", quietly = TRUE)) {
  sim <- simInit(
    inputs = data.frame(
      files = allTifs,
      functions = "raster",
      package = "raster",
      loadTime = 0,
      stringsAsFactors = FALSE)
    )

  ##############################
  #A fully described inputs object, including arguments:
  files <- dir(system.file("maps", package = "quickPlot"),
               full.names = TRUE, pattern = "tif")
  # arguments must be a list of lists. This may require I() to keep it as a list
  #   once it gets coerced into the data.frame.
  arguments = I(rep(list(native = TRUE), length(files)))
  filelist = data.frame(
     objectName = paste0("Maps", 1:5),
     files = files,
     functions = "raster::raster",
     arguments = arguments,
     loadTime = 0,
     intervals = c(rep(NA, length(files) - 1), 10)
  )
  inputs(sim) <- filelist
  spades(sim)
}


# Clean up after
unlink(tmpdir, recursive = TRUE)

########################
# outputs
########################

library(igraph) # for %>%
tmpdir <- file.path(tempdir(), "outputs") %>% checkPath(create = TRUE)
tmpFile <- file.path(tmpdir, "temp.rds")
tempObj <- 1:10
```

```
# Can add data.frame of outputs directly into simInit call
sim <- simInit(objects = c("tempObj"),
               outputs = data.frame(objectName = "tempObj"),
               paths = list(outputPath = tmpdir))
outputs(sim) # To see what will be saved, when, what filename
sim <- spades(sim)
outputs(sim) # To see that it was saved, when, what filename

# Also can add using assignment after a simList object has been made
sim <- simInit(objects = c("tempObj"), paths = list(outputPath = tmpdir))
outputs(sim) <- data.frame(objectName = "tempObj", saveTime = 1:10)
sim <- spades(sim)
outputs(sim) # To see that it was saved, when, what filename.

# can do highly variable saving
tempObj2 <- paste("val",1:10)
df1 <- data.frame(col1 = tempObj, col2 = tempObj2)
sim <- simInit(objects = c("tempObj", "tempObj2", "df1"),
  paths=list(outputPath = tmpdir))
outputs(sim) = data.frame(
     objectName = c(rep("tempObj", 2), rep("tempObj2", 3), "df1"),
     saveTime = c(c(1,4), c(2,6,7), end(sim)),
     fun = c(rep("saveRDS", 5), "write.csv"),
     package = c(rep("base", 5), "utils"),
     stringsAsFactors = FALSE)
# since write.csv has a default of adding a column, x, with rownames, must add additional
#   argument for 6th row in data.frame (corresponding to the write.csv function)
outputArgs(sim)[[6]] <- list(row.names=FALSE)
sim <- spades(sim)
outputs(sim)

# read one back in just to test it all worked as planned
newObj <- read.csv(dir(tmpdir, pattern = "second10.csv", full.name = TRUE))
newObj

# using saving with SpaDES-aware methods
# To see current ones SpaDES can do
.saveFileExtensions()

library(raster)
if (require(rgdal)) {
  ras <- raster(ncol = 4, nrow = 5)
  ras[] <- 1:20

  sim <- simInit(objects = c("ras"), paths = list(outputPath = tmpdir))
  outputs(sim) = data.frame(
    file = "test",
    fun = "writeRaster",
    package = "raster",
    objectName = "ras",
    stringsAsFactors = FALSE)
```

```
  outputArgs(sim)[[1]] <- list(format = "GTiff") # see ?raster::writeFormats
  simOut <- spades(sim)
  outputs(simOut)
  newRas <- raster(dir(tmpdir, full.name = TRUE, pattern = ".tif"))
  all.equal(newRas, ras) # Should be TRUE
}
# Clean up after
unlink(tmpdir, recursive = TRUE)
```

---

inSeconds *Convert time units*

Description

In addition to using the `lubridate` package, some additional functions to work with times are provided.

This function takes a numeric with a "unit" attribute and converts it to another numeric with a different time attribute. If the units passed to argument `units` are the same as `attr(time, "unit")`, then it simply returns input `time`.

Usage

```
inSeconds(unit, envir, skipChecks = FALSE)

convertTimeunit(time, unit, envir, skipChecks = FALSE)

.spadesTimes

spadesTimes()

checkTimeunit(unit, envir)

## S4 method for signature 'character,missing'
checkTimeunit(unit, envir)

## S4 method for signature 'character,environment'
checkTimeunit(unit, envir)
```

Arguments

`unit`	Character. One of the time units used in `SpaDES` or user defined time unit, given as the unit name only. See details.
`envir`	An environment. This is where to look up the function definition for the time unit. See details.
`skipChecks`	For speed, the internal checks for classes and missingness can be skipped. Default `FALSE`.
`time`	Numeric. With a unit attribute, indicating the time unit of the input numeric. See Details.

Format

An object of class `character` of length 6.

Details

Current pre-defined units are found within the `spadesTimes()` function. The user can define a new unit. The unit name can be anything, but the function definition must be of the form `"dunitName"`, e.g., `dyear` or `dfortnight`. The unit name is the part without the `d` and the function name definition includes the `d`. This new function, e.g., `dfortnight <- function(x) lubridate::duration(dday(14))` can be placed anywhere in the search path or in a module.

Because of R scoping, if `envir` is a simList environment, then this function will search there first, then up the current `search()` path. Thus, it will find a user defined or module defined unit before a SpaDES unit. This means that a user can override the `dyear` given in SpaDES, for example, which is 365.25 days, with `dyear <- function(x) lubridate::duration(dday(365))`.

If `time` has no `units` attribute, then it is assumed to be seconds.

Value

A numeric vector of length 1, with `unit` attribute set to "seconds".

Author(s)

Alex Chubaty & Eliot McIntire

Eliot McIntire

`loadPackages` *Load packages.*

Description

Load and optionally install additional packages.

Usage

```
loadPackages(packageList, install = FALSE, quiet = TRUE)

## S4 method for signature 'character'
loadPackages(packageList, install = FALSE,
  quiet = TRUE)

## S4 method for signature 'list'
loadPackages(packageList, install = FALSE, quiet = TRUE)

## S4 method for signature '`NULL`'
loadPackages(packageList, install = FALSE, quiet = TRUE)
```

Arguments

packageList	A list of character strings specifying the names of packages to be loaded.
install	Logical flag. If required packages are not already installed, should they be installed?
quiet	Logical flag. Should the final "packages loaded" message be suppressed?

Value

Specified packages are loaded and attached using `require()`, invisibly returning a logical vector of successes.

Author(s)

Alex Chubaty

See Also

`require`.

Examples

```
## Not run:
  pkgs <- list("raster", "lme4")
  loadPackages(pkgs) # loads packages if installed
  loadPackages(pkgs, install = TRUE) # loads packages after installation (if needed)

## End(Not run)
```

---

ls.simList *List simulation objects*

Description

Return a vector of character strings giving the names of the objects in the specified simulation environment. Can be used with a `simList` object, because the method for this class is simply a wrapper for calling `ls` on the simulation environment stored in the `simList` object.

Usage

```
ls.simList(name)

## S4 method for signature 'simList'
ls(name)

objects.simList(name)

## S4 method for signature 'simList'
objects(name)
```

Arguments

name	A simList object.

See Also

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

ls.str.simList *List simulation objects and their structure*

Description

A variation of applying str to each matched name. Can be used with a simList object, because the method for this class is simply a wrapper for calling ls on the simulation environment stored in the simList object.

export

Usage

```
ls.str.simList(name)

## S4 method for signature 'missing,simList'
ls.str(name)

## S4 method for signature 'simList,missing'
ls.str(pos)
```

Arguments

name	A simList object.
pos	A simList object, used only if name not provided.

See Also

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, modules, objs, packages, params, paths, progressInterval, times

maxTimeunit *Determine the largest timestep unit in a simulation*

Description

Determine the largest timestep unit in a simulation

Usage

```
maxTimeunit(sim)

## S4 method for signature 'simList'
maxTimeunit(sim)
```

Arguments

sim	A simList simulation object.

Value

The timeunit as a character string. This defaults to NA if none of the modules has explicit units.

Author(s)

Eliot McIntire and Alex Chubaty

minTimeunit *Determine the smallest timeunit in a simulation*

Description

When modules have different timeunit, SpaDES automatically takes the smallest (e.g., "second") as the unit for a simulation.

Usage

```
minTimeunit(sim)

## S4 method for signature 'simList'
minTimeunit(sim)

## S4 method for signature 'list'
minTimeunit(sim)
```

Arguments

sim	A simList simulation object.

Value

The timeunit as a character string. This defaults to "second" if none of the modules has explicit units.

Author(s)

Eliot McIntire

`moduleCoverage` *Calculate module coverage of unit tests*

Description

Calculate the test coverage by unit tests for the module and its functions.

Usage

```
moduleCoverage(name, path)

## S4 method for signature 'character,character'
moduleCoverage(name, path)

## S4 method for signature 'character,missing'
moduleCoverage(name)
```

Arguments

`name`	Character string. The module's name.
`path`	Character string. The path to the module directory (default is the current working directory).

Value

Return a list of two coverage objects and two data.table objects. The two coverage objects are named 'moduleCoverage' and 'functionCoverage'. The 'moduleCoverage' object contains the percent value of unit test coverage for the module. The 'functionCoverage' object contains percentage values for unit test coverage for each function defined in the module. Please use `shine` to view the coverage information. Two data.tables give the information of all the tested and untested functions in the module.

Note

When running this function, the test files must be strictly placed in the '`tests/testthat/`' directory under module path. To automatically generate this folder, please set `unitTests = TRUE` when creating a new module using `newModule`. To accurately test your module, the test filename must follw the format `test-functionName.R`.

Author(s)

Yong Luo

See Also

newModule.

Examples

```
## Not run:
 library(igraph) # for %>%
 library(SpaDES.core)
 tmpdir <- file.path(tempdir(), "coverage")
 modulePath <- file.path(tmpdir, "Modules") %>% checkPath(create = TRUE)
 moduleName <- "forestAge" # sample module to test
 downloadModule(name = moduleName, path = modulePath) # download sample module
 testResults <- moduleCoverage(name = moduleName, path = modulePath)
 shine(testResults$moduleCoverage)
 shine(testResults$functionCoverage)
 unlink(tmpdir, recursive = TRUE)

## End(Not run)
```

---

moduleDiagram *Simulation module dependency diagram*

Description

Create a network diagram illustrating the simplified module dependencies of a simulation. Offers a less detailed view of specific objects than does plotting the depsEdgeList directly with objectDiagram.

Usage

```
moduleDiagram(sim, type, showParents, ...)

## S4 method for signature 'simList,character,logical'
moduleDiagram(sim, type, showParents, ...)

## S4 method for signature 'simList,missing,ANY'
moduleDiagram(sim, type, showParents, ...)
```

Arguments

sim	A simList object (typically corresponding to a completed simulation).
type	Character string, either "rgl" for igraph::rglplot or "tk" for igraph::tkplot. Default missing, which uses regular plot.

showParents	Logical. If TRUE, then any children that are grouped into parent modules will be grouped together by colored blobs. Internally, this is calling moduleGraph. Default FALSE.
...	Additional arguments passed to plotting function specified by type.

Value

Plots module dependency diagram.

Author(s)

Alex Chubaty

See Also

igraph, moduleGraph for a version that accounts for parent and children module structure.

---

moduleGraph *Build a module dependency graph*

Description

This is still experimental, but this will show the hierarchical structure of parent and children modules and return a list with an igraph object and an igraph communities object, showing the groups. Currently only tested with relatively simple structures.

Usage

```
moduleGraph(sim, plot, ...)

## S4 method for signature 'simList,logical'
moduleGraph(sim, plot, ...)

## S4 method for signature 'simList,missing'
moduleGraph(sim, plot, ...)
```

Arguments

sim	A simList object.
plot	Logical indicating whether the edgelist (and subsequent graph) will be used for plotting. If TRUE, duplicated rows (i.e., multiple object dependencies between modules) are removed so that only a single arrow is drawn connecting the modules. Default is FALSE.
...	Arguments passed to Plot

Value

A list with 2 elements, an igraph object and an igraph communities object.

@importFrom igraph graph_from_data_frame cluster_optimal edges # already with import igraph

Author(s)

Eliot McIntire

See Also

moduleDiagram

---

moduleMetadata *Parse and extract module metadata*

Description

Parse and extract module metadata

Usage

```
moduleMetadata(module, path, sim)

## S4 method for signature 'character,character,missing'
moduleMetadata(module, path)

## S4 method for signature 'character,missing,missing'
moduleMetadata(module)

## S4 method for signature 'ANY,missing,simList'
moduleMetadata(module, sim)
```

Arguments

module	Character string. Your module's name.
path	Character string specifying the file path to modules directory. Default is to use the spades.modulePath option.
sim	A simList simulation object, generally produced by simInit.

Value

A list of module metadata, matching the structure in defineModule.

Author(s)

Alex Chubaty

See Also

defineModule

Examples

```
path <- system.file("sampleModules", package = "SpaDES.core")
sampleModules <- dir(path)
x <- moduleMetadata(sampleModules[3], path)

# using simList
mySim <- simInit(
   times = list(start = 2000.0, end = 2002.0, timeunit = "year"),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
)
moduleMetadata(sim = mySim)
```

---

modules *Simulation modules and dependencies*

Description

Accessor functions for the depends and modules slots in a simList object. These are included for advanced users.

depends	List of simulation module dependencies. (advanced)
modules	List of simulation modules to be loaded. (advanced)
inputs	List of loaded objects used in simulation. (advanced)

Usage

```
modules(sim, hidden = FALSE)

## S4 method for signature '.simList'
modules(sim, hidden = FALSE)

modules(sim) <- value

## S4 replacement method for signature '.simList'
modules(sim) <- value

depends(sim)

## S4 method for signature '.simList'
```

```
depends(sim)

depends(sim) <- value

## S4 replacement method for signature '.simList'
depends(sim) <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
hidden	Logical. If TRUE, show the default core modules.
value	The object to be stored at the slot.

Details

Currently, only get and set methods are defined. Subset methods are not.

Value

Returns or sets the value of the slot from the simList object.

Author(s)

Alex Chubaty

See Also

SpaDES.core-package, specifically the section 1.2.7 on Modules and dependencies.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, objs, packages, params, paths, progressInterval, times

---

moduleVersion *Parse and extract a module's version*

Description

Parse and extract a module's version

Usage

```
moduleVersion(module, path, sim)

## S4 method for signature 'character,character,missing'
moduleVersion(module, path)

## S4 method for signature 'character,missing,missing'
moduleVersion(module)

## S4 method for signature 'character,missing,simList'
moduleVersion(module, sim)
```

Arguments

module	Character string. Your module's name.
path	Character string specifying the file path to modules directory. Default is to use the spades.modulePath option.
sim	A simList simulation object, generally produced by simInit.

Value

numeric_version indicating the module's version.

Author(s)

Alex Chubaty

See Also

moduleMetadata

Examples

```
path <- system.file("sampleModules", package = "SpaDES.core")

# using filepath
moduleVersion("caribouMovement", path)

# using simList
mySim <- simInit(
   times = list(start = 2000.0, end = 2002.0, timeunit = "year"),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = path)
)
moduleVersion("caribouMovement", sim = mySim)
```

newModule *Create new module from template*

Description

Autogenerate a skeleton for a new SpaDES module, a template for a documentation file, a citation file, a license file, a 'README.txt' file, and a folder that contains unit tests information. The newModuleDocumentation will not generate the module file, but will create the other files.

Usage

```
newModule(name, path, ...)

## S4 method for signature 'character,character'
newModule(name, path, ...)

## S4 method for signature 'character,missing'
newModule(name, path, ...)
```

Arguments

name	Character string specifying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
...	Additional arguments. Currently, only the following are supported:
	open. Logical. Should the new module file be opened after creation? Default TRUE.
	unitTests. Logical. Should the new module include unit test files? Default TRUE. Unit testing relies on the testthat package.
	type. Character string specifying one of "child" (default), or "parent".
	children. Required when type = "parent". A character vector specifying the names of child modules.

Details

All files will be created within a subdirectory named name within the path:

- path/

```
- name/
- R/               # contains additional module R scripts
- data/            # directory for all included data
    * CHECKSUMS.txt  # contains checksums for data files
- tests/           # contains unit tests for module code
- citation.bib     # bibtex citation for the module
- LICENSE.txt      # describes module's legal usage
- README.txt       # provide overview of key aspects
- name.R           # module code file (incl. metadata)
- name.Rmd         # documentation, usage info, etc.
```

Value

Nothing is returned. The new module file is created at '`path/name.R`', as well as ancillary files for documentation, citation, '`LICENSE`', '`README`', and '`tests`' directory.

Note

On Windows there is currently a bug in RStudio that prevents the editor from opening when `file.edit` is called. Similarly, in RStudio on macOS, there is an issue opening files where they are opened in an overlayed window rather than a new tab. `file.edit` does work if the user types it at the command prompt. A message with the correct lines to copy and paste is provided.

Author(s)

Alex Chubaty and Eliot McIntire

See Also

Other module creation helpers: `newModuleCode`, `newModuleDocumentation`, `newModuleTests`

Examples

```
## Not run:
  ## create a "myModule" module in the "modules" subdirectory.
  newModule("myModule", "modules")

  ## create a new parent module in the "modules" subdirectory.
  newModule("myParentModule", "modules", type = "parent", children = c("child1", "child2"))

## End(Not run)
```

`newModuleCode` *Create new module code file*

Description

Create new module code file

Usage

```
newModuleCode(name, path, open, type, children)

## S4 method for signature 'character,character,logical,character,character'
newModuleCode(name,
  path, open, type, children)
```

Arguments

`name`	Character string specifying the name of the new module.
`path`	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
`open`	Logical. Should the new module file be opened after creation? Default `TRUE`.
`type`	Character string specifying one of `"child"` (default), or `"parent"`.
`children`	Required when `type = "parent"`. A character vector specifying the names of child modules.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Other module creation helpers: `newModuleDocumentation`, `newModuleTests`, `newModule`

`newModuleDocumentation` *Create new module documentation*

Description

Create new module documentation

Usage

```
newModuleDocumentation(name, path, open, type, children)


  ## S4 method for signature 'character,character,logical,character,character'
newModuleDocumentation(name,
  path, open, type, children)

## S4 method for signature 'character,missing,logical,ANY,ANY'
newModuleDocumentation(name, open)

## S4 method for signature 'character,character,missing,ANY,ANY'
newModuleDocumentation(name,
  path)

## S4 method for signature 'character,missing,missing,ANY,ANY'
newModuleDocumentation(name)
```

Arguments

name	Character string specifying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
open	Logical. Should the new module file be opened after creation? Default TRUE.
type	Character string specifying one of "child" (default), or "parent".
children	Required when type = "parent". A character vector specifying the names of child modules.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Other module creation helpers: newModuleCode, newModuleTests, newModule

---

newModuleTests *Create template testing structures for new modules*

Description

Create template testing structures for new modules

Usage

```
newModuleTests(name, path, open)

## S4 method for signature 'character,character,logical'
newModuleTests(name, path, open)
```

Arguments

name	Character string specifying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
open	Logical. Should the new module file be opened after creation? Default TRUE.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Other module creation helpers: newModuleCode, newModuleDocumentation, newModule

---

newProgressBar *Progress bar*

Description

Shows a progress bar that is scaled to simulation end time.

Usage

```
newProgressBar(sim)

setProgressBar(sim)
```

Arguments

sim	A simList simulation object.

Details

The progress bar object is stored in a separate environment, #' .pkgEnv.

Author(s)

Alex Chubaty and Eliot McIntire

objectDiagram *Simulation object dependency diagram*

Description

Create a sequence diagram illustrating the data object dependencies of a simulation. Offers a more detailed view of specific objects than does plotting the `depsEdgeList` directly with `moduleDiagram`.

Usage

```
objectDiagram(sim, ...)

## S4 method for signature 'simList'
objectDiagram(sim, ...)
```

Arguments

`sim`	A `simList` object (typically corresponding to a completed simulation).
`...`	Additional arguments passed to `mermaid`. Useful for specifying `height` and `width`.

Value

Plots a sequence diagram, invisibly returning a `mermaid` object.

Author(s)

Alex Chubaty

See Also

`mermaid`.

objs *Extract or replace an object from the simulation environment*

Description

The `[[` and `$` operators provide "shortcuts" for accessing objects in the simulation environment. I.e., instead of using `envir(sim)$object` or `envir(sim)[["object"]]`, one can simply use `sim$object` or `sim[["object"]]`.

Usage

```
objs(sim, ...)

## S4 method for signature 'simList'
objs(sim, ...)

objs(sim) <- value

## S4 replacement method for signature 'simList'
objs(sim) <- value

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

## S4 replacement method for signature 'simList,ANY,ANY,ANY'
x[[i]] <- value

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

## S4 replacement method for signature 'simList'
x$name <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
value	Any R object.
x	A simList object from which to extract element(s) or in which to replace element(s).
i, j, ...	Indices specifying elements to extract or replace.
drop	not implemented.
name	A literal character string or a name.

Details

objs can take ... arguments passed to ls, allowing, e.g. all.names=TRUE objs<- requires takes a named list of values to be assigned in the simulation environment.

Value

Returns or sets a list of objects in the simList environment.

See Also

SpaDES.core-package, specifically the section 1.2.1 on Simulation Parameters.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, packages, params, paths, progressInterval, times

---

openModules *Open all modules nested within a base directory*

Description

This is just a convenience wrapper for opening several modules at once, recursively. A module is defined as any file that ends in .R or .r and has a directory name identical to its filename. Thus, this must be case sensitive.

Usage

```
openModules(name, path)

## S4 method for signature 'character,character'
openModules(name, path)

## S4 method for signature 'missing,missing'
openModules()

## S4 method for signature 'missing,character'
openModules(path)

## S4 method for signature 'character,missing'
openModules(name)

## S4 method for signature 'simList,missing'
openModules(name)
```

Arguments

name	Character vector with names of modules to open. If missing, then all modules will be opened within the basedir.
path	Character string of length 1. The base directory within which there are only module subdirectories.

Value

Nothing is returned. All file are open via file.edit.

Note

On Windows there is currently a bug in RStudio that prevents the editor from opening when file.edit is called. file.edit does work if the user types it at the command prompt. A message with the correct lines to copy and paste is provided.

Author(s)

Eliot McIntire

Examples

```
## Not run: openModules("~\SpaDESModules")
```

---

packages *Get module or simulation package dependencies*

Description

Get module or simulation package dependencies

Usage

```
packages(sim, ...)

## S4 method for signature '.simList'
packages(sim, ...)

## S4 method for signature 'missing'
packages(sim, ...)
```

Arguments

sim	A simList object.
...	Additional arguments. Currently only module, specifying the name of a module, and filename, specifying a module filename, are supported.

Value

A sorted character vector of package names.

Author(s)

Alex Chubaty

See Also

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, params, paths, progressInterval, times

paddedFloatToChar *Convert numeric to character with padding*

Description

Convert numeric to character with padding

Usage

```
paddedFloatToChar(x, padL = ceiling(log10(x + 1)), padR = 3, pad = "0")
```

Arguments

x	numeric. Number to be converted to character with padding
padL	numeric. Desired number of digits on left side of decimal. If not enough, pad will be used to pad.
padR	numeric. Desired number of digits on right side of decimal. If not enough, pad will be used to pad.
pad	character to use as padding (nchar(pad)==1 must be TRUE). Passed to stri_pad

Value

Character string representing the filename.

Author(s)

Eliot McIntire and Alex Chubaty

Examples

```
paddedFloatToChar(1.25)
paddedFloatToChar(1.25, padL = 3, padR = 5)
```

params *Get and set simulation parameters.*

Description

params and P access the parameter slot in the simList. params has a replace method, so can be used to update a parameter value.

P is a concise way to access parameters within a module. It works more like a namespaced function in the sense that the module from which it is called is the default place it will look for the parameter. To access a parameter from within a module, you can use P(sim)$paramName instead of params(sim)$moduleName$paramName

Usage

```
params(sim)

## S4 method for signature '.simList'
params(sim)

params(sim) <- value

## S4 replacement method for signature '.simList'
params(sim) <- value

P(sim, module = NULL, param = NULL)

## S4 method for signature '.simList'
P(sim, module = NULL, param = NULL)

parameters(sim, asDF = FALSE)

## S4 method for signature '.simList'
parameters(sim, asDF = FALSE)
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
value	The object to be stored at the slot.
module	Optional character string indicating which module params should come from.
param	Optional character string indicating which parameter to choose.
asDF	Logical. For parameters, if TRUE, this will produce a single data.frame of all model parameters. If FALSE, then it will return a data.frame with 1 row for each parameter within nested lists, with the same structure as params.

Value

Returns or sets the value of the slot from the simList object.

Note

The differences between P, params and being explicit with passing arguments are mostly a question of speed and code compactness. The computationally fastest way to get a parameter is to specify moduleName and parameter name, as in: `P(sim, "moduleName", "paramName")` (replacing moduleName and paramName with your specific module and parameter names), but it is more verbose than P(sim)$paramName. Note: the important part for speed (e.g., 2-4x faster) is specifying the moduleName. Specifying the parameter name is <5

See Also

SpaDES.core-package, specifically the section 1.2.1 on Simulation parameters.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, paths, progressInterval, times

Examples

```
modules <- list("randomLandscapes")
paths <- list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
mySim <- simInit(modules = modules, paths = paths,
                 params = list(.globals = list(stackName = "landscape")))
parameters(mySim)
```

---

paths *Specify paths for modules, inputs, and outputs*

Description

Accessor functions for the paths slot in a simList object.

Usage

```
paths(sim)

## S4 method for signature '.simList'
paths(sim)

paths(sim) <- value

## S4 replacement method for signature '.simList'
paths(sim) <- value

cachePath(sim)

## S4 method for signature '.simList'
cachePath(sim)

cachePath(sim) <- value

## S4 replacement method for signature '.simList'
cachePath(sim) <- value

inputPath(sim)

## S4 method for signature '.simList'
```

```
inputPath(sim)

inputPath(sim) <- value

## S4 replacement method for signature '.simList'
inputPath(sim) <- value

outputPath(sim)

## S4 method for signature '.simList'
outputPath(sim)

outputPath(sim) <- value

## S4 replacement method for signature '.simList'
outputPath(sim) <- value

modulePath(sim)

## S4 method for signature '.simList'
modulePath(sim)

modulePath(sim) <- value

## S4 replacement method for signature '.simList'
modulePath(sim) <- value
```

Arguments

sim	A simList object from which to extract element(s) or in which to replace element(s).
value	The object to be stored at the slot.

Details

These are ways to add or access the file paths used by spades. There are four file paths: cachePath, modulePath, inputPath, and outputPath. Each has a function to get or set the value in a simList object. If no paths are specified, the defaults are as follows:

- cachePath: getOption("spades.cachePath");
- inputPath: getOption("spades.modulePath");
- modulePath: getOption("spades.inputPath");
- inputPath: getOption("spades.outputPath").

Value

Returns or sets the value of the slot from the simList object.

See Also

SpaDES.core-package, specifically the section 1.2.4 on Simulation Paths.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, progressInterval, times

---

Plot,simList-method *Plot method for* simList *objects*

Description

Extends Plot for simList objects.

Usage

```
## S4 method for signature 'simList'
Plot(..., new = FALSE, addTo = NULL, gp = gpar(),
  gpText = gpar(), gpAxis = gpar(), axes = FALSE, speedup = 1,
  size = 5, cols = NULL, col = NULL, zoomExtent = NULL,
  visualSqueeze = NULL, legend = TRUE, legendRange = NULL,
  legendText = NULL, pch = 19, title = TRUE, na.color = "#FFFFFF00",
  zero.color = NULL, length = NULL, arr = NULL, plotFn = "plot")
```

Arguments

...	A combination of spatialObjects or some non-spatial objects. See details.
new	Logical. If TRUE, then the previous named plot area is wiped and a new one made; if FALSE, then the ... plots will be added to the current device, adding or rearranging the plot layout as necessary. Default is FALSE. This currently works best if there is only one object being plotted in a given Plot call. However, it is possible to pass a list of logicals to this, matching the length of the ... objects. Use clearPlot to clear the whole plotting device.
addTo	Character vector, with same length as .... This is for overplotting, when the overplot is not to occur on the plot with the same name, such as plotting a SpatialPoints* object on a RasterLayer.
gp	A gpar object, created by gpar function, to change plotting parameters (see **grid** package).
gpText	A gpar object for the title text. Default gpar(col = "black").
gpAxis	A gpar object for the axes. Default gpar(col = "black").
axes	Logical or "L", representing the left and bottom axes, over all plots.
speedup	Numeric. The factor by which the number of pixels is divided by to plot rasters. See Details.
size	Numeric. The size, in points, for SpatialPoints symbols, if using a scalable symbol.

cols	(also col) Character vector or list of character vectors of colours. See details.
col	(also cols) Alternative to cols to be consistent with plot. cols takes precedence, if both are provided.
zoomExtent	An Extent object. Supplying a single extent that is smaller than the rasters will call a crop statement before plotting. Defaults to NULL. This occurs after any downsampling of rasters, so it may produce very pixelated maps.
visualSqueeze	Numeric. The proportion of the white space to be used for plots. Default is 0.75.
legend	Logical indicating whether a legend should be drawn. Default is TRUE.
legendRange	Numeric vector giving values that, representing the lower and upper bounds of a legend (i.e., 1:10 or c(1,10) will give same result) that will override the data bounds contained within the grobToPlot.
legendText	Character vector of legend value labels. Defaults to NULL, which results in a pretty numeric representation. If Raster* has a Raster Attribute Table (rat; see **raster** package), this will be used by default. Currently, only a single vector is accepted. The length of this must match the length of the legend, so this is mostly useful for discrete-valued rasters.
pch	see ?par.
title	Logical or character string. If logical, it indicates whether to print the object name as the title above the plot. If a character string, it will print this above the plot. NOTE: the object name is used with addTo, not the title.
na.color	Character string indicating the color for NA values. Default transparent.
zero.color	Character string indicating the color for zero values, when zero is the minimum value, otherwise, zero is treated as any other color. Default transparent.
length	Numeric. Optional length, in inches, of the arrow head.
arr	A vector of length 2 indicating a desired arrangement of plot areas indicating number of rows, number of columns. Default NULL, meaning let Plot function do it automatically.
plotFn	An optional function name to do the plotting internally, e.g., "barplot" to get a barplot() call. Default "plot".

Details

Plot for simList class objects

See Plot. This method strips out stuff from a simList class object that would make it otherwise not reproducibly digestible between sessions, operating systems, or machines. This will likely still not allow identical digest results across R versions.

See Also

Plot

POM *Use Pattern Oriented Modeling to fit unknown parameters*

Description

This is very much in alpha condition. It has been tested on simple problems, as shown in the examples, with up to 2 parameters. It appears that DEoptim is the superior package for the stochastic problems. This should be used with caution as with all optimization routines. This function can nevertheless take optim or genoud as optimizers, using stats::optim or rgenoud::genoud, respectively. However, these latter approaches do not seem appropriate for stochastic problems, and have not been widely tested and are not supported within POM.

Usage

```
POM(sim, params, objects = NULL, objFn, cl, optimizer = "DEoptim",
  sterr = FALSE, ..., objFnCompare = "MAD", optimControl = NULL,
  NaNRetries = NA, logObjFnVals = FALSE, weights, useLog = FALSE)

## S4 method for signature 'simList,character'
POM(sim, params, objects = NULL, objFn, cl,
  optimizer = "DEoptim", sterr = FALSE, ..., objFnCompare = "MAD",
  optimControl = NULL, NaNRetries = NA, logObjFnVals = FALSE, weights,
  useLog = FALSE)
```

Arguments

sim	A simList simulation object, generally produced by simInit.
params	Character vector of parameter names that can be changed by the optimizer. These must be accessible with params(sim) internally.
objects	A optional named list (must be specified if objFn is not). The names of each list element must correspond to an object in the .GlobalEnv and the list elements must be objects or functions of objects that can be accessed in the ls(sim) internally. These will be used to create the objective function passed to the optimizer. See details and examples.
objFn	An optional objective function to be passed into optimizer. If missing, then POM will use objFnCompare and objects instead. If using POM with a SpaDES simulation, this objFn must contain a spades call internally, followed by a derivation of a value that can be minimized but the optimizer. It must have, as first argument, the values for the parameters. See example.
cl	A cluster object. Optional. This would generally be created using parallel::makeCluster or equivalent. This is an alternative way, instead of beginCluster(), to use parallelism for this function, allowing for more control over cluster use.
optimizer	The function to use to optimize. Default is "DEoptim". Currently it can also be "optim" or "rgenoud", which use stats::optim or rgenoud::genoud, respectively. The latter two do not seem optimal for stochastic problems and have not been widely tested.

sterr	Logical. If using optimizer = "optim", the hessian can be calculated. If this is TRUE, then the standard errors can be estimated using that hessian, assuming normality.
...	All objects needed in objFn
objFnCompare	Character string. Either, "MAD" or "RMSE" indicating that inside the objective function, data and prediction will be compared by Mean Absolute Deviation or Root Mean Squared Error. Default is "MAD".
optimControl	List of control arguments passed into the control of each optimization routine. Currently, only passed to DEoptim.control when optimizer is "DEoptim"
NaNRetries	Numeric. If greater than 1, then the function will retry the objective function for a total of that number of times if it results in an NaN. In general this should not be used as the objective function should be made so that it doesn't produce NaN. But, sometimes it is difficult to diagnose stochastic results.
logObjFnVals	Logical or Character string indicating a filename. Ignored if objFn is supplied. If TRUE (and there is no objFn supplied), then the value of the individual patterns will be output the console if being run interactively or to a tab delimited text file named ObjectiveFnValues.txt (or that passed by the user here) at each evaluation of the POM created objective function. See details.
weights	Numeric. If provided, this vector will be multiplied by the standardized deviations (possibly MAD or RMSE) as described in objects. This has the effect of weighing each standardized deviation (pattern–data pair) to a user specified amount in the objective function.
useLog	Logical. Should the data patterns and output patterns be logged (log) before calculating the objFnCompare. i.e., mean(abs(log(output) - log(data))). This should be length 1 or length objects. It will be recycled if length >1, less than objects.

Details

There are two ways to use this function, via 1) objFn or 2) objects.

1. The user can pass the entire objective function to the objFn argument that will be passed directly to the optimizer. For this, the user will likely need to pass named objects as part of the ....
2. The slightly simpler approach is to pass a list of 'actual data–simulated data' pairs as a named list in objects and specify how these objects should be compared via objFnCompare (whose default is Mean Absolute Deviation or "MAD").

Option 1 offers more control to the user, but may require more knowledge. Option 1 should likely contain a call to simInit(Copy(simList)) and spades internally. See examples that show simple examples of each type, option 1 and option 2. In both cases, params is required to indicate which parameters can be varied in order to achieve the fit.

Currently, option 1 only exists when optimizer is "DEoptim", the default.

The upper and lower limits for parameter values are taken from the metadata in the module. Thus, if the module metadata does not define the upper and lower limits, or these are very wide, then the optimization may have troubles. Currently, there is no way to override these upper and lower

limits; the module metadata should be changed if there needs to be different parameter limits for optimization.

`objects` is a named list of data–pattern pairs. Each of these pairs will be assessed against one another using the `objFnCompare`, after standardizing each independently. The standardization, which only occurs if the abs(data value < 1), is: `mean(abs(derived value - data value))/mean(data value)`. If the data value is between -1 and 1, then there is no standardization. If there is more than one data–pattern pair, then they will simply be added together in the objective function. This gives equal weight to each pair. If the user wishes to put different weight on each pattern, a `weights` vector can be provided. This will be used to multiply the standardized values described above. Alternatively, the user may wish to weight them differently, in which case, their relative scales can be adjusted.

There are many options that can be passed to `DEoptim`, (the details of which are in the help), using `optimControl`. The defaults sent from `POM` to `DEoptim` are: steptol = 3 (meaning it will start assessing convergence after 3 iterations (WHICH MAY NOT BE SUFFICIENT FOR YOUR PROBLEM), `NP = 10 * length(params)` (meaning the population size is 10 x the number of parameters) and itermax = 200 (meaning it won't go past 200 iterations). These and others may need to be adjusted to obtain good values. NOTE: `DEoptim` does not provide a direct estimate of confidence intervals. Also, convergence may be unreliable, and may occur because `itermax` is reached. Even when convergence is indicated, the estimates are not guaranteed to be global optima. This is different than other optimizers that will normally indicate if convergence was not achieved at termination of the optimization.

Using this function with a parallel cluster currently requires that you pass `optimControl = list(parallelType = 1)`, and possibly package and variable names (and does not yet accept the `cl` argument). See examples. This setting will use all available threads on your computer. Future versions of this will allow passing of a custom cluster object via `cl` argument. `POM` will automatically determine packages to load in the spawned cluster (via `packages`) and it will load all objects in the cluster that are necessary, by sending `names(objects)` to `parVar` in `DEoptim.control`.

Setting `logObjFnVals` to `TRUE` may help diagnosing some problems. Using the POM derived objective function, essentially all patterns are treated equally. This may not give the correct behavior for the objective function. Because `POM` weighs the patterns equally, it may be useful to use the log files to examine the behaviour of the pattern–data pairs. The first file, ObjectiveFnValues.txt, shows the result of each of the (possibly logged), pattern–data deviations, standardized, and weighted. The second file, '`ObjectiveFnValues_RawPatterns.txt`', shows the actual value of the pattern (unstandardized, unweighted, unlogged). If `weights` is passed, then these weighted values will be reflected in the '`ObjectiveFnValues.txt`' file.

Value

A list with at least 2 elements. The first (or first several) will be the returned object from the optimizer. The second (or last if there are more than 2), named `args` is the set of arguments that were passed into the control of the optimizer.

Author(s)

Eliot McIntire

See Also

`spades`, `makeCluster`, `simInit`

Examples

```
if (interactive()) {
  set.seed(89462)
  library(parallel)
  library(raster)
  mySim <- simInit(
    times = list(start = 0.0, end = 2.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
      fireSpread = list(nFires = 5),
      randomLandscapes = list(nx = 300, ny = 300)
    ),
    modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
    paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
  )

  # Since this is a made up example, we don't have real data
  #  to run POM against. Instead, we will run the model once,
  #  take the values at the end of the simulation as if they
  #  are real data, then rerun the POM function next,
  #  comparing these "data" with the simulated values
  #  using Mean Absolute Deviation
  outData <- spades(Copy(mySim), .plotInitialTime = NA)

  # Extract the "true" data, in this case, the "proportion of cells burned"
  # Function defined that will use landscape$Fires map from simList,
  #  i.e., sim$landscape$Fires
  #  the return value being compared via MAD with propCellBurnedData
  propCellBurnedFn <- function(landscape) {
    sum(getValues(landscape$Fires) > 0) / ncell(landscape$Fires)
  }
  # visualize the burned maps of true "data"
  propCellBurnedData <- propCellBurnedFn(outData$landscape)
  clearPlot()
  if (interactive()) {
    Fires <- outData$landscape$Fires # Plot doesn't do well with many nested layers
    Plot(Fires)
  }

  # Example 1 - 1 parameter
  # In words, this says, "find the best value of spreadprob such that
  #  the proportion of the area burned in the simulation
  #  is as close as possible to the proportion area burned in
  #  the "data", using \code{DEoptim()}.

  # Can use cluster if computer is multi-threaded (but not yet via cl arg, which is not
  #                                                implemented yet in DEoptim)
  # This example uses parallelType = 1 in DEoptim. For this, you must manually
  #  pass all packages and variables as character strings.
  # cl <- makeCluster(detectCores() - 1) # not implemented yet in DEoptim
  out1 <- POM(mySim, "spreadprob",
              list(propCellBurnedData = propCellBurnedFn), # data = pattern pair
```

```
            #optimControl = list(parallelType = 1),
            logObjFnVals = TRUE)

## Once cl arg is available from DEoptim, this will work:
# out1 <- POM(mySim, "spreadprob", cl = cl,
#            list(propCellBurnedData = propCellBurnedFn)) # data = pattern pair

# Example 2 - 2 parameters
# Function defined that will use caribou from sim$caribou, with
#  the return value being compared via MAD with NPattern
#  module, parameter N, is from 10 to 1000)
caribouFn <- function(caribou) length(caribou)

# Extract "data" from simList object (normally, this would be actual data)
NPattern <- caribouFn(outData$caribou)

aTime <- Sys.time()
parsToVary <- c("spreadprob", "N")
out2 <- POM(mySim, parsToVary,
            list(propCellBurnedData = propCellBurnedFn,
                 NPattern = caribouFn), logObjFnVals = TRUE)
                 #optimControl = list(parallelType = 1))
                 #cl = cl) # not yet implemented, waiting for DEoptim
bTime <- Sys.time()
# check that population overlaps known values (0.225 and 100)
apply(out2$member$pop, 2, quantile, c(0.025, 0.975))
hists <- apply(out2$member$pop, 2, hist, plot = FALSE)
clearPlot()
for (i in seq_along(hists)) Plot(hists[[i]], addTo = parsToVary[i],
                                 title = parsToVary[i], axes = TRUE)

print(paste("DEoptim", format(bTime - aTime)))
#stopCluster(cl) # not yet implemented, waiting for DEoptim

# Example 3 - using objFn instead of objects

# list all the parameters in the simList, from these, we select to vary
params(mySim)

# Objective Function Example:
#   objective function must have several elements
#   - first argument must be parameter vector, passed to and used by DEoptim
#   - likely needs to take sim object, likely needs a copy
#      because of pass-by-reference semantics of sim objects
#   - pass data that will be used internally for objective function
objFnEx <- function(pars, # param values
                    sim, # simList object
                    NPattern, propCellBurnedData, caribouFn, propCellBurnedFn) {
  ### data

  # make a copy of simList because it will possibly be altered by spades call
  sim1 <- Copy(sim)
```

```
  # take the parameters and assign them to simList
  params(sim1)$fireSpread$spreadprob <- pars[1]
  params(sim1)$caribouMovement$N <- pars[2]

  # run spades, without plotting
  out <- spades(sim1, .plotInitialTime = NA)

  # calculate outputs
  propCellBurnedOut <- propCellBurnedFn(out$landscape)
  NPattern_Out <- caribouFn(out$caribou)

  minimizeFn <- abs(NPattern_Out - NPattern) +
                abs(propCellBurnedOut - propCellBurnedData)

  # have more info reported to console, if desired
  # cat(minimizeFn)
  # cat(" ")
  # cat(pars)
  # cat("\n")

  return(minimizeFn)
}

# Run DEoptim with custom objFn, identifying 2 parameters to allow
#  to vary, and pass all necessary objects required for the
#  objFn

# choose 2 of them to vary. Need to identify them in params & inside objFn
# Change optimization parameters to alter how convergence is achieved
out5 <- POM(mySim, params = c("spreadprob", "N"),
            objFn = objFnEx,
            NPattern = NPattern,
            propCellBurnedData = propCellBurnedData,
            caribouFn = caribouFn,
            propCellBurnedFn = propCellBurnedFn,
           #cl = cl, # uncomment for cluster # not yet implemented, waiting for DEoptim
            # see ?DEoptim.control for explanation of these options
            optimControl = list(
              NP = 100, # run 100 populations, allowing quantiles to be calculated
           initialpop = matrix(c(runif(100, 0.2, 0.24), runif(100, 80, 120)), ncol = 2),
              parallelType = 1
            )
          )

# Can also use an optimizer directly -- miss automatic parameter bounds,
#  and automatic objective function using option 2
library(DEoptim)
out7 <- DEoptim(fn = objFnEx,
                sim = mySim,
                NPattern = NPattern,
                propCellBurnedData = propCellBurnedData,
                caribouFn = caribouFn,
                propCellBurnedFn = propCellBurnedFn,
```

```
                  # cl = cl, # uncomment for cluster
                  # see ?DEoptim.control for explanation of these options
                  control = DEoptim.control(
                    steptol = 3,
                    parallelType = 1, # parallelType = 3,
                    packages = list("raster", "SpaDES.core", "RColorBrewer"),
                    parVar = list("objFnEx"),
                    initialpop = matrix(c(runif(40, 0.2, 0.24),
                                          runif(40, 80, 120)), ncol = 2)),
                  lower = c(0.2, 80), upper = c(0.24, 120))
}
```

`priority` *Event priority*

Description

Preset event priorities: 1 = first (highest); 5 = normal; 10 = last (lowest).

Usage

```
.first()

.highest()

.last()

.lowest()

.normal()
```

Value

A numeric.

Author(s)

Alex Chubaty

progressInterval *Get and set simulation progress bar details*

Description

The progress bar can be set in two ways in SpaDES. First, by setting values in the .progress list element in the params list element passed to `simInit`. Second, at the `spades` call itself, which can be simpler. See examples.

Usage

```
progressInterval(sim)

## S4 method for signature '.simList'
progressInterval(sim)

progressInterval(sim) <- value

## S4 replacement method for signature '.simList'
progressInterval(sim) <- value

progressType(sim)

## S4 method for signature '.simList'
progressType(sim)

progressType(sim) <- value

## S4 replacement method for signature '.simList'
progressType(sim) <- value
```

Arguments

`sim`	A `simList` object from which to extract element(s) or in which to replace element(s).
`value`	The object to be stored at the slot.

Details

Progress Bar: Progress type can be one of `"text"`, `"graphical"`, or `"shiny"`. Progress interval can be a numeric. These both can get set by passing a `.progress = list(type = "graphical", interval = 1)` into the `simInit` call. See examples.

See Also

Other functions to access elements of a `simList` object: `.addDepends`, `doEvent.checkpoint`, `envir`, `events`, `globals`, `inputs`, `ls.simList`, `ls.str.simList`, `modules`, `objs`, `packages`, `params`, `paths`, `times`

Examples

```
## Not run:
mySim <- simInit(
  times = list(start=0.0, end=100.0),
  params = list(.globals = list(stackName = "landscape"),
  .progress = list(type = "text", interval = 10),
  .checkpoint = list(interval = 10, file = "chkpnt.RData")),
  modules = list("randomLandscapes"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core")))

# progress bar
progressType(mySim) # "text"
progressInterval(mySim) # 10

# parameters
params(mySim) # returns all parameters in all modules
              # including .global, .progress, .checkpoint
globals(mySim) # returns only global parameters

# checkpoint
checkpointFile(mySim) # returns the name of the checkpoint file
                      # In this example, "chkpnt.RData"
checkpointInterval(mySim) # 10

## End(Not run)
```

`rasterToMemory` *Read raster to memory*

Description

Wrapper to the `raster` function, that creates the raster object in memory, even if it was read in from file.

Usage

```
rasterToMemory(x, ...)

## S4 method for signature 'ANY'
rasterToMemory(x, ...)
```

Arguments

`x`	An object passed directly to the function raster (e.g., character string of a filename).
`...`	Additional arguments to `raster`.

Value

A raster object whose values are stored in memory.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

raster.

`rndstr` *Generate random strings*

Description

Generate a vector of random alphanumeric strings each of an arbitrary length.

Usage

```
rndstr(n, len, characterFirst)

## S4 method for signature 'numeric,numeric,logical'
rndstr(n, len, characterFirst)

## S4 method for signature 'numeric,numeric,missing'
rndstr(n, len)

## S4 method for signature 'numeric,missing,logical'
rndstr(n, characterFirst)

## S4 method for signature 'missing,numeric,logical'
rndstr(len, characterFirst)

## S4 method for signature 'numeric,missing,missing'
rndstr(n)

## S4 method for signature 'missing,numeric,missing'
rndstr(len)

## S4 method for signature 'missing,missing,logical'
rndstr(characterFirst)

## S4 method for signature 'missing,missing,missing'
rndstr(n, len, characterFirst)
```

Arguments

n	Number of strings to generate (default 1). Will attempt to coerce to integer value.
len	Length of strings to generate (default 8). Will attempt to coerce to integer value.
characterFirst	Logical, if TRUE, then a letter will be the first character of the string (useful if being used for object names).

Value

Character vector of random strings.

Author(s)

Alex Chubaty and Eliot McIntire

Examples

```
set.seed(11)
rndstr()
rndstr(len = 10)
rndstr(characterFirst = FALSE)
rndstr(n = 5, len = 10)
rndstr(n = 5)
rndstr(n = 5, characterFirst = TRUE)
rndstr(len = 10, characterFirst = TRUE)
rndstr(n = 5, len = 10, characterFirst = TRUE)
```

---

saveFiles *Save objects using* .saveObjects *in* params *slot of* simInit

---

Description

In the simInit call, a parameter called .saveObjects can be provided in each module. This must be a character string vector of all object names to save. These objects will then be saved whenever a call to saveFiles is made.

Usage

```
saveFiles(sim)
```

Arguments

sim	A simList simulation object.

Details

The file names will be equal to the object name plus `time(sim)` is appended at the end. The files are saved as `.rds` files, meaning, only one object gets saved per file.

For objects saved using this function, the module developer must create save events that schedule a call to `saveFiles`.

If this function is used outside of a module, it will save all files in the outputs(sim) that are scheduled to be saved at the current time in the simList.

There are 3 ways to save objects using `SpaDES`.

1. Model-level saving

Using the `outputs` slot in the `simInit` call. See example in `simInit`. This can be convenient because it gives overall control of many modules at a time, and it gets automatically scheduled during the `simInit` call.

2. Module-level saving

Using the `saveFiles` function inside a module. This must be accompanied by a `.saveObjects` list element in the `params` slot in the `simList`. Usually a module developer will create this method for future users of their module.

3. Custom saving

A module developer can save any object at any time inside their module, using standard R functions for saving R objects (e.g., `save` or `saveRDS`). This is the least modular approach, as it will happen whether a module user wants it or not.

Note

It is not possible to schedule separate saving events for each object that is listed in the `.saveObjects`.

Author(s)

Eliot McIntire

Alex Chubaty

Examples

```
## Not run:

# This will save the "caribou" object at the save interval of 1 unit of time
#  in the outputPath location
outputPath <- file.path(tempdir(), "test_save")
times <- list(start = 0, end = 6, "month")
parameters <- list(
  .globals = list(stackName = "landscape"),
  caribouMovement = list(
    .saveObjects = "caribou",
    .saveInitialTime = 1, .saveInterval = 1
```

```
  ),
  randomLandscapes = list(.plotInitialTime = NA, nx = 20, ny = 20))

modules <- list("randomLandscapes", "caribouMovement")
paths <- list(
  modulePath = system.file("sampleModules", package = "SpaDES.core"),
  outputPath = savePath
)
mySim <- simInit(times = times, params = parameters, modules = modules,
                 paths = paths)

# The caribou module has a saveFiles(sim) call, so it will save caribou
spades(mySim)
dir(outputPath)

# remove the files
file.remove(dir(savePath, full.names = TRUE))


## End(Not run)
```

---

scheduleEvent *Schedule a simulation event*

Description

Adds a new event to the simulation's event queue, updating the simulation object.

Usage

```
scheduleEvent(sim, eventTime, moduleName, eventType, eventPriority)
```

Arguments

sim	A simList simulation object.
eventTime	A numeric specifying the time of the next event.
moduleName	A character string specifying the module from which to call the event.
eventType	A character string specifying the type of event from within the module.
eventPriority	A numeric specifying the priority of the event. Lower number means higher priority. See priority.

Details

Here, we implement a simulation in a more modular fashion so it's easier to add submodules to the simulation. We use S4 classes and methods, and use 'data.table' instead of 'data.frame' to implement the event queue (because it is much faster).

Value

Returns the modified `simList` object.

Author(s)

Alex Chubaty

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

See Also

`priority`

Examples

```
## Not run:
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn") # default priority
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .normal()) # default priority

 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .normal()-1) # higher priority
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .normal()+1) # lower priority

 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .highest()) # highest priority
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .lowest()) # lowest priority

## End(Not run)
```

`show,simList-method` *Show an Object*

Description

Show an Object

Usage

```
## S4 method for signature 'simList'
show(object)
```

Arguments

`object`	`simList`

Author(s)

Alex Chubaty

simInit *Initialize a new simulation*

Description

Create a new simulation object, the "sim" object. This object is implemented using an `environment` where all objects and functions are placed. Since environments in `R` are pass by reference, "putting" objects in the sim object does no actual copy. This is also the location of all parameters, and other important simulation information, such as times, paths, modules, and module load order. See more details below.

Usage

```
simInit(times, params, modules, objects, paths, inputs, outputs, loadOrder,
  notOlderThan = NULL)


  ## S4 method for signature
## 'list,list,list,list,list,data.frame,data.frame,character'
simInit(times,
  params, modules, objects, paths, inputs, outputs, loadOrder,
  notOlderThan = NULL)

## S4 method for signature 'ANY,ANY,ANY,character,ANY,ANY,ANY,ANY'
simInit(times, params,
  modules, objects, paths, inputs, outputs, loadOrder, notOlderThan = NULL)

## S4 method for signature 'ANY,ANY,character,ANY,ANY,ANY,ANY,ANY'
simInit(times, params,
  modules, objects, paths, inputs, outputs, loadOrder, notOlderThan = NULL)

## S4 method for signature 'ANY,ANY,ANY,ANY,ANY,ANY,ANY,ANY'
simInit(times, params, modules,
  objects, paths, inputs, outputs, loadOrder, notOlderThan = NULL)
```

Arguments

`times`	A named list of numeric simulation start and end times (e.g., `times = list(start = 0.0, end = 10.0)`
`params`	A list of lists of the form list(moduleName=list(param1=value, param2=value)). See details.
`modules`	A named list of character strings specifying the names of modules to be loaded for the simulation. Note: the module name should correspond to the R source file from which the module is loaded. Example: a module named "caribou" will be sourced form the file '`caribou.R`', located at the specified `modulePath(simList)` (see below).

objects (optional) A vector of object names (naming objects that are in the calling environment of the simInit, which is often the .GlobalEnv unless used programmatically – NOTE: this mechanism will fail if object name is in a package dependency), or a named list of data objects to be passed into the simList (more reliable). These objects will be accessible from the simList as a normal list, e.g,. mySim$obj.

paths An optional named list with up to 4 named elements, modulePath, inputPath, outputPath, and cachePath. See details.

inputs A data.frame. Can specify from 1 to 6 columns with following column names: objectName (character, required), file (character), fun (character), package (character), interval (numeric), loadTime (numeric). See inputs and vignette("ii-modules") section about inputs.

outputs A data.frame. Can specify from 1 to 5 columns with following column names: objectName (character, required), file (character), fun (character), package (character), saveTime (numeric). See outputs and vignette("ii-modules") section about outputs.

loadOrder An optional list of module names specifying the order in which to load the modules. If not specified, the module load order will be determined automatically.

notOlderThan A time, as in from Sys.time(). This is passed into the Cache function that wraps .inputObjects. If the module has a parameter, .useCache and it is TRUE, then the .inputObjects will be cached. Passing the current time into to notOlderThan will cause the Cache to be refreshed, i.e., rerun.

Details

Calling this simInit function does several things including the following: - sources all module files, placing all function definitions in the sim object - optionally copies objects from the global environment to the sim object - optionally loads objects from disk - schedules all "init" events from all modules - assesses module dependencies via the inputs and outputs identified in their metadata - determines time units of modules and how they fit together

params can only contain updates to any parameters that are defined in the metadata of modules. Take the example of a module named, Fire, which has a parameter named .plotInitialTime. In the metadata of that module, it says TRUE. Here we can override that default with: list(Fire=list(.plotInitialTime=NA) effectively turning off plotting. Since this is a list of lists, one can override the module defaults for multiple parameters from multiple modules all at once, with say: list(Fire = list(.plotInitialTime = NA, .plotInte

We implement a discrete event simulation in a more modular fashion so it is easier to add modules to the simulation. We use S4 classes and methods, and use data.table instead of data.frame to implement the event queue (because it is much faster).

paths specifies the location of the module source files, the data input files, and the saving output files. If no paths are specified the defaults are as follows:

- cachePath: getOption("spades.cachePath");
- inputPath: getOption("spades.modulePath");
- modulePath: getOption("spades.inputPath");
- inputPath: getOption("spades.outputPath").

Value

A `simList` simulation object, pre-initialized from values specified in the arguments supplied.

Note

The user can opt to run a simpler simInit call without inputs, outputs, and times. These can be added later with the accessor methods (See example). These are not required for initializing the simulation via simInit. `modules`, `paths`, `params`, and `objects` are all needed for initialization.

Author(s)

Alex Chubaty and Eliot McIntire

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

See Also

spades, times, params, objs, paths, modules, inputs, outputs

Examples

```
## Not run:
mySim <- simInit(
 times = list(start = 0.0, end = 2.0, timeunit = "year"),
 params = list(
   .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
 ),
 modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
 paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
)
spades(mySim, .plotInitialTime = NA)

# Change more parameters, removing plotting
mySim <- simInit(
 times = list(start = 0.0, end = 2.0, timeunit = "year"),
 params = list(
   .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
   fireSpread = list(.plotInitialTime = NA)
 ),
 modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
 paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
)
outSim <- spades(mySim)

# A little more complicated with inputs and outputs
if (require(rgdal)) {
 mapPath <- system.file("maps", package = "quickPlot")
 mySim <- simInit(
```

```
  times = list(start = 0.0, end = 2.0, timeunit = "year"),
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
  ),
  modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
               outputPath = tempdir()),
  inputs = data.frame(
    files = dir(file.path(mapPath), full.names = TRUE, pattern = "tif")[1:2],
    functions = "raster",
    package = "raster",
    loadTime = 1,
    stringsAsFactors = FALSE),
  outputs = data.frame(
    expand.grid(objectName = c("caribou","landscape"),
    saveTime = 1:2,
    stringsAsFactors = FALSE))
)

# Use accessors for inputs, outputs
mySim2 <- simInit(
  times = list(current = 0, start = 0.0, end = 2.0, timeunit = "year"),
  modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
  params = list(.globals = list(stackName = "landscape", burnStats = "nPixelsBurned")),
  paths = list(
    modulePath = system.file("sampleModules", package = "SpaDES.core"),
    outputPath = tempdir()
  )
)

# add by accessor is equivalent
inputs(mySim2) <- data.frame(
    files = dir(file.path(mapPath), full.names = TRUE, pattern = "tif")[1:2],
    functions = "raster",
    package = "raster",
    loadTime = 1,
    stringsAsFactors = FALSE)
outputs(mySim2) <- data.frame(
    expand.grid(objectName = c("caribou", "landscape"),
    saveTime = 1:2,
    stringsAsFactors = FALSE))
all.equal(mySim, mySim2) # TRUE

# Use accessors for times -- does not work as desired because times are
#   adjusted to the input timeunit during simInit
mySim2 <- simInit(
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
  ),
  modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
               outputPath = tempdir()),
  inputs = data.frame(
```

```
    files = dir(file.path(mapPath), full.names = TRUE, pattern = "tif")[1:2],
    functions = "raster",
    package = "raster",
    loadTime = 1,
    stringsAsFactors = FALSE),
  outputs = data.frame(
    expand.grid(objectName = c("caribou","landscape"),
    saveTime = 1:2,
    stringsAsFactors = FALSE))
)

# add times by accessor fails all.equal test because "year" was not
#   declared during module loading, so month became the default
times(mySim2) <- list(current = 0, start = 0.0, end = 2.0, timeunit = "year")
all.equal(mySim, mySim2) # fails because time units are all different, so
                         # several parameters that have time units in
                         # "months" because they were loaded that way
params(mySim)$fireSpread$.plotInitialTime
params(mySim2)$fireSpread$.plotInitialTime
events(mySim) # load event is at time 1 year
events(mySim2) # load event is at time 1 month, reported in years because of
               #   update to times above
}

## End(Not run)
```

---

spades *setMethod("scheduleEvent", signature(sim = "simList", eventTime = "NULL", moduleName = "character", eventType = "character", eventPriority = "numeric"), definition = function(sim, eventTime, moduleName, eventType, eventPriority) warning(paste("Invalid or missing eventTime. This is usually", "caused by an attempt to scheduleEvent at time NULL", "or by using an undefined parameter.")) return(invisible(sim)))*

Description

#' @rdname scheduleEvent setMethod("scheduleEvent", signature(sim = "simList", eventTime = "numeric", moduleName = "character", eventType = "character", eventPriority = "missing"), definition = function(sim, eventTime, moduleName, eventType, eventPriority) scheduleEvent(sim = sim, eventTime = eventTime, moduleName = moduleName, eventType = eventType, eventPriority = .normal())) Run a spatial discrete event simulation

Usage

```
spades(sim, debug = FALSE, progress = NA, cache, .plotInitialTime = NULL,
  .saveInitialTime = NULL, notOlderThan = NULL, ...)
```

```
## S4 method for signature 'simList,ANY,ANY,missing'
spades(sim, debug = FALSE,
  progress = NA, cache, .plotInitialTime = NULL, .saveInitialTime = NULL,
  notOlderThan = NULL, ...)

## S4 method for signature 'ANY,ANY,ANY,logical'
spades(sim, debug = FALSE, progress = NA,
  cache, .plotInitialTime = NULL, .saveInitialTime = NULL,
  notOlderThan = NULL, ...)
```

Arguments

sim	A simList simulation object, generally produced by simInit.
debug	Optional logical flag or character vector indicating what to print to console at each event. See details. Default is FALSE.
progress	Logical (TRUE or FALSE show a graphical progress bar), character ("graphical", "text") or numeric indicating the number of update intervals to show in a graphical progress bar.
cache	Logical. If TRUE, then the spades call will be cached. This means that if the call is made again with the same simList, then 'spades' will return the return value from the previous run of that exact same simList. Default FALSE. See Details. See also the vignette on caching for examples.
.plotInitialTime	Numeric. Temporarily override the .plotInitialTime parameter for all modules. See Details.
.saveInitialTime	Numeric. Temporarily override the .plotInitialTime parameter for all modules. See Details.
notOlderThan	Date or time. Passed to reproducible::Cache to update the cache. Default is NULL, meaning don't update the cache. If Sys.time() is provided, then it will force a recache, i.e., remove old value and replace with new value. Ignored if cache is FALSE.
...	Any. Can be used to make a unique cache identity, such as "replicate = 1". This will be included in the Cache call, so will be unique and thus spades will not use a cached copy as long as anything passed in ... is unique, i.e., not cached previously.

Details

Here, we implement a simulation in a more modular fashion so it's easier to add submodules to the simulation. We use S4 classes and methods, and use 'data.table' instead of 'data.frame' to implement the event queue (because it is much faster).

The is the workhorse function in the SpaDES package. It runs simulations by implementing the rules outlined in the simList.

This function gives simple access to two sets of module parameters: .plotInitialTime and with .plotInitialTime. The primary use of these arguments is to temporarily turn off plotting and saving. "Temporary" means that the simList is not changed, so it can be used again with the

simList values reinstated. To turn off plotting and saving, use `.plotInitialTime = NA` or `.saveInitialTime = NA`. NOTE: if a module did not use `.plotInitialTime` or `.saveInitialTime`, then these arguments will not do anything.

If `cache` is TRUE, this allows for a seamless way to "save" results of a simulation. The user does not have to intentionally do any saving manually. Instead, upon a call to `spades` in which the simList is identical, the function will simply return the result that would have come if it had been rerun. Use this with caution, as it will return exactly the result from a previous run, even if there is stochasticity internally. Caching is only based on the input simList. See also `experiment` for the same mechanism, but it can be used with replication. See also the vignette on caching for examples.

If `debug` is specified, it can be a logical or character vector. In all cases, something will be printed to the console immediately before each event is being executed. If `TRUE`, then the event immediately following will be printed as it runs (equivalent to `current(sim)`). If a character string, then it can be one of the many `simList` accessors, such as `events`, `params`, `"simList"` (print the entire simList), or any R expression. If an R expression it will be evaluated with access to the `sim` object. If this is more than one character string, then all will be printed to the screen in their sequence.

Value

Invisibly returns the modified `simList` object.

Note

The debug option is primarily intended to facilitate building simulation models by the user. Will print additional outputs informing the user of updates to the values of various `simList` slot components. See https://github.com/PredictiveEcology/SpaDES/wiki/Debugging for details.

Author(s)

Alex Chubaty and Eliot McIntire

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

See Also

`simInit`, `SpaDES.core-package`, `Cache`

`experiment` for using replication with spades.

Examples

```
## Not run:
mySim <- simInit(
 times = list(start = 0.0, end = 2.0, timeunit = "year"),
 params = list(
   .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
 ),
 modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
```

```
 paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
)
spades(mySim)

# Different debug options
spades(mySim, debug = TRUE) # Fastest
spades(mySim, debug = "simList")
spades(mySim, debug = "print(table(sim$landscape$Fires[]))")

# Can turn off plotting, and inspect the output simList instead
out <- spades(mySim, .plotInitialTime = NA) # much faster
completed(out) # shows completed events

# use cache -- simInit should generally be rerun each time a spades call is made
#   to guarantee that it is identical. Here, run spades call twice, first
#   time to establish cache, second time to return cached result
for (i in 1:2) {
 mySim <- simInit(
   times = list(start = 0.0, end = 2.0, timeunit = "year"),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
 )
 print(system.time(out <- spades(mySim, cache = TRUE)))
}

## End(Not run)
```

---

spadesClasses *Classes defined in SpaDES*

Description

These S4 classes are defined within SpaDES. "dot" classes are not exported and are therefore intended for internal use only.

Simulation classes

simList	The 'simList' class
.moduleDeps	Descriptor object for specifying SpaDES module dependencies
.simDeps	Defines all simulation dependencies for all modules within a SpaDES simulation

Author(s)

Eliot McIntire and Alex Chubaty

See Also

simInit

---

times *Time usage in* SpaDES

Description

Functions for the simtimes slot of a simList object and its elements. To maintain modularity, the behavior of these functions depends on where they are used. In other words, different modules can have their own timeunit. SpaDES converts these to seconds when running a simulation, but shows the user time in the units of the model as shown with timeunit(sim)

Usage

```
times(x, ...)

## S4 method for signature '.simList'
times(x)

times(x) <- value

## S4 replacement method for signature '.simList'
times(x) <- value

## S3 method for class '.simList'
time(x, unit, ...)

time(x) <- value

## S4 replacement method for signature '.simList'
time(x) <- value

end(x, unit, ...)

## S4 method for signature '.simList,missing'
end(x)

## S4 method for signature '.simList,character'
end(x, unit)

end(x) <- value
```

```
## S4 replacement method for signature '.simList'
end(x) <- value

start(x, unit, ...)

## S4 method for signature '.simList,missing'
start(x)

## S4 method for signature '.simList,character'
start(x, unit)

start(x) <- value

## S4 replacement method for signature '.simList'
start(x) <- value

timeunit(x)

## S4 method for signature '.simList'
timeunit(x)

timeunit(x) <- value

## S4 replacement method for signature '.simList'
timeunit(x) <- value

timeunits(x)

## S4 method for signature '.simList'
timeunits(x)
```

Arguments

x	A simList object from which to extract element(s) or in which to replace element(s).
...	Additional parameters.
value	A time, given as a numeric, optionally with a unit attribute, but this will be deduced from the model time units or module time units (if used within a module).
unit	Character. One of the time units used in SpaDES.

Details

timeunit will extract the current units of the time used in a simulation (i.e., within a spades call). If it is set within a simInit, e.g., times=list(start=0, end=52, timeunit = "week"), it will set the units for that simulation. By default, a simInit call will use the smallest unit contained within the metadata for the modules being used. If there are parent modules, then the parent module timeunit will be used even if one of its children is a smaller timeunit. If all modules, including

parents, are set to NA, timeunit defaults to seconds. If parents are set to NA, then the set of modules defined by that parent module will be given the smallest units of the children.

Currently, available units are "second", "hours", day", "week", "month", and "year" can be used in the metadata of a module.

The user can also define a new unit. The unit name can be anything, but the function definition must be of the form dunitName, e.g., dyear or dfortnight. The unit name is the part without the d and the function name definition includes the d. This new function, e.g., dfortnight <- function(x) lubridate::duration(can be placed anywhere in the search path or in a module.

timeunits will extract the current units of the time of all modules used in a simulation. This is different from timeunit because it is not necessarily associated with a spades call.

In many cases, the "simpler" use of each of these functions may be slower computationally. For instance, it is much faster to use time(sim, "year") than time(sim). So as a module developer, it is advantageous to write out the longer one, minimizing the looking up that R must do.

Value

Returns or sets the value of the slot from the simList object.

Note

These have default behavior that is based on the calling frame timeunit. When used inside a module, then the time is in the units of the module. If used in an interactive mode, then the time will be in the units of the simulation.

Additional methods are provided to access the current, start, and end times of the simulation:

time	Current simulation time.
start	Simulation start time.
end	Simulation end time.
timeunit	Simulation timeunit.
timeunits	Module timeunits.
times	List of all simulation times (current, start, end, timeunit).

Author(s)

Alex Chubaty and Eliot McIntire

See Also

SpaDES.core-package, specifically the section 1.2.5 on Simulation times.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval

updateList *Update elements of a named list with elements of a second named list*

Description

Merge two named list based on their named entries. Where any element matches in both lists, the value from the second list is used in the updated list. Subelements are not examined and are simply replaced. If one list is empty, then it returns the other one, unchanged.

Usage

```
updateList(x, y)

## S4 method for signature 'list,list'
updateList(x, y)

## S4 method for signature '`NULL`,list'
updateList(x, y)

## S4 method for signature 'list,`NULL`'
updateList(x, y)

## S4 method for signature '`NULL`,`NULL`'
updateList(x, y)
```

Arguments

x	a named list
y	a named list

Value

A named list, with elements sorted by name. The values of matching elements in list y replace the values in list x.

Author(s)

Alex Chubaty

Examples

```
L1 <- list(a = "hst", b = NA_character_, c = 43)
L2 <- list(a = "gst", c = 42, d = list(letters))
updateList(L1, L2)

updateList(L1, NULL)
updateList(NULL, L2)
updateList(NULL, NULL) # should return empty list
```

zipModule *Create a zip archive of a module subdirectory*

Description

The most common use of this would be from a "modules" directory, rather than inside a given module.

Usage

```
zipModule(name, path, version, data = FALSE, ...)

## S4 method for signature 'character,character,character'
zipModule(name, path, version,
  data = FALSE, ...)

## S4 method for signature 'character,missing,character'
zipModule(name, path, version,
  data = FALSE, ...)

## S4 method for signature 'character,missing,missing'
zipModule(name, path, version,
  data = FALSE, ...)

## S4 method for signature 'character,character,missing'
zipModule(name, path, version,
  data = FALSE, ...)
```

Arguments

name	Character string giving the module name.
path	A file path to a directory containing the module subdirectory.
version	The module version.
data	Logical. If TRUE, then the data subdirectory will be included in the zip. Default is FALSE.
...	Additional arguments to zip: e.g., add "-q" using flags="-q -r9X" (the default flags are "-r9X").

Author(s)

Eliot McIntire and Alex Chubaty

Index