Package ‘SpaDES.core’

August 19, 2022

Type Package

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

Description Provides the core framework for a discrete event system (DES) to implement a complete data-to-decisions, reproducible workflow. The core DES components facilitate modularity, and easily enable the user to include additional functionality by running user-built modules. Includes conditional scheduling, restart after interruption, packaging of reusable modules, tools for developing arbitrary automated workflows, automated interweaving of modules of different temporal resolution, and tools for visualizing and understanding the DES project. The suggested package 'NLMR' can be installed with 'install.packages("NLMR", repos = "https://PredictiveEcology.r-universe.dev")'.


Date 2022-08-13

Version 1.1.0

Depends R (>= 4.0), quickPlot (>= 0.1.4), reproducible (>= 1.2.9)

Imports crayon, data.table (>= 1.11.0), fastdigest, igraph (>= 1.0.1), lobstr, methods, qs (>= 0.21.1), raster (>= 2.5-8), Require (>= 0.7), stats, utils, whisker

Suggests codetools, covr, DiagrammeR (>= 0.8.2), dplyr, future, future.callr, ggplot2, googledrive, httr, knitr, logging, magrittr, NLMR (>= 1.1.1), pkgload, png, RColorBrewer (>= 1.1-2), rgdal, rmarkdown, roxygen2, rstudioapi, sp, SpaDES.tools (>= 0.2.0), tcltk, terra, testthat (>= 1.0.2)

Additional_repositories https://predictiveecology.r-universe.dev/

Encoding UTF-8

Language en-CA

License GPL-3

VignetteBuilder knitr, rmarkdown
BugReports https://github.com/PredictiveEcology/SpaDES.core/issues

ByteCompile yes

Collate 'Plots.R' 'module-dependencies-class.R' 'misc-methods.R'
  'environment.R' 'helpers.R' 'simList-class.R' 'cache.R'
  'check.R' 'priority.R' 'checkpoint.R' 'code-checking.R'
  'convertToPackage.R' 'copy.R' 'debugging.R' 'downloadData.R'
  'simulation-parseModule.R' 'simulation-simInit.R' 'load.R'
  'memory-leaks.R' 'memory.R' 'modActiveBinding.R'
  'module-define.R' 'module-dependencies-methods.R'
  'module-repository.R' 'module-template.R' 'moduleCoverage.R'
  'moduleMetadata.R' 'objectSynonyms.R' 'options.R' 'times.R'
  'simList-accessors.R' 'plotting-diagrams.R' 'plotting.R'
  'progress.R' 'project-template.R' 'reexports.R' 'restart.R'
  'save.R' 'saveLoadSimList.R' 'simulation-spades.R'
  'spades-classes.R' 'spades-core-deprecated.R'
  'spades-core-package.R' 'suppliedElsewhere.R' 'zzz.R'

RoxygenNote 7.2.1

NeedsCompilation no

Author Alex M Chubaty [aut, cre] (<https://orcid.org/0000-0001-7146-8135>),
  Eliot J B McIntire [aut] (<https://orcid.org/0000-0002-6914-8316>),
  Yong Luo [ctb],
  Steve Cumming [ctb],
  Ceres Barros [ctb] (<https://orcid.org/0000-0003-4036-977X>),
  Her Majesty the Queen in Right of Canada, as represented by the
  Minister of Natural Resources Canada [cph]

Maintainer Alex M Chubaty <achubaty@for-cast.ca>

Repository CRAN

Date/Publication 2022-08-19 13:20:05 UTC

R topics documented:

SpaDES.core-package ...................................................... 5
  .addChangedAttr,simList-method .................................... 13
  .addTagsToOutput,simList-method .................................. 14
  .cacheMessage,simList-method .................................... 15
  .checkCacheRepo,list-method ..................................... 15
  .fileExtensions ...................................................... 16
  .findSimList .......................................................... 17
  .parseElems,simList-method ......................................... 18
  .preDigestByClass,simList-method .................................. 18
  .prepareOutput,simList-method .................................... 19
  .quickCheck .......................................................... 20
  .robustDigest,simList-method ...................................... 20
  .tagsByClass,simList-method ....................................... 21
all.equal.simList ...................................................... 22
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyPlotting</td>
<td>22</td>
</tr>
<tr>
<td>append_attr</td>
<td>23</td>
</tr>
<tr>
<td>bindrows</td>
<td>24</td>
</tr>
<tr>
<td>checkModule</td>
<td>24</td>
</tr>
<tr>
<td>checkModuleLocal</td>
<td>25</td>
</tr>
<tr>
<td>checkObject</td>
<td>26</td>
</tr>
<tr>
<td>checkParams</td>
<td>27</td>
</tr>
<tr>
<td>checksums</td>
<td>27</td>
</tr>
<tr>
<td>classFilter</td>
<td>28</td>
</tr>
<tr>
<td>clearCache.simList-method</td>
<td>30</td>
</tr>
<tr>
<td>convertToPackage</td>
<td>32</td>
</tr>
<tr>
<td>Copy.simList-method</td>
<td>34</td>
</tr>
<tr>
<td>copyModule</td>
<td>35</td>
</tr>
<tr>
<td>createsOutput</td>
<td>36</td>
</tr>
<tr>
<td>defineEvent</td>
<td>37</td>
</tr>
<tr>
<td>defineModule</td>
<td>38</td>
</tr>
<tr>
<td>defineParameter</td>
<td>40</td>
</tr>
<tr>
<td>depsEdgeList</td>
<td>42</td>
</tr>
<tr>
<td>depsGraph</td>
<td>43</td>
</tr>
<tr>
<td>dhour</td>
<td>43</td>
</tr>
<tr>
<td>doEvent.checkpoint</td>
<td>45</td>
</tr>
<tr>
<td>downloadData</td>
<td>46</td>
</tr>
<tr>
<td>downloadModule</td>
<td>49</td>
</tr>
<tr>
<td>envir</td>
<td>51</td>
</tr>
<tr>
<td>eventDiagram</td>
<td>52</td>
</tr>
<tr>
<td>events</td>
<td>53</td>
</tr>
<tr>
<td>expectsInput</td>
<td>55</td>
</tr>
<tr>
<td>experiment</td>
<td>56</td>
</tr>
<tr>
<td>extractURL</td>
<td>57</td>
</tr>
<tr>
<td>fileName</td>
<td>57</td>
</tr>
<tr>
<td>getModuleVersion</td>
<td>58</td>
</tr>
<tr>
<td>globals</td>
<td>59</td>
</tr>
<tr>
<td>initialize.simList-method</td>
<td>60</td>
</tr>
<tr>
<td>inputObjects</td>
<td>60</td>
</tr>
<tr>
<td>inputs</td>
<td>62</td>
</tr>
<tr>
<td>inSeconds</td>
<td>65</td>
</tr>
<tr>
<td>loadPackages</td>
<td>67</td>
</tr>
<tr>
<td>loadSimList</td>
<td>68</td>
</tr>
<tr>
<td>makeMemoisable.simList</td>
<td>69</td>
</tr>
<tr>
<td>maxTimeunit</td>
<td>70</td>
</tr>
<tr>
<td>memoryUseThisSession</td>
<td>70</td>
</tr>
<tr>
<td>minTimeunit</td>
<td>71</td>
</tr>
<tr>
<td>moduleCodeFiles</td>
<td>72</td>
</tr>
<tr>
<td>moduleCoverage</td>
<td>72</td>
</tr>
<tr>
<td>moduleDefaults</td>
<td>73</td>
</tr>
<tr>
<td>moduleDiagram</td>
<td>74</td>
</tr>
<tr>
<td>moduleGraph</td>
<td>75</td>
</tr>
<tr>
<td>moduleMetadata</td>
<td>76</td>
</tr>
</tbody>
</table>
R topics documented:

moduleParams .................................................. 78
modules .......................................................... 79
moduleVersion .................................................... 80
newModule ........................................................ 82
newModuleCode ................................................... 83
newModuleDocumentation ..................................... 84
newModuleTests .................................................. 85
newProgressBar .................................................. 86
newProject ......................................................... 86
newProjectCode ................................................... 87
objectDiagram ...................................................... 88
objectSynonyms ................................................... 89
objs ................................................................. 90
objSize.simList .................................................... 91
openModules ......................................................... 92
outputs ............................................................. 93
packages ........................................................... 96
paramCheckOtherMods .......................................... 97
params ............................................................. 98
paths ............................................................... 100
Plot, simList-method ............................................. 103
Plots ............................................................... 105
priority ............................................................. 108
progressInterval .................................................. 108
rasterCreate ....................................................... 110
rasterToMemory ................................................... 111
remoteFileSize ..................................................... 112
restartR ............................................................. 112
restartSpades ...................................................... 114
rndstr ............................................................... 115
saveFiles ........................................................... 116
saveSimList ......................................................... 118
scheduleConditionalEvent ...................................... 120
scheduleEvent ...................................................... 122
show, simList-method ............................................. 123
simFile ............................................................. 124
simInit ............................................................. 124
simInitAndSpades ................................................ 130
simList-class ....................................................... 133
spades .............................................................. 135
spadesClasses ..................................................... 140
spadesOptions ..................................................... 141
suppliedElsewhere ............................................... 142
times .............................................................. 143
updateList .......................................................... 146
use_gha ............................................................ 147
writeEventInfo ...................................................... 147
writeRNGInfo ........................................................ 148
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 56 microseconds overhead for each event (including scheduling, sorting event queue, spawning event etc.) or 38 microseconds if there is no sorting (i.e., no sorting occurs under simple conditions). Under most event conditions, therefore, the DES itself will contribute very minimally compared to the content of the events, which may often be milliseconds to many seconds each event.

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’ In ‘SpaDES.experiment’ package. Run multiple `spades()` calls
- ‘experiment2’ In ‘SpaDES.experiment’ package. Run multiple `spades()` calls

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

- `scheduleEvent()` Schedule a simulation event
- `scheduleConditionalEvent()` Schedule a conditional 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.
[rasterPath()]  Global simulation temporary raster path.
[paths()]  Global simulation paths (cache, modules, inputs, outputs, rasters).

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.
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)
- **[elapsedTime()]** The amount of clock time that modules & events use

3.7 Modules, dependencies, packages: 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)
- **[packages()]** Vector of required R libraries of all modules. (advanced)

3.8 ‘simList’ environment: The [simList()] has a slot called ‘.xData’ 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 ‘.xData’ 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:

<table>
<thead>
<tr>
<th>Accessor method</th>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[checkpointFile()]</td>
<td>‘checkpoint’</td>
<td>Name of the checkpoint file. (advanced)</td>
</tr>
<tr>
<td>[checkpointInterval()]</td>
<td>‘checkpoint’</td>
<td>The simulation checkpoint interval. (advanced)</td>
</tr>
</tbody>
</table>

3.10 Progress Bar:

<table>
<thead>
<tr>
<th>Accessor method</th>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[progressType()]</td>
<td>‘.progress’</td>
<td>Type of graphical progress bar used. (advanced)</td>
</tr>
<tr>
<td>[progressInterval()]</td>
<td>‘.progress’</td>
<td>Interval for the progress bar. (advanced)</td>
</tr>
</tbody>
</table>

[times()] List of all simulation times (current, start, end), in units of longest module.
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](https://github.com/PredictiveEcology/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

There are also accessors for many of the metadata entries:

- `[timeunit()]` Accesses metadata of same name
- `[citation()]` Accesses metadata of same name
- `[documentation()]` Accesses metadata of same name
- `[reqdPkgs()]` Accesses metadata of same name
- `[inputObjects()]` Accesses metadata of same name
- `[outputObjects()]` Accesses metadata of same name

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:

- [SpaDES.tools::adj()] An optimized (i.e., faster) version of [raster::adjacent()]
- [SpaDES.tools::cir()] Identify pixels in a circle around a [‘SpatialPoints*()’]
- [‘directionFromEachPoint()’][SpaDES.tools::distanceFromEachPoint] Fast calculation of direction and distance surfaces
- [SpaDES.tools::distanceFromEachPoint()] Fast calculation of distance surfaces
- [SpaDES.tools::rings()] Identify rings around focal cells (e.g., buffers and donuts)
- [SpaDES.tools::spokes()] Identify outward radiating spokes from initial points
- [SpaDES.tools::spread()] Contagious cellular automata
- [SpaDES.tools::spread2()] Contagious cellular automata, different algorithm, more robust
- [SpaDES.tools::wrap()] Create a torus from a grid

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

- [SpaDES.tools::crw()] Simple correlated random walk function
- [SpaDES.tools::heading()] Determines the heading between ‘SpatialPoints*’
- [quickPlot::makeLines()] Makes ‘SpatialLines’ object for, e.g., drawing arrows
- [‘move()’][SpaDES.tools::crw] A meta function that can currently only take "crw"
- [‘specificNumPerPatch()’][SpaDES.tools::specnumperpatch-probs] 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.

- [SpaDES.tools::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()’][quickPlot::getSetColors] Set colours for plotting ‘Raster*‘ objects
- [getColors()] Get colours in a ‘Raster*‘ objects
- [divergentColors()] Create a colour palette with diverging colours 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.

- \[\text{SpaDES.tools::gaussMap()}\] Creates a random map using Gaussian random fields
- \[\text{SpaDES.tools::randomPolygons()}\] 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:

- \[\text{checkObject()}\] Check for a existence of an object within a ‘simList’
- \[\text{Require::checkPath()}\] 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 (https://www.gowlland.ca/). You must know how to use SELES for these to be useful:

- \[\text{agentLocation()}][\text{SpaDES.tools::SELESagentLocation}\] Agent location
- \[\text{SpaDES.tools::initiateAgents()}\] Initiate agents into a ‘SpatialPointsDataFrame’
- \[\text{numAgents()}][\text{SpaDES.tools::SELESnumAgents}\] Number of agents
- \[\text{probInit()}][\text{SpaDES.tools::SELESprobInit}\] Probability of initiating an agent or event
- \[\text{transitions()}][\text{SpaDES.tools::SELEStransitions}\] Transition probability

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

- \[\text{SpaDES.tools::inRange()}\] Test whether a number lies within range \([a,b]\)
- \[\text{layerNames()}\] Get layer names for numerous object classes
- \[\text{numLayers()}\] Return number of layers
- \[\text{paddedFloatToChar()}\] Wrapper for padding (e.g., zeros) floating numbers to character

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.

- \[\text{reproducible::Cache()}\] Caches a function, but often accessed as arg in \[spades()\]
- \[\text{‘showCache()’}[\text{reproducible::cache-tools}]\] Shows information about the objects in the cache
- \[\text{‘clearCache()’}[\text{reproducible::cache-tools}]\] Removes objects from the cache
- \[\text{‘keepCache()’}[\text{reproducible::cache-tools}]\] Keeps only the objects described
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 quickPlot.*

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("SpaDES.core"), "sampleModules")’.

- ‘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.browserOnError’: If ‘TRUE’, the default, then any error rerun the same event with ‘debugonce’ called on it to allow editing to be done. When that browser is continued (e.g., with ‘c’), then it will try to reparse it into the simList and rerun the edited version. This may allow a spades call to be recovered on error, though in many cases that may not be the correct behaviour. For example, if the simList gets updated inside that event in an iterative manner, then each run through the event will cause that iteration to occur. When this option is ‘TRUE’, then the event will be run at least 3 times: the first time makes the error, the second time has ‘debugonce’ and the third time is after the error is addressed. ‘TRUE’ is likely somewhat slower.
- ‘reproducible.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.debug’: The default debugging value ‘debug’ argument in ‘spades()’. Default is ‘TRUE’.
- ‘spades.lowMemory’: If true, some functions will use more memory efficient (but slower) algorithms. Default ‘FALSE’.
- ‘spades.moduleCodeChecks’: Should the various code checks be run during ‘simInit’. These are passed to codetools::checkUsage. Default is given by the function, plus these :‘list(suppressParamUnused = FALSE, suppressUndefined = TRUE, suppressPartialMatchArgs = FALSE, suppressNoLocalFun = TRUE, skipWith = TRUE)’.
- ‘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 via ‘downloadModule’. 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.recoveryMode’: If this a numeric > 0 or TRUE, then the discrete event simulator will take a snapshot of the objects in the simList that might change (based on metadata ‘outputObjects’ for that module), prior to initiating every event. This will allow the user to be able to recover in case of an error or manual interruption (e.g., ‘Esc’). If this is numeric, a copy of that number of “most recent events” will be maintained so that the user can recover and restart > 1 event in the past, i.e., redo some of the "completed" events. Default is ‘TRUE’, i.e., it will keep the state of the ’simList’ at the start of the current event. This can be recovered with ‘restartSpades’ and the differences can be seen in a hidden object in the stashed ’simList.’ There is a message which describes how to find that.

• ‘spades.switchPkgNamespaces’: Should the search path be modified to ensure a module’s required packages are listed first? Default ‘FALSE’ to keep computational overhead down. If ‘TRUE’, there should be no name conflicts among package objects, but it is much slower, especially if the events are themselves fast.

• ‘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 ‘"https://github.com/PredictiveEcology/SpaDES"’.

Author(s)

Maintainer: Alex M Chubaty <achubaty@for-cast.ca> (ORCID)
Authors:

• Eliot J B McIntire <eliot.mcintire@nrcan-rncan.gc.ca> (ORCID)

Other contributors:

• Yong Luo <Yong.Luo@gov.bc.ca> [contributor]
• Steve Cumming <Steve.Cumming@sbf.ulaval.ca> [contributor]
• Ceres Barros <cbarros@mail.ubc.ca> (ORCID) [contributor]
• Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [copyright holder]

See Also

[spadesOptions()]

Description

This will evaluate which elements in the simList object changed following this Cached function call. It will add a named character string as an attribute attr(x, ".Cache")$changed, indicating which ones changed. When this function is subsequently called again, only these changed objects will be returned. All other simList objects will remain unchanged.
Usage

## S4 method for signature 'simList'
.addChangedAttr(object, preDigest, origArguments, ...)

Arguments

- **object**: Any R object returned from a function
- **preDigest**: The full, element by element hash of the input arguments to that same function, e.g., from .robustDigest
- **origArguments**: These are the actual arguments (i.e., the values, not the names) that were the source for preDigest
- **...**: Anything passed to methods.

See Also

reproducible::.addChangedAttr.
reproducible::.addChangedAttr

Description

See reproducible::.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 list, environment (or similar) objects
- **FUN**: A function
- **preDigestByClass**: A list, usually from .preDigestByClass

Author(s)

Eliot McIntire

See Also

reproducible::.addTagsToOutput
Description
See reproducible::.cacheMessage.

Usage
## S4 method for signature 'simList'
.cacheMessage(
  object,
  functionName,
  fromMemoise = getOption("reproducible.useMemoise", TRUE)
)

Arguments
object Any R object.
functionName A character string indicating the function name
fromMemoise Logical. If TRUE, the message will be about recovery from memoised copy

See Also
reproducible::.cacheMessage

Description
See reproducible::.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

reproducible::.checkCacheRepo

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.

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()

Arguments

sim simList object.

filelist list or data.frame to call loadFiles directly from the filelist as described in Details

... Additional arguments.

Author(s)

Eliot McIntire and Alex Chubaty
.findSimList

Find simList in a nested list

Description

This is recursive, so it will find the all simLists even if they are deeply nested.
Usage

.findSimList(x)

Arguments

x any object, used here only when it is a list with at least one \texttt{simList} in it

Description

See \texttt{quickPlot::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

quickPlot::parseElems

Description

Pre-digesting method for \texttt{simList}

Usage

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

Arguments

object Any R object.
.prepareOutput, simList-method

Details

See reproducible::preDigestByClass.

Author(s)

Eliot McIntire

See Also

reproducible::preDigestByClass

Description

See reproducible::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 passed to FUN

See Also

reproducible::prepareOutput
### .quickCheck

*The SpaDES.core variable to switch between quick and robust checking*

**Description**

A variable that can be used by module developers and model users to switch between a quick check of functions like downloadData, Cache. The module developer must actually use this in their code.

**Usage**

```
.quickCheck
```

**Format**

An object of class `logical` of length 1.

### .robustDigest, simList-method

*robustDigest for simList 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, length, algo, quick, classOptions)
```

**Arguments**

- `object` an object to digest.
- `objects` Character vector of objects to be digested. This is only applicable if there is a list, environment (or similar) with named objects within it. Only this/these objects will be considered for caching, i.e., only use a subset of the list, environment or similar objects. In the case of nested list-type objects, this will only be applied outermost first.
- `length` Numeric. If the element passed to Cache is a Path class object (from e.g., asPath(filename)) or it is a Raster with file-backing, then this will be passed to digest::digest, essentially limiting the number of bytes to digest (for speed). This will only be used if `quick = FALSE`. Default is `getOption("reproducible.length")`, which is set to Inf.
The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32, sha256, sha512, xxhash32, xxhash64, murmur32, spookyhash and blake3.

Logical or character. If TRUE, no disk-based information will be assessed, i.e., only memory content. See Details section about quick in Cache.

Optional list. This will pass into .robustDigest for specific classes. Should be options that the .robustDigest knows what to do with.

See reproducible::.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.

Eliot McIntire

reproducible::.robustDigest()

Description
See reproducible::.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
reproducible::.tagsByClass
all.equal.simList  

All equal method for simLists

Description

This function removes a few attributes that are added internally by SpaDES.core and are not relevant to the all.equal. One key element removed is any time stamps, as these are guaranteed to be different.

Usage

## S3 method for class 'equal.simList'
all(target, current, ...)

Arguments

target  

R object.
current  

other R object, to be compared with target.
...

further arguments for different methods, notably the following two, for numerical comparison:

Value

See base::all.equal()

anyPlotting  

Test whether there should be any plotting from .plot parameter

Description

This will do all the various tests needed to determine whether plotting of one sort or another will occur. Testing any of the types as listed in Plots() argument types. Only the first 3 letters of the type are required.

Usage

anyPlotting(.plots)

Arguments

.plots  

Usually will be the P(sim)$plots is used within a module.
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, y

A list of items with optional attributes.

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)
bindrows  

*Simple wrapper around* data.table::rbindlist

**Description**

This simply sets defaults to fill = TRUE, and use.names = TRUE

**Usage**

bindrows(...)

**Arguments**

...  

1 or more data.frame, data.table, or list objects

---

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**

```r
checkModuleLocal(name, path, version)
```

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

```r
## 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim</td>
<td>A simList() object.</td>
</tr>
<tr>
<td>name</td>
<td>A character string specifying the name of an object to be checked.</td>
</tr>
<tr>
<td>object</td>
<td>An object. This is mostly used internally, or with layer, because it will fail if the object does not exist.</td>
</tr>
<tr>
<td>layer</td>
<td>Character string, specifying a layer name in a Raster, if the name is a Raster* object.</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments. Not implemented.</td>
</tr>
</tbody>
</table>

Value

Invisibly return TRUE indicating object exists; FALSE if not.

Author(s)

Alex Chubaty and Eliot McIntire
checkParams

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

Usage
checkParams(sim, coreParams, ...)

## S4 method for signature 'simList,list'
checkParams(sim, coreParams, ...)

Arguments
sim A simList simulation object.
coreParams List of default core parameters.
... 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

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 reproducible:::.digest, which is simply a wrapper around digest::digest.

Usage
checksums(module, path, ...)

Arguments

module Character string giving the name of the module.
path Character string giving the path to the module directory.
...

Passed to `reproducible::Checksums()`, notably, write, quickCheck, checksumFile and files.

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.

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:

1. specify your module (`moduleName <- "my_module"`);
2. use a temp dir to ensure all modules get fresh copies of the data (`tmpdir <- file.path(tempdir(), "SpaDES_modules")));
3. download your module’s data to the temp dir (`downloadData(moduleName, tmpdir)`);
4. initialize a dummy simulation to ensure any 'data prep' steps in the .inputObjects section are run (`simInit(modules = moduleName)`);
5. recalculate your checksums and overwrite the file (`checksums(moduleName, tmpdir, write = TRUE)`);
6. copy the new checksums file to your working module directory (the one not in the temp dir) (`file.copy(from = file.path(tmpdir, moduleName, 'data', 'CHECKSUMS.txt'), to = file.path('path/to/my/moduleDir', moduleName, 'data', 'CHECKSUMS.txt'), overwrite = TRUE)`).

---

classFilter

Filter objects by class

Description

Based on [https://stackoverflow.com/a/5158978/1380598](https://stackoverflow.com/a/5158978/1380598).
Usage

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

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

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

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

## S4 method for signature 'character,character,missing,missing'
```r
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 [https://stackoverflow.com/a/27923346/1380598](https://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)

clearCache,simList-method

clearCache for simList objects

---

Description

This will take the cachePath(object) and pass
This will take the cachePath(object) and pass
This will take the cachePath(object) and pass
clearCache.simList-method

Usage

```r
## S4 method for signature 'simList'
clearCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  ask =getOption("reproducible.ask"),
  useCloud = FALSE,
  cloudFolderID = getOption("reproducible.cloudFolderID", NULL),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

## S4 method for signature 'simList'
showCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)

## S4 method for signature 'simList'
keepCache(
  x,
  userTags = character(),
  after = NULL,
  before = NULL,
  ask =getOption("reproducible.ask"),
  drv = getOption("reproducible.drv", RSQLite::SQLite()),
  conn = getOption("reproducible.conn", NULL),
  ...
)
```

Arguments

- **x**: A simList or a directory containing a valid Cache repository. Note: For compatibility with Cache argument, cacheRepo can also be used instead of `x`, though `x` will take precedence.

- **userTags**: Character vector. If used, this will be used in place of the after and before. Specifying one or more userTag here will clear all objects that match those tags. Matching is via regular expression, meaning partial matches will work unless strict beginning (^) and end ($) of string characters are used. Matching will be against any of the 3 columns returned by `showCache()`, i.e., artifact,
**convertToPackage**

Convert standard module code into an R package

---

**Description**

*EXPERIMENTAL – USE WITH CAUTION.* This function will only create the necessary source files so that all the code can be used (and installed) like an R package. This function does not install anything (e.g., devtools::install). After running this function, simInit will automatically detect that this is now a package and will load the functions (via pkgload::load_all) from the source files. This will have the effect that it emulates the "non-package" behaviour of a SpaDES module exactly. After running this function, current tests show no impact on module behaviour, other than event-level and module-level Caching will show changes and will be rerun. Function-level Caching appears unaffected. In other words, this should cause no changes to running the module code via simInit and spades.

**Usage**

```r
convertToPackage(
  module = NULL,
  path = getOption("spades.modulePath"),
  buildDocuments = TRUE
)
```

---

The function `convertToPackage` can be used to convert standard module code into an R package. This function does not install anything but will create the necessary source files so that all the code can be used like an R package. After running this function, simInit will automatically detect that this is now a package and will load the functions from the source files. This will have the effect that it emulates the "non-package" behaviour of a SpaDES module exactly. After running this function, current tests show no impact on module behaviour, other than event-level and module-level Caching will show changes and will be rerun. Function-level Caching appears unaffected. In other words, this should cause no changes to running the module code via simInit and spades.
convertToPackage

Arguments

module  Character string of module name, without path
path    Character string of modulePath. Defaults to getOption("spades.modulePath")
buildDocuments  A logical. If TRUE, the default, then the documentation will be built, if any exists, using roxygen2::roxygenise

Details

This will move all functions that are not already in an .R file in the R folder into that folder, one function per file, including the doEvent.xxx function. It will not touch any other functions already in the "R" folder. It will also create and fill a minimal DESCRIPTION file. This will leave the defineModule function call as the only code in the main module file. This defineModule and a doEvent.xxx are the only 2 elements that are required for an R package to be considered a SpaDES module. With these changes, the module should still function normally, but will be able to act like an R package, e.g., for writing function documentation with roxygen2, using the testthat infrastructure, etc.

This function is intended to be run once for a module that was created using the "standard" SpaDES module structure (e.g., from a newModule call). There is currently no way to "revert" the changes from R (though it can be done using version control utilities if all files are under version control, e.g., GitHub). Currently SpaDES.core identifies a module as being a package if it has a DESCRIPTION file, or if it has been installed to the .libPaths() e.g., via devtools::install or the like. So one can simply remove the package from .libPaths and delete the DESCRIPTION file and SpaDES.core will treat it as a normal module.

Value

This is run for its side effects. There will be a new or modified DESCRIPTION file in the root directory of the module. Any functions that were in the main module script (i.e., the .R file whose filename is the name of the module and is in the root directory of the module) will be moved to individual .R files in the R folder. Any function with a dot prefix will have the dot removed in its respective filename, but the function name is unaffected.

Currently, SpaDES.core does not install the package under any circumstances. It will load it via pkgdown::load_all, and optionally (option("spades.moduleDocument" = TRUE)) build documentation via roxygen2::roxygenise within the simInit call. This means that any modifications to source code will be read in during the simInit call, as is the practice when a module is not a package.

Exported functions

The only function that will be exported by default is the doEvent.xxx, where xxx is the module name. If any other module is to be exported, it must be explicitly exported with e.g., @export, and then building the NAMESPACE file, e.g., via devtools::document(moduleRootPath). NOTE: as long as all the functions are being used inside each other, and they all can be traced back to a call in doEvent.xxx, then there is no need to export anything else.
DESCRIPTION

The DESCRIPTION file that is created (destroying any existing DESCRIPTION file) with this function will have several elements that a user may wish to change. Notably, all packages that were in reqdPkg in the SpaDES module metadata will be in the Imports section of the DESCRIPTION. To accommodate the need to see these functions, a new R script, imports.R will be created with @import for each package in reqdPkg of the module metadata. However, if a module already has used @importFrom for importing a function from a package, then the generic @import will be omitted for that (those) package(s). So, a user should likely follow standard R package best practices and use @importFrom to identify the specific functions that are required within external packages, thereby limiting function name collisions (and the warnings that come with them).

Other elements of a standard DESCRIPTION file that will be missing or possibly inappropriately short are Title, Description, URL, BugReports.

Installing as a package

There is no need to "install" the source code as a package because simInit will load it on the fly. But, there may be reasons to install it, e.g., to have access to individual functions, help manual, running tests etc. To do this, simply use the devtools::install(pathToModuleRoot). Even if it is installed, simInit will nevertheless run pkgload::load_all to ensure the spades call will be using the current source code.

---

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 reproducible::Copy().

Usage

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

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>An R object (likely containing environments) or an environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>objects</td>
<td>Whether the objects contained within the simList environment should be copied. Default TRUE, which may be slow.</td>
</tr>
<tr>
<td>queues</td>
<td>Logical. Should the events queues (events, current, completed) be deep copied via data.table::copy</td>
</tr>
<tr>
<td>...</td>
<td>Only used for custom Methods</td>
</tr>
</tbody>
</table>
copyModule

Details

simList objects can contain a lot of information, much of which could be in pass-by-reference objects (e.g., data.table class), and objects that are file-backed, such as some Raster*-class objects. For all the objects that are file-backed, it is likely very important to give unique file-backed directories. This should be passed here, which gets passed on to the many methods of Copy in reproducible.

Author(s)

Eliot McIntire

See Also

reproducible::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
createsOutput

**Define an output object of a module**

**Description**

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

**Usage**

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

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

## S4 method for signature 'character,character,character'
```r
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. If there are extra spaces or carriage returns, these will be stripped, allowing for multi-line character strings without using `paste` or multiple quotes.
- `...` 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**

```r
outputObjects <- bindrows(
  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")
)
```
defineEvent

Alternative way to define events in SpaDES.core

Description

There are two ways to define what occurs during an event: defining a function called doEvent.moduleName, where moduleName is the actual module name. This approach is the original approach used in SpaDES.core, and it must have an explicit switch statement branching on eventType. The newer approach (still experimental) uses defineEvent. Instead of creating the function called, doEvent.XXXX, where XXXX is the module name, it creates one function for each event, each with the name doEvent.XXXX.YYYY, where YYY is the event name. This may be a little bit cleaner, but both work.

Usage

defineEvent(sim, eventName = "init", code, moduleName = NULL, envir)

Arguments

- sim: A simList
- eventName: Character string of the desired event name to define. Default is "init"
- code: An expression that defines the code to execute during the event. This will be captured, and pasted into a new function (doEvent.XXXX.YYYY), where XXXX is the moduleName and YYY is the eventName, remaining unevaluated until that new function is called.
- moduleName: Character string of the name of the module. If this function is used within a module, then it will try to find the module name.
- envir: An optional environment to specify where to put the resulting function. The default will place a function called doEvent.moduleName.eventName in the module function location, i.e., sim$.mods[[moduleName]]. However, if this location does not exist, then it will place it in the parent.frame(), with a message. Normally, especially, if used within SpaDES module code, this should be left missing.

See Also

defineModule(), simInit(), scheduleEvent()

Examples

```r
sim <- simInit()

# these put the functions in the parent.frame() which is .GlobalEnv for an interactive user
defineEvent(sim, "init", moduleName = "thisTestModule", code = {
  sim <- Init(sim) # initialize
  # Now schedule some different event for "current time", i.e., will
  # be put in the event queue to run *after* this current event is finished
```
Define a new module.

**Description**

Specify a new module’s metadata as well as object and package dependencies. Packages are loaded during this call. Any or all of these can be missing, with missing values set to defaults.

**Usage**

```r
defineModule(sim, x)
```

## S4 method for signature 'simList, list'

```r
defineModule(sim, x)
```
defineModule

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; it is for human readers only.
- `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, then only this list entry will be read. For normal, i.e., 'child modules', this should be character(0).
- `authors`: Module author information (as a vector of person() objects). This is currently not parsed by SpaDES; it is for human readers only.
- `version`: Module version number (will be coerced to numeric_version() if a character or numeric are supplied). This is for human readers only.
- `spatialExtent`: The spatial extent of the module supplied via raster::extent. This is currently unimplemented. Once implemented, this should define what spatial region this module is scientifically reasonable to be used in.
- `timeframe`: A vector (length 2) of POSIXt dates specifying the temporal extent of the module. Currently unimplemented. Once implemented, this should define what time frame this module is scientifically reasonable to be used for.
- `timeunit`: Time scale of the module (e.g., "day", "year"). If this is not specified, then .timeunitDefault() will be used. If this is not a character, SpaDES interprets this and if modules have different units, this only identifies the objects that this module will output into the simList. The developer must create the necessary functions that will cause these objects to be put into the simList. This is currently unimplemented. Once implemented, this should define what timeunit this module is scientifically reasonable to be used for.
- `sourceURL`: List of filenames referring to module documentation sources. This is currently not parsed by SpaDES; it is for human readers only.
- `documentation`: List of character strings specifying module documentation information. Alternatively, a list of filenames of .bib or .R files or similar files. This is currently not parsed by SpaDES; it is for human readers only.
- `reqdPkgs`: List of R package names required by the module. These packages will be loaded when simInit is called. RRequire::Require() can also download from GitHub.com, these packages can specify package names stored on GitHub, e.g., https://semver.org/.
- `parameters`: A data.frame specifying the parameters used in the module. Usually produced by rbind-ing the outputs of defineParameter(). These parameters indicate the default values that will be used unless a module user overrides them with the argument in the simInit() call. The minimum and maximum are currently used by the simInit function, and they should indicate the range of values that are reasonable scientifically.
- `inputObjects`: A data.frame specifying the data objects expected as inputs to the module, with columns objectName (class gives the developer the opportunity to identify the source of a data file that can be used with the model. This URL will be used if the user calls downloadData function, and they should indicate the range of values that are reasonable scientifically.
- `outputObjects`: A data.frame specifying the data objects output by the module, with columns identical to those in inputObjects.

Author(s)

Alex Chubaty

See Also

moduleDefaults defineEvent()

Examples

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

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

Define a parameter used in a module

Description

Used to specify a parameter's name, value, and set a default. The min and max arguments are ignored by simInit or spades; they are for human use only. To ensure that a user cannot set parameters outside of a range of values, the module developer should use assertions in their module code.

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, min, max, 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. *These are not tested by simInit or spades.* These are primarily for human use, i.e., to tell a module user what values the module expects.
max With min, used to define a suitable range of values. Non-standard evaluation is used for the expression. *These are not tested by simInit or spades.* These are primarily for human use, i.e., to tell a module user what values the module expects.
desc Text string providing a brief description of the parameter. If there are extra spaces or carriage returns, these will be stripped, allowing for multi-line character strings without using paste or multiple quotes.
... A convenience that allows writing a long desc without having to use paste; any character strings after desc will be pasted together with desc.
**Value**

data.frame

**Note**

Be sure to use the correct NA type: logical (NA), integer (NA_integer_), real (NA_real_), complex (NA_complex_), or character (NA_character_). See NA().

**Author(s)**

Alex Chubaty and Eliot McIntire

**See Also**

P(), params() for accessing these parameters in a module.

**Examples**

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

# multi-line desc without quotes on each line -- spaces and carriage returns are stripped
defineParameter("rate", "numeric", 0.2, 0, 1,
  "rate of arrival. This is in individuals per day. This can be modified
  by the user"),
# multi-line desc with quotes on each line
defineParameter("times", "numeric", 0.2, 0, 1,
  desc = "The times during the year ",
  "that events will occur ",
  "with possibility of random arrival times")
```

```r
## Not run:
# Create a new module, then access parameters using 'P'
tmpdir <- file.path(tempdir(), "test")
checkPath(tmpdir, create = TRUE)

# creates a new, "empty" module -- it has defaults for everything that is required
newModule("testModule", tmpdir, open = FALSE)

# Look at new module code -- see defineParameter
if (interactive()) file.edit(file.path(tmpdir, "testModule", "testModule.R"))

# initialize the simList
mySim <- simInit(modules = "testModule",
                  paths = list(modulePath = tmpdir))

# Access one of the parameters -- because this line is not inside a module
# function, we must specify the module name. If used within a module,
```
## depsEdgeList

Build edge list for module dependency graph

### Description

Build edge list for module dependency graph

### Usage

```r
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

#### Usage

```r
depsGraph(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

An `igraph()` object.

#### Author(s)

Alex Chubaty

### dhour

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

dhour(x)
dday(x)
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)
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, and create a function to calculate the number of seconds in that unit using the "d" prefix (for duration), following the lubridate package standard: ddecade <- function(x) lubridate::duration(dyear(10)). Then the module developer can use "decade" as the module's time unit.

Value

Number of seconds within each unit
doEvent.checkpoint

Author(s)
Eliot McIntire

Simulation checkpoints.

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

Usage

```r
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".
downloadData

    debug  Optional logical flag determines whether sim debug info will be printed (default debug = FALSE).
    file   The checkpoint file.
    value  The parameter value to be set (in the corresponding module and param).

Details

RNG save code adapted from: http://www.cookbook-r.com/Numbers/Saving_the_state_of_the_random_number_generator/ and https://stackoverflow.com/q/13997444/1380598

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(), modules(), objs(), packages(), params(), paths(), progressInterval(), times()

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,  
    quickCheck = FALSE,  
    overwrite = FALSE,  
    files = NULL,  
    checked = NULL,  
    urls = NULL,  
    children = NULL,  
    ...  
)
## S4 method for signature 'character,character,logical'

```r
downloadData(
  module,
  path,
  quiet,
  quickCheck = FALSE,
  overwrite = FALSE,
  files = NULL,
  checked = NULL,
  urls = NULL,
  children = NULL,
  ...
)
```

## S4 method for signature 'character,missing,missing'

```r
downloadData(module, quickCheck, overwrite, files, checked, urls, children)
```

## S4 method for signature 'character,missing,logical'

```r
downloadData(
  module,
  quiet,
  quickCheck,
  overwrite,
  files,
  checked,
  urls,
  children
)
```

## S4 method for signature 'character,character,missing'

```r
downloadData(
  module,
  path,
  quickCheck,
  overwrite,
  files,
  checked,
  urls,
  children
)
```

### 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.
quickCheck Logical. If TRUE, then the check with local data will only use file.size instead of digest::digest. This is faster, but potentially much less robust.

overwrite Logical. Should local data files be overwritten in case they exist? Default is FALSE.

files A character vector of length 1 or more if only a subset of files should be checked in the ‘CHECKSUMS.txt’ file.

checked The result of a previous checksums call. This should only be used when there is no possibility that the file has changed, i.e., if downloadData is called from inside another function.

urls Character vector of urls from which to get the data. This is automatically found from module metadata when this function invoked with SpaDES.core::downloadModule(..., data = TRUE). See also prepInputs().

children The character vector of child modules (without path) to also run downloadData on

... Passed to reproducible::preProcess(), e.g., purge

Details

downloadData requires a checksums file to work, as it will only download the files specified therein. Hence, module developers should make sure they have manually downloaded all the necessary data and ran checksums to build a checksums file.

There is an experimental attempt to use the googledrive package to download data from a shared (publicly or with individual users) file. To try this, put the Google Drive URL in sourceURL argument of expectsInputs in the module metadata, and put the filename once downloaded in the objectName argument. If using RStudio Server, you may need to use "out of band" authentication by setting options(httr_oob_default = TRUE). To avoid caching of Oauth credentials, set options(httr_oauth_cache = TRUE).

There is also an experimental option for the user to make a new ‘CHECKSUMS.txt’ file if there is a sourceURL but no entry for that file. This is experimental and should be used with caution.

Value

Invisibly, a list of downloaded files.

Author(s)

Alex Chubaty & Eliot McIntire

See Also

prepInputs(), checksums(), and downloadModule() for downloading modules and building a checksums file.
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, quickCheck = FALSE, overwrite = FALSE
)

## S4 method for signature
## 'character,character,character,character,logical,logical,ANY,logical'
downloadModule(
  name, path, version, repo, data, quiet, quickCheck = FALSE, overwrite = FALSE
)

## S4 method for signature

Examples

## Not run:
# For a Google Drive example
# In metadata:
expectsInputs("theFilename.zip", "NA", "NA",
  sourceURL = "https://drive.google.com/open?id=1Ngb-8lRCs1G6zcuaaCEFUw1dbkI_K8Ez")
# create the checksums file
checksums("thisModule", "there", write = TRUE)
downloadData("thisModule", "there", files = "theFilename.zip")

## End(Not run)
downloadModule

```
## 'character,missing,missing,missing,missing,missing,ANY,ANY'
downloadModule(name, quickCheck, overwrite)

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

**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, specified as "username/repo". Default is "PredictiveEcology/SpaDES-modules" which is specified by the global option spades.moduleRepo. Only master branches can be used at this point.
- **data**: Logical. If TRUE, then the data that is identified in the module metadata will be downloaded, if possible. Default FALSE.
- **quiet**: Logical. This is passed to download.file (default FALSE).
- **quickCheck**: Logical. If TRUE, then the check with local data will only use file.size instead of digest::digest. This is faster, but potentially much less robust.
- **overwrite**: Logical. Should local module files be overwritten in case they exist? Default FALSE.

**Details**

Currently only works with GitHub repositories where modules are located in a modules directory in the root tree on the master branch. Module .zip files' names should contain the version number and be inside their respective module folders (see zipModule() for zip compression of modules).

**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, and whether it was renamed (because there was a local copy that had the wrong file name).
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 with no additional permissions at https://github.com/settings/tokens, and add this key to `.Renviron` as GITHUB_PAT=<your-github-pat-here>. The default is to overwrite any existing files in the case of a conflict.

Author(s)
Alex Chubaty

See Also
zipModule() for creating module .zip folders.

envir
Simulation environment

Description
Accessor functions for the `.xData` slot, which is the default virtual slot for an S4 class object that inherits from an S3 object (specifically, the simList inherits from environment) in a simList object. These are included for advanced users.

Usage

```r
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.
eventDiagram

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(), modules(), obj(), 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 in 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: https://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

DiagrammeR::mermaid.

Description

Accessor functions for the events and completed slots of a simList object. These path functions will extract the values that were provided to the simInit function in the path argument.

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
conditionalEvents(sim, unit)

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

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

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, times = TRUE)

## S4 method for signature 'simList,character'
completed(sim, unit, times = TRUE)

## S4 method for signature 'simList,missing'
completed(sim, unit, times = TRUE)

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.
times          Logical. Should this function report the clockTime

Details

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 simulation events.

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(), 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'
eventsInput(objectName, objectClass, desc, sourceURL, ...)

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

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectName</td>
<td>Character string to define the input object’s name.</td>
</tr>
<tr>
<td>objectClass</td>
<td>Character string to specify the input object’s class.</td>
</tr>
<tr>
<td>desc</td>
<td>Text string providing a brief description of the input object. If there are extra spaces or carriage returns, these will be stripped, allowing for multi-line character strings without using <code>paste</code> or multiple quotes.</td>
</tr>
<tr>
<td>sourceURL</td>
<td>Character string to specify an URL to reach the input object, default is <code>NA</code>.</td>
</tr>
</tbody>
</table>

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

```r
inputObjects <- bindrows(
  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")
)
```

---

**deprecated functions**

Description

These functions have been moved to `SpaDES.experiment` package.

Usage

```r
experiment(...)  
experiment2(...)  
POM(...)  
simInitAndExperiment(...)  
```

Arguments

... Unused.
### extractURL

**Extract a url from module metadata**

**Description**

This will get the sourceURL for the object named.

**Usage**

```
extractURL(objectName, sim, module)
```

### S4 method for signature 'character,missing'

```
extractURL(objectName, sim, module)
```

### S4 method for signature 'character,simList'

```
extractURL(objectName, sim, module)
```

**Arguments**

- `objectName` A character string of the object name in the metadata.
- `sim` A simList object from which to extract the sourceURL.
- `module` An optional character string of the module name whose metadata is to be used. If omitted, the function will use the currentModule(sim), if defined.

**Value**

The url.

**Author(s)**

Eliot McIntire

### 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
getModuleVersion

Value
A character vector.

Author(s)
Eliot McIntire

---

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

Description

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, specified as "username/repo". Default is "PredictiveEcology/SpaDES-modules", which is specified by the global option spades.moduleRepo. Only master branches can be used at this point.

Details

getModuleVersion extracts a module’s most recent version by looking at the module ‘.zip’ files contained in the module directory. It takes the most recent version, based on the name of the zip file.

See the modules vignette for details of module directory structure ([https://spades-core.predictiveecology.org/articles/ii-modules.html#module-directory-structure-modulename](https://spades-core.predictiveecology.org/articles/ii-modules.html#module-directory-structure-modulename)), and see our SpaDES-modules repo for details of module repository structure ([https://github.com/PredictiveEcology/SpaDES-modules](https://github.com/PredictiveEcology/SpaDES-modules)).

Author(s)

Alex Chubaty

See Also

zipModule() for creating module ‘.zip’ folders.
globals

Get and set global simulation parameters

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 parameter value to be set (in the corresponding module and param).

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(), 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**

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

```r
## S4 method for signature 'simList_'
initialize(.Object, ...)
```

**Arguments**

- `.Object` A simList object.
- `...` Optional Values passed to any or all slot

---

`inputObjects`  

**Metadata accessors**

**Description**

These accessors extract the metadata for a module (if specified) or all modules in a simList if not specified.

**Usage**

```r
inputObjects(sim, module, path)
```

```r
## S4 method for signature 'simList'
inputObjects(sim, module, path)
```

```r
## S4 method for signature 'missing'
inputObjects(sim, module, path)
```

```r
outputObjects(sim, module, path)
```

```r
## S4 method for signature 'simList'
```
outputObjects(sim, module, path)

## S4 method for signature 'missing'
outputObjects(sim, module, path)

outputObjectNames(sim, module)

## S4 method for signature 'simList'
outputObjectNames(sim, module)

reqdPkgs(sim, module, modulePath)

## S4 method for signature 'simList'
reqdPkgs(sim, module, modulePath)

## S4 method for signature 'missing'
reqdPkgs(sim, module, modulePath)

documentation(sim, module)

## S4 method for signature 'simList'
documentation(sim, module)

citation(package, lib.loc = NULL, auto = NULL, module = character())

## S4 method for signature 'simList'
citation(package, lib.loc = NULL, auto = NULL, module = character())

## S4 method for signature 'character'
citation(package, lib.loc = NULL, auto = NULL, module = character())

Arguments

**sim**  
A simList object from which to extract element(s) or in which to replace element(s).

**module**  
Character vector of module name(s)

**path**  
The path to the module, i.e., the modulePath. Only relevant if sim not supplied.

**modulePath**  
That path where module can be found. If set already using setPaths, it will use that. This will be ignored if sim is supplied and is required if sim not supplied

**package**  
For compatibility with `utils::citation()`
This can be a simList or a character string for a package name.

**lib.loc**  
a character vector with path names of R libraries, or the directory containing the source for package, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.

**auto**  
a logical indicating whether the default citation auto-generated from the package 'DESCRIPTION' metadata should be used or not, or NULL (default), indicating that
a 'CITATION' file is used if it exists, or an object of class "packageDescription" with package metadata (see below).

Examples

```r
## Not run:
# To pre-install and pre-load all packages prior to 'simInit'.

# set modulePath
setPaths(modulePath = system.file("sampleModules", package = "SpaDES.core"))
# use Require and reqdPkgs
if (!interactive()) chooseCRANmirror(ind = 1) #
pkgs <- reqdPkgs(module = c("caribouMovement", "randomLandscapes", "fireSpread"))
pkgs <- unique(unlist(pkgs))
Require(pkgs)

## End(Not run)
```

Description

Accessor functions for the `inputs` slots in a `simList` object.

Usage

```r
inputs(sim)

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

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

inputArgs(sim)

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

## S4 replacement method for signature 'simList'
inputArgs(sim) <- value
```
inputs 63

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

These functions are one of three mechanisms to add the information about which input files to load in a spades call.

1. As arguments to a simInit call. Specifically, inputs or outputs. See ?simInit.
2. With the 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.

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 simList. This object can therefore be accessed with sim$xxx in any module, where objectName = "xxx". Defaults to the filename without file extension or directory information.
- fun optional, a character string indicating the function to use to load that file. Defaults to the known extensions in SpaDES (found by examining .fileExtensions()). The package and fun can be jointly specified here as "packageName::functionName", e.g., "raster::raster".
- 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 priority at start(sim), i.e., at the very start.
- interval optional numeric, indicating at what interval should this same exact file be reloaded from disk, e.g., 10 would mean every 10 time units.
- arguments is a list of lists of named arguments, one list for each fun. For example, if fun="raster", arguments = list(native = TRUE) will pass the argument "native = TRUE" to raster. If there is only one list, then it is assumed to apply to all files and will be recycled as per normal R rules of recycling for each fun.

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 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:

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

### 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()`, `.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 function 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)
}

# Example showing loading multiple objects from global environment onto the
# same object in the simList, but at different load times
a1 <- 1
a2 <- 2
# Note arguments must be a list of NROW(inputs), with each element itself being a list,
# which is passed to do.call(fun[x], arguments[[x]]), where x is row number, one at a time
args <- lapply(1:2, function(x) {
  list(x = paste0("a", x),
       envir = environment()) # may be necessary to specify in which envir a1, a2
# are located, if not in an interactive sessino
})
inputs <- data.frame(objectName = "a", loadTime = 1:2, fun = "base::get", arguments = I(args))
a <- simInit(inputs = inputs, times = list(start = 0, end = 1))
a <- spades(a)
identical(a1, a$a)
end(a) <- 3
a <- spades(a) # different object (a2) loaded onto a$a
identical(a2, a$a)

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

---

### inSeconds

**Convert time units**

**Description**

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 (you will need to declare "lubridate" in your pkgDeps in the metadata).

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

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

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

```r
.spadesTimes
```

```r
spadesTimes()
```

```r
checkTimeunit(unit, envir)
```

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

```r
## 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 12.

Details

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 unit attribute, then it is assumed to be seconds.
loadPackages

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

Author(s)
Alex Chubaty & Eliot McIntire
Eliot McIntire

## 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

```r
## 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)
```

## loadSimList

### Load a saved simList and ancillary files

#### Description

Loading a simList from file can be problematic as there are non-standard objects that must be rebuilt. See description in `saveSimList()` for details.

`unzipSimList` is a convenience wrapper around `unzip` and `loadSimList` where all the files are correctly identified and passed to `loadSimList(..., otherFiles = xxx)`. See `zipSimList` for details.

#### Usage

```r
loadSimList(filename, paths = getPaths(), otherFiles = "")
unzipSimList(zipfile, load = TRUE, paths = getPaths(), ...)
```

#### Arguments

- **filename**: Character giving the name of a saved simulation file. Currently, only file types `.qs` or `.rds` are supported.
- **paths**: A list of character vectors for all the simList paths. When loading a simList, this will replace the paths of everything to these new paths. Experimental still.
- **otherFiles**: A character vector of (absolute) file names locating each of the existing file-backed Raster* files that are the real paths for the possibly incorrect paths in Filenames(sim) if the the file being read in is from a different computer, path, or drive. This could be the output from unzipSimList (which is calls loadSimList internally, passing the unzipped filenames)
- **zipfile**: Filename of a zipped simList
- **load**: Logical. If TRUE, the default, then the simList will also be loaded into R.
- **...**: passed to unzip

#### Details

If cache is used, it is likely that it should be trimmed before zipping, to include only cache elements that are relevant.
Value

- `loadSimList()`: A `simList` object.
- `unzipSimList()`: Either a character vector of file names unzipped (if `load = FALSE`), or a `simList` object.

See Also

`saveSimList()`, `zipSimList()`

Description

Because of the environment slot, `simList` objects don’t correctly memoise a `simList`. This method for `simList` converts the object to a `simList_` first.

Usage

```r
## S3 method for class 'simList'
makeMemoisable(x)

## S3 method for class 'simList_'
unmakeMemoisable(x)
```

Arguments

- `x` An object to make memoisable. See individual methods in other packages.

Value

A `simList_` object or a `simList`, in the case of `unmakeMemoisable`.

See Also

`reproducible::makeMemoisable()`
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

memoryUseThisSession

Estimate memory used with system("ps")

Description

This will give a slightly different estimate than pryr::mem_used, which uses gc() internally. The purpose of this function is to allow continuous monitoring, external to the R session. Normally, this is run in a different session.

This will only work if the user has specified before running the spades call, set the interval, in seconds, that ps is run with options("spades.memoryUseInterval" = 0.5), will assess memory use every 0.5 seconds. The default is 0, meaning no interval, "off".

Usage

memoryUseThisSession(thisPid)

memoryUse(sim, max = TRUE)
minTimeunit

Arguments

thisPid Numeric or integer, the PID of the process. If omitted, it will be found with Sys.getpid()

sim A completed simList

max Logical. If TRUE, then it the return value will be summarized by module/event, showing the maximum memory used. If FALSE, then the raw memory used during each event will be shown.

See Also

The vignette("iv-modules")

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
**moduleCodeFiles**

*Extract the full file paths for R source code*

**Description**

This can be used e.g., for Caching, to identify which files have changed.

**Usage**

```r
moduleCodeFiles(paths, modules)
```

**Arguments**

- **paths**
  - An optional named list with up to 4 named elements, `modulePath`, `inputPath`, `outputPath`, and `cachePath`. See details. NOTE: Experimental feature now allows for multiple `modulePath`s to be specified in a character vector. The modules will be searched for sequentially in the first `modulePath`, then if it doesn’t find it, in the second etc.

- **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).

**moduleCoverage**

*Calculate module coverage of unit tests*

**Description**

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

**Usage**

```r
moduleCoverage(mod, modulePath = "..")
```

**Arguments**

- **mod**
  - Character string. The module’s name. Default is `basename(getwd())`

- **modulePath**
  - Character string. The path to the module directory (default is "..", i.e., one level up from 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 covr::report() 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 follow the format test-functionName.R.

Author(s)

Yong Luo

See Also

newModule().

Examples

```r
## Not run:
library(moduleDefault)
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)
report(testResults$moduleCoverage)
report(testResults$functionCoverage)
unlink(tmpdir, recursive = TRUE)
mc1 <- moduleCoverage("Biomass_core", modulePath = ".")

## End(Not run)
```
Format

An object of class list of length 12.

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 = TRUE, ...)

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

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

Arguments

sim A simList object (typically corresponding to a completed simulation).

type Character string, either "rgl" for igraph::rglplot or "tk" for igraph::tkplot, "Plot" to use quickPlot::Plot or "plot" to use base::plot, the default.

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.

Author(s)

Eliot McIntire

See Also

moduleDiagram

moduleMetadata

Parse and extract module metadata

Description

Parse and extract module metadata

Usage

moduleMetadata(
  sim,
  module,
  path = getOption("spades.modulePath", NULL),
  defineModuleListItems = c("name", "description", "keywords", "childModules", "authors", "version", "spatialExtent", "timeframe", "timeunit", "citation", "documentation", "reqdPkgs", "parameters", "inputObjects", "outputObjects")
)

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

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

## S4 method for signature 'ANY,ANY,ANY'
moduleMetadata(
  sim,
  module,
  path = getOption("spades.modulePath", NULL),
moduleMetadata

```r
defineModuleListItems = c("name", "description", "keywords", "childModules", "authors", "version", "spatialExtent", "timeframe", "timeunit", "citation", "documentation", "reqdPkgs", "parameters", "inputObjects", "outputObjects")
```

### Arguments

- **sim**
  - A `simList` simulation object, generally produced by `simInit`.  
- **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.  
- **defineModuleListItems**
  - A vector of metadata entries to return values about.

### Value

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

### Author(s)

Alex Chubaty

### See Also

`defineModule()`

### Examples

```r
path <- system.file("sampleModules", package = "SpaDES.core")
sampleModules <- dir(path)
## turn off code checking -- don't need it here
opts <- options("spades.moduleCodeChecks" = FALSE, "spades.useRequire" = FALSE)

x <- moduleMetadata(sampleModules[3], path = path)
## using simList
if (require("SpaDES.tools", quietly = TRUE)) {
  mySim <- simInit(
    times = list(start = 2000.0, end = 2001.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape")
    ),
    modules = list("caribouMovement"),
    paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
  )
  moduleMetadata(sim = mySim)
}
# turn code checking back on -- don't need it here
options(opts)
```
moduleParams

### Extract a module’s parameters, inputs, or outputs

**Description**

These are more or less wrappers around moduleMetadata, with the exception that extraneous spaces and End-Of-Line characters will be removed from the desc arguments in defineParameters, defineInputs, and defineOutputs.

**Usage**

```r
moduleParams(module, path)
```

### S4 method for signature 'character,character'

```r
moduleParams(module, path)
```

```r
moduleInputs(module, path)
```

### S4 method for signature 'character,character'

```r
moduleInputs(module, path)
```

```r
moduleOutputs(module, path)
```

### S4 method for signature 'character,character'

```r
moduleOutputs(module, path)
```

**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.

**Value**

data.frame

**Author(s)**

Alex Chubaty

**See Also**

- moduleMetadata()
Examples

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

p <- moduleParams(sampleModules[3], path = path)
i <- moduleInputs(sampleModules[3], path = path)
o <- moduleOutputs(sampleModules[3], path = path)

#' 
dontrun{
## easily include these tables in Rmd files using knitr
knitr::kable(p)
knitr::kable(i)
knitr::kable(o)
#'}
```

---

**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

```r
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()`, `objs()`, `packages()`, `params()`, `paths()`, `progressInterval()`, `times()`

---

### moduleVersion

Parse and extract a module's version

### Description

Parse and extract a module’s version

### Usage

```r
moduleVersion(module, path, sim, envir = NULL)
```

## S4 method for signature 'character,character,missing'

`moduleVersion(module, path, envir)`

## S4 method for signature 'character,missing,missing'

`moduleVersion(module, envir)`

## S4 method for signature 'character,missing,simList'

`moduleVersion(module, sim, envir)`
moduleVersion

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`.
- **envir**: Optional environment in which to store parsed code. This may be useful if the same file is being parsed multiple times. This function will check in that envir for the parsed file before parsing again. If the `envir` is transient, then this will have no effect.

Value

A numeric version indicating the module’s version.

Author(s)

Alex Chubaty

See Also

`moduleMetadata()`

Examples

```r
# using filepath
path <- system.file("sampleModules", package = "SpaDES.core")
moduleVersion("caribouMovement", path)

# using simList
options("spades.useRequire" = FALSE)
if (require("SpaDES.tools", quietly = TRUE)) {
  mySim <- simInit(
    times = list(start = 2000.0, end = 2002.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
    ),
    modules = list("caribouMovement"),
    paths = list(modulePath = path)
  )
  moduleVersion("caribouMovement", sim = mySim)
}
```
newModule

Create new module from template

Description

Generate a skeleton for a new SpaDES module, a template for a documentation file, a citation file, a license file, a ‘README.md’ 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:
  - **children**: Required when type = "parent". A character vector specifying the names of child modules.
  - **open**: Logical. Should the new module file be opened after creation? Default TRUE.
  - **type**: Character string specifying one of "child" (default), or "parent".
  - **unitTests**: Logical. Should the new module include unit test files? Default TRUE. Unit testing relies on the testthat package.
  - **useGitHub**: Logical. Is module development happening on GitHub? Default TRUE.

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
```
newModuleCode

Create new module code file

Description

Create new module code file

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 overlaid 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

```r
## 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"))
```

```r
## End(Not run)
```
newModuleDocumentation

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 in an interactive session.
- **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()`

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)
**newModuleTests**

**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` in an interactive session.
- **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**

```r
newModuleTests(name, path, open, useGitHub)
```

## S4 method for signature 'character,character,logical,logical'

```r
newModuleTests(name, path, open, useGitHub)
```

**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` in an interactive session.
- **useGitHub**
  - Logical indicating whether GitHub will be used. If `TRUE` (default), creates suitable configuration files (e.g., `.gitignore`) and configures basic GitHub actions for module code checking.

**Author(s)**

Eliot McIntire and Alex Chubaty
newProject

Create new SpaDES project

Description
Initialize a project with subdirectories `cache/`, `modules/`, `inputs/`, `outputs/`, and setPaths accordingly. If invoked from Rstudio, will also create a new Rstudio project file.

Usage
```
newProject(name, path, open)
```

Arguments
- `name`: character
- `path`: character
- `open`: logical

Details
If invoked from Rstudio, will also create a new Rstudio project file.

Author(s)
Alex Chubaty and Eliot McIntire

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
newProjectCode

Arguments

- **name**: project name (name of project directory)
- **path**: path to directory in which to create the project directory
- **open**: Logical. Should the new project file be opened after creation? Default TRUE in an interactive session.

Examples

```r
myProjDir <- newProject("myProject", tempdir())

dir.exists(file.path(myProjDir, "cache"))
dir.exists(file.path(myProjDir, "inputs"))
dir.exists(file.path(myProjDir, "modules"))
dir.exists(file.path(myProjDir, "outputs"))
unlink(myProjDir, recursive = TRUE) # cleanup
```

newProjectCode Create new module code file

Description

Create new module code file

Usage

```r
newProjectCode(name, path, open)
```

## S4 method for signature 'character,character,logical'
```r
newProjectCode(name, path, open = interactive())
```

Arguments

- **name**: project name (name of project directory)
- **path**: path to directory in which to create the project directory
- **open**: Logical. Should the new project file be opened after creation? Default TRUE in an interactive session.

Author(s)

Alex Chubaty
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 DiagrammeR::mermaid. Useful for specifying height and width.

Value

Plots a sequence diagram, invisibly returning a DiagrammeR::mermaid object.

Author(s)

Alex Chubaty

See Also

DiagrammeR::mermaid.

Examples

## Not run:
objectDiagram(sim)
# if there are lots of objects, may need to increase width and/or height
objectDiagram(sim, height = 3000, width = 3000)

## End(Not run)
**objectSynonyms**

*Identify synonyms in a simList*

**Description**

This will create active bindings amongst the synonyms. To minimize copying, the first one that exists in the character vector will become the "canonical" object. All others named in the character vector will be activeBindings to that canonical one. This synonym list will be assigned to the envir, as an object named objectSynonyms. That object will have an attribute called, bindings indicating which one is the canonical one and which is/are the activeBindings. EXPERIMENTAL: If the objects are removed during a spades call by, say, a module, then at the end of the event, the spades call will replace the bindings. In other words, if a module deletes the object, it will "come back". This may not always be desired.

**Usage**

objectSynonyms(envir, synonyms)

**Arguments**

- envir: An environment, which in the context of SpaDES.core is usually a simList to find and/or place the objectSynonyms object.
- synonyms: A list of synonym character vectors, such as list(c("age", "ageMap", "age2"), c("veg", "vegMap"))

**Details**

This is very experimental and only has minimal tests. Please report if this is not working, and under what circumstances (e.g., please submit a reproducible example to our issues tracker)

This function will append any new objectSynonym to any pre-existing objectSynonym in the envir. Similarly, this function assumes transitivity, i.e., if age and ageMap are synonyms, and ageMap and timeSinceFire are synonyms, then age and timeSinceFire must be synonyms.

**Value**

Active bindings in the envir so that all synonyms point to the same canonical object, e.g., they would be at envir[[synonym[[1]][1] and envir[[synonym[[1]][2]], if a list of length one is passed into synonyms, with a character vector of length two. See examples.

**Examples**

```r
sim <- simInit()
sim$age <- 1:10;
sim <- objectSynonyms(sim, list(c("age", "ageMap")))
identical(sim$ageMap, sim$age)
```
objs <- 4
identical(sim$ageMap, sim$age)
sim$ageMap <- 2:5
sim$ageMap[3] <- 11
identical(sim$ageMap, sim$age)

# Also works to pass it in as an object
objectSynonyms <- list(c("age", "ageMap"))
sim <- simInit(objects = list(objectSynonyms = objectSynonyms))
identical(sim$ageMap, sim$age) # they are NULL at this point
sim$age <- 1:10
identical(sim$ageMap, sim$age) # they are not NULL at this point

## More complicated, with 'updating' i.e., you can add new synonyms to previous
sim <- simInit()
os <- list(c("age", "ageMap"), c("vegMap", "veg"), c("studyArea", "studyArea2"))
os2 <- list(c("ageMap", "timeSinceFire", "tsf"),
          c("systime", "systime2"),
          c("vegMap", "veg"))
sim <- objectSynonyms(sim, os)
sim <- objectSynonyms(sim, os2)

# check
sim$objectSynonyms

---

**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'
oobjs(sim) <- value
Arguments

`sim` A `simList` object from which to extract element(s) or in which to replace element(s).

... passed to `ls`

`value` objects to assign to the `simList`

Details

`objs` can take... arguments passed to `ls`, allowing, e.g. `all.names=TRUE` `obj<-` 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()`, `modules()`, `packages()`, `params()`, `paths()`, `progressInterval()`, `times()`

---

**objSize.simList**

Object size for `simList`

---

Description

Recursively, runs `reproducible::objSize()` on the `simList` environment, so it estimates the correct size of functions stored there (e.g., with their enclosing environments) plus, it adds all other 'normal' elements of the `simList`, e.g., `objSize(completed(sim))`. The output is structured into 2 elements: the sim environment and all its objects, and the other slots in the `simList` (e.g., events, completed, modules, etc.). The returned object also has an attribute, "total", which shows the total size.

Usage

```r
## S3 method for class 'simList'
objSize(x, quick = TRUE, ...)
```

Arguments

- `x` An object
- `quick` Logical. If FALSE, then an attribute, "objSize" will be added to the returned value, with each of the elements’ object size returned also.
- ... Additional arguments (currently unused), enables backwards compatible use.
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

```r
## Not run: openModules("~/path/to/my/modules")
```

```r
<table>
<thead>
<tr>
<th>outputs</th>
<th>Simulation outputs</th>
</tr>
</thead>
</table>
```

Description

Accessor functions for the `outputs` slots in a `simList` object.

Usage

```r
outputs(sim)
```

```r
## S4 method for signature 'simList'
outputs(sim)
```

```r
outputs(sim) <- value
```

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

```r
outputArgs(sim)
```

```r
## S4 method for signature 'simList'
outputArgs(sim)
```

```r
outputArgs(sim) <- value
```

```r
## 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

These functions are one of three mechanisms to add information about which output files to save.

1. As arguments to a simInit call. Specifically, inputs or outputs. See ?simInit.
2. With the 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.

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 the sim$ prefix).
- **file** optional, a character string indicating the file path to save to. The default is to concatenate objectName with the simulation time unit and saveTime, separated by underscore. So a default filename would be "Fires_year1.rds".
- **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 at end(sim), i.e., at the very end.
- **arguments** is a list of lists of named arguments, one list for each fun. For example, if fun = "write.csv", arguments = list(row.names = TRUE)

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.

Examples

```r
# 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 = "year10.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")[[1]])
  all.equal(newRas, ras) # Should be TRUE
}

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

Get module or simulation package dependencies

Description

Get module or simulation package dependencies

Usage

packages(sim, modules, paths, filenames, envir, clean = FALSE, ...)

## S4 method for signature 'ANY'
packages(sim, modules, paths, filenames, envir, clean = FALSE, ...)

Arguments

sim A simList object.
modules Character vector, specifying the name or vector of names of module(s)
paths Character vector, specifying the name or vector of names of paths(s) for those modules. If path not specified, it will be taken from getOption("spades.modulePath"), which is set with setPaths)
filenames Character vector specifying filenames of modules (i.e. combined path & module. If this is specified, then modules and path are ignored.
envir Optional environment in which to store parsed code. This may be useful if the same file is being parsed multiple times. This function will check in that envir for the parsed file before parsing again. If the envir is transient, then this will have no effect.
clean Optional logical. If TRUE, it will scrub any references to github repositories, e.g., "PredictiveEcology/reproducible" will be returned as "reproducible"
...

All simInit parameters.

Value

A sorted character vector of package names.

Author(s)

Alex Chubaty & Eliot McIntire

See Also

Other functions to access elements of a 'simList' object: .addDepends(), doEvent.checkpoint(), envir(), events(), globals(), inputs(), modules(), objs(), params(), paths(), progressInterval(), times()
paramCheckOtherMods  Test and update a parameter against same parameter in other modules

Description

This function is intended to be part of module code and will test whether the value of a parameter within the current module matches the value of the same parameter in other modules. This is a test for parameters that might expect to be part of a `params = list(.globals = list(someParam = "test"))` passed to the `simInit`

Usage

```
paramCheckOtherMods(
  sim, paramToCheck, moduleToUse = "all",
  ifSetButDifferent = c("error", "warning", "message", "silent")
)
```

Arguments

- `sim` A simList
- `paramToCheck` A character string, length one, of a parameter name to check and compare between the current module and one or more or all others
- `moduleToUse` A character vector of module names to check against. This can be "all" which will compare against all other modules.
- `ifSetButDifferent` A character string indicating whether to "error" the default, or send a "warning", message or just silently continue (any other value).

Value

If the value of the `paramToCheck` in the current module is either `NULL` or "default", and there is only one other value across all modules named in `moduleToUse`, then this will return a character string with the value of the single parameter value in the other module(s). It will return the current value if there are no other modules with the same parameter.

It is considered a "fail" under several conditions:

1. current module has a value that is not `NULL` or "default" and another module has a different value;
2. there is more than one value for the `paramToCheck` in the other modules, so it is ambiguous which one to return.

either the current module is different than other modules, unless it is "default" or `NULL`.
**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.

**Usage**

```r
params(sim)
## S4 method for signature 'simList'
params(sim)
params(sim) <- value
## S4 replacement method for signature 'simList'
params(sim) <- value
P(sim, param, module)
P(sim, param, module) <- value
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 parameter value to be set (in the corresponding module and param).
- **param**: Optional character string indicating which parameter to choose.
- **module**: Optional character string indicating which module params should come from.
- **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`.

**Details**

`parameters` will extract only the metadata with the metadata defaults, NOT the current values that may be overwritten by a user. See examples.
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, "paramName", "moduleName") (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% faster.

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(), modules(), objs(), packages(), paths(), progressInterval(), times()

Examples

```r
if (require("NLMR", quietly = TRUE) && require("SpaDES.tools", quietly = TRUE)) {
  modules <- list("randomLandscapes")
  paths <- list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
  mySim <- simInit(modules = modules, paths = paths,
                   params = list(.globals = list(stackName = "landscape")))

  # update some parameters using assignment -- currently only params will work
  params(mySim)$randomLandscapes$nx <- 200
  params(mySim)$randomLandscapes$ny <- 200

  parameters(mySim) # Does not contain these user overridden values

  # These next 2 are same here because they are not within a module
  P(mySim) # Does contain the user overridden values
  params(mySim) # Does contain the user overridden values

  # NOTE -- deleting a parameter will affect params and P, not parameters
  params(mySim)$randomLandscapes$nx <- NULL
  params(mySim)$randomLandscapes$ny <- NULL

  parameters(mySim) # Shows nx and ny

  # These next 2 are same here because they are not within a module
  P(mySim) # nx and ny are Gone
  params(mySim) # nx and ny are Gone
}
```
paths

Specify paths for modules, inputs, outputs, and temporary rasters

Description

Accessor functions for the paths slot in a simList object.

dataPath will return file.path(modulePath(sim), currentModule(sim), "data"). dataPath, like currentModule, is namespaced. This means that when it is used inside a module, then it will return that model-specific information. For instance, if used inside a module called "movingAgent", then currentModule(sim) will return "movingAgent". and dataPath(sim) will return file.path(modulePath(sim), "movingAgent", "data")

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

logPath(sim)

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

modulePath(sim, module)

## S4 method for signature 'simList'
modulePath(sim, module)

modulePath(sim) <- value

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

scratchPath(sim)

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

scratchPath(sim) <- value

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

rasterPath(sim)

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

rasterPath(sim) <- value

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

terraPath(sim)

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

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

dataPath(sim)

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

**Arguments**

- **sim**  
  A simList object from which to extract element(s) or in which to replace element(s).
- **value**  
  The parameter value to be set (in the corresponding module and param).
- **module**  
  The optional character string of the module(s) whose paths are desired. If omitted, will return all modulePaths, if more than one exist.

**Details**

These are ways to add or access the file paths used by `spades()`. There are five file paths: cachePath, modulePath, inputPath, outputPath, and rasterPath. 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("reproducible.cachePath")`
- inputPath: `getOption("spades.modulePath")`
- modulePath: `getOption("spades.inputPath")`
- outputPath: `getOption("spades.outputPath")`
- rasterPath: `file.path(getOption("spades.scratchPath"), "raster")`
- scratchPath: `getOption("spades.scratchPath")`
- terraPath: `file.path(getOption("spades.scratchPath"), "terra")`

**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()`, `modules()`, `objs()`, `packages()`, `params()`, `progressInterval()`, `times()`
Description

Extends quickPlot::Plot for simList objects.

Usage

```r
## 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 = NULL,
  na.color = "#FFFFFF00",
  zero.color = NULL,
  length = NULL,
  arr = NULL,
  plotFn = "plot"
)
```

Arguments

- `...`: A combination of spatialObjects or non-spatial objects. For many object classes, there are specific Plot methods. Where there are no specific ones, the base plotting will be used internally. This means that for objects with no specific Plot methods, many arguments, such as `addTo`, will not work. 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. NOTE if TRUE: *Everything that was there, including the legend and the end points of the colour palette, will be removed and re-initiated.*

**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(), 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. Default NULL, which means print the object name as title, if no other already exists on the plot, in which case, keep the previous title.

**na.color**
Character string indicating the colour for NA values. Default transparent.

**zero.color**
Character string indicating the colour for zero values, when zero is the minimum value, otherwise, zero is treated as any other colour. Default transparent.

**length**
Numeric. Optional length, in inches, of the arrow head.
Plots

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

See quickPlot::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

quickPlot::Plot

Plots Plot wrapper intended for use in a SpaDES module

Description

This is a single function call that allows a user to change which format in which the plots will occur. Specifically, the two common formats would be to "screen" or to disk as an image file, such as "png". THIS CURRENTLY HAS BEEN TESTED WITH ggplot2, RasterLayer, and tmap objects. The default (or change with e.g., fn = "print", usePlot = FALSE) uses Plot internally, so individual plots may be rearranged. When saved to disk (e.g., via type = 'png'), then Plot will not be used and the single object that is the result of this Plots call will be saved to disk. This function requires at least 2 things: a plotting function and arguments passed to that function (which could include data, but commonly would simply be named arguments required by fn). See below and examples.

Usage

Plots(
  data,
  fn,
  filename,
  types = quote(params(sim)[[currentModule(sim)]]$.plots),
  path = quote(file.path(outputPath(sim), "figures")),
  .plotInitialTime = quote(params(sim)[[currentModule(sim)]]$.plotInitialTime),
  ggsaveArgs = list(),
  usePlot = getOption("spades.PlotsUsePlot", FALSE),
  deviceArgs = list(),
  ...
)

Arguments

data An (optional) arbitrary data object. If supplied, it will be passed as the first argument to Plot function, and should contain all the data required for the inner plotting. If passing a RasterLayer, it may be a good idea to set names(RasterLayer) so that multiple layers can be plotted without overlapping each other. When a custom fn is used and all arguments for fn are supplied and named, then this can be omitted. See examples.

fn An arbitrary plotting function. If not provided, defaults to using quickPlot::Plot

filename A name that will be the base for the files that will be saved, i.e, do not supply the file extension, as this will be determined based on types. If a user provides this as an absolute path, it will override the path argument.

types Character vector, zero or more of types. If used within a module, this will be deduced from the P(sim)$type and can be omitted. See below.

path Currently a single path for the saved objects on disk. If filename is supplied as an absolute path, path will be set to dirname(filename), overriding this argument value.

.plotInitialTime A numeric. If NA then no visual on screen. Anything else will have visuals plotted to screen device. This is here for backwards compatibility. A developer should set in the module to the intended initial plot time and leave it, i.e., not NA.

ggsaveArgs An optional list of arguments passed to ggplot2::ggsave

usePlot Logical. If TRUE, the default, then the plot will occur with quickPlot::Plot, so it will be arranged with previously existing plots.

deviceArgs An optional list of arguments passed to one of png, pdf, tiff, bmp, or jpeg. This is useful when the plotting function is not creating a ggplot object, e.g., plotting a RasterLayer.

... Anything needed by fn, all named.

Details

- type
  - "screen" – Will plot to the current device, normally a plot window
  - "object" – Will save the plot object, e.g., ggplot object
  - "raw" – Will save the raw data prior to plotting, e.g., the data argument
  - "png" – or any other type save-able with ggsave

Recording of files saved

In cases where files are saved, and where Plots is used within a SpaDES module, the file(s) that is/are saved will be appended to the outputs slot of the simList of the module. This will, therefore, keep a record of figures saved within the simList
Note

THIS IS STILL EXPERIMENTAL and could change in the next release.

Plots now has experimental support for "just a Plot call", but with types specified. See example. The devices to save on disk will have some different behaviours to the screen representation, since "wiping" an individual plot on a device doesn’t exist for a file device.

This offers up to 4 different actions for a given plot:

• To screen device
• To disk as raw data (limited testing)
• To disk as a saved plot object (limited testing)
• To disk as a `.png’ or other image file, e.g., ‘.pdf’

To turn off plotting both to screen and disk, set both `.plotInitialTime = NA` and `.plots = NA` or any other value that will not trigger a TRUE with a grepl with the types argument (e.g., "" will omit all saving).

Examples

```r
## Not run:
# Note: if this is used inside a SpaDES module, do not define this
# function inside another function. Put it outside in a normal
# module script. It will cause a memory leak, otherwise.
if (!require("ggplot2")) stop("please install ggplot2")
fn <- function(d)
  ggplot(d, aes(a)) +
  geom_histogram()
sim <- simInit()
sim$something <- data.frame(a = sample(1:10, replace = TRUE))
Plots(data = sim$something, fn = fn,
      types = c("png"),
      path = file.path("figures"),
      filename = tempfile(),
      .plotInitialTime = 1)
  # plot to active device and to png
Plots(data = sim$something, fn = fn,
      types = c("png", "screen"),
      path = file.path("figures"),
      filename = tempfile(),
      .plotInitialTime = 1)
  
# Can also be used like quickPlot::Plot, but with control over output type
r <- raster::raster(extent(0,10,0,10), vals = sample(1:3, size = 100, replace = TRUE))
Plots(r, types = c("screen", "png"), deviceArgs = list(width = 700, height = 500))
```

## End(Not run) # end of dontrun
### priority vs Event priority

<table>
<thead>
<tr>
<th>priority</th>
<th>Event priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Preset event priorities: 1 = first (highest); 5 = normal; 10 = last (lowest).</td>
</tr>
</tbody>
</table>

**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**

```r
progressInterval(sim)
```

```r
## S4 method for signature 'simList'
progressInterval(sim)
```

```r
progressInterval(sim) <- value
```

```r
## 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 parameter value to be set (in the corresponding module and param).

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(), 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)

rasterCreate

Simple wrapper to load any Raster* object. This wraps either raster::raster, raster::stack, or raster::brick, allowing a single function to be used to create a new object of the same class as a template.

**Description**

Simple wrapper to load any Raster* object. This wraps either raster::raster, raster::stack, or raster::brick, allowing a single function to be used to create a new object of the same class as a template.

**Usage**

rasterCreate(x, ...)

## Default S3 method:
rasterCreate(x, ...)

## S3 method for class 'RasterBrick'
rasterCreate(x, ...)

## S3 method for class 'RasterLayer'
rasterCreate(x, ...)

## S3 method for class 'RasterStack'
rasterCreate(x, ...)

## S3 method for class 'Raster'
rasterCreate(x, ...)

**Arguments**

x                  An object, notably a Raster* object. All others will simply be passed through with no effect.
...

Passed to raster::raster, raster::stack, or raster::brick

**Details**

A new (empty) object of same class as the original.
Methods (by class)

- `rasterCreate(default)`: Simply passes through argument with no effect
- `rasterCreate(RasterBrick)`: Uses `raster::brick`
- `rasterCreate(RasterLayer)`: Uses `raster::raster`
- `rasterCreate(RasterStack)`: Uses `raster::stack`
- `rasterCreate(Raster)`: Uses `raster::raster` when one of the other, less commonly used `Raster*` classes, e.g., `RasterLayerSparse`

---

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. There is the default method which is just a pass through, so this can be safely used on large complex objects, recursively, e.g., a `simList`.

Usage

```r
rasterToMemory(x, ...)
```

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

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

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

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

Arguments

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

Value

A raster object whose values are stored in memory.

Author(s)

Eliot McIntire and Alex Chubaty
remoteFileSize

Determine the size of a remotely hosted file

Description

Deprecated.

Usage

remoteFileSize(url)

Arguments

url

The url of the remote file.

Value

A numeric indicating the size of the remote file in bytes.

Author(s)

Eliot McIntire and Alex Chubaty

restartR

Restart R programmatically

Description

This will attempt to restart the R session, reloading all packages, and saving and reloading the simList. Currently, this is not intended for general use: it has many specialized pieces for using inside a spades call. The main purpose for doing this is to clear memory leaks (possibly deep in R [https://github.com/r-lib/fastmap](https://github.com/r-lib/fastmap)) that are not fully diagnosed. *This is still very experimental.* This should only be used if there are RAM limitations being hit with long running simulations. It has been tested to work Linux within Rstudio and at a terminal R session. The way to initiate restarting of R is simply setting the spades.restartRInterval or setting the equivalent parameter in the restartR core module via: simInit(..., params = list(.restartR = list(.restartRInterval = 1)), ...) greater than 0, which is the default, e.g., options("spades.restartRInterval" = 100). This is only intended to restart a simulation in exactly the same place as it was (i.e., cannot change machines), and because of the restart, the assignment of the spades call will be either to sim or the user must make such an assignment manually, e.g., sim <- SpaDES.core:::.pkgEnv$.sim. This is stated in a message.
restartR

Usage

restartR(
    sim,
    reloadPkgs = TRUE,
    .First = NULL,
    .RDataFile = getOption("spades.restartR.RDataFilename"),
    restartDir = getOption("spades.restartR.restartDir", NULL)
)

Arguments

sim Required. A simList to be retained through the restart
reloadPkgs Logical. If TRUE, it will attempt to reload all the packages as they were in previous session, in the same order. If FALSE, it will load no packages beyond normal R startup. Default TRUE
.First A function to save to `~/.qs` which will be loaded at restart from `~/.qs` and run. Default is NULL, meaning it will use the non-exported SpaDES.core:::First. If a user wants to make a custom First file, it should built off that one.
.RDataFile A filename for saving the simList. Defaults to getOption("spades.restartR.filename"), and the directory will be in restartDir. The simulation time will be mid-pended to this name, as in: basename(file), ".time", paddedFloatToChar(time(sim), padL = nchar)

restartDir A character string indicating root directory to save simList and other ancillary files during restart. Defaults to getOption("spades.restartR.restartDir", NULL). If NULL, then it will try, in order, outputPath(sim), modulePath(sim), inputPath(sim), cachePath(sim), taking the first one that is not inside the tempdir(), which will disappear during restart of R. The actual directory for a given spades call that is restarting will be: file.path(restartDir, "restartR", paste0(sim$._startClockTime, ",", .rndString)). The random string is to prevent parallel processes that started at the same clock time from colliding.

Details

The process responds to several options. Though under most cases, the default behaviour should suffice. These are of 3 types: restartRInterval the arguments to restartR and the arguments to saveSimList, these latter two using a dot to separate the function name and its argument. The defaults for two key options are: options("spades.restartR.restartDir" = NULL, meaning use file.path(restartDir, "restartR", paste0(sim$._startClockTime, ",", .rndString)) and options("spades.saveSimList.fileBackend = FALSE during restartR by default, because it is assumed that the file backends will still be intact after a restart, so no need to move them all to memory.

Note

Because of the restarting, the object name of the original assignment of the spades call can not be preserved. The spades call will be assigned to sim in the .GlobalEnv.
Because this function is focused on restarting during a spades call, it will remove all objects in the .GlobalEnv, emulating q("no"). If the user wants to keep those objects, then they should be saved to disk immediately before the spades call. This can then be recovered immediately after the return from the spades call.

To keep the saved simList, use options("spades.restartR.clearFiles" = TRUE). The default is to treat these files as temporary files and so will be removed.

---

**Description**

This is very experimental and has not been thoroughly tested. Use with caution. This function will reparse a single module (currently) into the simList where its source code should reside, and then optionally restart a simulation that stopped on an error, presumably after the developer has modified the source code of the module that caused the break. This will restart the simulation at the next event in the event queue (i.e., returned by events(sim)). Because of this, this function will not do anything if the event queue is empty.

**Usage**

restartSpades(sim = NULL, module = NULL, numEvents = Inf, restart = TRUE, ...)

**Arguments**

- `sim`: A simList. If not supplied (the default), this will take the sim from SpaDES.core:::.pkgEnv$.sim, i.e., the one that was interrupted
- `module`: A character string length one naming the module that caused the error and whose source code was fixed. This module will be reparsed and placed into the simList
- `numEvents`: Numeric. Default is Inf (i.e., all available). In the simList, if options('spades.recoveryMode') is set to TRUE or a numeric, then there will be a list in the simList called .recoverableObjs. These will be replayed backwards in time to reproduce the initial state of the simList before the event that is numEvents back from the first event in events(sim).
- `restart`: Logical. If TRUE, then the call to spades will be made, i.e., restarting the simulation. If FALSE, then it will return a new simList with the module code parsed into the simList
- `...`: Passed to spades, e.g., debug, .plotInitialTime

**Details**

This will only parse the source code from the named module. It will not affect any objects that are in the mod or sim.

The random number seed will be reset to the state it was at the start of the earliest event recovered, thereby returning to the exact stochastic simulation trajectory.
**Value**

A simList as if spades had been called on a simList.

**Note**

This will only work reliably *if the simList was not modified yet during the event which caused the error*. The simList will be in the state it was at the time of the error.

**Examples**

```r
## Not run:
# options("spades.recoveryMode" = 1) # now the default
s <- simInit()
s <- spades(s) # if this is interrupted or fails
s <- restartSpades() # don't need to put simList
    # will take from SpaDES.core:::.pkgEnv$.sim automatically

## End(Not run)
```

### rndstr

<table>
<thead>
<tr>
<th>rndstr</th>
<th>Generate random strings</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```r
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

```r
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

```r
saveFiles(sim)
```
saveFiles

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(
saveSimList

Description

Saving a `simList` may not work using the standard approaches (e.g., `save`, `saveRDS`, and `qs::qsave`). There are 2 primary reasons why this doesn't work as expected: the `activeBindings` that are in place within modules (these allow the `mod` and `Par` to exist), and file-backed rasters. Because of these, a user should use `saveSimList` and `loadSimList` (and the `zipSimList/unzipSimList` alternatives). The most robust way if there are file-backed Raster* objects seems to be to set `fileBackend = 2`, though this may not be desirable if there are many large Raster* objects. When using `fileBackend = 0` or `fileBackend = 1`, and when errors are noticed, please file a bug report on GitHub.

`zipSimList` will save the `simList` and file-backed Raster* objects, plus, optionally, files identified in `outputs(sim)` and `inputs(sim)`. This uses Copy under the hood, to not affect the original `simList`. **VERY experimental.**

Usage

`saveSimList(sim, filename, fileBackend = 0, filebackedDir = NULL, envir, ...)`

`zipSimList(sim, zipfile, ..., outputs = TRUE, inputs = TRUE, cache = FALSE)`
Arguments

- **sim**: Either a `simList` or a character string of the name of a `simList` that can be found in `envir`. Using a character string will assign that object name to the saved `simList`, so when it is recovered it will be given that name.

- **filename**: Character string with the path for saving `simList` to or reading the `simList` from. Currently, only `.rds` and `.qs` file types are supported.

- **fileBackend**: Numeric. 0 means don’t do anything with file backed rasters. Leave their file intact as is, in its place. 1 means save a copy of the file backed rasters in `fileBackedDir`. 2 means move all data in file-backed rasters to memory. This means that the objects will be part of the main `qs` file of the `simList`. Default is 0.

- **filebackedDir**: Only used if `fileBackend` is 1. `NULL`, the default, or Character string. If `NULL`, then the files will be copied to the directory: `file.path(dirname(filename), "rasters")`. A character string will be interpreted as a path to copy all rasters to.

- **envir**: If `sim` is a character string, then this must be provided. It is the environment where the object named `sim` can be found.

- **...**: Passed to `saveSimList()`, including non-optional ones such as `filename`. Also see `fileBackend` and `filebackedDir` arguments in that function.

- **zipfile**: A character string indicating the filename for the zip file. Passed to `zip`.

- **outputs**: Logical. If `TRUE`, all files identified in `outputs(sim)` will be included in the zip.

- **inputs**: Logical. If `TRUE`, all files identified in `inputs(sim)` will be included in the zip.

- **cache**: Logical. Not yet implemented. If `TRUE`, all files in `cachePath(sim)` will be included in the zip archive. Defaults to `FALSE` as this could be large, and may include many out of date elements. See Details.

Details

There is a family of 4 functions that are mutually useful for saving and loading `simList` objects and their associated files (e.g., file-backed Raster*, inputs, outputs, cache) `saveSimList()`, `loadSimList()`, `zipSimList()`, `unzipSimList()`

**Save - Move - Load:**

There are 3 different workflows for "save - move files - load" that work in our tests:

1. `filebackend = 0`: No renaming of file-backed rasters, on recovery attempts to rebuild
   - This approach is attempting to emulate a "relative filenames" approach, i.e., attempt to treat the file-backed raster file names as if they were relative (which they are not – raster package forces absolute file paths). To do this, all the renaming occurs within `loadSimList` or `unzipSimList`. These function will use the paths argument to rewrite the paths of the files that are identified with Filenames(sim) so that they are in the equivalent (relative) position as they were. This will only work if all files were in one of the paths of the original `simList`, so that they can be matched up with the new paths passed in `loadSimList`. This is not guaranteed to work correctly, though it works in a wide array of testing.

```r
zipSimList(sim, zipfile = tmpZip, filename = "sim.qs")
pths <- getPaths(mySim)
out <- unzipSimList(tmpZip, paths = pths)
```
2. filebackend = 1: On-the-fly renaming of file-backed rasters;
   (a) Save the sim object with a filename, e.g., file,
   (b) make a copy of all file-backed rasters to fileBackedDir,
   (c) update all the pointers to those files so that they are correct in the raster metadata

   \[
   \text{saveSimList(sim, file = "sim.qs", fileBackend = 1, fileBackedDir = "here")}
   \]

   \[
   \text{simNew <- loadSimList(file = "sim.qs")}
   \]

3. filebackend = 2: On-the-fly bringing to memory of all rasters
   All rasters are brought to memory, and then saved into sim.qs

   \[
   \text{saveSimList(sim, file = "sim.qs", fileBackend = 2)}
   \]

   \[
   \text{simNew <- loadSimList(file = "sim.qs")}
   \]

   If cache is used, it is likely that it should be trimmed before zipping, to include only cache
   elements that are relevant.

Value

\[
\text{saveSimList(): A saved .qs file in filename location.}
\]

\[
\text{zipSimList(): A saved .zip file in zipfile location.}
\]

See Also

\[
\text{loadSimList(), unzipSimList()}
\]

---

**scheduleConditionalEvent**

*Schedule a conditional simulation event*

Description

Adds a new event to the simulation’s conditional event queue, updating the simulation object by creating or appending to sim$._conditionalEvents. *This is very experimental. Use with caution.*

Usage

\[
\text{scheduleConditionalEvent(}
\]

\[
\text{sim,}
\]

\[
\text{condition,}
\]

\[
\text{moduleName,}
\]

\[
\text{eventType,}
\]

\[
\text{eventPriority = .normal()},
\]

\[
\text{minEventTime = start(sim)},
\]

\[
\text{maxEventTime = end(sim)}
\]

\[
\text{)}
\]
scheduleConditionalEvent

Arguments

sim A simList simulation object.
condition A string, call or expression that will be assessed for TRUE after each event in the regular event queue. It can access objects in the simList by using functions of sim, e.g., "sim$age > 1"
moduleName A character string specifying the module from which to call the event. If missing, it will use currentModule(sim)
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. As a best practice, it is recommended that decimal values are conceptually grouped by their integer values (e.g., 4.0, 4.25, 4.5 are conceptually similar). See priority().
minEventTime A numeric specifying the time before which the event should not occur, even if the condition is met. Defaults to start(sim)
maxEventTime A numeric specifying the time after which the event should not occur, even if the condition is met. Defaults to end(sim)

Value

Returns the modified simList object, i.e., sim$.conditionalEvents.

This conditional event queue will be assessed at every single event in the normal event queue. If there are no conditional events, then spades will proceed as normal. As conditional event conditions are found to be true, then it will trigger a call to scheduleEvent(...) with the current time passed to eventTime and it will remove the conditional event from the conditional queue. If the user would like the triggered conditional event to occur as the very next event, then a possible strategy would be to set eventPriority of the conditional event to very low or even negative to ensure it gets inserted at the top of the event queue.

Author(s)

Eliot McIntire

References


See Also

scheduleEvent(), conditionalEvents()

Examples

sim <- simInit(times = list(start = 0, end = 2))
condition <- "sim$age > 1" # provide as string
condition <- quote(sim$age > 1) # provide as a call
condition <- expression(sim$age > 1) # provide as an expression
sim <- scheduleConditionalEvent(sim, condition, "firemodule", "burn")
conditionalEvents(sim)
sim <- spades(sim) # no changes to sim$age, i.e., it is absent
events(sim) # nothing scheduled
sim$age <- 2 # change the value
sim <- spades(sim) # Run spades, the condition is now true, so event is
# scheduled at current time
events(sim) # now scheduled in the normal event queue

---

**scheduleEvent**

*Schedule a simulation event*

**Description**

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

**Usage**

```r
scheduleEvent(
  sim,  
  eventTime,  
  moduleName,  
  eventType,  
  eventPriority = .pkgEnv$.normalVal,  
  .skipChecks = FALSE
)
```

**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. If missing, it will use `currentModule(sim)`
- `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. As a best practice, it is recommended that decimal values are conceptual grouped by their integer values (e.g., 4.0, 4.25, 4.5 are conceptually similar). See `priority()`.
- `.skipChecks` Logical. If TRUE, then internal checks that arguments match expected types are skipped. Should only be used if speed is critical.

**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


See Also

priority(), scheduleConditionalEvent()

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)
**simFile**  
*Generate simulation file name*

**Description**
Assists with saving and retrieving simulations (e.g., with `saveSimList` and `loadSimList`).

**Usage**

```r
simFile(name, path, time = NULL, ext = "rds")
```

**Arguments**

- **name**: Object name (e.g., “mySimOut”)
- **path**: Directory location in where the file will be located (e.g., an `outputPath`).
- **time**: Optional simulation time to use as filename suffix. Default `NULL`.
- **ext**: The file extension to use (default "rds").

---

**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. The `simList` also stores all parameters, and other important simulation information, such as times, paths, modules, and module load order. See more details below.

**Usage**

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

```r
## S4 method for signature
```
## 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,
simInit

loadOrder,
notOlderThan = NULL

simInitDefaults()

Arguments

times A named list of numeric simulation start and end times (e.g., times = list(start = 0.0, end = 10.0, timeunit = "year")), with the final optional element, timeunit, overriding the default time unit used in the simulation which is the "smallest time unit" across all modules. See examples.

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 from 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. NOTE: Experimental feature now allows for multiple modulePaths to be specified in a character vector. The modules will be searched for sequentially in the first modulePath, then if it doesn’t find it, in the second etc.

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) and eventPriority (numeric). If eventPriority is not set, it defaults to .last(). If eventPriority is set to a low value, e.g., 0, 1, 2 and saveTime is start(sim), it should give "initial conditions". See outputs() and vignette("ii-modules") section about outputs.

loadOrder An optional character vector 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 uses the .useCache parameter and it is set to TRUE or ".inputObjects", then the .inputObjects will be cached. Setting
not OlderThan = Sys.time() will cause the cached versions of .inputObjects to be refreshed, i.e., rerun.

Details

Calling this simInit function does the following::

<table>
<thead>
<tr>
<th>What</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>fills simList slots</td>
<td>places the arguments times, params, modules, paths into equivalently named simList slots</td>
</tr>
<tr>
<td>sources all module files</td>
<td>places all function definitions in the simList, specifically, into a sub-environment of the main simList environment</td>
</tr>
<tr>
<td>copies objects</td>
<td>from disk into the simList</td>
</tr>
<tr>
<td>loads objects</td>
<td>Objects can be loaded into the simList at any time during a simulation</td>
</tr>
<tr>
<td>schedule object loading/copying</td>
<td>Objects can be saved to disk at any arbitrary time during the simulation. If specified here, this saves from all modules (see events())</td>
</tr>
<tr>
<td>schedule object saving</td>
<td>via the inputs and outputs identified in their metadata. This gives the order of the .inputObjects and .init events.</td>
</tr>
<tr>
<td>assesses module dependencies</td>
<td>takes time units of modules and how they fit together</td>
</tr>
<tr>
<td>determines time unit</td>
<td>from every module in the module order as determined above</td>
</tr>
<tr>
<td>runs .inputObjects functions</td>
<td></td>
</tr>
</tbody>
</table>

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, .plotInterval = 2), caribouModule = list(N = 1000)).

The params list can contain a list (named .globals) of named objects e.g., .globals = list(climateURL = "https:\something.com") entry. Any and every module that has a parameter with that name (in this case climateURL) will be overridden with this value as passed.

params can be used to set the seed for a specific event in a module. This is done using the normal params argument, specifying .seed as a list where the elements are a numeric for the seed and the name is the event. Since parameters must be specific to a module, this creates a module and event specific seed e.g., params = list(moduleName = list(.seed = list(init = 123))) will set the init event of module named moduleName to 123. The RN stream will be reset to its state prior to the set.seed call after the event.

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 fast lists to manage the event queue.

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

Value

A simList simulation object, pre-initialized from values specified in the arguments supplied.
**Parsing and Checking Code**

The `simInit` function will attempt to find usage of `sim$xxx` or `sim[['xxx']]` on either side of the assignment (`<-`) operator. It will compare these to the module metadata, specifically `inputObjects` for cases where objects or "gotten" from the `simList` and `outputObjects` for cases where objects are assigned to the `simList`.

It will also attempt to find potential, common function name conflicts with things like `scale` and `stack` (both in `base` and `raster`), and `Plot` (in `quickPlot` and some modules).

*This code checking is young and may get false positives and false negatives, i.e., miss things.* It also takes computational time, which may be undesirable in operational code. To turn off checking (i.e., if there are too many false positives and negatives), set `options(spades.moduleCodeChecks = FALSE)`.

**Caching**

Using caching with SpaDES is vital when building re-usable and reproducible content. Please see the vignette dedicated to this topic.

**Note**

Since the objects in the `simList` are passed-by-reference, it is useful to create a copy of the initialized `simList` object prior to running the simulation (e.g., `mySimOut <- spades(Copy(mySim))`). This ensures you retain access to the original objects, which would otherwise be overwritten/modified during the simulation.

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`. All of `modules`, `paths`, `params`, and `objects` are needed for successful initialization.

**Author(s)**

Alex Chubaty and Eliot McIntire

**References**


**See Also**

`spades()`, `times()`, `params()`, `objs()`, `paths()`, `modules()`, `inputs()`, `outputs()`

**Examples**

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

# 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(
```
```r
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,
      eventPriority = c(0,10), # eventPriority 0 may give "initial" conditions
      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)
```

Call `simInit` and `spades` together
simInitAndSpades

Description
These functions are convenience wrappers that may allow for more efficient Caching. Passes all arguments to simInit, then passes the created simList to spades.

Usage

simInitAndSpades(
  times,
  params,
  modules,
  objects,
  paths,
  inputs,
  outputs,
  loadOrder,
  notOlderThan,
  debug,
  progress,
  cache,
  .plots,
  .plotInitialTime,
  .saveInitialTime,
  events,
  ...
)

Arguments

times A named list of numeric simulation start and end times (e.g., times = list(start = 0.0, end = 10.0, timeunit = "year")), with the final optional element, timeunit, overriding the default time unit used in the simulation which is the "smallest time unit" across all modules. See examples.

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 from 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. NOTE: Experimental feature now
allows for multiple modulePaths to be specified in a character vector. The modules will be searched for sequentially in the first modulePath, then if it doesn’t find it, in the second etc.

**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) and eventPriority (numeric). If eventPriority is not set, it defaults to .last(). If eventPriority is set to a low value, e.g., 0, 1, 2 and saveTime is start(sim), it should give "initial conditions". See outputs() and vignette("ii-modules") section about outputs.

**loadOrder**
An optional character vector 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 uses the .useCache parameter and it is set to TRUE or ".inputObjects", then the .inputObjects will be cached. Setting notOlderThan = Sys.time() will cause the cached versions of .inputObjects to be refreshed, i.e., rerun.

**debug**
Optional tools for invoking debugging. Supplying a list will invoke the more powerful logging package. See details. Default is to use the value in getOption("spades.debug").

**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.

**.plots**
Character. Sets the parameter of this name in all modules. See Plots() for possible values. The parameter is intended to slowly take over from .plotInitialTime as a mechanism to turn on or off plotting. For backwards compatibility, if .plotInitialTime is not set in this spades call, but this .plots is used, two things will happen: setting this without "screen" will turn off all plotting; setting this with "screen" will trigger plotting for any modules that use this parameter but will have no effect on other modules. To get plotting, therefore, it may be necessary to also set .plotInitialTime = start(sim).

**.plotInitialTime**
Numeric. Temporarily override the .plotInitialTime parameter for all modules. See Details.

**.saveInitialTime**
Numeric. Temporarily override the .plotInitialTime parameter for all modules. See Details.

**events**
A character vector or a named list of character vectors. If specified, the simulations will only do the events indicated here. If a named list, the names must
correspond to the modules and the character vectors can be specific events within each of the named modules. With the list form, all unspecified modules will run all their events, including internal spades modules, e.g., save, that get invoked with the outputs argument in simInit. See example.

Arguments passed to simInit and spades

Value

Same as spades() (a simList) or

See Also

simInit(), spades()

---

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), which can be converted to a sorted data.table with events(sim). See 'Event Lists' for more information.
- **current** The current event, as a data.table. See 'Event Lists' for more information.
- **completed** An environment consisting of completed events, with each object named a character representation of the order of events. This was converted from a previous version which was a list. This was changed because the list became slow as number of events increased. See 'Event Lists' for more information. It is kept as an environment 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 data.frame or data.table of files and metadata
outputs a data.frame or data.table of files and metadata
paths Named list of modulePath, inputPath, and outputPath paths. Partial matching is performed.
.xData 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 .xData slot: $, [[], ls, ls.str, objs. See examples.
.envir Deprecated. Please do not use any more.

Accessor Methods

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

- `simList-accessors-envir()` Simulation environment.
- `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 environment, by adding several slots that provide information about the metadata for a discrete event simulation. The environment slot, if accessed directly is .xData and this is where input and output objects from modules are placed. The simList_() class is similar, but it extends the list class. All other slots are the same. 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 when saving/loading, because saving/loading a list behaves more reliably than saving/loading an environment.
Author(s)
Alex Chubaty and Eliot McIntire

References

spades Run a spatial discrete event simulation

Description
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).

Usage
spades(
  sim,
  debug = getOption("spades.debug"),
  progress = NA,
  cache,
  .plotInitialTime = NULL,
  .saveInitialTime = NULL,
  notOlderThan = NULL,
  events = NULL,
  .plots = NULL,
  ...
)

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

## S4 method for signature 'ANY,ANY,ANY,logical'

---

# 135
spades(
  sim,
  debug = getOption("spades.debug"),
  progress = NA,
  cache,
  .plotInitialTime = NULL,
  .saveInitialTime = NULL,
  notOlderThan = NULL,
  events = NULL,
  .plots = NULL,
  ...
)

Arguments

sim A simList simulation object, generally produced by simInit.
debug Optional tools for invoking debugging. Supplying a list will invoke the more
powerful logging package. See details. Default is to use the value in getOption("spades.debug").
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 mod-
ules. See Details.
.saveInitialTime Numeric. Temporarily override the .plotInitialTime parameter for all mod-
ules. 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.
events A character vector or a named list of character vectors. If specified, the sim-
ulations will only do the events indicated here. If a named list, the names must
correspond to the modules and the character vectors can be specific events within
each of the named modules. With the list form, all unspecified modules will run all their events, including internal spades modules, e.g., save, that get in-
voked with the outputs argument in simInit. See example.
.plots Character. Sets the parameter of this name in all modules. See Plots() for pos-
sible values. The parameter is intended to slowly take over from .plotInitialTime
as a mechanism to turn on or off plotting. For backwards compatibility, if
.plotInitialTime is not set in this spades call, but this .plots is used, two
things will happen: setting this without "screen" will turn off all plotting; setting this with "screen" will trigger plotting for any modules that use this parameter but will have no effect on other modules. To get plotting, therefore, it may be necessary to also set .plotInitialTime = start(sim).

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

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.

Value

Invisibly returns the modified simList object.

Caching with SpaDES

There are numerous ways in which Caching can be used within SpaDES. Please see the vignette https://CRAN.R-project.org/package=SpaDES.core/vignettes/iii-cache.html for many examples. Briefly, functions, events, modules, entire spades calls or experiment calls (see https://github.com/PredictiveEcology/SpaDES.experiment) can be cached and mixtures of all of these will work. For functions, simply wrap the call with Cache, moving the original function name into the first argument of Cache. For events or modules, set the module parameters, .useCache, e.g., simInit(..., parameters = list(myModule = list(.useCache = "init"))). This can be set to an event name, which will cache that event, or a logical (e.g., ), which will cache every event in that module. Event and module caching makes most sense when the event or module only runs once, such as an initialization or data preparation event/module. Caching an entire simulation is actually just a function call to simInitAndSpades, for example. So, simply writing Cache(simInitAndSpades, modules = ...) will effectively cache a whole simulation. Finally for experiments, it is just like a function call: Cache(simInitandExperiment, ...). The final way Caching can be done is in experiment or spades, by setting the cache argument.

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 the vignette on caching for examples.
debug

The most powerful way to use debug is to invoke the logging R package. To invoke this, debug must be a list with up to 3 named elements: console, file, and debug. Each of these list elements must be a list (including empty list() for defaults) with the sub-list elements here:

<table>
<thead>
<tr>
<th>console</th>
<th>level</th>
<th>The level, see below, of information shown</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>append</td>
<td>Logical. If TRUE, the default, then log entries are appended to file, if it exists</td>
</tr>
<tr>
<td>file</td>
<td>level</td>
<td>A filename. Defaults to log.txt</td>
</tr>
<tr>
<td>debug</td>
<td>level</td>
<td>The level, see below, of information shown</td>
</tr>
<tr>
<td>debug</td>
<td></td>
<td>See possible values below</td>
</tr>
</tbody>
</table>

level can be a number from 0 to 100 or a character string matching one of the values in logging::loglevels. These are hierarchical levels of information passed to the console. Set a lower number for more information and a higher number for less information. Errors in code will be shown if level is set to "ERROR" or 40 or above; warnings in code will be shown if level is set to "WARN" or 30 or above; normal messages in code will be shown if level is set to "INFO" or 20 or above. For consistency with base R messaging, if default level is used, then normal messaging via message will be shown; this means that suppressMessages will work to suppress messaging only when level is set to "INFO" or 20. Some functions in the SpaDES ecosystem may have information at the lower levels, but currently, there are few to none.

ddebug is specified as a non-list argument to spades or as list(debug = ...), then it can be a logical, a quoted call, a character vector or a numeric scalar (currently 1 or 2) or a list of any of these to get multiple outputs. This will be run at the start of every event. The following options for debug are available. Each of these can also be in a list to get multiple outputs:

TRUE
a function name (as character string)
moduleName (as character string)
eventName (as character string)
c(<moduleName>, <eventName>)
Any other R expression expressed as a character string or quoted call
A numeric scalar, currently 1 or 2 (maybe others)

current(sim) will be printed at the start of each event.
If a function, then it will be run on the simList, e.g., "time" will run time(sim).
All calls to that module will be entered interactively.
All calls that have that event name (in any module) will be shown.
Only the event in that specified module will be entered interactively.
Will be evaluated with access to the simList as 'sim'.
This will print out alternative forms of event information.

If not specified in the function call, the package option spades.debug is used.

If options("spades.browserOnError" = TRUE) (experimental still) if there is an error, it will attempt to open a browser in the event where the error occurred. You can edit, and then press c to continue or Q to quit, plus all other normal interactive browser tools. c will trigger a reparse and events will continue as scheduled, starting with the one just edited. There may be some unexpected consequences if the simList objects had already been changed before the error occurred.

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

See Also
SpaDES.core-package(), simInit(), and the caching vignette (very important for reproducibility): https://CRAN.R-project.org/package=SpaDES.core/vignettes/iii-cache.html which uses reproducible::Cache().

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)

# set default debug printing for the current session
# setOption(spades.debug = TRUE)

# Different debug options (overrides the package option 'spades.debug')
spades(mySim, debug = TRUE) # Fastest
spades(mySim, debug = "simList")
spades(mySim, debug = "print(table(sim$landscape$Fires[]))")
# To get a combination -- use list(debug = list(......))
spades(mySim, debug = list(debug = list(1, quote(as.data.frame(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("sample Modules", package = "SpaDES.core"))
  )
  spades(mySim, .plotInitialTime = NA)
}
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
Description

These provide top-level, powerful settings for a comprehensive SpaDES workflow. To see defaults, run `spadesOptions()`. See Details below.

Usage

```
spadesOptions()
```

Details

Below are options that can be set with `options("spades.xxx" = newValue)`, where `xxx` is one of the values below, and `newValue` is a new value to give the option. Sometimes these options can be placed in the user’s `.Rprofile` file so they persist between sessions.

The following options are likely of interest to most users

*OPTION* | DEFAULT VALUE |
---|---|
`spades.browserOnError` | `FALSE` |
`reproducible.cachePath` | `getOption('reproducible.cachePath')` |
`spades.debug` | `TRUE` |
`spades.DTthreads` | `1L` |
`spades.futureEvents` | `FALSE` |
`spades.inputPath` | Default is a temporary directory (typically `/tmp/RtmpXXX/SpaDES/inputs`) |
`spades.lowMemory` | `FALSE` |
`spades.memoryUseInterval` | `FALSE` |
`spades.messagingNumCharsModule` | `21` |
`spades.moduleCodeChecks` | `list(suppressParamUnused = FALSE, suppressUndefined = TRUE, suppressPartialMatchArgs = FALSE, ... checks be run during 'simInit'. These are passed to codetools::checkUsage. Default is given by the function, plus these:

- `moduleDocument` | `NULL` |
- `spades.modulePath` | `file.path(tempdir(), "SpaDES")` |
- `spades.moduleRepo` | "PredictiveEcology/SpaDES" |
- `spades.nCompleted` | `1000L` |
- `spades.outputPath` | `file.path(tempdir(), "SpaDES")` |
- `spades.recoveryMode` | `FALSE` |
- `spades.scratchPath` | `file.path(tempdir(), "SpaDES")` |
- `spades.switchPkgNamespaces` | `FALSE` to keep computational overhead down. Should the search path be modified to ensure a computational environment? There is no name conflicts among package objects, but it is much slower, especially if the events are themselves fast. |
- `spades.testMemoryLeaks` | `TRUE` |
- `spades.tolerance` | `.Machine$double.eps^0.5` |
'spades.useragent'  "https://github.com/PredictiveEcology/SpaDES".

suppliedElsewhere

Assess whether an object has or will be supplied from elsewhere

Description

When loading objects into a simList, especially during the simInit call, and inside the .inputObjects functions of modules, it is often useful to know if an object in question will or has been by the user via the inputs or objects arguments, or by another module's .inputObjects while preparing its expected inputs (via expectsInputs in metadata), or if it will be supplied by another module during its "init" event. In all these cases, it may not be necessary for a given module to load any default value for its expectsInputs. This function can be used as a check to determine whether the module needs to proceed in getting and assigning its default value.

Usage

suppliedElsewhere(
  object,
  sim,
  where = c("sim", "user", "initEvent"),
  returnWhere = FALSE
)

Arguments

object Character vector
sim A simList in which to evaluated whether the object is supplied elsewhere
where Character vector with one to three of "sim", "user", or "initEvent". Default is all three. Partial matching is used. See details.
returnWhere Logical, default FALSE, whether the vector of length 3 logical should be returned, or a logical of length one

Details

where indicates which of three places to search, either "sim" i.e., the simList, which would be equivalent to is.null(sim\$objName), or "user" which would be supplied by the user in the simInit function call via outputs or inputs (equivalent to (!('defaultColor' %in% sim$.userSuppliedObjNames))), or "initEvent", which would test whether a module that gets loaded before the present one will create it as part of its outputs (i.e., as indicated by createsOutputs in that module's metadata). There is a caveat to this test, however; if that other event also has the object as an expectsInput, then it would fail this test, as it also needs it as an input. This final one ("initEvent") does not explicitly test that the object will be created in the "init" event, only that it is in the outputs of that module, and that it is a module that is loaded prior to this one.
Examples

```r
mySim <- simInit()
suppliedElsewhere("test", mySim) # FALSE

# supplied in the simList
mySim$test <- 1
suppliedElsewhere("test", mySim) # TRUE
test <- 1

# supplied from user at simInit time -- note, this object would eventually get into the simList
# but the user supplied values come *after* the module's .inputObjects, so
# a basic is.null(sim$test) would return TRUE even though the user supplied test
mySim <- simInit(objects = list("test" = test))
suppliedElsewhere("test", mySim) # TRUE

## Not run:
# Example with prepInputs
# Put chunks like this in your .inputObjects
if (!suppliedElsewhere("test", mySim))
  sim$test <- Cache(prepInputs, "raster.tif", "downloadedArchive.zip",
                    destinationPath = dataPath(sim), studyArea = sim$studyArea, rasterToMatch = sim$otherRasterTemplate, overwrite = TRUE)
```

## End(Not run)

---

### times

**Time usage in SpaDES**

#### Description

Functions for the `times` slot of a `simList` object and its elements. To maintain modularity, the behaviour 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

```r
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, ...)

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

end(x) <- value

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

start(x, ...

## S3 method for class 'simList'
start(x, unit = NULL, ...)

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)

timeunits(x) <- value

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

elapsedTime(x, ...)

## S3 method for class 'simList'
elapsedTime(x, byEvent = TRUE, units = "auto", ...)
Arguments

- \( x \) A simList
- 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.
- byEvent Logical. If TRUE, the elapsed time will be by module and event; FALSE will report only by module. Default is TRUE.
- units character string. Units in which the results are desired. Can be abbreviated.

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(dday(14)) 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 behaviour 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.
```
**updateList**

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

**Description**

Being deprecated. Use `utils::modifyList()` (which can not handle NULL) or `Require::modifyList2()` for case with >2 lists and can handle NULL lists.

**Usage**

```r
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
**use_gha**

Use GitHub actions for automated module checking

**Description**

See corresponding vignette for more information.

**Usage**

```r
use_gha(name, path)
```

**Arguments**

- `name`: module name
- `path`: module path

---

**writeEventInfo**

Write simulation event info to file

**Description**

Useful for debugging.

**Usage**

```r
writeEventInfo(sim, file = "events.txt", append = FALSE)
```

**Arguments**

- `sim`: A `simList` object.
- `file`: Character specifying the filename (default "events.txt").
- `append`: Logical indicating whether to append to the file (default FALSE).

**Value**

Nothing returned. Invoked for its side-effect of writing to file.

**Author(s)**

Alex Chubaty
writeRNGInfo  Write RNG state info to file

Description
Useful for debugging and ensuring reproducibility.

Usage
writeRNGInfo(file = "seed.txt", append = FALSE)

Arguments
file Character specifying the filename (default "seed.txt").
append Logical indicating whether to append to the file (default FALSE).

Value
Nothing returned. Invoked for its side-effect of writing to file.

Author(s)
Alex Chubaty

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 \texttt{zip()}: e.g., add "-q" using \texttt{flags="-q -r9X"} (the default flags are "-r9X").

Author(s)

Eliot McIntire and Alex Chubaty
Index

* datasets
  .quickCheck, 20
  inSeconds, 65
  moduleDefaults, 73

* functions to access elements of a 'simList' object
  doEvent.checkpoint, 45
  envir, 51
  events, 53
  globals, 59
  inputs, 62
  modules, 79
  objs, 90
  packages, 96
  params, 98
  paths, 100
  progressInterval, 108
  times, 143

* module creation helpers
  newModule, 82
  newModuleCode, 83
  newModuleDocumentation, 84
  newModuleTests, 85
  .Random.seed(), 46
  .addChangedAttr, simList-method, 13
  .addDepends, 46, 52, 55, 59, 64, 80, 91, 96, 99, 102, 109, 146
  .addTagsToOutput, simList-method, 14
  .cacheMessage, simList-method, 15
  .checkCacheRepo, list-method, 15
  .checkpointSave (doEvent.checkpoint), 45
  .fileExtensions, 16
  .findSimList, 17
  .first (priority), 108
  .highest (priority), 108
  .last (priority), 108
  .lowest (priority), 108
  .moduleDeps(), 133, 140
  .normal (priority), 108
  .parseElems, simList-method, 18
  .preDigestByClass, simList-method, 18
  .prepareOutput, simList-method, 19
  .quickCheck, 20
  .robustDigest, simList-method, 20
  .saveFileExtensions (.fileExtensions), 16
  .simDeps(), 140
  .spadesTimes (inSeconds), 65
  .tagsByClass, simList-method, 21

  all.equal.simList, 22
  anyPlotting, 22
  append_attr, 23
  append_attr, list, list-method
    (append_attr), 23
  base::all.equal(), 22
  bindrows, 24

  Cache, 21
  Cache (.robustDigest, simList-method), 20
  cachePath (paths), 100
  cachePath, simList-method (paths), 100
  cachePath<- (paths), 100
  cachePath<-, simList-method (paths), 100
  checkModule, 24
  checkModule, character, character-method
    (checkModule), 24
  checkModule, character, missing-method
    (checkModule), 24
  checkModuleLocal, 25
  checkModuleLocal, character, ANY, ANY-method
    (checkModuleLocal), 25
  checkModuleLocal, character, character-method
    (checkModuleLocal), 25
  checkObject, 26
  checkObject, missing, ANY, missing, ANY-method
    (checkObject), 26
checkObject, simList, character, missing, character-method
(checkObject), 26
c(checkObject), 26
c(checkObject), 26
c(checkObject), 26
c(checkObject), 26

checkParams, 27

checkpointFile (doEvent.checkpoint), 45

checkpointInterval
(doEvent.checkpoint), 45

checkpointInterval<-
(doEvent.checkpoint), 45

checkpointInterval<-
, simList-method
(doEvent.checkpoint), 45

checkpointLoad (doEvent.checkpoint), 45

checkTimeunit (inSeconds), 65

checkTimeunit, character, environment-method
(inSeconds), 65

checkTimeunit, character, missing-method
(inSeconds), 65

citation (inputObjects), 60

citation, character-method
(inputObjects), 60

citation, simList-method (inputObjects), 60

classFilter, 28
classFilter, character, character, character, environment-method
(classFilter), 28
classFilter, character, character, character, missing-method
(classFilter), 28
classFilter, character, character, missing, environment-method
(simList, list-method
(classFilter), 28
classFilter, character, character, missing, missing-method
(classFilter), 28

classFilter, character, character, missing, character-method
(classFilter), 28

classFilter, character, character, missing, missing-method
(classFilter), 28

classFilter, character, character, missing, missing-method
(classFilter), 28

classFilter, character, character, missing, missing-method
(classFilter), 28

createOutput, character, character, character-method
(createOutput), 36

current (events), 53

current, simList, character-method
(events), 53

current, simList, missing-method
(events), 53

current<-, (events), 53

current<-, simList-method (events), 53

cov::report(), 73

cover::report(), 73

data.frame(), 133

data.table(), 55, 133, 134

dataPath (paths), 100

dataPath, simList-method (paths), 100

dday (dhour), 43

defineEvent(), 39

defineEvent(), 39

defineModule(), 38

defineModule(), 37, 77

defineModule(), 38

defineParameter(), 39

defineParameter(), 39

defineParameter(), 39

defineParameter(), 39
outputObjects, simList-method
(inputObjects), 60
outputPath (paths), 100
outputPath, simList-method (paths), 100
outputPath<- (paths), 100
outputPath<-, simList-method (paths), 100
outputs, 93
outputs[, 126, 128, 132]
outputs<- (outputs), 93
outputs<-, simList-method (outputs), 93

P (params), 98
P(), 41
P<- (params), 98
packageDescription, 62
packages, 46, 52, 55, 59, 64, 80, 91, 96, 99,
102, 109, 146
packages, ANY-method (packages), 96
paramCheckOtherMods, 97
parameters (params), 98
parameters, simList-method (params), 98
params, 46, 52, 55, 59, 64, 80, 91, 96, 99, 102,
109, 146
params(), 41, 128
params<- (params), 98
params<-, simList-method (params), 98
paths, 46, 52, 55, 59, 64, 80, 91, 96, 99, 100,
109, 146
paths(), 128
paths, simList-method (paths), 100
paths<- (paths), 100
paths<-, simList-method (paths), 100
person(), 39
Plot, simList-method, 103
Plots, 105
Plots(), 22, 132, 136
POM (experiment), 56
prepInputs(), 48
priority, 108
priority(), 121–123
progressInterval, 46, 52, 55, 59, 64, 80, 91,
96, 99, 102, 108, 146
progressInterval, simList-method
(progressInterval), 108
progressInterval<- (progressInterval), 108
progressInterval<-, simList-method
(progressInterval), 108
progressType (progressInterval), 108
progressType, simList-method
(progressInterval), 108
progressType<- (progressInterval), 108
progressType<-, simList-method
(progressInterval), 108
quickPlot::.parseElems, 18
raster(), 112
rasterCreate, 110
rasterPath (paths), 100
rasterPath, simList-method (paths), 100
rasterPath<- (paths), 100
rasterPath<-, simList-method (paths), 100
rasterToMemory, 111
rasterToMemory, ANY-method
(rasterToMemory), 111
rasterToMemory, list-method
(rasterToMemory), 111
rasterToMemory, Raster-method
(rasterToMemory), 111
rasterToMemory, simList-method
(rasterToMemory), 111
remoteFileSize, 112
reproducible::.addChangedAttr, 14
reproducible::.addTagsToOutput, 14
reproducible::.cacheMessage, 15
reproducible::.checkCacheRepo, 15, 16
reproducible::.preDigestByClass, 19
reproducible::.prepareOutput, 19
reproducible::.robustDigest(), 21
reproducible::.tagsByClass, 21
reproducible::.Cache(), 139
reproducible::.Checksums(), 28
reproducible::.Copy(), 34, 35
reproducible::.makeMemoisable(), 69
reproducible::.objSize(), 91
reproducible::.preProcess(), 48
reqdPkgs (inputObjects), 60
reqdPkgs, missing-method (inputObjects),
60
reqdPkgs, simList-method (inputObjects),
60
require(), 67
Require::modifyList2(), 146
Require::Require(), 39, 67
INDEX

restartR, 112
restartSpades, 114
rndstr, 115
rndstr,missing,missing,logical-method (rndstr), 115
rndstr,missing,missing,missing-method (rndstr), 115
rndstr,missing,numeric,logical-method (rndstr), 115
rndstr,missing,numeric,missing-method (rndstr), 115
rndstr,numeric,missing,logical-method (rndstr), 115
rndstr,numeric,missing,missing-method (rndstr), 115
rndstr,numeric,numeric,logical-method (rndstr), 115
rndstr,numeric,numeric,missing-method (rndstr), 115

saveFiles, 116
saveRDS(), 94
saveSim(saveSimList), 118
saveSimList, 118
saveSimList(), 68, 69, 119, 120
scheduleConditionalEvent, 120
scheduleConditionalEvent(), 123
scheduleEvent, 122
scheduleEvent(), 37, 121
scratchPath (paths), 100
scratchPath, simList-method (paths), 100
scratchPath<-(paths), 100
scratchPath<-,simList-method (paths), 100
setPaths(), 35
setProgressBar (newProgressBar), 86
show, simList-method, 123
showCache, simList-method
  (clearCache, simList-method), 30
simFile, 124
simInit, 124
simInit(), 37, 39, 108, 133, 139, 141
simInit,ANY,ANY,ANY,ANY,ANY,ANY,ANY-method (simInit), 124
simInit,ANY,ANY,ANY,character,ANY,ANY,ANY-method (simInit), 124
simInit,ANY,ANY,character,ANY,ANY,ANY,ANY-method (simInit), 124
simInit,ANY,ANY,character,ANY,ANY,ANY,ANY-method (simInit), 124
simInitAndExperiment (experiment), 56
simInitAndSpades, 130
simInitDefaults (simInit), 124
simList (simList-class), 133
simList(), 26, 140
simList-accessors-envir (envir), 51
simList-accessors-envir(), 134
simList-accessors-events (events), 53
simList-accessors-events(), 134
simList-accessors-inout (inputs), 62
simList-accessors-inout(), 134
simList-accessors-metadata (inputObjects), 60
simList-accessors-modules (modules), 79
simList-accessors-modules(), 134
simList-accessors-objects (objs), 90
simList-accessors-objects(), 134
simList-accessors-packages (packages), 96
simList-accessors-params (params), 98
simList-accessors-params(), 134
simList-accessors-paths (paths), 100
simList-accessors-paths(), 134
simList-accessors-times (times), 143
simList-accessors-times(), 134
simList-class, 133
simList-class (simList-class), 133
simList(), 134
simList-class (simList-class), 133
spades, 135
spades(), 102, 108, 128, 133
spades, ANY, ANY, ANY, logical-method (spades), 135
spades, simList, ANY, ANY, ANY, missing-method (spades), 135
SpaDES.core (SpaDES.core-package), 5
SpaDES.core-package, 5
spadesClasses, 140
spadesOptions, 141
spadesTimes (inSeconds), 65
start (times), 143
start<-(times), 143
start<-, simList-method (times), 143
suppliedElsewhere, 142
terraPath, simList-method (paths), 100
terraPath<-(paths), 100
terraPath<-, simList-method (paths), 100
time.simList (times), 143
time<-(times), 143
time<-, simList-method (times), 143
times, 46, 52, 55, 59, 64, 80, 91, 96, 99, 102, 109, 143
times(), 128
times, simList-method (times), 143
times<-(times), 143
times<-, simList-method (times), 143
timeunit (times), 143
timeunit, simList-method (times), 143
timeunit<-(times), 143
timeunit<-, simList-method (times), 143
timeunits (times), 143
timeunits, simList-method (times), 143

unmakeMemoisable.simList_
   (makeMemoisable.simList), 69
unzipSimList (loadSimList), 68
unzipSimList (), 69, 119, 120
updateList, 146
use_gha, 147
utils::citation (), 61
utils::modifyList (), 146

writeEventInfo, 147
writeRNGInfo, 148

zip(), 149
zipModule, 148
zipModule(), 50, 51, 58
zipModule, character, character, character-method
   (zipModule), 148
zipModule, character, character, missing-method
   (zipModule), 148
zipModule, character, missing, character-method
   (zipModule), 148
zipModule, character, missing, missing-method
   (zipModule), 148
zipSimList, 68
zipSimList (saveSimList), 118
zipSimList (), 69, 119, 120