Package 'Simile'

December 11, 2024

Type Package

Title Interact with Simile Models

Version 1.3.4

Depends tcltk

Date 2024-12-10

Description Allows a Simile model saved as a compiled binary to be loaded, parameterized, executed and interrogated. This version works with Simile v6 on.

License Unlimited

LazyLoad yes

RoxygenNote 7.3.2

NeedsCompilation no

Author Simulistics Ltd [cph], Jasper Taylor [aut, cre]

Maintainer Jasper Taylor <support@simulistics.com>

Repository CRAN

Date/Publication 2024-12-11 18:00:02 UTC

Contents

Simile-package 2
consult.parameter.metafile 4
create.model 5
create.param.array 6
execute.model 6
get.model.property 8
get.model.time 9
get.value.array 10
get.value.list 11
is.dummy 12
list.objects 12

load.model 13
reset.model 14
set.model.parameter 15
set.model.step 15
tcl.paired.to.array 16
use.simile.at 17

Index **18**

`Simile-package` *Interface to executable Simile models*

Description

This package loads, parameterizes, executes and interrogates executable models saved by Simile.

Details

Package:	Simile
Type:	Package
Version:	1.3.4
Date:	2024-12-10
License:	Unrestricted
LazyLoad:	yes

The package loads some Simile libraries into R's tcltk extension, so you need Simile installed to use it. Initialize the package by calling use.simile.at() to tell it where to find Simile in the file system. Tell it where to find the saved executable model with load.model() which returns a handle to the loaded model. This can be used to interrogate static model info via list.objects() and get.model.property(), and to create executable instances with create.model(), which returns their handles.

An executable instance can be parameterized either by loading a Simile parameter metafile with consult.parameter.metafile(), or directly from R arrays with create.param.array() and set.model.parameter(). The functions set.model.step(), reset.model() and execute.model() control execution. get.model.time() returns the time at which execution finished. Use get.value.array() to get the values from any model component as an R array, or get.value.list() to get them as an R list.

This version works with Simile versions 6.0 on.

Author(s)

Simulistics Ltd

References

Simile modelling environment: http://simulistics.com

Examples

```
require("Simile")
exec.extn <- as.character(tcl("info","sharedlibextension"))
if (interactive()) {
  path.to.installation <- tcl("tk_chooseDirectory", "-title",
          "Folder where Simile is installed:")

  path.to.spiro <- tcl("tk_getOpenFile",
       "-title", "Compiled binary for Spiro example:",
                       "-initialfile", paste("spiro",exec.extn,sep=""))
} else {
  path.to.installation <- "dummy/path"
  path.to.spiro <- "dummy.dll"
}

use.simile.at(path.to.installation)
mHandle <- load.model(path.to.spiro)
objs <- list.objects(mHandle)

for (obj in objs) {
  print(c(obj, get.model.property(mHandle, obj, "Class")))
}

iHandle <- create.model(mHandle)
# model step is 0.1 by default but spiro only needs 1.0
set.model.step(iHandle, 1, 1)

# initialize the model, including default slider values
reset.model(iHandle, -2)

xs <- list(get.value.list(iHandle, "/runs/x"))
ys <- list(get.value.list(iHandle, "/runs/y"))
for (count in 1:1739) {
  execute.model(iHandle, count)
  xs[[count+1]] <- get.value.list(iHandle, "/runs/x")
  ys[[count+1]] <- get.value.list(iHandle, "/runs/y")
}
xs <- mapply(c,xs)
ys <- mapply(c,ys)
print("View default pattern -- now try to plot dancer")
plot(xs, ys, type="l")

# now we are going to parameterize it using a state file for the slider helper
# -- to make this work we load the mime library, which is needed only because
# the dancer.spf in Simile v6 is an older v4.x format file. No need to load
# if testing, and will always be available if live because required for Simile.

if (!is.dummy(iHandle)) {
   tcl("package","require","mime")
}
pFile  <- tcl("file", "join", path.to.installation, "Examples", "dancer.spf")
consult.parameter.metafile(iHandle, pFile)
```

```
# also the file from the v6 distribution has no value for "Wheel outside?"
# so we set this boolean directly
pHandle <- create.param.array(iHandle, "/start/Wheel outside?")
set.model.parameter(pHandle, FALSE)

# apply reset at level 0 to propagate input values
reset.model(iHandle, 0)

xs <- list(get.value.list(iHandle, "/runs/x"))
ys <- list(get.value.list(iHandle, "/runs/y"))
for (count in 1:419) {
  execute.model(iHandle, count)
  xs[[count+1]] <- get.value.list(iHandle, "/runs/x")
  ys[[count+1]] <- get.value.list(iHandle, "/runs/y")
}
xs <- mapply(c,xs)
ys <- mapply(c,ys)
plot(xs, ys, type="l")

print("OK, but that's not how it looks on Simile is it? Try this...")
xs <- aperm(xs, c(2,1))
ys <- aperm(ys, c(2,1))
plot(xs, ys, type="l")
```

`consult.parameter.metafile` *Set a model's parameters from a file saved by Simile*

Description

Parameter metafiles (.spf) saved from within Simile can contain information about a model's parameter values, either locally or as references to other files. They may refer to the whole model or a submodel.

Usage

```
consult.parameter.metafile(instance.handle,param.file,target.submodel="")
```

Arguments

`instance.handle`	The handle returned by create.model() identifying the model instance.
`param.file`	Location of the parameter metafile.
`target.submodel`	Pathname of the submodel into which the parameters are to be loaded. Default is the top level.

Value

None

Author(s)

Jasper Taylor

Examples

```
consult.parameter.metafile("dummy_ih", "../data/base_vals.spf")
```

---

create.model *Create an executable model instance*

Description

A script can create many independent instances of the same model description, each with its own execution state.

Usage

```
create.model(model.handle)
```

Arguments

model.handle	The handle returned by load.model() identifying the model type.

Value

A handle to a new instance of the model, which will have its own component values, parameter values and execution settings

Author(s)

Jasper Taylor

Examples

```
instance.handle <- create.model("dummy_mh")
```

create.param.array *Allocates memory for interactively loading model parameters*

Description

If a script is to provide values directly for a Simile model parameter, this command must be called first to set up a location for the parameters to be held

Usage

```
create.param.array(instance.handle,param.name)
```

Arguments

instance.handle	The handle returned by create.model() identifying the model instance.
param.name	Caption path to the model component whose value is to be specified

Value

A handle to the location that has been created.

Author(s)

Jasper Taylor

Examples

```
param.handle <- create.param.array("dummy_ih", "/submodel1/trees/larch")
```

execute.model *Execute a Simile model to a given time point*

Description

The model will be executed using the time steps specified by earlier calling set.model.step until either the finish.time is reached or an exception occurs.

Usage

```
execute.model(instance.handle, finish.time, integration.method,
start.time, error.limit, pause.on.event)
```

Arguments

`instance.handle`
: The handle returned by create.model(), identifying an executable model instance.

`finish.time`
: Time returned by functions in the model on the last execution step, when incrementing the time takes it to or beyond this value.

`integration.method`
: One of "Euler" or "Runge-Kutta", the latter being 4th-order. Default is "Euler".

`start.time`
: Time returned by functions in the model on the first execution step. On each subsequent step this will be incremented by the value specified by earlier calling set.model.step. Default is NA, which starts the model at the time to which it was previously reset or executed.

`error.limit`
: Maximum integration error allowed by adaptive step size variation, as a fraction of the allowed range of each model compartment value. The integration error is estimated by comparing each compartment's rate of change with that predicted on the previous time step (equal to its previous rate of change if integration method is Euler). If error limit is exceeded, the time step is temporarily shortened and the model re-executed from the end of the last time step. Default is 0, in which case no adaptive step size variation is done.

`pause.on.event`
: Controls whether execution should be paused in the case of potentially interesting model occurrences. These are limit events (discrete events triggered by the value of an expression reaching a preset maximum or minimum), and compartment over/underruns (the level of a compartment exceeding or underrunning its specified maximum or minimum). Default is FALSE.

Value

Result code: value of system error (-ve), user-defined interruption (+ve) or 0 if model runs to finish.time

Author(s)

Jasper Taylor

Examples

```
for (count in 0:1738) {
  execute.model("dummy_ih", count+1, "Euler", count, 0, FALSE)
}
```

`get.model.property` *Get properties of model components*

Description

This function can return any of several pieces of static information about a component in a loaded Simile model

Usage

```
get.model.property(model.handle,caption.path,requested.property)
```

Arguments

`model.handle`	Model handle created when calling load.model()
`caption.path`	Path made from submodel and component captions separated by forward slashes, like a member of the list returned by list.objects()
`requested.property`	One of the following strings: **Class** Returns the component class, i.e., which Simile symbol it is. **Type** Data type of component's value. **Eval** Source of component's value. **Dims** Returns list of the component value's array dimensions. **MinVal, MaxVal, Desc, Comment** Return other values or text associated with component in Simile.

Value

Form of returned value depends on the requested.property as described above.

- For Class it is one of SUBMODEL VARIABLE COMPARTMENT FLOW CONDITION CREATION REPRODUCTION IMMIGRATION LOSS ALARM EVENT SQUIRT STATE
- For Type: one of VALUELESS REAL INTEGER FLAG EXTERNAL or ENUM(n), where n is the index of a set of enumerated type values.
- For Eval: one of EXOGENOUS DERIVED TABLE INPUT SPLIT GHOST. Fixed parameters have TABLE and variable parameters INPUT for this property.
- For Dims it is a list containing integers or the special types RECORDS MEMBERS SEPARATE START_VM END_VM for components in submodels without a preset member count.
- For MinVal and MaxVal it is a number of the same type as the component.
- For Desc and Comment it is a character string.

Author(s)

Jasper Taylor

Examples

```
get.model.property("dummy_mh","/sector/output","Class")
# [1] "COMPARTMENT"
get.model.property("dummy_mh","/sector/output","Dims")
# [1] 2 5
```

`get.model.time` *Gets the time in the simulation.*

Description

The simulation time is initialized by the reset.model command, and incremented by the execute.model command. After execute.model it may not be the expected end time because the model may stop executing before that point for various reasons.

Usage

```
get.model.time(instance.handle)
```

Arguments

`instance.handle`
: The handle returned by create.model(), identifying an executable model instance.

Value

The current model time

Author(s)

Jasper Taylor

Examples

```
get.model.time("dummy_ih")
```

`get.value.array` *Get values from model components*

Description

Retrieves data from the model, as either a single quantity or an array of values with the appropriate dimensions

Usage

```
get.value.array(instance.handle, value.name, as.enum.types)
```

Arguments

`instance.handle`	The handle returned by create.model(), identifying an executable model instance.
`value.name`	Caption path to the model component whose values are required
`as.enum.types`	Logical: whether to supply values as character strings, using enumerated type names if applicable. Default is false: values are numerical.

Value

A value or array of values, from the model

Note

Will not work on components inside variable-membership submodels. For these you should use get.value.list instead, as R's list structure is more appropriate than the array structure when not all values exist.

Author(s)

Jasper Taylor

See Also

`get.value.list`

Examples

```
get.value.array("dummy_ih", "/sector/output", FALSE)
# [1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243 -0.2794155
# [7]  0.6569866  0.9893582  0.4121185 -0.5440211
```

`get.value.list` *Get values from model components*

Description

Retrieves data from the model, as either a single quantity or a possibly nested list of values

Usage

```
get.value.list(instance.handle, value.name, as.enum.types)
```

Arguments

`instance.handle`	The handle returned by create.model(), identifying an executable model instance.
`value.name`	Caption path to the model component whose values are required
`as.enum.types`	Logical: whether to supply indices and values as character strings, using enumerated type members if applicable. Default is false: indices are integers and values are numerical.

Value

A value or index-value list, from the model

Author(s)

Jasper Taylor

See Also

`get.value.array`

Examples

```
get.value.list("dummy_ih", "/runs/x", "FALSE")
# [[1]]
# [1] 40.76668
#
# [[2]]
# [1] 37.52907
#
# [[3]]
# [1] 33.82021
#
# [[4]]
# [1] 29.69413
```

`is.dummy` *Test if a path or handle is a 'dummy' whose only purpose is to allow tests to run.*

Description

When any of the functions in the package are called, if this returns 'true' on their first argument, no actual Simile interaction will be attempted but some sample output may be returned.

Usage

```
is.dummy(path.or.handle)
```

Arguments

`path.or.handle` The file path or handle which is being tested for dummy-ness.

Value

A boolean which is true in the case of a dummy argument.

Author(s)

Jasper Taylor

Examples

```
is.dummy("dummy.dll")
# [1] TRUE
is.dummy("~/.simile/Examples/spiro.so")
# [1] FALSE
```

`list.objects` *List Simile model components*

Description

Lists all the model components with values. Components are specified by path, i.e., a string made from their caption preceded by the caption of their parent submodel, its parent and so on, separated by forward-slashes like a directory path.

Usage

```
list.objects(model.handle)
```

Arguments

`model.handle` Model handle created when calling load.model()

Value

List of component paths as described above, in tree traversal order

Author(s)

Jasper Taylor

Examples

```
list.objects("dummy_mh")
# [1] "/sector"           "/sector/output"    "/sector/flow1"
# [4] "/sector/variation"
```

---

load.model *Load a Simile executable model*

Description

Loads a simile model saved as a compiled binary (a .dll, .so or .dylib file depending on platform)

Usage

```
load.model(path.to.binary)
```

Arguments

`path.to.binary` Absolute or relative path to file to be loaded

Value

Handle to be used for querying or instantiating model, as a Tcl object

Author(s)

Jasper Taylor

Examples

```
model.handle <- load.model("dummy.dll")
```

`reset.model` *resets the model to its initial state.*

Description

A Simile model instance must be reset before it is first run, whenever the parameters are changed, and in order to run it again with new values for random constants.

Usage

```
reset.model(instance.handle, depth, integration.method, starting.time)
```

Arguments

`instance.handle`	The handle returned by create.model(), identifying an executable model instance.
`depth`	Simile does a sort of 'lazy execution', with values being left untouched if they do not need to be recalculated for a given type of reset. This argument tells it what to reset, with each action including those that follow it: **-2** Recalculate constant atrithmetic expressions including numerals **-1** Recalculate values that depend on fixed parameters **0** Recalculate random constants and set state variables to their initial values **+ve** Recalculate rate variables from the current state values at this time step
`integration.method`	One of "Euler" or "Runge-Kutta", the latter being 4th-order. Default is "Euler".
`starting.time`	Value for model time after resetting. Used for indexing time series, time plots etc in the model. Default is 0.

Value

None

Author(s)

Jasper Taylor

Examples

```
reset.model("dummy_ih", -2)
```

set.model.parameter *Sets a model parameter with data in an array*

Description

A model parameter may be a single value or an array of values. This function allows it to be set.

Usage

```
set.model.parameter(param.handle,data, as.enum.types)
```

Arguments

param.handle	The handle returned by create.param.array() identifying the location to send the data.
data	The data to be used in the model, either a single value or an array with the appropriate dimensions
as.enum.types	Logical: whether the values are supplied using enumerated type names if applicable. Default is false: values are numerical.

Value

None

Author(s)

Jasper Taylor

Examples

```
set.model.parameter("dummy_ph", array(c(4,4,3,4,2,5,5,2,1,5), c(2,5)))
```

set.model.step *Sets the time step used to execute a model.*

Description

The time step is the amount by which the model time advances each time the state variables are updated. A model may have more than one time step.

Usage

```
set.model.step(instance.handle,step.index,step.size)
```

Arguments

instance.handle	The handle returned by create.model(), identifying an executable model instance.
step.index	The level of the time step to be set. If a model only has one time step, this is step 1. Higher levels are set to shorter steps.
step.size	The duration for the time step.

Value

None

Author(s)

Jasper Taylor

Examples

```
set.model.step("dummy_ih", 1, 0.1)
```

---

tcl.paired.to.array *Re-format model value array*

Description

Convert a set of values from a Simile model component from a nested list of alternating indices and values to an R array structure

Usage

```
tcl.paired.to.array(paired, dims, as.enum.types)
```

Arguments

paired	Nested list of alternating indices and values
dims	Dimensions of R array structure
as.enum.types	If TRUE, values will be converted to R character strings, otherwise they are numeric

Value

A numerical value or array of numerical values, from the Tcl value or array

Note

Note that the dimensions of the array returned will be in the opposite order from those supplied in the 'dims' argument. This is because Simile's convention is to list the outermost dimension first, whereas R's is to list the outermost dimension last.

Author(s)

Jasper Taylor

See Also

tcl.paired.to.list

Examples

```
Simile:::tcl.paired.to.array("1 {1 7 2 4} 2 {1 8 2 6} 3 {1 9 2 1}",
c(3,2), FALSE)
#     [,1] [,2] [,3]
#[1,]    7    8    9
#[2,]    4    6    1
```

---

use.simile.at *Initialize the Simile interface*

Description

This tells the package where to find an installed copy of Simile. It then loads the Tcl command implementations in the Simile installation to enable R's tcltk package to communicate with Simile's saved executable models. It should be called only once in a session.

Usage

```
use.simile.at(path.to.installation)
```

Arguments

`path.to.installation`
: Absolute or realtive path to top directory of Simile installation, e.g., "c:/Program files/Simile5.97" or "/usr/lib64/Simile-6.3"

Value

undefined.

Author(s)

Simulistics Ltd

Examples

```
use.simile.at("dummy/path")
```

Index

∗ **model**
Simile-package, 2

consult.parameter.metafile, 4
create.model, 5
create.param.array, 6

execute.model, 6

get.model.property, 8
get.model.time, 9
get.value.array, 10, *11*
get.value.list, *10*, 11

is.dummy, 12

list.objects, 12
load.model, 13

reset.model, 14

set.model.parameter, 15
set.model.step, 15
Simile *(*Simile-package*)*, 2
Simile-package, 2

tcl.paired.to.array, 16
tcl.paired.to.list, *17*

use.simile.at, 17