Package 'logolink'

May 8, 2026

Title An Interface for Running 'NetLogo' Simulations

Version 1.0.0

Description An interface for 'NetLogo' <https://www.netlogo.org> that enables programmatic setup and execution of simulations. Designed to facilitate integrating 'NetLogo' models into reproducible workflows by creating and running 'BehaviorSpace' experiments and retrieving their results.

License GPL (>= 3)

URL https://danielvartan.github.io/logolink/, https://github.com/danielvartan/logolink/

BugReports https://github.com/danielvartan/logolink/issues/

Depends R (>= 4.4)

Imports checkmate (>= 2.3.3), cli (>= 3.6.5), dplyr (>= 1.1.4), fs (>= 1.6.6), glue (>= 1.8.0), janitor (>= 2.2.1), magrittr (>= 2.0.4), purrr (>= 1.2.0), readr (>= 2.0.0), tidyr (>= 1.3.1), stringr (>= 1.6.0), xml2 (>= 1.4.0)

Suggests bslib (>= 0.9.0), colorspace (>= 2.1.2), covr (>= 3.6.5), curl (>= 7.0.0), ggplot2 (>= 4.0.1), ggimage (>= 0.3.4), ggtext (>= 0.1.2), here (>= 1.0.2), httr2 (>= 1.2.1), knitr (>= 1.50), lifecycle (>= 1.0.4), magick (>= 2.9.0), quarto (>= 1.5.1), ragg (>= 1.5.0), remotes (>= 2.5.0), rsvg (>= 2.7.0), scales (>= 1.4.0), spelling (>= 2.3.2), tibble (>= 3.3.0), testthat (>= 3.3.0)

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Daniel Vartanian [aut, cre, ccp, cph] (ORCID: <https://orcid.org/0000-0001-7782-759X>)

Maintainer Daniel Vartanian <danielvartan@proton.me>

Repository CRAN

Date/Publication 2026-01-08 15:10:14 UTC

Contents

`create_experiment` *Create NetLogo BehaviorSpace experiment*

Description

`create_experiment()` creates a NetLogo BehaviorSpace experiment XML file that can be used to run headless experiments with the `run_experiment()` function.

For complete guidance on setting up and running experiments in NetLogo, please refer to the BehaviorSpace Guide.

Usage

```
create_experiment(
  name = "",
  repetitions = 1,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = 1,
  pre_experiment = NULL,
  setup = "setup",
  go = "go",
  post_run = NULL,
  post_experiment = NULL,
  exit_condition = NULL,
  run_metrics_condition = NULL,
  metrics = "count turtles",
  constants = NULL,
```

```
  sub_experiments = NULL,
  file = tempfile(pattern = "experiment-", fileext = ".xml")
)
```

Arguments

`name`	(optional) A `character` string specifying the name of the experiment (default: `""`).
`repetitions`	(optional) An integer number specifying the number of times to run the experiment (default: `1`).
`sequential_run_order`	(optional) A `logical` flag indicating whether to run the experiments in sequential order (default: `TRUE`).
`run_metrics_every_step`	(optional) A `logical` flag indicating whether to record metrics at every step (default: `FALSE`).
`time_limit`	(optional) An integer number specifying the maximum number of steps to run for each repetition. Set to `0` or `NULL` to have no time limit (default: `1`).
`pre_experiment`	(optional) A `character` vector specifying the NetLogo command(s) to run before the experiment starts (default: `NULL`).
`setup`	(optional) A `character` vector specifying the NetLogo command(s) to set up the model (default: `'setup'`).
`go`	(optional) A `character` vector specifying the NetLogo command(s) to run the model (default: `'go'`).
`post_run`	(optional) A `character` vector specifying the NetLogo command(s) to run after each run (default: `NULL`).
`post_experiment`	(optional) A `character` vector specifying the NetLogo command(s) to run after the experiment ends (default: `NULL`).
`exit_condition`	(optional) A `character` vector specifying the NetLogo command that defines the exit condition for the experiment (default: `NULL`).
`run_metrics_condition`	(optional) A `character` vector specifying the NetLogo command that defines the condition to record metrics (default: `NULL`).
`metrics`	A `character` vector specifying the NetLogo commands to record as metrics (default: `'count turtles'`).
`constants`	(optional) A named `list` specifying the parameters for the experiment. Each element can be either a scalar, vector (for fixed/enumerated values), or a `list` with `first`, `step`, and `last` elements (for stepped/varying values). See the *Details* and *Examples* sections to learn more (default: `NULL`).
`sub_experiments`	(optional) A `list` where each element is also a `list` specifying the constants for a sub-experiment. Each sub-experiment uses the same structure as the `constants` argument. See the `constants` argument documentation for details on how to specify parameter values (default: `NULL`).
`file`	(optional) A `character` string specifying the path to save the created XML file (default: `tempfile(pattern = "experiment-", fileext = ".xml")`).

Details

Enclosing:

Since NetLogo only accepts double quotes for strings inside commands, we recommend always using single quotes when writing NetLogo commands in R to avoid mistakes. For example, to run the `[1 "a" true]` command, use `'[1 "a" true]'`, not `"[1 \"a\" true]"`.

Multiple Commands:

Some arguments accept multiple NetLogo commands to be run in sequence. In such cases, you can provide a `character` vector with each command as a separate element.

For example, to run two commands in sequence for the `go` argument, you can provide:

```
go = c("command-1", "command-2")
```

`constants` Argument:

The `constants` argument allows you to specify the parameters to vary in the experiment. It should be a named `list` where each name corresponds to a NetLogo global variable. The value for each name can be either:

- A scalar or vector (for enumerated values). For example, to set the variable `initial-number-of-turtles` to `10`, you would use `list("initial-number-of-turtles" = 10)`.
- A `list` with `first`, `step`, and `last` elements (for stepped values). For example, to vary the variable `initial-number-of-turtles` from `10` to `50` in steps of `10`, you would use `list("initial-number-of-turtles" = list(first = 10, step = 10, last = 50))`.

When passing values to constants, `character` strings should be passed as is, without adding quotes to them. For example, to set the variable `pathway` to `"SSP-585"`, you should use `list("pathway" = "SSP-585")`, not `list("pathway" = '"SSP-585"')`.

Value

A `character` string with the path to the created XML file.

See Also

Other BehaviorSpace functions: `inspect_experiment()`, `read_experiment()`, `run_experiment()`

Examples

```
# The examples below reproduce experiments from the NetLogo Models Library.
# Try exporting these experiments from NetLogo and compare the XML files.

## Examples from the Wolf Sheep Predation Model (Sample Models) ----

### BehaviorSpace Combinatorial

setup_file <- create_experiment(
  name = "BehaviorSpace Combinatorial",
  repetitions = 1,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = 1500,
```

```
  setup = 'setup',
  go = 'go',
  post_run = 'wait .5',
  run_metrics_condition = 'ticks mod 10 = 0',
  metrics = c(
    'count sheep',
    'count wolves',
    'count grass'
  ),
  constants = list(
    "model-version" = "sheep-wolves-grass",
    "wolf-reproduce" = c(3, 5, 10, 15),
    "wolf-gain-from-food" = c(10, 15, 30, 40)
  )
)

setup_file

setup_file |> inspect_experiment()

### Behaviorspace Run 3 Experiments

setup_file <- create_experiment(
  name = "Behaviorspace Run 3 Experiments",
  repetitions = 1,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = 1500,
  setup = c(
    'setup',
    paste0(
      'print (word "sheep-reproduce: " sheep-reproduce ", wolf-reproduce: ',
      '" wolf-reproduce)'
    ),
    paste0(
      'print (word "sheep-gain-from-food: " sheep-gain-from-food ", ',
      'wolf-gain-from-food: " wolf-gain-from-food)'
    )
  ),
  go = 'go',
  post_run = c(
    'print (word "sheep: " count sheep ", wolves: " count wolves)',
    'print ""',
    'wait 1'
  ),
  run_metrics_condition = 'ticks mod 10 = 0',
  metrics = c(
    'count sheep',
    'count wolves',
    'count grass'
  ),
  constants = list(
    "model-version" = "sheep-wolves-grass"
```

```
  ),
  sub_experiments = list(
    list(
      "sheep-reproduce" = 1,
      "sheep-gain-from-food" = 1,
      "wolf-reproduce" = 2,
      "wolf-gain-from-food" = 10
    ),
    list(
      "sheep-reproduce" = 6,
      "sheep-gain-from-food" = 8,
      "wolf-reproduce" = 5,
      "wolf-gain-from-food" = 20
    ),
    list(
      "sheep-reproduce" = 20,
      "sheep-gain-from-food" = 15,
      "wolf-reproduce" = 15,
      "wolf-gain-from-food" = 30
    )
  )
)

setup_file

setup_file |> inspect_experiment()

### BehaviorSpace Run 3 Variable Values Per Experiments

setup_file <- create_experiment(
  name = "BehaviorSpace Run 3 Variable Values Per Experiments",
  repetitions = 1,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = 1500,
  setup = c(
    'setup',
    paste0(
      'print (word "sheep-reproduce: " sheep-reproduce ", ',
      'wolf-reproduce: " wolf-reproduce)'
    ),
    paste0(
      'print (word "sheep-gain-from-food: " sheep-gain-from-food ", ',
      'wolf-gain-from-food: " wolf-gain-from-food)'
    )
  ),
  go = 'go',
  post_run = c(
    'print (word "sheep: " count sheep ", wolves: " count wolves)',
    'print ""',
    'wait 1'
  ),
  run_metrics_condition = 'ticks mod 10 = 0',
```

```
  metrics = c(
    'count sheep',
    'count wolves',
    'count grass'
  ),
  constants = list(
    "model-version" = "sheep-wolves-grass",
    "sheep-reproduce" = 4,
    "wolf-reproduce" = 2,
    "sheep-gain-from-food" = 4,
    "wolf-gain-from-food" = 20
  ),
  sub_experiments = list(
    list(
      "sheep-reproduce" = c(1, 6, 20)
    ),
    list(
      "wolf-reproduce" = c(2, 7, 15)
    ),
    list(
      "sheep-gain-from-food" = c(1, 8, 15)
    ),
    list(
      "wolf-gain-from-food" = c(10, 20, 30)
    )
  )
)

setup_file

setup_file |> inspect_experiment()

## Examples from the Spread of Disease Model (IABM Textbook) ----

### Population Density

setup_file <- create_experiment(
  name = "Population Density",
  repetitions = 10,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = NULL,
  setup = 'setup',
  go = 'go',
  metrics = 'ticks',
  constants = list(
    "variant" = "mobile",
    "connections-per-node" = 4.1,
    "num-people" = list(
      first = 50,
      step = 50,
      last = 200
    ),
```

```
    "num-infected" = 1,
    "disease-decay" = 0
  )
)

setup_file

setup_file |> inspect_experiment()

### Degree

setup_file <- create_experiment(
  name = "Degree",
  repetitions = 10,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = 50,
  setup = 'setup',
  go = 'go',
  metrics = 'count turtles with [infected?]',
  constants = list(
    "num-people" = 200,
    "connections-per-node" = list(
      first = 0.5,
      step = 0.5,
      last = 4
    ),
    "disease-decay" = 10,
    "variant" = "network",
    "num-infected" = 1
  )
)

setup_file

setup_file |> inspect_experiment()

### Environmental

setup_file <- create_experiment(
  name = "Environmental",
  repetitions = 10,
  sequential_run_order = TRUE,
  run_metrics_every_step = FALSE,
  time_limit = NULL,
  setup = 'setup',
  go = 'go',
  metrics = 'ticks',
  constants = list(
    "num-people" = 200,
    "connections-per-node" = 4,
    "disease-decay" = list(
      first = 0,
```

```
      step = 1,
      last = 10
    ),
    "variant" = "environmental",
    "num-infected" = 1
  )
)

setup_file

setup_file |> inspect_experiment()
```

`find_netlogo_console` *Find NetLogo executable file*

Description

`find_netlogo_console()` attempts to locate the NetLogo executable file on the user's system.

Usage

```
find_netlogo_console()
```

Details

The function uses the following search order:

1. Checks the `NETLOGO_CONSOLE` environment variable. If set and the file exists, returns that path.
2. If the environment variable is not set or the file does not exist, constructs and expands the path based on the output of `find_netlogo_home()` (`<NETLOGO_HOME>/NetLogo_Console.exe` on Windows or `<NETLOGO_HOME>/NetLogo_Console` for other systems).

Value

A `character` string specifying the path to the NetLogo executable file. Returns `NA` if the executable cannot be found at any location.

See Also

Other system functions: `find_netlogo_home()`, `find_netlogo_version()`

Examples

```
## Not run:
  find_netlogo_console()

## End(Not run)
```

`find_netlogo_home` *Find NetLogo installation directory*

Description

`find_netlogo_home()` attempts to locate the installation directory of NetLogo on the user's system.

Usage

```
find_netlogo_home()
```

Details

The function uses the following search order:

1. Checks the `NETLOGO_HOME` environment variable. If set and the directory exists, returns that path.
2. If the environment variable is not set or the directory does not exist, searches through common installation paths for directories containing "NetLogo" (case-insensitive) in their name. If multiple NetLogo installations are found in the same directory, the last one (alphabetically) is returned.

Value

A `character` string specifying the path to the NetLogo installation directory. Returns `NA` if no installation can be found.

See Also

Other system functions: `find_netlogo_console()`, `find_netlogo_version()`

Examples

```
## Not run:
  find_netlogo_home()

## End(Not run)
```

`find_netlogo_version` *Find NetLogo version*

Description

`find_netlogo_version()` attempts to determine the NetLogo version installed on the user's system.

Usage

```
find_netlogo_version()
```

Details

The function uses the following detection methods in order:

1. If the NetLogo console executable is found by `find_netlogo_console()`, it runs `NetLogo_Console --headless --version` command to retrieve the version information. This is the most reliable method.
2. If the executable is not found, it attempts to extract the version number from the installation directory name returned by `find_netlogo_home()` (e.g., `NetLogo 7.0.2` yields `"7.0.2"`). Note that this fallback may produce slightly different results if the directory was renamed or uses a non-standard naming convention.

Value

A `character` string specifying the NetLogo version (e.g., `"7.0.3"`). Returns `NA` if the version cannot be determined.

See Also

Other system functions: `find_netlogo_console()`, `find_netlogo_home()`

Examples

```
## Not run:
  find_netlogo_version()

## End(Not run)
```

`get_netlogo_shape` *Download NetLogo shapes from LogoShapes*

Description

`get_netlogo_shape()` downloads NetLogo shapes from the LogoShapes project on GitHub.

The collections and shapes available for download can be found in the LogoShapes project `svg` directory. Refer to the LogoShapes documentation for more information about the different collections.

Note: This function requires an active internet connection and the `httr2` package.

Usage

```
get_netlogo_shape(
  shape,
  collection = "netlogo-refined",
  dir = tempdir(),
  user_agent = "logolink <https://CRAN.R-project.org/package=logolink>",
  auth_token = Sys.getenv("GH_TOKEN")
)
```

Arguments

`shape`	A `character` vector indicating the names of the shapes to download.
`collection`	(optional) A `character` string indicating the collection of shapes to download from (default: `"netlogo-refined"`).
`dir`	(optional) A `character` string indicating the directory where the shapes will be saved (default: `tempdir()`).
`user_agent`	(optional) A `character` string indicating the user agent to use for the GitHub API requests. (default: `"logolink <https://CRAN.R-project.org/package=logolink>"`).
`auth_token`	(optional) A `character` string indicating a GitHub Personal Access Token (PAT) for authentication with the GitHub API. This is useful when dealing with rate limits. (default: `Sys.getenv("GH_TOKEN")`).

Value

A named `character` vector with the file paths to the downloaded NetLogo shapes as SVG files.

Examples

```
## Not run:
  library(fs)
  library(magick)

## End(Not run)
```

```
## Not run:
  shape <- get_netlogo_shape("turtle")

  file_size(shape)

  shape |> image_read_svg() |> image_ggplot()

## End(Not run)

## Not run:
  shape <- get_netlogo_shape("turtle", collection = "netlogo-simplified")

  file_size(shape)

  shape |> image_read_svg() |> image_ggplot()

## End(Not run)

## Not run:
  shape <- get_netlogo_shape("turtle", collection = "netlogo-7-0-3")

  file_size(shape)

  shape |> image_read_svg() |> image_ggplot()

## End(Not run)
```

`inspect_experiment` *Inspect NetLogo BehaviorSpace experiment file*

Description

`inspect_experiment()` reads and prints the content of a NetLogo BehaviorSpace experiment XML file to the R console. This is useful for debugging and verifying the structure of experiment files created by `create_experiment()`.

For complete guidance on setting up and running experiments in NetLogo, please refer to the BehaviorSpace Guide.

Usage

```
inspect_experiment(file)
```

Arguments

`file`	A `character` string specifying the path to the BehaviorSpace experiment XML file.

Value

An invisible NULL. This function is called for its side effect of printing the XML content to the R console.

See Also

Other BehaviorSpace functions: create_experiment(), read_experiment(), run_experiment()

Examples

```
file <- create_experiment(name = "My Experiment")

file |> inspect_experiment()
```

---

parse_netlogo_color *Parse NetLogo colors*

Description

parse_netlogo_color() parses NetLogo color codes into their approximate hexadecimal color representations.

Note: This function requires the colorspace, and scales packages.

Usage

```
parse_netlogo_color(x, bias = 0.1)
```

Arguments

x	A numeric vector containing NetLogo color codes (ranging from 0 to 140) to be parsed into hexadecimal color representations.
bias	(optional) A numeric value between -1 and 1 that adjusts the lightness or darkness of the resulting colors. Positive values lighten colors, while negative values darken them. This only affects shaded colors (those with shades other than 5) (default: 0.1).

Details

NetLogo color codes are based on 14 hues, which can be visualized by running base-colors in the NetLogo console. Each hue can be adjusted using shades from 0 to 9, where 0 represents the darkest shade and 5 represents the base shade. Shades 6 through 9 represent progressively lighter variations.

Note that NetLogo also supports extracting RGB components directly with extract-rgb. This function provides an alternative approach for obtaining color representations from NetLogo color codes.

Value

A character vector containing the approximate hexadecimal color representations corresponding to the input NetLogo color codes.

See Also

Other parsing functions: parse_netlogo_list()

Examples

```
# Simple Parsing Examples -----

netlogo_base_colors <- c(
  "gray" = 5,
  "red" = 15,
  "orange" = 25,
  "brown" = 35,
  "yellow" = 45,
  "green" = 55,
  "lime" = 65,
  "turquoise" = 75,
  "cyan" = 85,
  "sky" = 95,
  "blue" = 105,
  "violet" = 115,
  "magenta" = 125,
  "pink" = 135
)

parse_netlogo_color(netlogo_base_colors)

parse_netlogo_color(seq(10, 20, by = 1))

parse_netlogo_color(seq(10, 20, by = 0.5))

# Bias Adjustment Examples -----

parse_netlogo_color(17.5, bias = 0)

parse_netlogo_color(17.5, bias = -0.5)

parse_netlogo_color(17.5, bias = 0.5)
```

---

parse_netlogo_list *Parse NetLogo lists*

Description

`parse_netlogo_list()` parses NetLogo-style lists represented as strings (e.g., `"[1 2 3]"`) into R lists. It automatically detects `numeric`, `integer`, `logical`, and `character` types within the lists and converts them accordingly.

Note: We recommend using this function **only when necessary**, as it can be computationally intensive for large datasets and may not handle all edge cases. NetLogo provides a special output format called lists that exports list metrics in a tabular structure. If your experiment includes metrics that return NetLogo lists, include `"lists"` in the `outputs` argument of `run_experiment()` to capture this output.

Usage

```
parse_netlogo_list(x)
```

Arguments

x	An `atomic` vector potentially containing NetLogo-style lists.

Details

The function handles the following cases:

- **Homogeneous lists**: Lists containing elements of the same type are returned as atomic vectors (e.g., `"[1 2 3]"` becomes `c(1L, 2L, 3L)`).
- **Mixed-type lists**: Lists containing elements of different types are returned as R lists (e.g., `'[1.1 "a" true]'` becomes `list(1.1, "a", TRUE)`).
- **Nested lists**: Lists containing other lists are fused with the main list (e.g., `'["a" "b" [1 2]]'` becomes `list(c("a", "b"), c(1L, 2L))`).

NetLogo boolean values (`true`/`false`) are converted to R `logical` values (`TRUE`/`FALSE`). NetLogo `NaN` values are parsed as R `NaN` .

Value

A `list` where each element is the parsed result of the corresponding input element. Parsed elements may be atomic vectors (for homogeneous lists) or nested lists (for mixed-type or nested lists). If a NetLogo list is not detected in an input element, that element is returned as a single-element `list` containing the original string.

See Also

Other parsing functions: `parse_netlogo_color()`

Examples

```
# Scalar Examples -----

'test' |> parse_netlogo_list() # Not a NetLogo list.
```

```
'[1]' |> parse_netlogo_list()

'["a" "b" "c"]' |> parse_netlogo_list()

'[1 2 3]' |> parse_netlogo_list()

'[1.1 2.1 3.1]' |> parse_netlogo_list()

'[true false true]' |> parse_netlogo_list()

# Vector Examples -----

c(1, 2, 3) |> parse_netlogo_list() # Not a NetLogo list.

c('["a" "b" "c"]', '["d" "e" "f"]') |> parse_netlogo_list()

c('[1 2 3]', '[4 5 6]') |> parse_netlogo_list()

c('[1.1 2.1 3.1]', '[4.1 5.1 6.1]') |> parse_netlogo_list()

c('[true false true]', '[false true false]') |> parse_netlogo_list()

# Mixed-Type Examples -----

'["a" "b" 1 2]' |> parse_netlogo_list()

'[1.1 2.1 3.1 true false]' |> parse_netlogo_list()

'[1.1 "a" true]' |> parse_netlogo_list()

# Nested Examples -----

'["a" "b" "c" [1 2]]' |> parse_netlogo_list()

'["a" "b" "c" [1 2] true ["d" "c"]]' |> parse_netlogo_list()

'[[1 2] [3 4] [5 6]]' |> parse_netlogo_list()
```

`read_experiment` *Read NetLogo BehaviorSpace Experiment output*

Description

`read_experiment()` reads NetLogo BehaviorSpace experiment output files as tidy data frames. It automatically detects the output format (Table, Spreadsheet, Lists, or Stats) and parses the data accordingly. The function also extracts metadata from the files.

Only version 2.0 (NetLogo 6.4 and later) of BehaviorSpace output files is supported.

Usage

```
read_experiment(file, tidy_output = TRUE)
```

Arguments

file	A character string specifying the path to the BehaviorSpace output CSV file.
tidy_output	(optional) A logical flag indicating whether to tidy the output data frames. If TRUE, output data frames are arranged according to tidy data principles. If FALSE, only the default transformations from read_delim() and clean_names() are applied to the output data (default: TRUE).

Value

A list containing the experiment results. The list includes the following elements, depending on the output file provided:

- metadata: A list with metadata about the experiment run (present in all cases).
- table: A tibble with the results of the table output.
- spreadsheet: A list with the results of the spreadsheet output containing two elements:
  - statistics: A tibble with data from the output first section.
  - data: A tibble with data from the output second section.
- lists: A tibble with the results of the lists output.
- statistics: A tibble with the results of the statistics output.

See Also

Other BehaviorSpace functions: create_experiment(), inspect_experiment(), run_experiment()

Examples

```
file <- tempfile()

c(
  'BehaviorSpace results (NetLogo 7.0.3), "Table version 2.0"',
  paste0(
    '"/opt/NetLogo 7-0-3/models/',
    'IABM Textbook/chapter 4/Wolf Sheep Simple 5.nlogox"'
  ),
  '"Wolf Sheep Simple Model Analysis"',
  '"01/05/2026 06:37:48:683 -0300"',
  '"min-pxcor","max-pxcor","min-pycor","max-pycor"',
  '"-17","17","-17","17"',
  paste0(
    '"[run number]","number-of-sheep","number-of-wolves",',
    '"movement-cost","grass-regrowth-rate","energy-gain-from-grass",',
    '"energy-gain-from-sheep","[step]","count wolves","count sheep"'
  ),
  '"3","500","5","0.5","0.3","2","5","0","5","500"',
  '"5","500","5","0.5","0.3","2","5","0","5","500"',
  '"4","500","5","0.5","0.3","2","5","0","5","500"',
  '"6","500","5","0.5","0.3","2","5","0","5","500"',
  '"1","500","5","0.5","0.3","2","5","0","5","500"',
  '"8","500","5","0.5","0.3","2","5","0","5","500"',
```

```
  '"9","500","5","0.5","0.3","2","5","0","5","500"',
  '"2","500","5","0.5","0.3","2","5","0","5","500"'
) |>
  writeLines(file)

read_experiment(file)
```

`run_experiment` *Run NetLogo BehaviorSpace experiment*

Description

`run_experiment()` runs a NetLogo BehaviorSpace experiment in headless mode and returns a `list` with results as tidy data frames. It can be used with `create_experiment()` to create and run experiments on the fly, or with an existing experiment stored in the NetLogo model file.

To avoid issues with list parsing, `run_experiment()` includes support for the special lists output format. If your experiment includes metrics that return NetLogo lists, include `"lists"` in the `output` argument to capture this output. Columns containing NetLogo lists are returned as `character` vectors.

The function tries to locate the NetLogo installation automatically. This is usually successful, but if it fails, you will need to set it manually. See the *Details* section for more information.

For complete guidance on setting up and running experiments in NetLogo, please refer to the BehaviorSpace Guide.

Usage

```
run_experiment(
  model_path,
  setup_file = NULL,
  experiment = NULL,
  output = "table",
  other_arguments = NULL,
  timeout = Inf,
  tidy_output = TRUE
)
```

Arguments

`model_path`	A `character` string specifying the path to the NetLogo model file (with extension `.nlogo`, `.nlogo3d`, `.nlogox`, or `.nlogox3d`).
`setup_file`	(optional) A `character` string specifying the path to an XML file containing the experiment definition. This file can be created using `create_experiment()` or exported from the NetLogo BehaviorSpace interface (default: `NULL`).
`experiment`	(optional) A `character` string specifying the name of the experiment defined in the NetLogo model file (default: `NULL`).

`output` (optional) A `character` vector specifying which output types to generate from the experiment. Valid options are: `"table"`, `"spreadsheet"`, `"lists"`, and `"statistics"`. At least one of `"table"` or `"spreadsheet"` must be included. See the BehaviorSpace documentation on formats for details about each output type (default: `c("table", "lists")`).

`other_arguments` (optional) A `character` vector specifying any additional command-line arguments to pass to the NetLogo executable. For example, you can use `c("--threads 4")` to specify the number of threads. See the *Details* section for more information (default: `NULL`).

`timeout` (optional) A `numeric` value specifying the maximum time (in seconds) to wait for the NetLogo process to complete. If the process exceeds this time limit, it will be terminated, and the function will return the available output up to that point. Use `Inf` for no time limit (default: `Inf`).

`tidy_output` (optional) A `logical` flag indicating whether to tidy the output data frames. If `TRUE`, output data frames are arranged according to tidy data principles. If `FALSE`, only the default transformations from `read_delim()` and `clean_names()` are applied to the output data (default: `TRUE`).

Details

Setting the NetLogo Installation Path:

If `run_experiment()` cannot find the NetLogo installation, you will need to set the path manually using the `NETLOGO_HOME` environment variable. On Windows, a typical path is something like `C:\Program Files\NetLogo 7.0.3`. You can set this variable temporarily in your R session with:

```
Sys.setenv(NETLOGO_HOME = "PATH/TO/NETLOGO/INSTALLATION")
```

or permanently by adding it to your `.Renviron` file.

If even after setting the `NETLOGO_HOME` variable you still encounter issues, try setting a `NETLOGO_CONSOLE` environment variable with the path to the NetLogo executable or binary. On Windows, a typical path is something like `C:\Program Files\NetLogo 7.0.3\NetLogo.exe`.

NetLogo 3D:

The function automatically detects whether the provided model is a 3D model (based on the file extension) and adjusts the command-line arguments accordingly. You do not need to set the `--3D` flag to the `other_arguments` parameter manually.

Handling NetLogo Lists:

NetLogo uses a specific syntax for lists (e.g., `"[1 2 3]"`) that is incompatible with standard CSV formats. To address this, NetLogo provides a special output format called lists that exports list metrics in a tabular structure. If your experiment includes metrics that return NetLogo lists, include `"lists"` in the `output` argument to capture this output. Columns containing NetLogo lists are returned as `character` vectors.

The `parse_netlogo_list()` function is available for parsing NetLogo list values embedded in other outputs. However, we recommend using it only when necessary, as it can be computationally intensive for large datasets and may not handle all edge cases.

Additional Command-Line Arguments:

You can pass additional command-line arguments to the NetLogo executable using the `other_arguments` parameter. This can be useful for specifying options such as the number of threads to use or other NetLogo-specific flags.

For example, to specify the number of threads, you can use:

```
run_experiment(
  model_path = "path/to/model.nlogox",
  setup_file = "path/to/experiment.xml",
  other_arguments = c("--threads 4")
)
```

There are a variety of command-line options available, but some are reserved for internal use by `run_experiment()` and cannot be modified. These are:

- `--headless`: Ensures NetLogo runs in headless mode.
- `--3D`: Specifies if the model is a 3D model (automatically set based on the model file extension).
- `--model`: Specifies the path to the NetLogo model file.
- `--setup-file`: Specifies the path to the experiment XML file.
- `--experiment`: Specifies the name of the experiment defined in the model.
- `--table`: Specifies the output file for the table results.
- `--spreadsheet`: Specifies the output file for the spreadsheet results.
- `--lists`: Specifies the output file for the lists results.
- `--stats`: Specifies the output file for the statistics results.

For a complete list of available options, refer to the BehaviorSpace Guide.

Non-Tabular Output:

If the experiment generates any non-tabular output (e.g., prints, error messages, warnings), it will be captured and displayed as an informational message after the results data frame is returned. This allows you to see any important messages generated during the experiment run. Keep in mind that excessive non-tabular output may clutter your R console.

Value

A `list` containing the experiment results. The `list` includes the following elements, depending on the values specified in the `output` parameter:

- `metadata`: A `list` with metadata about the experiment run (present in all cases).
- `table`: A `tibble` with the results of the `table` output.
- `spreadsheet`: A `list` with the results of the `spreadsheet` output containing two elements:
  - `statistics`: A `tibble` with data from the output first section.
  - `data`: A `tibble` with data from the output second section.
- `lists`: A `tibble` with the results of the `lists` output.
- `statistics`: A `tibble` with the results of the `statistics` output.

See Also

Other BehaviorSpace functions: create_experiment(), inspect_experiment(), read_experiment()

Examples

```
# Defining the Model -----

## Not run:
  # This model is included with NetLogo installations.
  model_path <-
    find_netlogo_home() |>
    file.path(
      "models",
      "IABM Textbook",
      "chapter 4",
      "Wolf Sheep Simple 5.nlogox"
    )

## End(Not run)

# Creating an Experiment -----

## Not run:
  setup_file <- create_experiment(
    name = "Wolf Sheep Simple Model Analysis",
    repetitions = 10,
    sequential_run_order = TRUE,
    run_metrics_every_step = TRUE,
    setup = "setup",
    go = "go",
    time_limit = 1000,
    metrics = c(
      'count wolves',
      'count sheep'
    ),
    run_metrics_condition = NULL,
    constants = list(
      "number-of-sheep" = 500,
      "number-of-wolves" = list(
        first = 5,
        step = 1,
        last = 15
      ),
      "movement-cost" = 0.5,
      "grass-regrowth-rate" = 0.3,
      "energy-gain-from-grass" = 2,
      "energy-gain-from-sheep" = 5
    )
  )

## End(Not run)
```

```
# Running the Experiment -----

## Not run:
  model_path |>
    run_experiment(
      setup_file = setup_file
    )

## End(Not run)

# Running an Experiment Defined in the NetLogo Model File -----

## Not run:
  model_path |>
    run_experiment(
      experiment = "Wolf Sheep Simple model analysis"
    )

## End(Not run)
```

Index