Package ‘envsetup’

October 16, 2025

Title Support the Setup of the R Environment for Clinical Trial Programming Workflows

Version 0.3.0

Description The purpose of this package is to support the setup the R environment. The two main features are 'autos', to automatically source files and/or directories into your environment, and 'paths' to consistently set path objects across projects for input and output. Both are implemented using a configuration file to allow easy, custom configurations that can be used for multiple or all projects.

License Apache License 2.0

Encoding UTF-8

RoxygenNote 7.3.2

Imports config, fs, purrr, rlang, usethis, envnames, utils

Suggests rmarkdown, testthat, knitr, kableExtra, magrittr, devtools, readr, tidyr, withr, lintr, styler, renv, covr

VignetteBuilder knitr

Config/testthat/edition 3

URL `https://github.com/pharmaverse/envsetup`

BugReports `https://github.com/pharmaverse/envsetup/issues`

NeedsCompilation no

Author Nicholas Masel [aut, cre],
Mike Stackhouse [aut] (ORCID: <`https://orcid.org/0000-0001-6030-723X`>),
Aidan Ceney [aut],
Johnson & Johnson Innovative Medicine [cph, fnd],
Atorus Research, Inc. [cph]

Maintainer Nicholas Masel <`nmasel@its.jnj.com`>

Repository CRAN

Date/Publication 2025-10-16 11:50:02 UTC

Contents

`build_from_config` *Build directory structure from a configuration file*

Description

Build directory structure from a configuration file

Usage

```
build_from_config(config, root = NULL)
```

Arguments

`config`	configuration object from config::get() containing paths
`root`	root directory to build from. Leave as NULL if using absolute paths. Set to working directory if using relative paths.

Value

Called for its side-effects. The directories build print as a tree-like format from `fs::dir_tree()`.

Examples

```
tmpdir <- tempdir()

hierarchy <- "default:
  paths:
    data: !expr list(DEV = '/demo/DEV/username/project1/data',
                     PROD = '/demo/PROD/project1/data')
    output: !expr list(DEV = '/demo/DEV/username/project1/output',
                       PROD = '/demo/PROD/project1/output')
    programs: !expr list(DEV = '/demo/DEV/username/project1/programs',
                         PROD = '/demo/PROD/project1/programs')
    docs: !expr list(DEV = 'docs',
                     PROD = 'docs')"
```

```
writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))

config <- config::get(file = file.path(tmpdir, "hierarchy.yml"))

build_from_config(config, tmpdir)
```

`detach_autos` *Detach the autos from the current session*

Description

This function will remove any autos that have been set from the search path

Usage

```
detach_autos()
```

Value

Called for its side-effects.

Examples

```
tmpdir <- tempdir()
print(tmpdir)

# account for windows
if (Sys.info()['sysname'] == "Windows") {
  tmpdir <- gsub("\\", "\\\\", tmpdir, fixed = TRUE)
}

# Create an example config file\
hierarchy <- paste0("default:
  paths:
    functions: !expr list(DEV = file.path('",tmpdir,"',
                                         'demo',
                                         'DEV',
                                         'username',
                                         'project1',
                                         'functions'),
                          PROD = file.path('",tmpdir,"',
                                          'demo',
                                          'PROD',
                                          'project1',
                                          'functions'))
  autos:
     my_functions: !expr list(DEV = file.path('",tmpdir,"',
                                             'demo',
                                             'DEV',
                                           'username',
                                           'project1',
                                           'functions'),
                                  PROD = file.path('",tmpdir,"',
                                            'demo',
                                            'PROD',
                                            'project1',
                                            'functions'))")

# write config
writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))

config <- config::get(file = file.path(tmpdir, "hierarchy.yml"))

build_from_config(config)

# write function to DEV
writeLines("dev_function <- function() {print(environment(dev_function))}",
       file.path(tmpdir, 'demo', 'DEV', 'username', 'project1', 'functions', 'dev_function.r'))

# write function to PROD
writeLines("prod_function <- function() {print(environment(prod_function))}",
          file.path(tmpdir, 'demo', 'PROD', 'project1', 'functions', 'prod_function.r'))

# setup the environment
Sys.setenv(ENVSETUP_ENVIRON = "DEV")
rprofile(config::get(file = file.path(tmpdir, "hierarchy.yml")))

# show dev_function() and prod_function() are available and print their location
dev_function()
prod_function()

# remove autos from search
detach_autos()
```

envsetup_environment *Environment Setup Environment*

Description

A dedicated environment object used to store and manage path configurations and other setup variables for the envsetup package. This environment provides an isolated namespace for storing path objects that can be retrieved using the package's path management functions.

Usage

```
envsetup_environment
```

Format

An object of class `environment` of length 0.

Details

This environment serves as the default storage location for path objects when using envsetup package functions. It helps maintain clean separation between user workspace and package-managed paths.

Value

An environment object created with `new.env()`.

See Also

`get_path`, `new.env`

Examples

```
# Store a path in the envsetup environment
assign("project_root", "/path/to/project", envir = envsetup_environment)

# List objects in the environment
ls(envir = envsetup_environment)

# Check if the environment exists and is an environment
exists("envsetup_environment")
is.environment(envsetup_environment)
```

`get_path` *Get Path Object from Environment*

Description

Retrieves a path object from the specified environment using non-standard evaluation. The function uses `substitute()` to capture the unevaluated expression and `get()` to retrieve the corresponding object.

Usage

```
get_path(path, envir = getOption("envsetup.path.environment"))
```

Arguments

`path`	An unquoted name of the path object to retrieve from the environment.
`envir`	The environment to search for the path object. Defaults to the value of `getOption("envsetup.path.env`

Value

The path object stored in the specified environment under the given name.

See Also

get, substitute

Examples

```
# Create a custom environment and store some paths
path_env <- new.env()
assign("data_dir", "/home/user/data", envir = path_env)
assign("output_dir", "/home/user/output", envir = path_env)

# Set up the option to use our custom environment
options(envsetup.path.environment = path_env)

# Retrieve paths using the function
data_path <- get_path(data_dir)
output_path <- get_path(output_dir)

print(data_path)    # "/home/user/data"
print(output_path)  # "/home/user/output"

# Using with a different environment
temp_env <- new.env()
assign("temp_dir", "/tmp/analysis", envir = temp_env)
temp_path <- get_path(temp_dir, envir = temp_env)
print(temp_path)    # "/tmp/analysis"
```

---

init *Initialize the R environment with envsetup*

Description

Initialize the R environment with envsetup

Usage

```
init(project, config_path = NULL, create_paths = NULL)
```

Arguments

project	Character. The path to the project directory.
config_path	Character. The path of the config file. Defaults to NULL.
create_paths	Logical indicating if missing paths should be created. Defaults to NULL.

Value

Called for its side-effects.

Examples

```
tmpdir <- tempdir()
print(tmpdir)

# account for windows
if (Sys.info()['sysname'] == "Windows") {
  tmpdir <- gsub("\\", "\\\\", tmpdir, fixed = TRUE)
}

# Create an example config file\
hierarchy <- paste0("default:
  paths:
    data: !expr list(
      DEV = file.path('",tmpdir,"', 'demo', 'DEV', 'username', 'project1', 'data'),
      PROD = file.path('",tmpdir,"', 'demo', 'PROD', 'project1', 'data'))
    output: !expr list(
      DEV = file.path('",tmpdir,"', 'demo', 'DEV', 'username', 'project1', 'output'),
      PROD = file.path('",tmpdir,"', 'demo', 'PROD', 'project1', 'output'))
    programs: !expr list(
      DEV = file.path('",tmpdir,"', 'demo', 'DEV', 'username', 'project1', 'programs'),
      PROD = file.path('",tmpdir,"', 'demo', 'PROD', 'project1', 'programs'))")


writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))

init(project = tmpdir,
     config_path = file.path(tmpdir, "hierarchy.yml"),
     create_paths = TRUE)
```

`read_path` *Read path*

Description

Check each environment for the file and return the path to the first.

Usage

```
read_path(
  lib,
  filename,
  full.path = TRUE,
  envsetup_environ = Sys.getenv("ENVSETUP_ENVIRON"),
  envir = getOption("envsetup.path.environment")
)
```

Arguments

`lib`	object containing the paths for all environments of a directory
`filename`	name of the file you would like to read
`full.path`	logical to return the path including the file name
`envsetup_environ`	name of the environment you would like to read the file from; default values comes from the value in the system variable ENVSETUP_ENVIRON which can be set by Sys.setenv(ENVSETUP_ENVIRON = "environment name")
`envir`	The environment to search for the path object. Defaults to the value of `getOption("envsetup.path.env`

Details

The environments searched depends on the current environment. For example, if your workflow contains a development (dev) area and production area (prod), and the code is executing in the dev environment, we search dev and prod. If in prod, we only search prod.

Value

string containing the path of the first directory the file is found

Examples

```
tmpdir <- tempdir()

# account for windows
if (Sys.info()['sysname'] == "Windows") {
  tmpdir <- gsub("\\", "\\\\", tmpdir, fixed = TRUE)
}

# add config for just the data location
hierarchy <- paste0("default:
  paths:
    data: !expr list(
      DEV = file.path('",tmpdir,"', 'demo', 'DEV', 'username', 'project1', 'data'),
      PROD = file.path('",tmpdir,"', 'demo', 'PROD', 'project1', 'data'))")

# write config file to temp directory
writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))

config <- config::get(file = file.path(tmpdir, "hierarchy.yml"))

# build folder structure from config
build_from_config(config)

# setup environment based on config
rprofile(config::get(file = file.path(tmpdir, "hierarchy.yml")))

# place data in prod data folder
saveRDS(mtcars, file.path(tmpdir, "demo/PROD/project1/data/mtcars.rds"))
```

```
# find the location of mtcars.rds
read_path(data, "mtcars.rds")
```

rprofile *Function used to pass through code to the .Rprofile*

Description

Function used to pass through code to the .Rprofile

Usage

```
rprofile(
  config,
  envir = getOption("envsetup.path.environment"),
  overwrite = TRUE
)
```

Arguments

`config`	configuration object from config::get()
`envir`	The environment to search for the path object. Defaults to the value of `getOption("envsetup.path.env`
`overwrite`	logical indicating if sourcing of autos should overwrite an object in global if it already exists

Value

Called for its side effects. Directory paths and autos are added to the search path based on your config.

Examples

```
# temp location to store configuration files
tmpdir <- tempdir()
print(tmpdir)

# Create an example config file
hierarchy <- "default:
  paths:
    data: !expr list(DEV = '/demo/DEV/username/project1/data',
                     PROD = '/demo/PROD/project1/data')
    output: !expr list(DEV = '/demo/DEV/username/project1/output',
                       PROD = '/demo/PROD/project1/output')
    programs: !expr list(DEV = '/demo/DEV/username/project1/programs',
                         PROD = '/demo/PROD/project1/programs')"

writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))

rprofile(config::get(file = file.path(tmpdir, "hierarchy.yml")))
```

validate_config *Validate a configuration file*

Description

A helper function to help troubleshoot common problems that can occur when building your configuration file.

Usage

```
validate_config(config)
```

Arguments

`config`	configuration object from config::get()

Value

Called for its side-effects. Prints findings from validation checks.

Examples

```
# temp location to store configuration files
tmpdir <- tempdir()
print(tmpdir)

# Each path only points to one location, i.e. there is no hierarchy for a path
no_hierarchy <- 'default:
  paths:
    data: "/demo/DEV/username/project1/data"
    output: "/demo/DEV/username/project1/output"
    programs: "/demo/DEV/username/project1/programs"'

writeLines(no_hierarchy, file.path(tmpdir, "no_hierarchy.yml"))

validate_config(config::get(file = file.path(tmpdir, "no_hierarchy.yml")))

# A path can point to multiple locations, i.e. there is a hierarchy
hierarchy <- "default:
  paths:
    data: !expr list(DEV = '/demo/DEV/username/project1/data',
                     PROD = '/demo/PROD/project1/data')
    output: !expr list(DEV = '/demo/DEV/username/project1/output',
                       PROD = '/demo/PROD/project1/output')
    programs: !expr list(DEV = '/demo/DEV/username/project1/programs',
                         PROD = '/demo/PROD/project1/programs')
    envsetup_environ: !expr Sys.setenv(ENVSETUP_ENVIRON = 'DEV'); 'DEV'"

writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))
```

```
validate_config(config::get(file = file.path(tmpdir, "hierarchy.yml")))

# A hierarchy is present for paths, but they are not named
hierarchy_no_names <- "default:
  paths:
    data: !expr list('/demo/DEV/username/project1/data', '/demo/PROD/project1/data')
   output: !expr list('/demo/DEV/username/project1/output', '/demo/PROD/project1/output')
   programs: !expr list('/demo/DEV/username/project1/programs', '/demo/PROD/project1/programs')
    envsetup_environ: !expr Sys.setenv(ENVSETUP_ENVIRON = 'DEV'); 'DEV'"


writeLines(hierarchy_no_names, file.path(tmpdir, "hierarchy_no_names.yml"))

validate_config(config::get(file = file.path(tmpdir, "hierarchy_no_names.yml")))


# No paths are specified
no_paths <- "default:
  autos:
    my_functions: '/demo/PROD/project1/R'"

writeLines(no_paths, file.path(tmpdir, "no_paths.yml"))

validate_config(config::get(file = file.path(tmpdir, "no_paths.yml")))
```

`write_path` *Retrieve a file path from an envsetup object containing paths*

Description

Paths will be filtered to produce the lowest available level from a hierarchy of paths based on envsetup_environ

Usage

```
write_path(
  lib,
  filename = NULL,
  envsetup_environ = Sys.getenv("ENVSETUP_ENVIRON"),
  envir = getOption("envsetup.path.environment")
)
```

Arguments

`lib`	Object containing the paths for all environments of a directory
`filename`	Name of the file you would like to write
`envsetup_environ`	Name of the environment to which you would like to write. Defaults to the ENVSETUP_ENVIRON environment variable
`envir`	The environment to search for the path object. Defaults to the value of `getOption("envsetup.path.envi`

Value

path to write

Examples

```
tmpdir <- tempdir()

# account for windows
if (Sys.info()['sysname'] == "Windows") {
  tmpdir <- gsub("\\", "\\\\", tmpdir, fixed = TRUE)
}

# add config for just the data location
hierarchy <- paste0("default:
  paths:
    data: !expr list(
      DEV = file.path('",tmpdir,"', 'demo', 'DEV', 'username', 'project1', 'data'),
      PROD = file.path('",tmpdir,"', 'demo', 'PROD', 'project1', 'data'))")

# write config file to temp directory
writeLines(hierarchy, file.path(tmpdir, "hierarchy.yml"))

config <- config::get(file = file.path(tmpdir, "hierarchy.yml"))

# build folder structure from config
build_from_config(config)

# setup environment based on config
rprofile(config::get(file = file.path(tmpdir, "hierarchy.yml")))

# find location to write mtcars.rds
write_path(data, "mtcars.rds")

# save data in data folder using write_path
saveRDS(mtcars, write_path(data, "mtcars.rds"))
```

Index