Package 'link2GI'

May 8, 2026

Type Package

Title Linking Geographic Information Systems, Remote Sensing and Other Command Line Tools

Version 0.7-2

Date 2025-12-21

Encoding UTF-8

Maintainer Chris Reudenbach <reudenbach@uni-marburg.de>

Description Functions and tools for using open GIS and remote sensing command-line interfaces in a reproducible environment.

URL https://github.com/r-spatial/link2GI/, https://r-spatial.github.io/link2GI/

BugReports https://github.com/r-spatial/link2GI/issues/

License GPL (>= 3) | file LICENSE

Depends R (>= 3.5.0)

Imports devtools, R.utils, roxygen2, sf (>= 0.9), brew, yaml, terra, methods, utils, rstudioapi, renv

SystemRequirements GNU make

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, sp, rgrass, stars, curl, markdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Chris Reudenbach [cre, aut], Tim Appelhans [ctb]

Repository CRAN

Date/Publication 2025-12-23 07:30:02 UTC

Contents

`createFolders` *Compile folder list and create folders*

Description

Compile folder list with absolut paths and create folders if necessary.

Usage

```
createFolders(root_folder, folders, create_folders = TRUE)
```

Arguments

`root_folder`	root directory of the project.
`folders`	list of subfolders within the project directory.
`create_folders`	create folders if not existing already.

Value

List with folder paths and names.

Examples

```
## Not run:
 createFolders(root_folder = tempdir(), folders = c('data/', 'data/tmp/'))

## End(Not run)
# Create folder list and set variable names pointing to the path values
```

`findGDAL` *Search recursivly existing 'GDAL binaries' installation(s) at a given drive/mountpoint*

Description

Provides an list of valid 'GDAL' installation(s) on your 'Windows' system. There is a major difference between osgeo4W and stand_alone installations. The functions trys to find all valid installations by analysing the calling batch scripts.

Usage

```
findGDAL(searchLocation = "default", quiet = TRUE)
```

Arguments

`searchLocation`	drive letter to be searched, for Windows systems default is `C:/`, for Linux systems default is `/usr/bin`.
`quiet`	boolean switch for supressing console messages default is TRUE

Value

A dataframe with the 'GDAL' root folder(s), and command line executable(s)

Author(s)

Chris Reudenbach

Examples

```
run = FALSE
if (run) {
# find recursively all existing 'GDAL' installations folders starting
# at the default search location
findGDAL()
}
```

findGRASS *Returns attributes of valid 'GRASS GIS' installation(s) on the system.*

Description

Retrieve a list of valid 'GRASS GIS' installation(s) on your system. On Windows, uses searchGRASSW() (cmd-free). On Unix, uses searchGRASSX().

Usage

```
findGRASS(searchLocation = "default", ver_select = FALSE, quiet = TRUE)
```

Arguments

searchLocation	On Windows MUST start with drive letter + colon, e.g. "C:", "C:/", "C:/Users/...". Defaults to "C:/". On Unix defaults to "/usr/bin".
ver_select	If TRUE and more than one installation is found, interactively select one.
quiet	Suppress messages.

Value

FALSE or data.frame(instDir, version, installation_type)

findOTB *Locate Orfeo ToolBox (OTB) installations*

Description

Dispatcher that calls the OS-specific search function and returns a normalized installations table.

Usage

```
findOTB(searchLocation = NULL, quiet = TRUE)
```

Arguments

searchLocation	Character. On Linux: mountpoints/roots to search. If NULL, defaults to "default" which expands to c("~","/opt","/usr/local","/usr").
quiet	Logical.

Value

data.frame of installations or FALSE.

`findSAGA` *Search recursively existing 'SAGA GIS' installation(s) at a given drive/mount point*

Description

Provides an list of valid 'SAGA GIS' installation(s) on your 'Windows' system. There is a major difference between osgeo4W and stand_alone installations. The functions tries to find all valid installations by analyzing the calling batch scripts.

Usage

```
findSAGA(searchLocation = "default", quiet = TRUE)
```

Arguments

`searchLocation`	drive letter to be searched, for Windows systems default is `C:/`, for Linux systems default is `/usr/bin`.
`quiet`	boolean switch for suppressing console messages default is TRUE

Value

A dataframe with the 'SAGA GIS' root folder(s), version name(s) and installation type code(s)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# find recursively all existing 'SAGA GIS' installation folders starting
# at the default search location
findSAGA()

## End(Not run)
```

`gvec2sf` *Converts from an existing 'GRASS' environment an arbitrary vector dataset into a sf object*

Description

Converts from an existing 'GRASS' environment an arbitrary vector dataset into a sf object

Usage

```
gvec2sf(x, obj_name, gisdbase, location, gisdbase_exist = TRUE)
```

Arguments

x	sf object corresponding to the settings of the corresponding GRASS container
obj_name	name of GRASS layer
gisdbase	GRASS gisDbase folder
location	GRASS location name containing obj_name
gisdbase_exist	logical switch if the GRASS gisdbase folder exist default is TRUE

Note

have a look at the sf capabilities to read direct from sqlite

Author(s)

Chris Reudenbach

Examples

```
run = FALSE
if (run) {
## example
require(sf)
require(sp)
require(link2GI)
data(meuse)
meuse_sf = st_as_sf(meuse,
                    coords = c('x', 'y'),
                    crs = 28992,
                    agr = 'constant')


# write data to GRASS and create gisdbase
sf2gvec(x = meuse_sf,
        obj_name = 'meuse_R-G',
        gisdbase = '~/temp3/',
        location = 'project1')

# read from existing GRASS
gvec2sf(x = meuse_sf,
        obj_name = 'meuse_r_g',
        gisdbase = '~/temp3',
        location = 'project1')

}
```

initProj *Simple creation and reproduction of an efficient project environment*

Description

Set up the project environment with a defined folder structure, an RStudio project, initial scripts and configuration files and optionally with Git and Renv support.

Usage

```
initProj(
  root_folder = ".",
  folders = NULL,
  init_git = NULL,
  init_renv = NULL,
  code_subfolder = c("src", "src/functions", "src/configs"),
  global = FALSE,
  openproject = NULL,
  newsession = TRUE,
  standard_setup = "baseSpatial",
  loc_name = NULL,
  ymlFN = NULL,
  appendlibs = NULL,
  OpenFiles = NULL
)
```

Arguments

root_folder	root directory of the project.
folders	list of sub folders within the project directory that will be created.
init_git	logical: init git repository in the project directory.
init_renv	logical: init renv in the project directory.
code_subfolder	sub folders for scripts and functions within the project directory that will be created. The folders src, src/functions and src/config are mandatory.
global	logical: export path strings as global variables?
openproject	default NULL if TRUE the project is opened in a new session
newsession	open project in a new session? default is FALSE
standard_setup	select one of the predefined settings c('base', 'baseSpatial', 'advancedSpatial'). In this case, only the name of the base folder is required, but individual additional folders can be specified under 'folders' name of the git repository must be supplied to the function.
loc_name	NULL by default, defines the research area of the analysis in the data folder as a subfolder and serves as a code tag
ymlFN	filename for a yaml file containing a non standard_setup

appendlibs	vector with the names of libraries that are required for the initial project. settings required for the project, such as additional libraries, optional settings, colour schemes, etc. Important: It should not be used to control the runtime parameters of the scripts. This file is not read in automatically, even if it is located in the 'fcts_folder' folder.
OpenFiles	default NULL

Details

The function uses [setupProj] for setting up the folders. Once the project is creaeted, manage the overall configuration of the project by the 'src/functions/000_settings.R script'. It is sourced at the begining of the template scripts that are created by default. Define additional constans, required libraries etc. in the 000_settings.R at any time. If additonal folders are required later, just add them manually. They will be parsed as part of the 000_settings.R and added to a variable called dirs that allows easy acces to any of the folders. Use this variable to load/save data to avoid any hard coded links in the scripts except the top-level root folder which is defined once in the main control script located at src/main.R.

Value

dirs, i.e. a list containing the project paths.

Note

For yaml based setup you need to use one of the default configurations c('base', 'baseSpatial','advancedSpatial') or you provide a yaml file this MUST contain the standard_setup arguments, where `mysetup` is the yaml root, all other items are mandatory keywords that can be filled in as needed.

```
mysetup:
  dataFolder:
  docsFolder:
  tmpFolder:
  init_git: true/false
  init_renv: true/false
  code_subfolder: ['src', 'src/functions' , 'src/config']
  global: true/false
  libs:
  create_folders: true/false
  files:
```

Alternatively you may set default_setup to NULL and provide the arguments via command line.

Examples

```
## Not run:
root_folder <- tempdir() # Mandatory, variable must be in the R environment.
dirs <- initProj(root_folder = root_folder, standard_setup = 'baseSpatial')

## End(Not run)
```

linkGDAL *Locate and set up 'GDAL' API bindings*

Description

Locate and set up 'GDAL - Geospatial Data Abstraction Librar' API bindings

Usage

```
linkGDAL(
  bin_GDAL = NULL,
  searchLocation = NULL,
  ver_select = FALSE,
  quiet = TRUE,
  returnPaths = TRUE
)
```

Arguments

bin_GDAL	string contains path to where the gdal binaries are located
searchLocation	string hard drive letter default is C:/
ver_select	Boolean default is FALSE. If there is more than one 'GDAL' installation and ver_select = TRUE the user can select interactively the preferred 'GDAL' version
quiet	Boolean switch for suppressing messages default is TRUE
returnPaths	Boolean if set to FALSE the paths of the selected version are written to the PATH variable only, otherwise all paths and versions of the installed GRASS versions ae returned.

Details

It looks for the gdalinfo(.exe) file. If the file is found in a bin folder it is assumed to be a valid 'GDAL' binary installation.

if called without any parameter linkGDAL() it performs a full search over the hard drive C:. If it finds one or more 'GDAL' binaries it will take the first hit. You have to set ver_select = TRUE for an interactive selection of the preferred version.

Value

add gdal paths to the environment and creates global variables path_GDAL

Note

You may also set the path manually. Using a 'OSGeo4W64' https://trac.osgeo.org/osgeo4w/ installation it is typically C:/OSGeo4W64/bin/

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# call if you do not have any idea if and where GDAL is installed
gdal<-linkGDAL()
if (gdal$exist) {
# call it for a default OSGeo4W installation of the GDAL
print(gdal)
}

## End(Not run)
```

`linkGRASS` *Locate and set up GRASS GIS API bindings*

Description

Initializes a GRASS GIS 7.x/8.x runtime environment and prepares a valid temporary or permanent GRASS location and mapset for use from R.

Usage

```
linkGRASS(
  x = NULL,
  epsg = NULL,
  default_GRASS = NULL,
  search_path = NULL,
  ver_select = FALSE,
  gisdbase_exist = FALSE,
  gisdbase = NULL,
  use_home = FALSE,
  location = NULL,
  spatial_params = NULL,
  resolution = NULL,
  quiet = TRUE,
  returnPaths = TRUE
)
```

Arguments

`x`	A spatial object used to initialize the GRASS location. Supported classes are 'terra::SpatRaster', 'sf', 'sp', 'stars', or a file path to a raster dataset.
`epsg`	Integer EPSG code used to define the GRASS projection. If 'NULL', the EPSG code is inferred from 'x' when possible.

default_GRASS	Optional character vector defining a GRASS installation (e.g. ‘c("/usr/lib/grass83", "8.3.2", "grass")‘).
search_path	Character path used to search for GRASS installations.
ver_select	Logical or numeric value controlling interactive or indexed GRASS version selection.
gisdbase_exist	Logical; if ‘TRUE‘, ‘gisdbase‘ and ‘location‘ must already exist and will only be linked.
gisdbase	Path to the GRASS database directory.
use_home	Logical; if ‘TRUE‘, the user home directory is used for GISRC.
location	Name of the GRASS location to create or link.
spatial_params	Optional numeric vector defining extent manually (‘xmin, ymin, xmax, ymax[, proj]‘).
resolution	Numeric raster resolution used for ‘g.region‘.
quiet	Logical; suppress console output if ‘TRUE‘.
returnPaths	Logical; return detected GRASS installation paths.

Details

The function detects installed GRASS versions, initializes required environment variables, and derives spatial reference information either from an existing spatial object or from manually provided parameters.

GRASS requires a fully initialized runtime environment (PATH, GISBASE, PROJ, GDAL). On some platforms, R must be started from a shell where GRASS is already available.

The function ensures that ‘PROJ_INFO‘ and ‘PROJ_UNITS‘ are written by explicitly calling ‘g.proj‘ before region initialization.

Value

A list describing the selected GRASS installation and status, or ‘NULL‘ if no valid installation was found.

Author(s)

Chris Reudenbach

See Also

initGRASS, execGRASS

Examples

```
## Not run:
library(link2GI)
library(sf)

# Example 1: initialize a temporary GRASS location from an sf object
```

```
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
grass <- linkGRASS(nc)

# Example 2: select GRASS version interactively if multiple installations exist
linkGRASS(nc, ver_select = TRUE)

# Example 3: create a permanent GRASS location
root <- tempdir()
linkGRASS(
  x        = nc,
  gisdbase = root,
  location = "project1"
)

# Example 4: link to an existing GRASS location without recreating it
linkGRASS(
  gisdbase        = root,
  location        = "project1",
  gisdbase_exist  = TRUE
)

# Example 5: manual setup using spatial parameters only
epsg <- 28992
linkGRASS(
  spatial_params = c(178605, 329714, 181390, 333611),
  epsg = epsg
)

## End(Not run)
```

---

linkOTB *Locate and describe Orfeo ToolBox (OTB) API bindings*

Description

Public wrapper that dispatches to OS-specific implementations. No PATH mutation, no environment setup here.

Dispatcher that selects the platform-specific OTB locator. This function is non-invasive: it does NOT modify PATH or ENV.

Usage

```
linkOTB(
  bin_OTB = NULL,
  root_OTB = NULL,
  type_OTB = NULL,
  searchLocation = NULL,
  ver_select = FALSE,
  quiet = TRUE,
```

```
  returnPaths = TRUE
)

linkOTB(
  bin_OTB = NULL,
  root_OTB = NULL,
  type_OTB = NULL,
  searchLocation = NULL,
  ver_select = FALSE,
  quiet = TRUE,
  returnPaths = TRUE
)
```

Arguments

bin_OTB	Optional. Path to the OTB 'bin/' directory.
root_OTB	Optional. Path to the OTB installation root directory.
type_OTB	Optional installation type filter (if available from discovery).
searchLocation	Optional search location for autodetect (e.g. mountpoint).
ver_select	Selection logic: FALSE = newest, TRUE = interactive, numeric = row index.
quiet	Logical. If FALSE, print selection tables and messages.
returnPaths	Logical. If TRUE, return the gili descriptor.

Value

A gili list describing the selected OTB installation.

---

linkSAGA *Identifies SAGA GIS Installations and returns linking Informations*

Description

Finds the existing SAGA GIS installation(s), generates and sets the necessary path and system variables for a seamless use of the command line calls of the 'SAGA GIS' CLI API, setup valid system variables for calling a default `rsaga.env` and by this makes available the RSAGA wrapper functions.
All existing installation(s) means that it looks for the `saga_cmd` or `saga_cmd.exe` executables. If the file is found it is assumed to be a valid 'SAGA GIS' installation. If it is called without any argument the most recent (i.e. highest) SAGA GIS version will be linked.

Usage

```
linkSAGA(
  default_SAGA = NULL,
  searchLocation = "default",
  ver_select = FALSE,
  quiet = TRUE,
  returnPaths = TRUE
)
```

Arguments

default_SAGA	string contains path to RSAGA binaries
searchLocation	drive letter to be searched, for Windows systems default is C:, for Linux systems default is /usr/bin.
ver_select	boolean default is FALSE. If there is more than one 'SAGA GIS' installation and ver_select = TRUE the user can select interactively the preferred 'SAGA GIS' version
quiet	boolean switch for supressing console messages default is TRUE
returnPaths	boolean if set to FALSE the paths of the selected version are written to the PATH variable only, otherwise all paths and versions of the installed SAGA versions ae returned.#'@details If called without any parameter linkSAGA() it performs a full search over C:. If it finds one or more 'SAGA GIS' binaries it will take the first hit. You have to set ver_select = TRUE for an interactive selection of the preferred version. Additionally the selected SAGA paths are added to the environment and the global variables sagaPath, sagaModPath and sagaCmd will be created.

Value

A list containing the selected RSAGA path variables $sagaPath,$sagaModPath,$sagaCmd and potentially other installations $installed

Note

The 'SAGA GIS' wrapper RSAGA package was updated several times however it covers currently (May 2014) only 'SAGA GIS' versions from 2.3.1 LTS - 8.4.1 The fast evolution of 'SAGA GIS' makes it highly impracticable to keep the wrapper adaptions in line (currently 9.4). RSAGA will meet all linking needs perfectly if you use 'SAGA GIS' versions from 2.0.4 - 7.5.0.
However you must call rsaga.env using the rsaga.env(modules = saga$sagaModPath) assuming that saga contains the returnPaths of linkSAGA In addition the very promising Rsagacmd wrapper package is providing a new list oriented wrapping tool.

Examples

```
## Not run:

# call if you do not have any idea if and where SAGA GIS is installed
# it will return a list with the selected and available SAGA installations
```

```
# it prepares the system for running the selected SAGA version via RSAGA or CLI
linkSAGA()

# overriding the default environment of rsaga.env call

saga<-linkSAGA()
if (saga$exist) {
require(RSAGA)
RSAGA::rsaga.env(path = saga$installed$binDir[1],modules = saga$installed$moduleDir[1])
}

## End(Not run)
```

`loadEnvi` *Load data from rds format and associated yaml metadata file.*

Description

Load data from rds format and associated yaml metadata file.

Usage

```
loadEnvi(file_path)
```

Arguments

`file_path`	name and path of the rds file.

Value

list of 2 containing data and metadata.

Examples

```
## Not run:
a <- 1
meta <- list(a = 'a is a variable')
saveEnvi(a, file.path(tempdir(), 'test.rds'), meta)
b <- loadEnvi(file.path(tempdir(), 'test.rds'))

## End(Not run)
```

otb_api *Orfeo ToolBox (OTB) helpers: introspection and command construction*

Description

Public helpers to introspect OTB applications (by parsing '-help' output) and to build/modify command lists consumable by [runOTB()].

Calls the OTB application with '-help' and parses the 'Parameters:' block into a parameter table.

Normalizes the parsed parameter table (from [otb_capabilities()]) into a stable schema used for command building.

Convenience accessor based on [otb_args_spec()].

Like [otb_required()], but can ensure that a best-effort output key is included.

Convenience accessor based on [otb_args_spec()]. Returns a named list of optional parameters with 'NA_character_' placeholders or default values.

Creates a command list suitable for [runOTB()], with mandatory parameters always present as 'NA_character_' placeholders. Optional parameters can be omitted, filled with defaults, or included as 'NA_character_'.

Prints a compact overview (help line count, parameter count) and the full normalized parameter spec table.

Updates 'cmd[[key]]' to a normalized output path. If the parameter is pixel-typed ('[pixel]' in OTB help), the value is set as 'c(path, pixel_type)'.

Usage

```
otb_capabilities(algo, gili = NULL, include_param_help = FALSE)

otb_args_spec(algo, gili = NULL)

otb_required(algo, gili = NULL)

otb_required_with_output(algo, gili = NULL, enforce_output = TRUE)

otb_optional(algo, gili = NULL, with_defaults = TRUE)

otb_build_cmd(
  algo,
  gili = NULL,
  include_optional = c("none", "defaults", "all_na"),
  require_output = TRUE
)

otb_show(algo, gili = NULL)
```

```
otb_set_out(
  cmd,
  gili = NULL,
  key = "out",
  path,
  pixel_type = NULL,
  overwrite = TRUE,
  create_dir = TRUE
)
```

Arguments

algo	Character scalar. OTB application name.
gili	Optional list from [linkOTB()]. If 'NULL', [linkOTB()] is called.
include_param_help	Logical. If 'TRUE', additionally queries '-help <param>' for each parameter and returns these blocks as a named list.
enforce_output	Logical. If 'TRUE', attempts to add an output key from a small set of common output parameter names (e.g. '"out"', '"io.out"').
with_defaults	Logical. If 'TRUE', populate optional parameters with their default where available; otherwise use 'NA_character_'.
include_optional	One of '"none"', '"defaults"', '"all_na"'.
require_output	Logical. If 'TRUE', ensures that a best-effort output key placeholder exists if the application exposes one of the common output keys.
cmd	Command list as produced by [otb_build_cmd()].
key	Character scalar. Output parameter key to set (default '"out"').
path	Character scalar. Output file path.
pixel_type	Optional character scalar pixel type (e.g. '"float"'). Only used if the output parameter is pixel-typed; if 'NULL', uses the pixel default from spec or '"float"' as fallback.
overwrite	Logical. If 'FALSE', error if the file already exists.
create_dir	Logical. If 'TRUE', create the output directory if missing.

Details

The functions in this family are **non-invasive**: they do not mutate 'PATH' or global environment variables. They rely on a valid OTB descriptor as returned by [linkOTB()]. On Linux/macOS the implementation expects a working launcher ('gili$launcher') and uses an explicit environment map internally.

The common command representation is a list where: - 'cmd[[1]]' is the application name (character scalar), e.g. '"DimensionalityReduction"'. - subsequent named entries represent CLI parameters **without** a leading dash, e.g. 'cmd[["in"]]', 'cmd[["out"]]', 'cmd[["method"]]'. - values are character scalars, 'NA_character_' (placeholder), or for pixel-typed output parameters a character vector 'c(path, pixel_type)'.

This function performs basic checks on directory existence and overwrite policy.

Value

A list with components: - ‘text‘: character vector of help lines. - ‘params‘: data.frame parsed from the ‘Parameters:‘ block. - ‘param_help‘: ‘NULL‘ or named list of character vectors (per-parameter help).

A data.frame with (at least) the columns: ‘key‘, ‘type‘, ‘mandatory‘, ‘has_pixel‘, ‘pixel_default‘, ‘has_default‘, ‘default‘, ‘class‘, ‘desc‘.

Character vector of mandatory parameter keys.

Character vector of required parameter keys (including output if enforced).

Named list of optional parameters.

A command list with ‘cmd[[1]] == algo‘ and named entries for parameters.

Invisibly returns a list with components ‘caps‘ and ‘spec‘.

The modified command list.

Introspection

- [otb_capabilities()] returns the raw help text and a parsed parameter table. - [otb_args_spec()] normalizes the parsed table into a stable schema used by the helper functions below.

Command helpers

- [otb_required()], [otb_required_with_output()], [otb_optional()] - [otb_build_cmd()] creates a template command list using the spec. - [otb_set_out()] sets/validates an output parameter path (optionally pixel-typed). - [otb_show()] prints a compact overview for interactive use.

See Also

[linkOTB()], [runOTB()], [runOTB_isolated()]

`parseOTBAlgorithms` *Linux/macOS: lists plugin libs* `otbapp_*.so`, `otbapp_*.dylib`, `otbapp_*.dll` *Windows: lists wrappers* `otbcli_<Algo>.ps1`, `otbcli_<Algo>.bat`, `otbcli_<Algo>.exe`*# Retrieve available OTB applications*

Description

Linux/macOS: lists ‘otbapp_*.{so,dylib,dll}‘ under OTB_APPLICATION_PATH derived from OTB root.

Usage

```
parseOTBAlgorithms(gili = NULL)
```

Arguments

`gili`	Optional list returned by [linkOTB()]. If ‘NULL‘, [linkOTB()] is called.

Details

Windows: lists wrappers ‘otbcli_<Algo>.{ps1,bat,exe}‘ in ‘gili$pathOTB‘ (binDir). #’

Value

Character vector of application names.

`parseOTBFunction` *Retrieve the argument list from an OTB application*

Description

Legacy convenience wrapper that returns a list containing: - first element: algo name - named entries: parameter defaults (if any) and "mandatory" markers - ‘$help‘: per-parameter help text (if available)

Usage

```
parseOTBFunction(algo = NULL, gili = NULL)
```

Arguments

`algo`	Character. OTB application name (see [parseOTBAlgorithms()]).
`gili`	Optional list returned by [linkOTB()]. If ‘NULL‘, [linkOTB()] is called.

Details

Under the hood this uses the NEW introspection API: [otb_capabilities()] and [otb_args_spec()].

Value

List (legacy format).

`runOTB` *Run an OTB application (new workflow C)*

Description

Executes an Orfeo ToolBox application via the launcher/wrapper described by ‘gili‘ (typically returned by [linkOTB()]). This wrapper is non-invasive: it does not permanently modify PATH or the user environment.

Usage

```
runOTB(
  otbCmdList,
  gili = NULL,
  retRaster = TRUE,
  retCommand = FALSE,
  quiet = TRUE
)
```

Arguments

otbCmdList	List. OTB command list. The first element must be the algorithm name; remaining named elements are parameter keys/values.
gili	List. OTB installation descriptor as returned by [linkOTB()]. If 'NULL', [linkOTB()] is called.
retRaster	Logical. If 'TRUE', return a 'terra::SpatRaster' for the primary raster output (when detectable). If 'FALSE', return the output path(s) (character) or a status code depending on implementation.
retCommand	Logical. If 'TRUE', do not execute; return the exact CLI command string that would be run.
quiet	Logical. If 'TRUE', suppress console output from OTB (best-effort).

Details

The command is provided as a list in "link2GI style": - 'otbCmdList[[1]]' is the application name (e.g., '"DimensionalityReduction"') - named elements are OTB parameter keys (without leading '-')

Parameter values can be: - a scalar character/numeric (converted to character) - 'NA' / 'NA_character_' to omit the parameter - for pixel-typed outputs: a character vector of length 2 'c("<path>", "<pixel_type>")' (e.g. 'c("out.tif","float")')

Value

Depending on 'retCommand' / 'retRaster', returns either a command string, a 'terra::SpatRaster', or a character vector/status describing the produced output.

---

runOTB_isolated *Execute an OTB application in an isolated OTB environment (mainly Windows)*

Description

- Windows: dot-sources 'otbenv.ps1' (preferred) or calls 'otbenv.bat', then runs 'otbcli' within the same shell session. - Linux/macOS: delegates to [runOTB()] (launcher + explicit env already used).

Usage

```
runOTB_isolated(otbCmdList, gili = NULL, retCommand = FALSE, quiet = TRUE)
```

Arguments

otbCmdList	Non-empty list. ‘otbCmdList[[1]]‘ is the OTB application name. Named entries are parameter keys without leading dashes.
gili	Optional list from [linkOTB()]. If ‘NULL‘, [linkOTB()] is called.
retCommand	Logical. If ‘TRUE‘, returns the exact shell command that would be executed instead of running it.
quiet	Logical. If ‘TRUE‘, suppresses stdout/stderr (best-effort).

Value

If ‘retCommand=TRUE‘, a character scalar command line. Otherwise an invisible status code.

---

saveEnvi *Saves data in rds format and adds a yaml metadata file.*

Description

Saves data in rds format and saves metadata in a corresponding yaml file.

Usage

```
saveEnvi(variable, file_path, meta)
```

Arguments

variable	name of the data variable to be saved.
file_path	name and path of the rds file.
meta	name of the metadata list.

Examples

```
## Not run:
a <- 1
meta <- list(a = 'a is a variable')
saveEnvi(a, file.path(tempdir(), 'test.rds'), meta)

## End(Not run)
```

searchOTBW *Search for OTB installations on Windows (bounded, cmd-free)*

Description

Detects Orfeo Toolbox (OTB) installations on Windows using a bounded set of plausible roots (no full-disk crawl). Modern standalone bundles (OTB 9.x) are detected by the presence of:

- an environment script: `otbenv.ps1` (preferred) or `otbenv.bat`
- a launcher: `bin/otbApplicationLauncherCommandLine.exe`
- at least one CLI wrapper in `bin/`: `otbcli_*.ps1`, `otbcli_*.bat`, or `otbcli_*.exe`

Usage

```
searchOTBW(searchLocation = "C:/", DL = NULL, maxdepth = 8L, quiet = FALSE)
```

Arguments

`searchLocation`	Character scalar. Root directory to search (default `"C:/"`).
`DL`	Character scalar. Deprecated alias for `searchLocation`.
`maxdepth`	Integer. Best-effort maximum recursion depth for the recursive `list.files()` search (default 8).
`quiet`	Logical. If `TRUE`, suppress messages.

Details

Backward compatibility: older callers may pass `DL` instead of `searchLocation`. Internally, `DL` is treated as an alias for `searchLocation`.

Value

A `data.frame` with one row per detected installation and columns:

binDir Normalized path to `<root>/bin`.

baseDir Normalized OTB root directory.

otbCmd Path to a detected CLI wrapper (ps1/bat/exe).

envScript Path to `otbenv.ps1` or `otbenv.bat`.

launcher Path to `otbApplicationLauncherCommandLine.exe`.

installation_type Classification string (e.g., `"OTB_STANDALONE_PS1"`).

Examples

```
## Not run:
# bounded search under C:/
searchOTBW("C:/", quiet = FALSE)

# legacy alias
searchOTBW(DL = "C:/", quiet = FALSE)

## End(Not run)
```

`setupProj` *Setup project folder structure*

Description

Defines folder structures and creates them if necessary, loads libraries, and sets other project relevant parameters.

Usage

```
setupProj(
  root_folder = tempdir(),
  folders = c("data", "data/tmp"),
  code_subfolder = NULL,
  global = FALSE,
  libs = NULL,
  setup_script = "000_setup.R",
  fcts_folder = NULL,
  source_functions = !is.null(fcts_folder),
  standard_setup = NULL,
  create_folders = TRUE
)
```

Arguments

`root_folder`	root directory of the project.
`folders`	list of sub folders within the project directory.
`code_subfolder`	sub folders for scripts and functions within the project directory that will be created. The folders src, src/functions and src/config are recommended.
`global`	logical: export path strings as global variables?
`libs`	vector with the names of libraries
`setup_script`	Name of the installation script that contains all the settings required for the project, such as additional libraries, optional settings, colour schemes, etc. Important: It should not be used to control the runtime parameters of the scripts. This file is not read in automatically, even if it is located in the 'fcts_folder' folder.

fcts_folder	path of the folder holding the functions. All files in this folder will be sourced at project start.
source_functions	logical: should functions be sourced? Default is TRUE if fcts_folder exists.
standard_setup	select one of the predefined settings c('base', 'baseSpatial', 'advancedSpatial'). In this case, only the name of the base folder is required, but individual additional folders can be specified under 'folders' name of the git repository must be supplied to the function.
create_folders	default is TRUE so create folders if not existing already.

Value

A list containing the project settings.

Examples

```
## Not run:
setupProj(
  root_folder = '~/edu', folders = c('data/', 'data/tmp/'),
  libs = c('link2GI')
)

## End(Not run)
```

---

setup_default *Define working environment default settings*

Description

Define working environment default settings

Usage

```
setup_default(
  default = NULL,
  new_folder_list = NULL,
  new_folder_list_name = NULL
)
```

Arguments

default	name of default list
new_folder_list	containing a list of arbitrary folders to be generated
new_folder_list_name	name of this list

Details

After adding new project settings run [setup_default()] to update and savew the default settings. For compatibility reasons you may also run [lutUpdate()].

Value

A list containing the default project settings

Examples

```
## Not run:
# Standard setup for baseSpatial
setup_default()

## End(Not run)
```

---

sf2gvec *Write sf object directly to 'GRASS' vector utilising an existing or creating a new GRASS environment*

Description

Write sf object directly to 'GRASS' vector utilising an existing or creating a new GRASS environment

Usage

```
sf2gvec(x, epsg, obj_name, gisdbase, location, gisdbase_exist = FALSE)
```

Arguments

x	sf object corresponding to the settings of the corresponding GRASS container
epsg	numeric epsg code
obj_name	name of GRASS layer
gisdbase	GRASS gisDbase folder
location	GRASS location name containing obj_name)
gisdbase_exist	logical switch if the GRASS gisdbase folder exist default is TRUE

Note

have a look at the sf capabilities to write direct to sqlite

Author(s)

Chris Reudenbach

Examples

```
run = FALSE
if (run) {
## example
require(sf)
require(sp)
require(link2GI)
data(meuse)
meuse_sf = st_as_sf(meuse,
                    coords = c('x', 'y'),
                    crs = 28992,
                    agr = 'constant')


# write data to GRASS and create gisdbase
sf2gvec(x = meuse_sf,
        obj_name = 'meuse_R-G',
        gisdbase = '~/temp3/',
        location = 'project1')

# read from existing GRASS
gvec2sf(x = meuse_sf,
        obj_name = 'meuse_r_g',
        gisdbase = '~/temp3',
        location = 'project1')

}
```

Index