Package: link2GI (via r-universe)

October 29, 2024

Type Package

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

Version 0.6-2

Date 2024-10-28

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, xml2, xfun, rstudioapi, renv

SystemRequirements GNU make

RoxygenNote 7.3.1

Suggests knitr, rmarkdown, sp, rgrass, stars, curl, markdown

VignetteBuilder knitr

NeedsCompilation no

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

Repository CRAN

Date/Publication 2024-10-28 11:50:50 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. There is a big difference between osgeo4W and stand_alone installations. The function tries to find all valid installations by analyzing the calling batch scripts.

Usage

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

Arguments

searchLocation	Location to search for the grass executable, i.e. one executable for each GRASS installation on the system. For Windows systems it is mandatory to include an uppercase Windows drive letter and a colon. Default for Windows systems is C:/, for Linux systems the default is /usr/bin.
ver_select	boolean, Default is FALSE. If there is more than one 'GRASS GIS' installation and ver_select = TRUE, the user can interactively select the preferred 'GRASS GIS' version.
quiet	boolean, default is TRUE. switch to suppress console messages

Value

data frame with the 'GRASS GIS' binary folder(s) (i.e. where the individual individual GRASS commands are installed), version name(s) and installation type code(s)

Author(s)

Chris Reudenbach

Examples

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

## End(Not run)
```

findOTB *Search recursivly existing 'Orfeo Toolbox' installation(s) at a given drive/mountpoint*

Description

Provides an list of valid 'OTB' 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

```
findOTB(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 'OTB' root folder(s), and command line executable(s)

Author(s)

Chris Reudenbach

Examples

```
## Not run:
# find recursively all existing 'Orfeo Toolbox' installations folders starting
# at the default search location
findOTB()

## End(Not run)
```

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' API bindings*

Description

Initializes the session environment and the system paths for an easy access to 'GRASS GIS 7.x/8.x'. The correct setup of the spatial and projection parameters is automatically performed by using either an existing and valid `raster`, `terra`, `sp` or `sf` object, or manually by providing a list containing the minimum parameters needed.

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`	raster/terra or sf/sp object
`epsg`	manual epsg override
`default_GRASS`	default is `NULL` If is `NULL` an automatic search for all installed versions is performed. If you provide a valid list the corresponding version is initialized. An example for OSGeo4W64 is: `c('C:/OSGeo4W64','grass-7.0.5','osgeo4w')`
`search_path`	Path or mount point to search for.
`ver_select`	Boolean if TRUE you may choose interactively the binary version (if found more than one), by default FALSE

gisdbase_exist	default is FALSE if set to TRUE the arguments gisdbase and location are expected to be an existing GRASS gisdbase
gisdbase	default is NULL, invoke tempdir() to the 'GRASS' database. Alternatively you can provide a individual path.
use_home	default is FALSE, set the GISRC path to tempdir(), if TRUE the HOME or USERPROFILE setting is used for writing the GISRC file
location	default is NULL, invoke basename(tempfile()) for defining the 'GRASS' location. Alternatively you can provide a individual path.
spatial_params	default is NULL. Instead of a spatial object you may provide the geometry as a list. E.g. c(xmin,ymin,xmax,ymax,proj4_string)
resolution	resolution in map units for the GRASS raster cells
quiet	Boolean switch for suppressing 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 GRASS versions ae returned.

Note

GRASS GIS is excellently supported by the rgrass wrapper package. Nevertheless, 'GRASS GIS' is known for its its high demands on the correct spatial and reference setup and environment requirements. This becomes even worse on 'Windows platforms or when there are several alternative 'GRASS GIS' installations available. If you know how to use the rgrass package setup function rgrass::initGRASS works fine on Linux. This is also true for known configurations under the 'Windows' operating system. However, on university labs or corporate machines with limited privileges and/or different releases such as the 'OSGeo4W' distribution and the 'GRASS' stand-alone installation, or different software releases (e.g. 'GRASS 7.0.5 and GRASS 8.1.0), it often becomes inconvenient or even to get the correct links.
The function linkGRASS tries to find all valid 'GRASS GIS' binaries by #' analyzing the startup script files. GRASS GIS' startup script files. After identifying the 'GRASS GIS' binaries, all #' necessary system variables and settings are system variables and settings are generated and passed to a temporary R environment. The concept is simple, but helpful for everyday use. You need to either provide a raster or sp sf spatial object that has the correct spatial and projection properties, or you can link directly to an existing 'GRASS' gisdbase and mapset. If you choose a spatial object to initialize a correct 'GRASS' mapset, it will be used to create either a temporary or permanent mapset. rgrass environment with the correct 'GRASS' structure.

The most time consuming part on Windows systems is the search process. This can easily take 10 minutes or more. To speed up this process, you can also provide a correct parameter set. The best way to do this is to manually call searchGRASSW or for 'Linux' searchGRASSX. and call linkGRASS with the version arguments of your choice. linkGRASS will initialize the use of GRASS. If you have more than one valid installation and call linkGRASS() without arguments, you will be asked to select one.

Author(s)

Chris Reudenbach

Examples

```
run = FALSE
if (run) {
library(link2GI)
require(sf)

# get  data
nc = st_read(system.file('shape/nc.shp', package='sf'))
# Automatic linking of GRASS binaries using the nc data object for spatial referencing
# This is the best practice linking procedure for on-the-fly jobs.
# NOTE: If more than one GRASS installation is found, you will have to select one.
grass = linkGRASS(nc)

# Select the GRASS installation (if more than one)
linkGRASS(nc, ver_select = TRUE)

# Select the GRASS installation and define the search location
linkGRASS(nc, ver_select = TRUE, search_path = '~/')

# Set up GRASS manually with spatial parameters of the nc data
epsg = 28992
proj4_string <- sp::CRS(paste0('+init=epsg:',epsg))

linkGRASS(spatial_params = c(178605,329714,181390,333611,proj4_string@projargs),epsg=epsg)


# create some temporary project folders for a permanent gisdbase
root_folder = tempdir()
grass_path = link2GI::createFolder(root_folder = root_folder,
                         folders = c('project1/'))
if (grass$exist){
# CREATE and link to a permanent GRASS folder at 'root_folder', location named 'project1'
linkGRASS(nc, gisdbase = root_folder, location = 'project1')

# ONLY LINK to a permanent GRASS folder in 'root_folder', location named 'project1'
linkGRASS(gisdbase = root_folder, location = 'project1', gisdbase_exist = TRUE )

# Manual creation of a GRASS gisdbase with the spatial parameters of the NC data.
# additional use of a permanent directory 'root_folder' and the location 'nc_spatial_params'.
epsg = 4267
proj4_string = sp::CRS(paste0('+init=epsg:',epsg))
linkGRASS(gisdbase = root_folder,
           location = 'nc_spatial_params',
       spatial_params = c(-84.32385, 33.88199,-75.45698,36.58965,proj4_string),epsg = epsg)
}
}
```

---

linkOTB *Locate and set up 'Orfeo ToolBox' API bindings*

Description

Locate and set up 'Orfeo ToolBox' API bindings

Usage

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

Arguments

bin_OTB	string contains path to where the otb binaries are located
root_OTB	string provides the root folder of the bin_OTB
type_OTB	string
searchLocation	string hard drive letter (Windows) or mounting point (Linux) default for Windows is C:, default for Linux is ~
ver_select	Boolean, default is FALSE. If there is more than one 'OTB' installation and ver_select = TRUE the user can interactively select the preferred 'OTB' version, conversely if FALSE the latest version is automatically selected.
quiet	Boolean switch for suppressing messages default is TRUE
returnPaths	Boolean, if set to FALSE the paths of the selected version are written. in the PATH variable only, otherwise all paths and versions of the installed OTB versions are returned.

Details

It looks for the otb_cli.bat file. If the file is found in a bin folder it is assumed to be a valid 'OTB' binary installation.

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

Value

add otb paths to the environment and creates global variables path_OTB

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 OTB is installed
otb<-linkOTB()
if (otb$exist) {
# call it for a default OSGeo4W installation of the OTB
print(otb)
}

## End(Not run)
```

---

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)
```

---

parseOTBAlgorithms *Retrieve available OTB modules*

Description

Read in the selected OTB module folder and create a list of available functions.

Usage

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

Arguments

gili	optional list of available ‘OTB‘ installations, if not specified, ‘linkOTB()‘ is called to automatically try to find a valid OTB installation

Examples

```
## Not run:
## link to the OTB binaries
otblink<-link2GI::linkOTB()

 if (otblink$exist) {

 ## parse all modules
 moduleList<-parseOTBAlgorithms(gili = otblink)
```

```
 ## print the list
 print(moduleList)

 }

## End(Not run)
```

---

parseOTBFunction *Retrieve the argument list from a selected OTB function*

Description

retrieve the selected function and returns a full argument list with the default settings

Usage

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

Arguments

algo	either the number or the plain name of the 'OTB' algorithm that is wanted. Note the correct (of current/selected version) information is provided by 'parseOTBAlgorithms()'
gili	optional list of available 'OTB' installations, if not specified, 'linkOTB()' is called to automatically try to find a valid OTB installation

Examples

```
## Not run:
otblink<-link2GI::linkOTB()
if (otblink$exist) {

## parse all modules
algos<-parseOTBAlgorithms(gili = otblink)


## take edge detection
cmdList<-parseOTBFunction(algo = algos[27],gili = otblink)
## print the current command
print(cmdList)
}

## End(Not run)
##+##
```

runOTB *Execute the OTB command via system call*

Description

Wrapper function that inserts the OTB command list into a system call compatible string and executes that command.

Usage

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

Arguments

otbCmdList	the correctly populated OTB algorithm parameter list
gili	optional list of available 'OTB' installations, if not specified, 'linkOTB()' is called to automatically try to find a valid OTB installation
retRaster	boolean if TRUE a raster stack is returned default is FALSE
retCommand	boolean if TRUE only the OTB API command is returned default is FALSE
quiet	boolean if TRUE suppressing messages default is TRUE

Details

#' Please NOTE: You must check the help to identify the correct input file argument ($input_in or $input_il).

Examples

```
## Not run:
require(link2GI)
require(terra)
require(listviewer)

## link to OTB
otblink<-link2GI::linkOTB()

if (otblink$exist) {
 root_folder<-tempdir()
 fn <- system.file('ex/elev.tif', package = 'terra')

## for an image output example we use the Statistic Extraction,
```

```
algoKeyword<- 'LocalStatisticExtraction'

## extract the command list for the choosen algorithm
cmd<-parseOTBFunction(algo = algoKeyword, gili = otblink)

## Please NOTE:
## You must check the help to identify the correct argument codewort ($input_in or $input_il)
listviewer::jsonedit(cmd$help)

## define the mandatory arguments all other will be default
cmd$input_in  <- fn
cmd$out <- file.path(tempdir(),'test_otb_stat.tif')
cmd$radius <- 7

## run algorithm
retStack<-runOTB(cmd,gili = otblink)

## plot image
terra::plot(retStack)

## for a data output example we use the

algoKeyword<- 'ComputeImagesStatistics'

## extract the command list for the chosen algorithm
cmd<-parseOTBFunction(algo = algoKeyword, gili = otblink)

## get help using the convenient listviewer
listviewer::jsonedit(cmd$help)

## define the mandatory arguments all other will be default
cmd$input_il  <- file.path(tempdir(),'test.tif')
cmd$ram <- 4096
cmd$out.xml <- file.path(tempdir(),'test_otb_stat.xml')
cmd$progress <- 1

## run algorithm
ret <- runOTB(cmd,gili = otblink, quiet = F)

## as vector
print(ret)

## as xml
XML::xmlParse(cmd$out)

}

## End(Not run)
```

---

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)
```

---

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