R topics documented:

- findGDAL .................................................. 2
- findGRASS ................................................. 3
- findOTB ..................................................... 4
- findSAGA .................................................... 5
- gvec2sf ..................................................... 5
- initProj ..................................................... 7
- link2GI ....................................................... 8
- linkAll ....................................................... 8
- linkGDAL ..................................................... 10
- linkGRASS .................................................. 11
- linkGRASS7 ............................................... 14
- linkOTB ...................................................... 17
- linkSAGA ..................................................... 18
- paramGRASSw .............................................. 20
- paramGRASSx .............................................. 21
- parseOTBAlggorithms ................................. 22
- parseOTBFuntion ......................................... 23
- rgb ......................................................... 23
- runOTB ....................................................... 24
- setenvGDAL ............................................... 25
- setenvGRASSw ............................................ 26
- setenvOTB .................................................. 27
- sf2gvec ..................................................... 28

Index ......................................................... 30

---

**findGDAL**

*Search recursively 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 tries 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.
- `quiet` boolean switch for supressing console messages default is TRUE
**findGRASS**

Return attributes of valid ‘GRASS GIS’ installation(s) on the system

**Description**

Provides a list of valid ‘GRASS GIS’ installation(s) on your system. There is a major difference between osgeo4W and stand-alone installations. The functions tries to find all valid installations by analysing the calling batch scripts.

**Usage**

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

**Arguments**

- `searchLocation`: location to be searched 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 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 suppressing console messages default is TRUE

**Value**

A dataframe with the ‘GRASS GIS’ binary folder(s) (i.e. where the individual GRASS commands are installed), version name(s) and installation type code(s)

**Author(s)**

Chris Reudenbach
findOTB

### Description

Search recursively existing 'Orfeo Toolbox' installation(s) at a given drive/mountpoint

### Usage

```r
findOTB(searchLocation = "default", quiet = TRUE)
```

### Arguments

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

### Value

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

### Author(s)

Chris Reudenbach

### Examples

```r
## 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/mountpoint

Description

Provides a list of valid 'SAGA GIS' installation(s) on your 'Windows' system. There is a major difference between osgeo4W and stand-alone installations. The function tries to find all valid installations by analysing 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.
- 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

```r
## 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 7/8 environment an arbitrary vector dataset into a sf object

Description

Converts from an existing GRASS 7/8 environment an arbitrary vector dataset into a sf object
gvec2sf

Usage

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

Arguments

\(x\) \hspace{1cm} \text{sf object corresponding to the settings of the corresponding GRASS container}

\(\text{obj\_name}\) \hspace{1cm} \text{name of GRASS layer}

\(\text{gisdbase}\) \hspace{1cm} \text{GRASS gisDbase folder}

\(\text{location}\) \hspace{1cm} \text{GRASS location name containing \text{obj\_name})}

\(\text{gisdbase\_exist}\) \hspace{1cm} \text{logical switch if the GRASS gisdbase folder exist default is TRUE}

Note

have a look at the \text{sf} capabilities to read direct from sqlite

Author(s)

Chris Reudenbach

Examples

## Not run:
## example
# get meuse data as sf object
require(sf)
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")

## End(Not run)
initProj

Defines and creates folders and variables

Description

Defines and creates (if necessary) all folders variables. Returns a list with the project folder pathes. Optionally exports all pathes to a global sub environment.

Usage

```r
initProj(
  projRootDir = tempdir(),
  GRASSlocation = "tmp/",
  projFolders = c("data/", "result/", "run/", "log/")
  path_prefix = "",
  global = FALSE
)
```

Arguments

- `projRootDir` project github root directory (your github name)
- `GRASSlocation` folder for GRASS data
- `projFolders` list of subfolders in project
- `path_prefix` character a prefix for the path variables names default is ""
- `global` boolean export path strings as global variables default is false

Examples

```r
## Not run:
link2GI::initProj(projRootDir = tempdir(),
  projFolders = c("data/",
                 "data/level0/",
                 "data/level1/",
                 "output/",
                 "run/",
                 "fun/" )
)

## End(Not run)
```
Description

A straightforward helper tool for linking GI/RS functionality to R. The goal of the package is to correctly initialize both the existing wrapper packages rgrass and RSAGA and to smoothly enable the necessary system variables and path parameters for a direct access of the binaries via direct system calls on all operating systems. In particular, rgrass and RSAGA can cause severe problems during initialization of parallel installations of GRASS GIS or SAGA GIS under the Windows operating system(s). link2GI tries to set the correct system settings and returns if system calls are required the necessary paths and command strings.

Furthermore the package provides a linkage to the Orfeo Toolbox (OTB) software. Due to the difficulties linking the correct GDAL binaries a new system wide search for GDAL binaries is implemented.

Finally there are some useful functions for creating project folder structures and project environments. To deal with the not always consistent API-calls of OTB a list based command parser and generator is provided.

Details

Functions for linking GI/RS functionality to R

Note

To utilize the power of the open source GI tools from within R you need to install them first. The link2GI package just tries to generate correct environment settings as system and path variables for the most of the known issues. The installation of the QGIS, GRASS GIS 7.x and SAGA-GIS GIS software is described in the vignettes.

link2GI is tested under Windows 7/10 as well as on the Ubuntu/Debian/Arch Linux distributions. The OSX operation system should run but is not tested (Any help is highly appreciated).

Author(s)

Chris Reudenbach Tim Appelhans

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

Description

brute force search, find and linkl of all link2GI link functions. This is helpfull if your system is well setup and the standard linkage procedure will provide the correct linkages.
Usage

```r
linkAll(
    links = NULL,
    simple = TRUE,
    linkItems = c("saga", "grass7", "otb", "gdal"),
    sagaArgs = "default",
    grassArgs = "default",
    otbArgs = "default",
    gdalArgs = "default",
    quiet = FALSE
)
```

Arguments

- **links**: character. links
- **simple**: logical. true make all
- **linkItems**: character. list of c("saga", "grass7", "otb", "gdal")
- **sagaArgs**: character. full string of sagaArgs
- **grassArgs**: character. grassArgs full string of grassArgs
- **otbArgs**: character. full string of otbArgs
- **gdalArgs**: character. full string of gdalArgs
- **quiet**: supress all messages default is FALSE

Note

You may also use the full list of arguments that is made available from the link2GI package, but it is strongly recommended in this case to use directly the single linkage functions from link2GI.

Examples

```r
## Not run:
# required packages
require(uavRst)
require(link2GI)

# search, find and create the links to all supported GI software
giLinks<-uavRst::linkAll()

# makes the GDAL linkage verbose
giLinks<-uavRst::linkAll(gdalArgs= "quiet = TRUE")
```

## End(Not run)
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 supressing messages default is TRUE
- `returnPaths` boolean if set to FALSE the pathes 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 harddrive 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 pathes to the enviroment and creates global variables path_GDAL

Note

You may also set the path manually. Using a 'OSGeo4W64' [https://trac.osgeo.org/osgeo4w/](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, sp or sf object, or manually by providing a list containing the minimum parameters needed.

Usage

linkGRASS(
  x = 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 or sp object
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 mounting point that will be searched

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 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 GRASS versions are returned.

Details

The concept is straightforward but for an all days usage helpful. Either you need to provide a raster or sp sf spatial object which has correct spatial and projection properties or you may link directly to an existing 'GRASS' gisdbase and mapset. If you choose an spatial object to initialize a correct 'GRASS' mapset it is used to create either a temporary or a permanent rgrass environment including the correct 'GRASS 7/8' structure.

The most time consuming part on 'Windows' Systems is the search process. This can easily take 10 or more minutes. To speed up this process you can also provide a correct parameter set. Best way to do so is to call searchGRASSW for 'Linux' searchGRASSX manually. and call linkGRASS with the version arguments of your choice. linkGRASS initializes the usage of GRASS7.

Note

'GRASS GIS' is excellently supported by the rgrass wrapper package. Nevertheless 'GRASS GIS' is well known for its high demands regarding the correct spatial and reference setup of workspace and environment requirements. This becomes even worse on 'Windows' platforms or if several alternative 'GRASS GIS' installations are available. If one knows what to do the rgrass package setup function rgrass::initGRASS works fine under Linux. This is also valid for well known configurations under the 'Windows' operation system. Nevertheless on university lab or on company computers with restricted privileges and/or using different releases like the 'OSGeo4W' distribution and the 'GRASS 7/8' stand-alone installation, or different software releases (e.g. 'GRASS 7.0.5 and GRASS 8.1.0), it becomes often cumbersome or even impossible to get the correct linkages.
The function `linkGRASS` tries to find all valid 'GRASS GIS' binaries by analyzing the startup script files of 'GRASS GIS'. After identifying the 'GRASS GIS' binaries all necessary system variables and settings will be generated and passed to a temporary R environment.

If you have more than one valid installation and run `linkGRASS()` without arguments, you will be asked to select one.

**Author(s)**

Chris Reudenbach

**Examples**

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

# proj folders
projRootDir<-tempdir()
paths<-link2GI::initProj(projRootDir = projRootDir, projFolders = c("project1/"))

# get data
nc <- st_read(system.file("shape/nc.shp", package="sf"))

# Automatic search and find of GRASS binaries
# using the nc sf data object for spatial referencing
# This is the highly recommended linking procedure for on the fly jobs
# NOTE: if more than one GRASS installation is found you have to choose.
grass<-linkGRASS(nc,returnPaths = TRUE)
if (grass$exist){

# CREATE and link to a permanent GRASS folder at "projRootDir", location named "project1"
linkGRASS(nc, gisdbase = projRootDir, location = "project1")

# ONLY LINK to a permanent GRASS folder at "projRootDir", location named "project1"
linkGRASS(gisdbase = projRootDir, location = "project1", gisdbase_exist = TRUE )

# setting up GRASS manually with spatial parameters of the nc data
proj4_string <- as.character(sp::CRS("+init=epsg:28992"))
linkGRASS(spatial_params = c(178605,329714,181390,333611,proj4_string))

# creating a GRASS gisdbase manually with spatial parameters of the nc data
# additionally using a peramanent directory "projRootDir" and the location "nc_spatial_params"
proj4_string <- as.character(sp::CRS("+init=epsg:4267"))
linkGRASS(gisdbase = projRootDir,
  location = "nc_spatial_params",
  spatial_params = c(-84.32385, 33.88199,-75.45698,36.58965,proj4_string))
}
```

## Some more examples related to interactive selection or OS specific settings
## linkGRASS7

### 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, `sp` or `sf` object, or manually by providing a list containing the minimum parameters needed.

### Usage

```r
linkGRASS7(
  x = 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 or sp object
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 mounting point that will be searched.

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 an 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 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 GRASS versions are returned.

Details

The concept is straightforward but for an all days usage helpful. Either you need to provide a raster or sp sf spatial object which has correct spatial and projection properties or you may link directly to an existing 'GRASS' gisdbase and mapset. If you choose an spatial object to initialize a correct 'GRASS' mapset it is used to create either a temporary or a permanent rgrass environment including the correct 'GRASS 7/8' structure.

The most time consuming part on 'Windows' Systems is the search process. This can easily take 10 or more minutes. To speed up this process you can also provide a correct parameter set. Best way to do so is to call searchGRASSW or for 'Linux' searchGRASSX manually. and call linkGRASS with the version arguments of your choice. linkGRASS initializes the usage of GRASS7/8.

Note

'GRASS GIS 7/8' is excellently supported by the rgrass wrapper package. Nevertheless 'GRASS GIS' is well known for its high demands regarding the correct spatial and reference setup of workspace and environment requirements. This becomes even worse on 'Windows' platforms or if several alternative 'GRASS GIS' installations are available. If one knows what to do the rgrass package setup function rgrass::initGRASS works fine under Linux. This is also valid for well known configurations under the 'Windows' operation system. Nevertheless on university lab or on company computers with restriced privileges and/or using different releases like the 'OSGeo4W' distribution and the 'GRASS 7/8' stand-alone installation, or different software releases (e.g. 'GRASS 7.0.5 and GRASS 8.1.0), it becomes often cumbersome or even impossible to get the
correct linkages.
The function linkGRASS tries to find all valid 'GRASS GIS' binaries by analyzing the startup script
files of 'GRASS GIS'. After identifying the 'GRASS GIS' binaries all necessary system variables
and settings will be generated and passed to a temporary R environment.

If you have more than one valid installation and run linkGRASS() without arguments, you will be
ask to select one.

Author(s)
Chris Reudenbach

Examples

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

# proj folders
projRootDir<-tempdir()
paths<-'link2GI::initProj(projRootDir = projRootDir,
                           projFolders = c("project1/"))

# get data
nc <- st_read(system.file("shape/nc.shp", package="sf"))

# Automatic search and find of GRASS binaries
# using the nc sf data object for spatial referencing
# This is the highly recommended linking procedure for on the fly jobs
# NOTE: if more than one GRASS installation is found you have to choose.
grass<-linkGRASS(nc,returnPaths = TRUE)
if (grass$exist){

  # CREATE and link to a permanent GRASS folder at "projRootDir", location named "project1"
  linkGRASS(nc, gisdbase = projRootDir, location = "project1")

  # ONLY LINK to a permanent GRASS folder at "projRootDir", location named "project1"
  linkGRASS(gisdbase = projRootDir, location = "project1", gisdbase_exist = TRUE)

  # setting up GRASS manually with spatial parameters of the nc data
  proj4_string <- as.character(sp::CRS("+init=epsg:28992"))
  linkGRASS(spatial_params = c(178605,329714,181390,333611,proj4_string))

  # creating a GRASS gisdbase manually with spatial parameters of the nc data
  # additionally using a peramanent directory "projRootDir" and the location "nc_spatial_params"
  proj4_string <- as.character(sp::CRS("+init=epsg:4267"))
  linkGRASS(gisdbase = projRootDir,
            location = "nc_spatial_params",
            spatial_params = c(-84.32385, 33.88199,-75.45698,36.58965,proj4_string))
}
## Some more examples related to interactive selection or OS specific settings

# SELECT the GRASS installation and define the search location
linkGRASS(nc, ver_select = TRUE, search_path = "~")

# SELECT the GRASS installation
linkGRASS(nc, ver_select = TRUE)

# Typical osge4W installation (QGIS), using the meuse sp data object for spatial referencing
linkGRASS(nc,c("C:/Program Files/QGIS 2.18","grass-7.2.1","osgeo4W"))

# Typical osgeo4W installation (rootdir), using the meuse sp data object for spatial referencing
linkGRASS(nc,c("C:/OSGeo4W64/","grass-7.2.2","osgeo4W"))

## End(Not run)

---

### Description

Locate and set up 'Orfeo ToolBox' API bindings

### Usage

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

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_OTB</td>
<td>string contains path to where the otb binaries are located</td>
</tr>
<tr>
<td>root_OTB</td>
<td>string provides the root folder of the bin_OTB</td>
</tr>
<tr>
<td>type_OTB</td>
<td>string</td>
</tr>
<tr>
<td>searchLocation</td>
<td>string hard drive letter (Windows) or mounting point (Linux) default for Windows is C:, default for Linux is ~</td>
</tr>
<tr>
<td>ver_select</td>
<td>boolean default is FALSE. If there is more than one 'OTB' installation and ver_select = TRUE the user can select interactively the preferred 'OTB' version In opposite if FALSE the newest version is automatically choosen.</td>
</tr>
<tr>
<td>quiet</td>
<td>boolean switch for supressing messages default is TRUE</td>
</tr>
</tbody>
</table>
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 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 link0TB() it performs a full search over the hardrive 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<-link0TB()
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

```r
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.
- `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 are returned.

Value

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

Note

The excellent 'SAGA GIS' wrapper RSAGA package was updated several times however it covers currently (Dec 2019) only 'SAGA GIS' versions from 2.3.1 - 6.3.0 The fast evolution of 'SAGA GIS' makes it highly impracticable to keep the wrapper adaptions in line (currently 7.5). 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 most recently 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)

---

`paramGRASSw`  
*Usually for internally usage get 'GRASS GIS' and rgrass parameters on ‘Windows’ OS*

### Description

Initialize the environment variables on a ‘Windows’ OS for using ‘GRASS GIS’ via `rgrass`

### Usage

```r
paramGRASSw(
  set_default_GRASS = NULL,
  DL = "C:",
  ver_select = FALSE,
  quiet = TRUE
)
```

### Arguments

- **set_default_GRASS**  
  default = NULL forces a full search for ‘GRASS GIS’ binaries. You may alternatively provide a vector containing paths and keywords. `c("C:/OSGeo4W64","grass-7.0.5","osgeo4w")` is valid for a typical osgeo4w installation.

- **DL**  
  raster or sp object

- **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 suppressing console messages default is TRUE

### Details

The concept is very straightforward but for an all days usage pretty helpful. You need to provide a `raster` or a `sp` object. The derived properties are used to initialize a temporary but static `rgrass` environment. During the session you will have full access to GRASS7 both via the wrapper package as well as the command line. `paramGRASSw` initializes the usage of GRASS7.
Examples

## Not run:

# automatic retrieval of valid 'GRASS GIS' environment settings
# if more than one is found the user has to choose.
paramGRASSw()

# typical OSGeo4W64 installation
paramGRASSw(c("C:/OSGeo4W64","grass-7.0.5","osgeo4W"))

## End(Not run)

---

**paramGRASSx**  
*Usually for internally usage, get 'GRASS GIS' and rgrass parameters on 'Linux' OS*

Description

Initialize and set up rgrass for 'Linux'

Usage

```r
paramGRASSx(
  set_default_GRASS = NULL,
  MP = "/usr/bin",
  ver_select = FALSE,
  quiet = TRUE
)
```

Arguments

- **set_default_GRASS**
  - default = NULL will force a search for 'GRASS GIS' You may provide a valid combination as c("/usr/lib/grass74","7.4.1","grass74")
- **MP**
  - mount point to be searched. default is "/usr/bin"
- **ver_select**
  - if TRUE you must interactively select between alternative installations
- **quiet**
  - boolean switch for suppressing console messages default is TRUE

Details

During the session you will have full access to GRASS7 GIS via the rgrass wrapper. Additionally you may use also use the API calls of GRASS7 via the command line.
parseOTBAlgorithms

Examples

## Not run:
# automatic retrieval of the GRASS7 enviroment settings
paramGRASSx()

# typical stand_alone installation
paramGRASSx("/usr/bin/grass72")

# typical user defined installation (compiled sources)
paramGRASSx("/usr/local/bin/grass72")

## End(Not run)

parseOTBAlgorithms  Get OTB modules

Description

retrieve the OTB module folder content and parses the module names

Usage

parseOTBAlgorithms(gili = NULL)

Arguments

gili  optional list of avaliable ‘OTB’ binaries if not provided ‘linkOTB()’ is called

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

*Get OTB function argument list*

**Description**

retrieves chosen function and returns a full argument list with the default settings.

**Usage**

```r
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/choosen version) information is provided by 'parseOTBAlgorithms()'.

- **gili**
  - optional list of available ‘OTB’ binaries if not provided 'linkOTB()' is called.

**Examples**

```r
## 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)
##
```

### rgb

*RGB ortho-image from an arbitrary Marburg Open Forest (MOF) plot*

**Description**

Example data set containing a RGB ortho-image of a small plot sampled in the Marburg University Forest aka Marburg Open Forest (MOF). The resolution is 10 cm, projection ETRS89 UTM32.

**Format**

"raster::raster"
runOTB

Execute the OTB command list via system call

Description

Wrapper function which paste the OTB command list into a system call compatible string and execute this command.

Usage

runOTB(otbCmdList = NULL, gili = NULL, retRaster = TRUE, quiet = TRUE)

Arguments

otbCmdList the OTB algorithm parameter list
gili optional gis linkage as done by 'linkOTB()'
retRaster boolean if TRUE a raster stack is returned
quiet boolean switch for suppressing messages default is TRUE

Details

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

Examples

## Not run:
require(link2GI)
require(raster)
require(listviewer)
rgdal::set_thin_PROJ6_warnings(TRUE)

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

if (otblink$exist) {
  projRootDir<-tempdir()
data('rgb', package = 'link2GI')
raster::plotRGB(rgb)
r<-raster::writeRaster(rgb,
  filename=file.path(projRootDir,"test.tif"),
  format="GTiff",
  overwrite=TRUE)

  ## for an image output example we use the Statistic Extraction,
  algoKeyword<- "LocalStatisticExtraction"

  ## extract the command list for the choosen algorithm

setenvGDAL

Usually for internally usage, initializes and set up access to the 'GDAL' command line interface
Description

Initializes and set up access to the 'GDAL' command line interface

Usage

setenvGDAL(bin_GDAL = NULL)

Arguments

bin_GDAL string contains the path to the 'GDAL' binaries

Value

Adds 'GDAL' pathes to the enviroment and creates the variable global string variable gdalCmd, that contains the path to the 'GDAL' binaries.

Examples

## Not run:
## example for the most common default OSGeo4W64 installation of GDAL
setenvGDAL(bin_GDAL = "C:/OSGeo4W64/bin/",
root_GDAL = "C:/OSGeo4W64")

## End(Not run)

---

Description

Usually for internally usage, create valid ‘GRASS GIS 7.xx’ rsession environment settings according to the selected GRASS GIS 7.x and Windows Version

Usage

setenvGRASSw(
  root_GRASS = NULL,
  grass_version = NULL,
  installation_type = NULL,
  jpgmem = 1e+06,
  quiet = TRUE
)

## Not run:
## example for the most common default OSGeo4W64 installation of GRASS
setenvGRASSw(root_GRASS = "C:/OSGeo4W64/GRASS7/",
  grass_version = "7.8.0",
  installation_type = "C:\\Program Files\\OSGeo4W64\\GRASS7\\grass Visio\",
  jpgmem = 1e+06,
  quiet = TRUE)

## End(Not run)
setenvOTB

Arguments

- **root_GRASS**  
  Grass root directory i.e. "C:\OSGEO4~1",

- **grass_version**  
  Grass version name i.e. "grass-7.0.5"

- **installation_type**  
  Two options "osgeo4w" as installed by the 'OSGeo4W'-installer and "NSIS" that is typical for a stand-alone installation of 'GRASS GIS'.

- **jpgmem**  
  Jpeg2000 memory allocation size. Default is 1000000

- **quiet**  
  Boolean switch for supressing console messages default is TRUE

Author(s)

Chris Reudenbach

Examples

```r
## Not run:
# set chosen 'GRASS GIS' installation folders
setenvGRASSw(root_GRASS = "C:\\PROGRA~1\\QGIS2~1.18",
            grass_version = "grass-7.2.1",
            installation_type = "osgeo4W")
## End(Not run)
```

Description

Usually for internally usage, initializes and set up access to the 'OTB' command line interface

Usage

```r
setenvOTB(bin_OTB = NULL, root_OTB = NULL)
```

Arguments

- **bin_OTB**  
  String contains the path to the 'OTB' binaries

- **root_OTB**  
  String contains the full string to the root folder containing the 'OTB' installation

Value

Adds 'OTB' paths to the environment and creates the variable global string variable otbCmd, that contains the path to the 'OTB' binaries.
Examples

## Not run:
## example for the most common default OSGeo4W64 installation of OTB
setenvOTB(bin_OTB = "C:\\OSGeo4W64\\bin\\",
root_OTB = "C:\\OSGeo4W64")

## End(Not run)

---

sf2gvec Write sf object to GRASS 7/8 vector utilising an existing or creating a new GRASS environment

Description

Write sf object to GRASS 7/8 vector utilising an existing or creating a new GRASS environment

Usage

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

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 write direct to sqlite

Author(s)

Chris Reudenbach

Examples

## Not run:
## example
# get meuse data as sf object
require(sf)
nc <- st_read(system.file("shape/nc.shp", package="sf"))

# write data to GRASS and create gisdbase
sf2gvec(x = nc,
       obj_name = "nc_R-G",
sf2gvec

```r
gisdbase = "~/temp3",
location = "project1")

# read from existing GRASS
gvec2sf(x = nc_R-G,
obj_name = "nc_R-G",
gisdbase = "~/temp3",
location = "project1")

## End(Not run)
```

## End(Not run)
Index

findGDAL, 2
findGRASS, 3
findOTB, 4
findSAGA, 5

gvec2sf, 5
initProj, 7
link2GI, 8
linkAll, 8
linkGDAL, 10
linkGRASS, 11
linkGRASS7, 14
linkOTB, 17
linkSAGA, 18

paramGRASSw, 20
paramGRASSx, 21
parseOTBAlgortihms, 22
parseOTBFunction, 23

rgb, 23
runOTB, 24

setenvGDAL, 25
setenvGRASSw, 26
setenvOTB, 27
sf, 6
sf2gvec, 28