Package 'LPJmL.RUtil'

March 23, 2015

Type Package

Title What the package does (short line)

Version 1.0

Date 2015-03-23

Author Who wrote it

Maintainer Who to complain to <yourfault@somewhere.net>

Description More about what it does (maybe more than one line)

License What license is it under?

R topics documented:

LPJmL.RUtil-package *LPJmL Utilities*

Description

reading, writting, visualising and data manipulating utility functions related with LPJmL.

Package:	LPJmL.RUtil
Type:	Package
Version:	1.0
Date:	2015-03-23
License:	What license is it under?

~~ An overview of how to use the package, including the most important ~~ ~~ functions ~~

S.SHI

Maintainer: Who to complain to <yourfault@somewhere.net> S.SHI

~~ Literature or other references for background information ~~

~~ Optionally other standard keywords, one per line, from file KEYWORDS in ~~ ~~ the R documentation directory ~~ package

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

~~ simple examples of the most important functions ~~

deg2area *convert degree to km^2*

Usage

```
deg2area(lat, res = 0.5)
```

Arguments

`lat`

`res`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lat, res = 0.5)
{
    deg2rad <- function(deg) {
        return(deg * pi * 0.00555555555555)
    }
```

```
  area <- (111000 * res) * (111000 * res) * cos(deg2rad(lat))/10000
  return(area)
}
```

HEADER_SIZE *header size*

Usage

```
data("HEADER_SIZE")
```

Format

The format is: num 43 the header size of LPJmL input.

Examples

```
data(HEADER_SIZE)
## maybe str(HEADER_SIZE) ; plot(HEADER_SIZE) ...
```

Lindex *function to do ...*

Usage

```
Lindex(NPIX, NBANDS, year, pix, band)
```

Arguments

NPIX

NBANDS

year

pix

band

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (NPIX, NBANDS, year, pix, band)
{
    temp <- (year - 1) * NPIX * NBANDS + (pix - 1) * NBANDS +
        band
    return(temp)
  }
```

map.build *build map*

Description

convert vectors to matrix for display. read.input.grid() has to ben run before.

Usage

```
map.build(raw_)
```

Arguments

```
raw_
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (raw_)
{
    map <- array(NA, dim = c(NR, NC))
    for (i in 1:length(raw_)) map[ind_lon[i], ind_lat[i]] <- raw_[i]
    return(map)
  }
```

map.create *map create*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
map.create(data.raw, startyear, endyear)
```

Arguments

```
data.raw

startyear

endyear
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (data.raw, startyear, endyear)
{
    npixel.out <- get.output.info(path.out)[1]
    simyears <- get.output.info(path.out)[2]
    startyear <- startyear - simstartyear + 1
    endyear <- endyear - simstartyear + 1
    if (startyear < 0) {
      print("map.create: the chosen start year should be later than simulation start year.")
        stop()
    }
    if (length(dim(data.raw)) == 2) {
        map.data <- array(NA, dim = c(NCOLS, NROWS, simyears))
        for (y in startyear:endyear) {
            for (i in 1:npixel.out) map.data[ind_lon[i], ind_lat[i],
                y] <- data.raw[i, y]
        }
    }
    if (length(dim(data.raw)) == 3) {
        map.data <- array(NA, dim = c(NCOLS, NROWS, 12, simyears))
        for (y in startyear:endyear) {
            for (m in 1:12) {
                for (i in 1:npixel.out) map.data[ind_lon[i],
                  ind_lat[i], m, y] <- data.raw[i, m, y]
            }
        }
    }
    map.data
  }
```

`map.interact` *map interact*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
map.interact(map.data, startyear, endyear, colour, data.info)
```

Arguments

`map.data`

`startyear`

`endyear`

`colour`

`data.info`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (map.data, startyear, endyear, colour, data.info)
{
    map.itc.data <<- array(NA, dim = dim(map.data))
    map.itc.data <<- map.data
    startyear <- startyear - simstartyear + 1
    endyear <- endyear - simstartyear + 1
    switch.year <<- startyear
    switch.month <<- 1
    yhscale <- 5.5
    Myvscale <- 5.5
    CopyToClip <- function() {
        png(file = paste(as.character(switch.year + simstartyear -
            1), "-", as.character(switch.month)), bg = "white")
        map.show(map.itc.data, switch.year, switch.month, data.info = data.info)
        dev.off()
    }
    NewWin <- function() {
        X11(type = "Xlib")
        map.show(map.itc.data, switch.year, switch.month, data.info = data.info)
    }
    NewSwitchWin <- function() {
        X11(type = "Xlib")
        map.switch(map.itc.data, startyear, endyear, data.info = data.info)
        getGraphicsEvent()
    }
    tt <- tktoplevel()
    tkwm.title(tt, "LPJmL Data")
    copy.but <- tkbutton(tt, text = "Copy to Clipboard", command = CopyToClip)
    newwins.but <- tkbutton(tt, text = "New Window(Static)",
        command = NewWin)
    newwind.but <- tkbutton(tt, text = "New Window(Dynamic)",
        command = NewSwitchWin)
    monorun.but <- tkbutton(tt, text = "Mono Pixel run", command = monop.graph)
    l.general <- tklabel(tt, text = "You may choose the functions here")
    l.ym <- tklabel(tt, text = "Year/Month")
    e.year <- tkentry(tt, width = 5)
    e.month <- tkentry(tt, width = 3)
    tkgrid(l.general)
    tkgrid(copy.but, sticky = "w")
    tkgrid(newwins.but, sticky = "w")
    tkgrid(newwind.but, sticky = "w")
    tkgrid(monorun.but, sticky = "w")
    tkgrid(l.ym, e.year, e.month)
    cat("for dynamic windows, press 'q' to quit")
    getGraphicsEvent()
  }
```

---

map.show *unction to do*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
map.show(map.data, y, m, data.info)
```

Arguments

```
map.data
y
m
data.info
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (map.data, y, m, data.info)
{
    yg.palette <- colorRampPalette(c("yellow3", "yellow", "greenyellow",
        "green", "green3", "darkgreen"))
    colo = yg.palette(101)
    colo.cm = cm.colors(32)
    tit <- paste(data.info$des, "(", data.info$name, ")")
    if (length(dim(map.data)) == 4) {
        ylab <- paste(paste(as.character(y + simstartyear - 1),
            "-", as.character(m)))
        image(x = seq(west, east, len = ((east - west + 0.5) *
            2)), y = seq(south, north, len = ((north - south +
            0.5) * 2)), map.data[, , m, y], col = colo, xlab = "",
            ylab = ylab, axes = T)
        map(add = T, boundary = T)
        title(main = tit)
        image.plot(x = seq(west, east, len = ((east - west +
            0.5) * 2)), y = seq(south, north, len = ((north -
            south + 0.5) * 2)), map.data[, , m, y], col = colo,
            legend.only = T, horizontal = T, legend.args = list(text = data.info$unit,
                col = "black", cex = 1, side = 1, line = 0),
            legend.width = 0.3)
    }
    if (length(dim(map.data)) == 3) {
        xlab <- as.character(y + simstartyear - 1)
        image(x = seq(west, east, len = ((east - west + 0.5) *
            2)), y = seq(south, north, len = ((north - south +
            0.5) * 2)), map.data[, , y], col = colo, xlab = xlab,
            ylab = "", axes = T)
        map(add = T, boundary = T)
        title(main = tit)
        image.plot(x = seq(west, east, len = ((east - west +
            0.5) * 2)), y = seq(south, north, len = ((north -
            south + 0.5) * 2)), map.data[, , y], col = colo,
```

```
            legend.only = T, horizontal = T, legend.args = list(text = data.info$unit,
                col = "black", cex = 1, side = 1, line = 0),
            legend.width = 0.3)
    }
  }
```

---

map.switch *function to do*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
map.switch(map.data, startyear, endyear, xlim = NULL, ylim = NULL, xaxs = "r", yaxs = "r", data.inf
```

Arguments

map.data
startyear
endyear
xlim
ylim
xaxs
yaxs
data.info

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (map.data, startyear, endyear, xlim = NULL, ylim = NULL,
    xaxs = "r", yaxs = "r", data.info)
{
    map.show(map.data, switch.year, switch.month, data.info = data.info)
    devset <- function() if (dev.cur() != eventEnv$which)
        dev.set(eventEnv$which)
    keydown <- function(key) {
        if (key == "q")
            return(invisible(1))
        eventEnv$onMouseMove <- NULL
        if (key == "6") {
            if (length(dim(map.data)) == 4) {
                if (switch.year == endyear && switch.month ==
                  12) {
                  switch.year <<- startyear
                  switch.month <<- 1
```

```
                }
                else {
                  if (switch.month == 12) {
                    switch.year <<- switch.year + 1
                    switch.month <<- 1
                  }
                  else switch.month <<- switch.month + 1
                }
            }
            if (length(dim(map.data)) == 3) {
                if (switch.year == endyear)
                  switch.year <<- startyear
                else switch.year <<- switch.year + 1
            }
            map.show(map.data, switch.year, switch.month, data.info)
            Sys.sleep(0.1)
        }
        if (key == "4") {
            if (length(dim(map.data)) == 4) {
                if (switch.year == startyear && switch.month ==
                  1) {
                  switch.year <<- endyear
                  switch.month <<- 12
                }
                else {
                  if (switch.month == 1) {
                    switch.year <<- switch.year - 1
                    switch.month <<- 12
                  }
                  else switch.month <<- switch.month - 1
                }
            }
            if (length(dim(map.data)) == 3) {
                if (switch.year == startyear)
                  switch.year <<- endyear
                else switch.year <<- switch.year - 1
            }
            map.show(map.data, switch.year, switch.month, data.info)
            Sys.sleep(0.1)
            NULL
        }
    }
    setGraphicsEventHandlers(prompt = "hit q to quit", onKeybd = keydown)
    eventEnv <- getGraphicsEventEnv()
}
```

`path.in` *data name/kind ...*

Usage

```
data("path.in")
```

Format

The format is: chr "../../NelasInputs/OUTPUT/"

Examples

```
data(path.in)
## maybe str(path.in) ; plot(path.in) ...
```

`read.cow` *read country file*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.cow(path)
```

Arguments

```
path
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path)
{
    npixel.out <- get.output.info(path)[1]
    country.fn.out <- file(paste(path, "country.bin", sep = ""),
        "rb")
    country.data <<- readBin(country.fn.out, integer(), n = npixel.out,
        size = 2)
    close(country.fn.out)
  }
```

`read.daily.output` *read daily output*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.daily.output(path)
```

Arguments

```
path
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path)
{
    nyear <- get.output.daily.info(paste(path, daily.files.list[1],
        ".bin", sep = ""))
    temp <- array(NA, dim = nyear * 365)
    daily.data.frame <<- data.frame(d_cleaf = temp, d_cpool = "",
        d_croot = "", d_cso = "", d_daylength = "", d_evap = "",
        d_fhiopt = "", d_fphu = "", d_froot = "", d_gpp = "",
        d_gresp = "", d_growingday = "", d_hi = "", d_himind = "",
        d_husum = "", d_irrig = "", d_lai = "", d_laimax_adjusted = "",
        d_laimaxnppdeficit = "", d_npp = "", d_par = "", d_perc = "",
        d_pet = "", d_phen = "", d_phu = "", d_prec = "", d_pvd = "",
        d_rd = "", d_rpool = "", d_rroot = "", d_rso = "", d_sun = "",
        d_temp = "", d_trans = "", d_vdsum = "", d_w0 = "", d_w1 = "",
        d_wdf = "", d_wevap = "", d_wscal = "")
    for (i in 1:length(daily.files.list)) {
        daily.fn <- file(paste(path, daily.files.list[i], ".bin",
            sep = ""), "rb")
        nyear <- get.output.daily.info(paste(path, daily.files.list[i],
            ".bin", sep = ""))
        cat("Reading", daily.files.list[i], "...")
        temp <- readBin(daily.fn, double(), 365 * nyear, size = sizeof.data)
        daily.data.frame[, daily.files.list[i]] <- temp
        cat("done\n")
        closeAllConnections()
    }
    return(daily.data.frame)
  }
```

`read.grid` *read grid*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.grid(path)
```

Arguments

`path`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path)
{
    pixel_year <- get.output.info(path)
    npixel.out <- pixel_year[1]
    grid.fn.out <- file(paste(path, "grid.bin", sep = ""), "rb")
    grid.data <<- readBin(grid.fn.out, integer(), n = 2 * npixel.out,
        size = 2)/100
    lon <<- grid.data[c(1:npixel.out) * 2 - 1]
    lat <<- grid.data[c(1:npixel.out) * 2]
    ind_lon <<- as.integer((grid.data[c(1:npixel.out) * 2 - 1] -
        bound_west)/res + 1.01)
    ind_lat <<- as.integer((grid.data[c(1:npixel.out) * 2] -
        bound_south)/res + 1.01)
    close(grid.fn.out)
  }
```

---

read.harvest.data *read harvest data*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.harvest.data(path, harvest.name.seclect)
```

Arguments

```
path
harvest.name.seclect
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path, harvest.name.seclect)
{
    cat("reading harvest output...")
    npixel.out <- get.output.info(path)[1]
    nyear.out <- get.output.info(path)[2]
    harvest.fn.out <- file(paste(path, "pft_harvest.pft.bin",
        sep = ""), "rb")
    harvest.temp <- array(NA, c(npixel.out, nft, nyear.out))
```

```
    harvest.temp[, , ] <- readBin(harvest.fn.out, double(), npixel.out *
        nyear.out * 32, size = 4)
    harvest.data <- array(NA, c(npixel.out, nyear.out, 32))
    for (i in 1:length(harvest.name)) harvest.data[, , i] <- harvest.temp[,
        i, ]
    close(harvest.fn.out)
    cat("done\n")
    return(harvest.data)
  }
```

`read.input.files` *read input files*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.input.files(filename, data.size)
```

Arguments

`filename`

`data.size`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (filename, data.size)
{
    fileHeader <- read.input.header(filename)
    file.in <- file(sprintf(filename), "rb")
    data.in <- array(NA, dim = c(fileHeader$nbands, fileHeader$nyears,
        fileHeader$ncells))
    seek(file.in, where = HEADER_SIZE, origin = "start")
    for (i in 1:fileHeader$nyears) {
        for (j in 1:fileHeader$ncells) {
            data.in[, i, j] <- readBin(file.in, integer(), n = fileHeader$nbands,
                size = data.size) * fileHeader$scalar
        }
    }
    close(file.in)
    return(data.in)
  }
```

read.input.files.one *read one input file*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.input.files.one(filename, data.size, year, nband)
```

Arguments

filename

data.size

year

nband

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (filename, data.size, year, nband)
{
    fileHeader <- read.input.header(filename)
    file.in <- file(sprintf(filename), "rb")
    data.in <- array(NA, dim = c(fileHeader$ncells))
    year.check <- 1 + fileHeader$firstyear - year
    seek(file.in, where = HEADER_SIZE, origin = "start")
    for (p in 1:fileHeader$ncells) {
        seek(file.in, where = (HEADER_SIZE + Lindex(fileHeader$ncells,
            fileHeader$nbands, year.check, p, nband)), origin = "start")
        data.in[p] <- readBin(file.in, integer(), n = 1, size = data.size) *
            fileHeader$scalar
    }
    close(file.in)
    return(data.in)
  }
```

read.input.grid *read input grid*

Description

read input grid

Usage

```
read.input.grid(path.in)
```

Arguments

```
path.in
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path.in)
{
    input.list <- dir(path.in)
    grid.name <- paste(path.in, input.list[grep("grid", input.list)],
        sep = "")
    grid.header <- read.input.header(grid.name)
    prec <- abs(log(grid.header$scalar)/log(10))
    gridfile <- file(grid.name, "rb")
    seek(gridfile, HEADER_SIZE, origin = "start")
    grid.temp <<- readBin(gridfile, integer(), n = 2 * grid.header$ncells,
        size = 2)
    grid.data <<- round(trunc(grid.temp, digits = 0) * grid.header$scalar,
        digits = 2)
    lon <<- grid.data[c(1:grid.header$ncells) * 2 - 1]
    lat <<- grid.data[c(1:grid.header$ncells) * 2]
    EAST <<- round(max(lon), prec)
    SOUTH <<- round(min(lat), prec)
    WEST <<- round(min(lon), prec)
    NORTH <<- round(max(lat), prec)
    RES <<- grid.header$cellsize
    NC <<- (NORTH - SOUTH)/RES + 1
    NR <<- (EAST - WEST)/RES + 1
    ind_lon <<- ceiling(lon/RES - min(lon)/RES + 1)
    ind_lat <<- ceiling(lat/RES - min(lat)/RES + 1)
    close(gridfile)
  }
```

---

read.input.header *read input header*

Description

read input header

Usage

```
read.input.header(filename)
```

Arguments

filename

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (filename)
{
    file.in <- file(filename, "rb")
    name <- readChar(file.in, nchar = 7)
    version <- readBin(file.in, integer(), n = 1, size = 4)
    order <- readBin(file.in, integer(), n = 1, size = 4)
    firstyear <- readBin(file.in, integer(), n = 1, size = 4)
    nyears <- readBin(file.in, integer(), n = 1, size = 4)
    firstcell <- readBin(file.in, integer(), n = 1, size = 4)
    ncells <- readBin(file.in, integer(), n = 1, size = 4)
    nbands <- readBin(file.in, integer(), n = 1, size = 4)
    cellsize <- readBin(file.in, numeric(), n = 1, size = 4)
    scalar <- readBin(file.in, numeric(), n = 1, size = 4)
    header <- data.frame(name, version, order, firstyear, nyears,
        firstcell, ncells, nbands, cellsize, scalar)
    close(file.in)
    return(header)
  }
```

---

read.input.yearband *read input year band*

Description

read input year band

Usage

```
read.input.yearband(filename, data.size, year, band)
```

Arguments

filename

data.size

year

band

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (filename, data.size, year, band)
{
    fileHeader <- read.input.header(filename)
    data.year <- year - fileHeader$firstyear + 1
    file.in <- file(sprintf(filename), "rb")
    data.in <- array(NA, dim = c(fileHeader$ncells))
    seek(file.in, where = HEADER_SIZE + data.size * ((data.year -
        1) * fileHeader$nband * fileHeader$ncells + (band - 1)),
        origin = "start")
    for (i in 1:fileHeader$ncells) {
        data.in[i] <- readBin(file.in, integer(), n = 1, size = data.size) *
            fileHeader$scalar
        seek(file.in, where = (fileHeader$nbands - 1) * 2, origin = "current")
    }
    close(file.in)
    return(data.in)
  }
```

`read.monthly.output` *read monthly output*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.monthly.output(path)
```

Arguments

path

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path)
{
    ncell <- get.output.info(path)[1]
    nyear <- get.output.info(path)[2]
    temp <- array(NA, dim = nyear * 12)
    monthly.data.frame <- data.frame(mevap = temp, mfpar = "",
        mgpp = "", minterc = "", mnpp = "", mrh = "", mrunoff = "",
        mswc1 = "", mswc2 = "", mtransp = "")
```

```
    for (i in 1:length(monthly.files.list)) {
        monthly.fn <- file(paste(path, monthly.files.list[i],
            ".bin", sep = ""), "rb")
        cat("Reading", monthly.files.list[i], "...")
        temp <- readBin(monthly.fn, double(), nyear * 12, size = sizeof.data)
        monthly.data.frame[, monthly.files.list[i]] <- temp
        cat("done\n")
        closeAllConnections()
    }
    return(monthly.data.frame)
  }
```

`read.output.all` *read all outputs*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.output.all(path)
```

Arguments

```
path
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path)
{
    pixel_year <- get.output.info(path)
    cat("=====================================================\n")
    cat("Path=", path, "\n")
    cat("pixel number=", pixel_year[1], "simulation years=",
        pixel_year[2], "\n")
    cat("Reading the output data...\n")
    read.grid(path)
    read.cow(path)
    read.output.carbon(path, pixel_year[1], pixel_year[2])
    read.output.flux(path, pixel_year[1], pixel_year[2])
    cat("read: <sucessful>\n")
    cat("=====================================================\n")
  }
```

read.output.carbon *function to do ... d*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.output.carbon(path, npixel.out, nyear.out)
```

Arguments

path

npixel.out

nyear.out

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path, npixel.out, nyear.out)
{
    vegc.data.out <<- array(NA, c(npixel.out, nyear.out))
    soilc.data.out <<- array(NA, c(npixel.out, nyear.out))
    litc.data.out <<- array(NA, c(npixel.out, nyear.out))
    firec.data.out <<- array(NA, c(npixel.out, nyear.out))
    fluxestab.data.out <<- array(NA, c(npixel.out, nyear.out))
    vegc.fn.out <- file(paste(path, "vegc.bin", sep = ""), "rb")
    vegc.data.out[, ] <<- readBin(vegc.fn.out, double(), npixel.out *
        nyear.out, size = 4)
    soilc.fn.out <- file(paste(path, "soilc.bin", sep = ""),
        "rb")
    soilc.data.out[, ] <<- readBin(soilc.fn.out, double(), npixel.out *
        nyear.out, size = 4)
    litc.fn.out <- file(paste(path, "litc.bin", sep = ""), "rb")
    litc.data.out[, ] <<- readBin(litc.fn.out, double(), npixel.out *
        nyear.out, size = 4)
    firec.fn.out <- file(paste(path, "firec.bin", sep = ""),
        "rb")
    firec.data.out[, ] <<- readBin(firec.fn.out, double(), npixel.out *
        nyear.out, size = 4)
    fluxestab.fn.out <- file(paste(path, "flux_estab.bin", sep = ""),
        "rb")
    fluxestab.data.out[, ] <<- readBin(fluxestab.fn.out, double(),
        npixel.out * nyear.out, size = 4)
    closeAllConnections()
  }
```

read.output.flux *function to do ... ~~*

Description

A concise (1-5 lines) description of what the function does.

Usage

```
read.output.flux(path, npixel.out, nyear.out)
```

Arguments

path

npixel.out

nyear.out

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path, npixel.out, nyear.out)
{
    mnpp.data.out <<- array(NA, c(npixel.out, 12, nyear.out))
    mrh.data.out <<- array(NA, c(npixel.out, 12, nyear.out))
    mevap.data.out <<- array(NA, c(npixel.out, 12, nyear.out))
    mtransp.data.out <<- array(NA, c(npixel.out, 12, nyear.out))
    mrunoff.data.out <<- array(NA, c(npixel.out, 12, nyear.out))
    mnpp.fn.out <- file(paste(path, "mnpp.bin", sep = ""), "rb")
    mnpp.data.out[, , ] <<- readBin(mnpp.fn.out, double(), npixel.out *
        nyear.out * 12, size = 4)
    mrh.fn.out <- file(paste(path, "mrh.bin", sep = ""), "rb")
    mrh.data.out[, , ] <<- readBin(mrh.fn.out, double(), npixel.out *
        nyear.out * 12, size = 4)
    mevap.fn.out <- file(paste(path, "mevap.bin", sep = ""),
        "rb")
    mevap.data.out[, , ] <<- readBin(mevap.fn.out, double(),
        npixel.out * nyear.out * 12, size = 4)
    mtransp.fn.out <- file(paste(path, "mtransp.bin", sep = ""),
        "rb")
    mtransp.data.out[, , ] <<- readBin(mtransp.fn.out, double(),
        npixel.out * nyear.out * 12, size = 4)
    mrunoff.fn.out <- file(paste(path, "mrunoff.bin", sep = ""),
        "rb")
    mrunoff.data.out[, , ] <<- readBin(mrunoff.fn.out, double(),
        npixel.out * nyear.out * 12, size = 4)
    closeAllConnections()
  }
```

read.output.harvest *function to do ... ~~*

Description

A concise (1-5 lines) description of what the function does. ~~

Usage

```
read.output.harvest(filename, ncells, nbands, startyear, year, data.size = 4, band = "ALL", par = 1
```

Arguments

filename

ncells

nbands

startyear

year

data.size

band

par

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (filename, ncells, nbands, startyear, year, data.size = 4,
    band = "ALL", par = 1)
{
    nyears <- file.info(filename)$size/ncells/nbands/data.size
    if (nyears/as.integer(nyears) != 1)
        stop("nyears:", nyears, " error\n")
    if (band[1] == "ALL")
        band <- c(1:nbands)
    harvest <- list()
    data <- array(NA, c(ncells, length(band), length(year)))
    fn <- file(filename, "rb")
    for (y in 1:length(year)) {
        for (b in 1:length(band)) {
            seek(fn, where = (year[y] - startyear) * nbands *
                ncells * data.size + (band[b] - 1) * ncells *
                data.size, origin = "start")
            data[, b, y] <- readBin(fn, double(), size = data.size,
                n = ncells)
        }
    }
    harvest$data <- data
    if (length(par) != 1) {
```

```
        for (i in 1:length(band)) {
            harvest$data[, i, ] <- data[, i, ] * par[i]
        }
    }
    else harvest$data <- harvest$data * par
    harvest$band <- band
    harvest$year <- year
    close(fn)
    return(harvest)
  }
```

`read.yearly.output` *function to do ... ~~*

Description

A concise (1-5 lines) description of what the function does. ~~

Usage

```
read.yearly.output(path)
```

Arguments

```
path
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (path)
{
    ncell <- get.output.info(path)[1]
    nyear <- get.output.info(path)[2]
    temp <- array(NA, dim = nyear)
    yearly.data.frame <- data.frame(firec = temp, firef = "",
        flux_estab = "", flux_harvest = "", litc = "", soilc = "",
        vegc = "")
    for (i in 1:length(yearly.files.list)) {
        yearly.fn <- file(paste(path, yearly.files.list[i], ".bin",
            sep = ""), "rb")
        cat("Reading", yearly.files.list[i], "...")
        temp <- readBin(yearly.fn, double(), nyear, size = sizeof.data)
        yearly.data.frame[, yearly.files.list[i]] <- temp
        cat("done\n")
        closeAllConnections()
    }
    return(yearly.data.frame)
  }
```

Index