Package: zooimage (via r-universe)

July 19, 2024

Type Package

Title Analysis of Numerical Plankton Images

Version 5.5.2

Date 2018-06-28

Author Philippe Grosjean [aut, cre], Kevin Denis [aut], Guillaume Wacquet [aut]

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 2.15.0), mlearning

Imports svMisc (>= 0.9-67), svDialogs (>= 0.9-53), filehash, jpeg, png, tiff, utils, digest, tools, MASS, mda, shiny, DT

Suggests rJava, mlbench

Description A free (open source) solution for analyzing digital images of plankton. In combination with ImageJ, a free image analysis system, it processes digital images, measures individuals, trains for automatic classification of taxa, and finally, measures plankton samples (abundances, total and partial size spectra or biomasses, etc.).

License GPL (>= 2)

URL http://www.sciviews.org/zooimage

NeedsCompilation no

Date/Publication 2018-06-29 16:00:26 UTC

Repository https://phgrosjean.r-universe.dev

RemoteUrl https://github.com/cran/zooimage

RemoteRef HEAD

RemoteSha f1f6cebc1faae5f49087275c69b6c67037a20b0c

Contents

`zooimage-package` *ZooImage: analysis of digital plankton images*

Description

ZooImage is a free (open source) solution for analyzing digital images of plankton. In combination with ImageJ, a free image analysis system, it processes digital images, measures individuals, trains for automatic classification of taxa, and finally, measures plankton samples (abundances, total and partial size spectra or biomasses, etc.)

Details

Package:	zooimage
Type:	Package
Version:	5.5.2
Date:	2018-06-28
License:	GPL 2 or above at your convenience.

Author(s)

Philippe Grosjean, Kevin Denis & Guillaume Wacquet, Numerical Ecology of Aquatic Systems, Mons University, Belgium. A part of the early concepts of ZooImage are inspired from PVA by Xabier Irigoien, Guillermo Boyra & Igor Arregi, AZTI Technalia, Spain, with their autorization. Angel Lopez-Urrutia, Centro Oceanografico de Gijon, IEO, Spain, also contributed to the initial concept during early discussions. No code come from them, however. Mike Sieracki, Ben Tupper

et al for the FIT VIS plugin (FlowCAM images process in ImageJ -FitVIS-, but this pulgin is not used any more by default in the current version).

Maintainer: Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

The software was funded, and is the property of:

- The University of Mons, Belgium, - Belgian Scientific Policy (BelSpo), Belgium, that contributed by funding the AMORE III project. - IFREMER, France, through a co-funded research collaboration with the University of Mons.

`correctError` *The ZooImage error correction (manual validation) tools*

Description

Open a web page for manual validation and error correction of predicted abundances in samples.

Usage

```
correctError(zidb, classifier, data = zidbDatRead(zidb), mode = "validation",
fraction = 0.05, sample.min = 100, sample.max = 200, grp.min = 2,
random.sample = 0.1, algorithm = "rf", diff.max = 0.2, prop.bio = NULL,
reset = TRUE, result = NULL)

addItemsToTrain(train, CtxSmp, add.mode = "SV+NSV", threshold = NA,
    dropItemsToTrain = dropItemsToTrain)
dropItemsToTrain(train, cl, drop.nb)

activeLearning(train, add.mode = "SV+NSV", threshold = NA)
```

Arguments

`zidb`	Path to a Zidb file.
`classifier`	A ZIClass object appropriate for this sample and the desired classification.
`data`	A ZIDat or a ZITest object matching that sample (by default, it is the ZIDat object contained in the zidb file).
`mode`	The mode to use for error correction. By default, `mode = "validation"`, where particles are manually validated. `mode = "demo"` is the same one, but it sorts particles according to the Class variable in data, ignoring changes made in the user interface (so that one can explain the logic of the process without care about how particles are manually resorted). Finally, `mode = "stat"` do not display the user interface at all and calculates all steps directly to show gain from the process from 0 to 100% of the particles validated.
`fraction`	The fraction of items to validate at each step (1/20th by default).
`sample.min`	Minimal number of items to take at each step.

sample.max	Maximal number of items to take at each step. In case the sample contains a very large number of items, the number of particles that are validated at each step are constrained by this parameter, and consequently, the total number of steps becomes large than 1/fraction, but usually, error correction allows to stop earlier.
grp.min	Minimal number of items to take for each group, on average.
random.sample	Fraction of random sample considered, when validating suspect items.
algorithm	Machine learning algorithm used to detect suspect items.
diff.max	Maximum difference allowed between probabilities in first and second class before considering the item is suspect.
prop.bio	Weight to apply to the groups for considering them as suspects (use biological or external considerations to build this).
reset	Do we reset analysis in the case a temporary file already exists for that sample (recommended).
result	Name of the object in the calling environment where the results will be stored (ZITest object). If not provided or NULL, it is the basename of the zidb file without extension plus _valid.
train	the training set to complete.
CtxSmp	the contextual samples containing validated items.
add.mode	the mode for adding items, "SV": Validated Suspects, "NSV": Validated Non-Suspects, or "SV+NSV": both (by default)).
threshold	the maximal number of items in each class of training set. This is used to decide when to drop items for the reworked training set.
dropItemsToTrain	the function to use to drop items in the training set (depending on threshold). By default, it is dropItemsToTrain(), but it could also be a custom function that has the same arguments.
cl	the class to consider.
drop.nb	the number of items to drop.

Value

correctError() returns nothing. It is called for its side-effect to install a web interface for manual validation of samples.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

zidbDatRead

Examples

```
# TODO...
```

file-utilities *Various file utility functions used by ZooImage*

Description

These functions are usually not called directly by the user, but they are interesting for developers. They help to manage files in the context of ZooImage processes.

Usage

```
extensionPattern(extension = "r", add.dot = !grepl("[.]", extension))
hasExtension(file, extension = "r", pattern = extensionPattern(extension))
noExtension(file)

listFilesExt(dir, extension = "r", pattern = extensionPattern(extension), ... )
zimList(dir, ...)
zimDatList(dir, ...)
zipList(dir, ...)
zidList(dir, ...)
zidbList(dir, ...)
jpgList(dir, ...)
pngList(dir, ...)

checkFileExists(file, extension, message = "file not found: %s",
    force.file = FALSE)
checkDirExists(dir, message = 'Path "%s" does not exist or is not a directory')
checkEmptyDir(dir, message = 'dir "%s" is not empty')
forceDirCreate(dir)

checkFirstLine(file, expected = c("ZI1", "ZI2", "ZI3", "ZI4", "ZI5"),
    message = 'file "%s" is not a valid ZooImage version <= 5 file')
```

Arguments

extension	lowercase version of the extension (the pattern will be constructed to be case-insensitive).
add.dot	if a dot is not provided, it is added by default in front of the pattern.
file	one or more file names or file paths to check.
pattern	a pattern matching a given file extension.
...	further arguments passed to the function. Currently, not in use.
dir	the directory to work with.
message	a warning message to provide (file/dirname replacement using %s).
force.file	make sure the item is a file, not a directory.
expected	the expected content of the first line of the file.

Details

All these function issue only warnings, no errors. Those functions that return TRUE or FALSE are designed to be used in batch mode.

Value

A string with suitable pattern to match a file extension for `extensionPattern()`.

The function `noExtension()` return base filenames without extensions.

A list of files with given extension for `listFilesExt()`, and `xxxList()` functions.

The other functions return TRUE or FALSE, depending if the tested condition is met or not.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

sampleInfo

Examples

```
# Construct a suitable pattern to match extensions of TIFF image files
extensionPattern("tif")
# Test if file names match given extensions (first 2 items only)
hasExtension(c("test1.tif", "test2.TIF", "test3.R"), "tif")
noExtension(c("test1.tif", "test2.TIF", "test3.R"))

# List all files with a given extension in a directory
ziDir <- system.file("examples", package = "zooimage")
listFilesExt(ziDir, "zid")
zidList(ziDir) # Idem

# Check that a file or a directory exists
checkDirExists(ziDir)
zisFile <- file.path(ziDir, "Description.zis")
checkFileExists(zisFile)

# Is this directory empty? (no)
checkEmptyDir(ziDir)

# force (re)creation of a directory
tmpDir <- file.path(tempdir(), "testdir")
forceDirCreate(tmpDir)
file.info(tmpDir)$isdir # yes
checkEmptyDir(tmpDir)   # yes
file.remove(tmpDir)
file.exists(tmpDir)

# Every .zis file must start with ZI1-5 => check this...
checkFirstLine(zisFile)
```

```
# Clean up
rm(ziDir, zisFile, tmpDir)
```

---

gui *The ZooImage GUI (Graphical User Interface)*

Description

These function display menus and dialog boxes to access ZooImages function without programming. Most of them are not intended to be called directly.

Usage

```
ZIDlg()
aboutZI(graphical = FALSE)
exitZI()
closeAssistant()
closeZooImage()
viewManual()
viewFrenchManual()
focusR()
focusGraph()
acquireImg()
importImg()
processImg()
makeZid()
makeZidb()
makeZidbFlowCAM()
makeTrain()
countCellsGUI()
activeLearningGUI()
collectTrain()
compTrain()
addVigsToTrain()
makeClass()
analyzeClass()
vignettesClass()
validClass()
editDescription()
processSamples()
processSamplesWithCells()
viewResults()
exportResults
loadObjects()
saveObjects()
listObjects()
```

```
removeObjects()
calib()
optInOutDecimalSep()
ZIUI()
```

Arguments

graphical	If TRUE, a graphical dialog box displays the information. Otherwise, information is printed at the R console.

Value

ZIDlg() is called for its side effect of dispalying the main ZooImage dialog box. aboutZI() gives sone information about the current ZooImage version. exitZI() unloads the zooimage package (for instance, to allow updating it). ZIUI() launches the error correction GUI using Shiny. The working dir must be in _analyze subdirectory of an analysis directory tree.

All the other functions are called for their side-effect and return value, if any, is not of primary importance.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

zidCompress

Examples

```
# The dialog box can be started just by issuing
# > ZIDlg()
```

---

gui-utilities *Various GUI utility functions used by ZooImage*

Description

These functions are usually not called directly by the user, but they are interesting for developers. They allow to select elements through dialog boxes.

Usage

```
selectObject(class = "data.frame", default = "", multiple = FALSE,
    title = paste0("Choose a ", class, ":"))
selectList(class = "data.frame", default = "", multiple = FALSE,
    title = paste0("Choose a list (of ", class, "s):"))
selectFile(type = c("ZipZid", "ZimZis", "LstZid", "ZidZidb", "Zip", "Zid", "Zidb",
    "Zim", "Zis", "Zie", "Zic", "Img", "TifPgm", "RData"), multiple = FALSE,
```

```
    quote = TRUE, title = NULL)
selectGroups(groups, multiple = TRUE, title = "Select taxa you want to plot")

createThreshold(ZIDat)

imageViewer(dir = getwd(), pgm = getOption("ZI.ImageViewer"))
startPgm(program, cmdline = "", switchdir = FALSE, iconize = FALSE, wait = FALSE)
modalAssistant(title, text, init, options = NULL, check = NULL,
    select.file = NULL, returnValOnCancel = "ID_CANCEL", help.topic = NULL)
```

Arguments

class	the class of objects to retrieve (or class of list components for selectList()).
default	the default item selected in the list.
multiple	are multiple selections allowed?
title	the title of the dialog box.
type	the type of file to list in selection dialog box.
quote	do we add quotes (\") around file names?
groups	a list of groups to select from.
ZIDat	a ZIDat object.
dir	directory to open in the image viewer.
pgm	program to use as image viewer. If not provided and not defined in the option(ZI.ImageViewer = ...), then, a reasonable default program is used (the file explorer is no better program found).
program	name of the program to start. It must match an entry in R options giving the actual executable that correspond to that program.
cmdline	the command line to run to start this program.
switchdir	do we switch R current directory to the directory where the program is located?
iconize	in case the ZooImage assistant is open, do we iconize it?
wait	do we wait that the external program is closed?
text	textual explanations to show in the modal assistant.
init	initial values for the modal assistant.
options	options to select in the modal assistant.
check	checkbox to add in the modal assistant.
select.file	prompt for a file to select in the modal assistant.
returnValOnCancel	what to return if the user clicks the Cancel button in the modal assistant dialog box?
help.topic	help topic to associate with the Help button of the modal assistant dialog box.

Details

As these functions are not made to be directly used by end-users, We don't give more details here. Developers interested to use these functions are encouraged to look at their code in the zooimage package source!

Value

A string or vector of strings of selected items. `character(0)` is returned to indicate the user clicked 'Cancel', while an empty string (`""`) is returned in case there is no corresponding element found.

`createThreshold()` proposes a dialog box to create a threshold on one variable in a ZIDat object (indicate minimum and maximum value allowed for that variable).

For `imageViewer()`, `TRUE` or `FALSE` is returned invisibly, depending if the program could be lauched or not. The problem is reported in a warning.

`startPgm()` is mostly invoked for its side effect of starting an external program. Status code returned by the program is returned if `wait = TRUE`.

`modalAssistant()` is currently disabled, and it will thus display no dialog box and return `returnValOnCancel` directly.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`listSamples`, `noExtension`

Examples

```
## Not run:
## Create two datasets in R and ask for selecting one:
df1__ <- data.frame(x = 1:3, y = 4:6)
df2__ <- data.frame(z = 1:10)
selectObject() # Try also to click 'Cancel'
## Can select both too
selectObject(multiple = TRUE, title = "Choose one or more data.frames")
selectObject("nonexistingclass") # Returns an empty string!

## Create lists containing only data frames as components
lst1__ <- list(A = df1__, B = df2__)
lst2__ <- list(C = df1__)
selectList() # Try also to click 'Cancel'
## Can select both too
selectList(multiple = TRUE, title = "Select one or more lists")
selectList("nonexistingclass")
rm(df1__, df2__, lst1__, lst2__)

## Select one or more ZooImage files
selectFile() # One Zip or Zid file

## Select groups to process
selectGroups(c("Copepods", "Appendicularians", "Medusae"))

## Start default image viewer on the current working directory
imageViewer()

## TODO: examples for createThreshold(), startPgm() and modalAssistant()
```

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

`import` *Function to import data from the FlowCAM*

Description

These functions read data and metadata from FlowCAM runs and import them into ZooImage objects.

Usage

```
readFlowCAMctx(ctx, stop.it = TRUE)
readFlowCAMlst(lst, skip = 2, read.ctx = TRUE)
importFlowCAM(lst, rgb.vigs = FALSE,  type = "ZI3", replace = FALSE)
```

Arguments

`ctx`	the path to a .ctx FlowCAM file.
`stop.it`	should the process stop in case of an error? Set this to `FALSE` when you run in batch mode and want the error to be repported back to the calling code.
`lst`	the path to a .lst FlowCAM file.
`skip`	the number of lines to skip in the .lst table before reading the data. This is usally two lines.
`read.ctx`	should we also read the .ctx file with `readFlowCAMctx()`?
`rgb.vigs`	do we build color vignettes that mix OD, visual and mask in the three RGB channels? By default, not (FALSE)
`type`	the type of `.zidb` file to create. Currently, only supports `type = "ZI3"` (defaulf value).
`replace`	a boolean indicating if an existing `.zidb` file should be replaced by a new one.

Details

The Visual Spreadsheet software provided with the FlowCAM is constantly changing specifications from version to version. This is mainly tested with VS 1.5.14 and 3.2.3.

Value

A `zidb` object with the converted data, metadata and images.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

selectObject

Examples

```
## TODO...
```

---

utilities *Various utility functions used by ZooImage*

Description

These functions are usually not called directly by the user, but they are interesting for developers.

Usage

```
sampleInfo(filename, type = c("sample", "fraction", "image", "scs",
    "date", "id", "frac", "imgnbr"), ext = "_dat[135][.]zim$")

underscoreToSpace(string)
trimString(string)

listSamples(ZIobj)
makeId(ZIDat)
addClass(ZIDat, ZIobj)
calcVars(x, drop.vars = NULL, drop.vars.def = dropVars())
dropVars()

ecd(area, cells = 1)

parseIni(data, label = "1")

calibrate(ODfile)

getDec()

zipNoteAdd(zipfile, zimfile)
zipNoteGet(zipfile, zimfile = NULL)

makeZIVignettes(orig.dir = getwd(), target.dir = dirname(orig.dir), clean.work = FALSE)
```

Arguments

filename	name of a file from which to extract information. It is supposed to be spelled as: SCS.xxxx-xx-xx.SS+Fnn.ext where 'SCS' is the series-cruise-station code, 'xxxx-xx-xx' is the date of collection (year-month-day), 'SS' is the unique sample identifier, 'F' is the fraction, 'nn' is the image number (when there are several

	images per fraction) or 'nn.mmm' when there are 'mmm' pictures taken to cover a bigger area of cell 'nn', and 'ext' is the file extension.
`type`	the type of data to extract (see examples) for `sampleInfo()`, or the type of file to select in `selectFile()`.
`ext`	the pattern to use (regular expression) to eliminate file extension from the 'filename'.
`string`	a character string to rework, or a vector of character strings.
`ZIobj`	a ZooImage object (here, a 'ZIDat', 'ZIDesc', 'ZITrain' or 'ZITest' object; most probably one of the last two for `addClass()`).
`ZIDat`	a 'ZIDat' object, or a data frame with correct column labels.
`x`	a data frame, but most probably, a 'ZIDat' object.
`drop.vars`	a character vector with names of variables to drop, or `NULL` (by default) to keep them all.
`drop.vars.def`	a second list of variables to drop contained in a character vector. That list is supposed to match the name of variables that are obviously non informative and that are dropped by default. It can be gathered automatically using `dropVars()`. That list includes Id, Label, Dil, ... (see details, variables with an asterisk).
`area`	a numerical vector with areas from which ECDs are calculated (Equivalent Circular Diameter, a more suitable term for 2D images than ESD, Equivalent Spherical Diameter).
`cells`	the number of cells in the particle (colony). If different from 1, the area is first divided by the number of cells bevore calculating the cell individual ECD.
`data`	a vector containing the data to parse.
`label`	a label to include for the parsed data.
`ODfile`	an image file of O.D. calibrated items that can be used to calibrate grayscales.
`zipfile`	a zip archive.
`zimfile`	a .zim file to use, or to create. If `NULL` (default) in `zipNoteGet()`, the data are not written in a file, but returned.
`orig.dir`	the directory containing the data (current directory by default)
`target.dir`	where to place the results, by defaut, the parent directory of `orig.dir`
`clean.work`	should we clean intermediary items (`FALSE`, by default)

Details

As these functions are not made to be directly used by end-users, We don't give more details here. Developers interested to use these functions are encouraged to look at their code in the zooimage package source!

Here is the list of all variables you got after running the standard version of `calcVars()` on ZIDat objects made by one of the ZooImage ImageJ plugins (you can provide your own version for, e.g., calculating more features):

Variable	Description	Origin

Area	Area of the region of interest (ROI)	ImageJ
Mean	Average gray value of the ROI	ImageJ
StdDev	Standard deviation of the gray values	ImageJ
Mode	Most frequent gray value within the ROI	ImageJ
Min	Minimum gray value within the ROI	ImageJ
Max	Maximum gray value within the ROI	ImageJ
X*	X coordinate of the centroid of the ROI in the image	ImageJ
Y*	Y coordinate of the centroid of the ROI in the image	ImageJ
XM*	X coordinate of the center of mass of the ROI in the image	ImageJ
YM*	Y coordinate of the center of mass of the ROI in the image	ImageJ
Perim.	Perimeter of the ROI	ImageJ
BX*	X coordinate of the upper left corner of the bounding rectangle (BR)	ImageJ
BY*	Y coordinate of the upper left corner of the BR	ImageJ
Width*	Width of the rectangle enclosing the ROI	ImageJ
Height*	Height of the rectangle enclosing the ROI	ImageJ
Major	Length of the longest axis of the ellipse fitted to the ROI	ImageJ
Minor	Length of the smallest axis of ellipse fitted to the ROI	ImageJ
Angle*	Angle between longest axis and an horizontal line	ImageJ
Circ.	Circularity of the ROI	ImageJ
Feret	Longest Feret diameter	ImageJ
IntDen	Sum of the gray values within the ROI	ImageJ
Median	Median value of the gray values within the ROI	ImageJ
Skew	Third order moment for the gray value	ImageJ
Kurt	Fourth order moment for the gray value	ImageJ
XStart*	X coordinate of initial point for the outline of the ROI	ImageJ
YStart*	Y coordinate of initial point for the outline of the ROI	ImageJ
Id*	Unique identifier of the ROI (Label_Item)	zooimage
Label*	Unique name of the image	zooimage
Item*	Name of the ROI	zooimage
ECD	Equivalent circular diameter of the ROI	zooimage
Dil*	Dilution coefficient to use for that ROI	zooimage
AspectRatio	Aspect ratio of the ROI	zooimage
CentBoxD	Distance between the centroid and the center of the BR	zooimage
GrayCentBoxD	Distance between the center of mass and the center of the BR	zooimage
CentroidsD	Distance between the centroid and the center mass	zooimage
Range	Range of the gray values in the ROI	zooimage
MeanPos	Position of mean gray value in the range of gray values	zooimage
SDNorm	Normalized standard deviation of the gray values	zooimage
CV	Coefficient of variation of gray values	zooimage
MeanDia	Mean diameter calculated on Major and Minor	zooimage
MeanFDia	Mean diameter calculated on Feret and Minor	zooimage
Transp1	Transparency calculated using ECD and MeanDia	zooimage
Transp2	Transparency calculated using ECD and MeanFDia	zooimage
Elongation	Elongation of the ROI	zooimage
Compactness	Compactness of the ROI	zooimage
Roundness	Roundness of the ROI	zooimage
Class*	Manual identification of the vignette for that ROI	zooimage
Predicted*	Automatic identification of the vignette for that ROI	zooimage

Predicted2*	Second automatic identification of the vignette for that ROI	zooimage

For the origin, ImageJ = measured during image ananlysis plugin in ImageJ, zooimage = calculated either during importation of data, or by `calcVars()`. Variables whose name ends with an asterisk are dropped by default.

Value

A string or vector of strings for `sampleInfo()`, `listSamples()` and `makeId()`. For those functions, `character(0)` is returned to indicate a problem (usually with a warning issued to explain it), while an empty string (`""`) is returned in case there is no corresponding element found.

The data.frame with additional columns for calculated variables with `calcVars()`. Variables to drop are gathered using `dropVars()`, altogether with a list provided explicitly in the `drop.vars =` argument. The list of variable names to drop automatically and silently can be stored in a variable named `ZI.dropVarsDef` or in `options(ZI.dropVarsDef = ....)`.

A vector of numerical values for `ecd()`.

Transformed strings for `trimstring()` and `underscoreToSpace()`

`parseIni()` reads the data and creates a list of data frames. Each entry in the list maps one section in the ini file (with the same name). For 'key=value' pairs, a one line data frame containing values and with keys as column names. The first column of these data frames is named label and get the corresponding value passed by the 'label' argument. That way, one can easily keep track of entries when data frames originated from various different ini files are merged together.

`calibrate()` returns a vector of two numbers with white and black point calibration (gray levels corresponding, respectively to O.D. = 0 and O.D. = 1.024), plus a "msg" attribute with some explanation in case of problem.

`zipNoteAdd()` returns `TRUE` or `FALSE` depending if the data from the zimfile was successfully added to the zip archive or not. Problem is returned in a warning. `zipNoteGet()` returns the comment included in the zip archive (invisibly if 'zimfile' is not `NULL`), `character(0)` if no comment if found, or `NULL` in case of a problem. The problem is detailled in a warning.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`noExtension`, `selectObject`

Examples

```
# Given a correct ZooImage name for a sample, return parts of it
smp__ <- "MTLG.2010-03-15.H1+A1.03_dat1.zim"
sampleInfo(smp__, "sample")
sampleInfo(smp__, "fraction")
sampleInfo(smp__, "image")
sampleInfo(smp__, "scs")
sampleInfo(smp__, "date")
sampleInfo(smp__, "id")
```

```
sampleInfo(smp__, "frac")
sampleInfo(smp__, "imgnbr")
rm(smp__)
sampleInfo(c("ScanG16.2004-10-20+A1.tif", "ScanG16.2004-10-20+B1.tif"),
    type = "sample", ext = extensionPattern("tif"))

# Character strings manipulation functions
underscoreToSpace("Some_string_to_convert")
trimString("    \tString with\textra spaces  \t")

# Variables calculation utilities
df__ <- data.frame(Label = c("Alabel", "AnotherLabel"), Item = c("01", "02"))
makeId(df__)
rm(df__)
ecd(1:10)
ecd(1:10, cells = 2)
ecd(1:10, cells = 1:10)

### TODO: addClass(), calibrate(), calcVars(), parseIni(), zipNoteAdd() and zipNoteGet() examples
```

`zic` *Check .zic files (ZooImage Categories)*

Description

Categories (i.e., plankton taxa), with possibly several sub-levels are defined in .zic files. This function check the files are correct.

Usage

```
zicCheck(zicfile)
```

Arguments

`zicfile`	the name of the .zic file to test.

Value

This function return `TRUE` or `FALSE`, depending on the content of the tested file.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`prepareTrain`

Examples

```
## Check that Detailed.zic file in the /etc subdir is correct
zicCheck(file.path(svMisc::getTemp("ZIetc"), "Detailed.zic"))
```

`ZIClass` *Create and manipulate 'ZIClass' objects*

Description

'ZIClass' objects are key items in ZooImage. They contain all what is required for automatically classify plancton from .zid files. They can be used as blackboxes by all users (but require users trained in machine learning techniques to build them). Hence, ZooImage is made very simple for biologists that just want to use classifiers but do not want to worry about all the complexities of what is done inside the engine!

Usage

```
ZIClass(formula, data, method = getOption("ZI.mlearning", "mlRforest"),
    calc.vars = getOption("ZI.calcVars", calcVars), drop.vars = NULL,
    drop.vars.def = dropVars(), cv.k = 10, cv.strat = TRUE,
    ..., subset, na.action = na.omit)

## S3 method for class 'ZIClass'
print(x, ...)
## S3 method for class 'ZIClass'
summary(object, sort.by = "Fscore", decreasing = TRUE,
    na.rm = FALSE, ...)
## S3 method for class 'ZIClass'
predict(object, newdata, calc = TRUE, class.only = TRUE,
    type = "class", ...)
## S3 method for class 'ZIClass'
confusion(x, y = response(x), labels = c("Actual", "Predicted"),
    useNA = "ifany", prior, use.cv = TRUE, ...)
```

Arguments

`formula`	a formula with left member being the class variable and the right member being a list of predicting variables separated by a '+' sign. Since `data` is supposed to be previously filtered using `calc.vars` and the class variable in 'ZITrain' object is always named `Class`, the formula almost always reduces to `Class ~ .`
`data`	a data frame (a 'ZITrain' object usually), containing both measurement and manual classification (a factor variables usually named 'Class').
`method`	the machine learning method to use. It should produce results compatible with `mlearning` objects as returned by the various `mlXXX()` functions in the `mlearning` package. By default, the random forest algorithm is used (it is among the ones that give best result with plankton).

calc.vars	a function to use to calculate variables from the original data frame.
drop.vars	a character vector with names of variables to drop for the classification, or NULL (by default) to keep them all.
drop.vars.def	a second list of variables to drop contained in a character vector. That list is supposed to match the name of variables that are obviously non informative and are dropped by default. It can be gathered automatically using dropVars(). See ?calcVars for more details.
cv.k	the k times for cross-validation.
cv.strat	do we use a stratified sampling for cross-validation? (recommended).
...	further arguments to pass to the classification algorithm (see help of that particular function).
subset	an expression for subsetting to original data frame.
na.action	the function to filter the initial data frame for missing values. Althoung the default in R is na.fail, leading to failure if at least one NA is found in the data frame, the default here is na.omit which leads to elimination of all lines containing at least one NA. Take care about how many items remain, if you encounter many NAs in your dataset!
x	a 'ZIClass' object.
object	a 'ZIClass' object.
newdata	a 'ZIDat' object, or a 'data.frame' to use for prediction.
sort.by	the statistics to use to sort the table (by default, F-score).
decreasing	do we sort in increasing or decreasing order?
na.rm	do we eliminate entries with missing data first (using na.omit())?
calc	a boolean indicating if variables have to be recalculated before running the prediction.
class.only	if TRUE, return just a vector with classification, otherwise, return the 'ZIDat' object with 'Predicted' column appended to it.
type	the type of result to return, "class" by default. No other value is permitted if class.only is FALSE.
y	a factor with reference classes.
labels	labels to use for, respectively, the reference class and the predicted class.
useNA	do we keep NAs as a separate category? The default "ifany" creates this category only if there are missing values. Other possibilities are "no", or "always". The default is suitable for test sets because unclassified items (those in the "_" directory or one of its subdirectories) get NA for Class.
prior	class frequencies to use for first classifier that is tabulated in the rows of the confusion matrix. This is either a single positive numeric to set all class frequencies to this value (use 1 for relative frequencies and 100 for relative freqs in percent), or a vector of positive numbers of the same length as the levels in the object. If the vector is named, names must match levels. Alternatively, providing NULL or an object of null length resets row class prefencies into their initial values.

use.cv — the predicted values extracted from the 'ZIClass' object can either be the predicted values from the training set, or the cross-validated predictions (by default). Most of the time, you want the cross-validated predictions, which allows for not (or less) biased evaluation of the classifier prediction... So, if you don't know, you are probably better leaving the default value.

Value

`ZIClass()` is the constructor that build the 'ZIClass' object. `print()`, `summary()` and `predict()`) are the methods to print the object, to calculate statistics on this classifier based on the confusion matrix and to predict groups for ZooImage samples, using one 'ZIClass' object.

Note

Always analyze carefully the properties, performances and limitations of a 'ZIClass' object before using it to classify objects of one series. For instance, you can use `confusion()` to compare two classifiers, or an automatic classifier with a manual classification done by a taxonomists. Always respect the limitations in the use of a 'ZIClass' object (for instance, a classifier specific of one given series should not be used to classify items in a different series)! It is a good practice to make a report, documenting a 'ZIClass' object, together with the comments of taxonomists that made the reference training set, and with details on the analysis of the performances of the classifier.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`getTrain`, `confusion`

Examples

```
##TODO...
```

---

zid — *Manage .zid files (ZooImage Data)*

Description

Compress, uncompress and verify ZooImage Data files.

Usage

```
zidCompress(zidir, type = c("ZI1", "ZI2", "ZI3", "ZI4", "ZI5"), check = TRUE,
    check.vignettes = TRUE, replace = FALSE, delete.source = replace)
zidCompressAll(path = ".", samples = NULL, type = c("ZI1", "ZI2", "ZI3", "ZI4", "ZI5"),
    check = TRUE, check.vignettes = TRUE, replace = FALSE,
    delete.source = replace)
```

```
zidClean(path = ".", samples = NULL)
zidDatMake(zidir, type = "ZI5", replace = FALSE)
zidDatRead(zidfile)

zidUncompress(zidfile, path = dirname(zidfile), delete.source = FALSE)
zidUncompressAll(path = ".", zidfiles = zidList(path, full.names = TRUE),
    path.extract = path, skip.existing.dirs = TRUE, delete.source = FALSE)

zidVerify(zidir, type = c("ZI1", "ZI2", "ZI3", "ZI4", "ZI5"), check.vignettes = TRUE)
zidVerifyAll(path = ".", samples = NULL, type = c("ZI1", "ZI2", "ZI3", "ZI4", "ZI5"),
    check.vignettes = TRUE)
```

Arguments

zidir	a directory containing data to put in a .zid files.
type	the ZI file format, currently 'ZI1', 'ZI2', 'ZI3', 'ZI4', or 'ZI5' types are supported (ZooImage1-5).
check	do we check the files in this directory before/after compression?
check.vignettes	do we check if the future .zid file contains all vignettes?
replace	do we replace existing files?
delete.source	do we delete source files after compression?
path	(un)compress or verify all subdirectories (except those starting with '_') in respective .zid files. Also, a place where to put uncompressed files (in the 'sample' subdirectory).
samples	a list of 'samples', i.e., subdirectories to process.
zidfile	a .zid file to uncompress or read from.
zidfiles	a series of .zid files to uncompress.
path.extract	a place where to extract data from .zid files.
skip.existing.dirs	do not unzip if the subdir already exists.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

zimMake

Examples

```
##TODO...
```

zidb *Manage .zidb files (ZooImage Databases)*

Description

Compress, uncompress .zidb files that contain data for a sample. Starting from ZooImage 3, the new format uses filehash tables for better performances. Conversion from and to the old .zid format (a zip archive indeed) is supported for compatibility with old datasets. Display content of a .zidb file is a simple way (both data/metadata and vignettes)

Usage

```
zidbMake(zidir, zidbfile = paste0(sub("[/\\]+$", "", zidir), ".zidb"), zisfile =
    file.path(dirname(zidir), "Description.zis"), type = "ZI5", smptime = "",
   check = FALSE, check.vignettes = FALSE, replace = FALSE, delete.source = replace)
zidbMakeAll(path = ".", samples, zisfiles = file.path(dirname(samples),
    "Description.zis"), type = "ZI5", check = FALSE,
    check.vignettes = FALSE, replace = FALSE, delete.source = replace)

zidToZidb(zidfile, zisfile = file.path(dirname(zidfile),
    "Description.zis"), replace = FALSE, delete.source = replace)
zidToZidbAll(path = ".", zidfiles, zisfiles = file.path(dirname(zidfiles),
    "Description.zis"), replace = FALSE, delete.source = replace)
zidbToZid(zidbfile, zisfile = file.path(dirname(zidbfile),
    "Description.zis"), replace = FALSE, delete.source = replace)
zidbToZidAll(path = ".", zidbfiles, zisfiles = file.path(dirname(zidbfiles),
    "Description.zis"), replace = FALSE, delete.source = replace)

zidbLink(zidbfile)
zidbDatRead(zidbfile)
zidbSampleRead(zidbfile)
zidbSummary(zidbfile, n = 3)

zidbPlotNew(main = "ZooImage collage", ...)
zidbDrawVignette(rawimg, type, item, nx = 5, ny = 5, vmar = 0.01)
zidbPlotPage(zidbfile, page = 1, title = NULL, type = "guess", method = NULL,
class = NULL)
```

Arguments

zidir	a directory containing data to put in a .zidb files.
zidbfile	the path of the .zidb file.
zidbfiles	the path of a series of .zidb files.
zidfile	the path of a .zid file.
zidfiles	the path of a series of .zid files.

zisfile	the path of the .zis file that contains description of this sample.
zisfiles	the path of a series of .zis files that contain description of the processed samples.
type	the ZI file format, currently only 'ZI5' type is supported. For `zidbDrawVignette()` and `zidbPlotPage()`, it is the type of vignette image. Currently, it can be either "jpg", or "png". If not provided, or if `type = "guess"` it will be guessed by the function.
smptime	the time the sample was collected. This value will replace a `<<TIME>>` placeholder in the metadata. You can also use `<<SMP>>` and `<<DATE>>` placeholders in corresponding metadata and they will be replaced by values computed from the sample name.
check	do we check the files in this directory before/after compression?
check.vignettes	do we check if the future .zidb file contains all vignettes? This is `FALSE` by default because they may not exist yet. This is the case when they are generated on the fly, like in the new version 5.5 processes.
replace	do we replace existing files?
delete.source	do we delete source files after compression?
n	the number of line of the data to print. If `n <= 0` no data is printed (only metadata).
path	look for files in this path.
samples	a list of 'samples', i.e., subdirectories to process.
main	the title of the new plot.
...	further arguments passed to the `plot()` function.
rawimg	the raw content of a vignette, as stored in a .zidb file.
item	the item number to draw (enumeration from left to right and from top to bottom).
nx	the number of vignettes in a column.
ny	the number of vignettes in a row.
vmar	the relative size of vignette margins.
page	the page to display (each page contains 25 vignettes).
title	the title of the page.
method	the name of the validation method to use to extract validation data.
class	a character vector with one or more classes for the validation data that we want to keep.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

zidCompress

Examples

```
##TODO...
```

zie *ZIE (ZooImage Extension) manipulate .zie files to add custom import, export, etc. routines to ZooImage*

Description

A .zie file describes sequentially images and tells how to convert and rename them and automates creation of associated metadata (.zim files). At the end of the process, one ends with complete and fully usable information (both the images with correct format/name and the metadata), so that you can proceed in ZooImage with imported data.

Usage

```
zieMake(path = ".", Filemap = "Import_Table.zie", check = TRUE, replace = FALSE,
    move.to.raw = TRUE, zip.images = "[.]tif$")

zieCompile(path = ".", Tablefile = "Table.txt", Template = "ImportTemplate.zie",
    Filemap = paste("Import_", noExtension(Tablefile), ".zie", sep = ""),
    Nrange = c(1, 1000), replace = TRUE, make.it = FALSE,
    zip.images = "[.]tif$")
zieCompileFlowCAM(path = ".", Tablefile, Template = "ImportTemplate.zie",
    check.names = TRUE)

ZIE(title, filter, description, pattern, command, author, version, date,
    license, url, depends = "R (>= 2.4.0), zooimage (>= 1.0-0)",
    type = c("import", "export"))
## S3 method for class 'ZIE'
print(x, ...)
```

Arguments

path	path where the images and the .zie file are located.
check.names	do wr check names in the .zie and .txt files?
Filemap	name of the .zie file mapping images to convert.
check	do we check if conversion programs (if any needed) are available?
replace	do we replace existing files (not used yet)?
move.to.raw	do we move processed files to the '_raw' subdirectory (currently, it is always the case)?
zip.images	the pattern to use for images extension in the zipped file.
Tablefile	a tab-delimited ASCII file containing a table defining samples and samples characteristics used to compile .zim files and to convert/rename images.
Template	a template to use for creating the .zie file.
Nrange	A range (two integers) giving min and max number of images for each sample/fraction.

make.it	do we make the .zie (run its commands) after compiling it?
title	the title of the ZooImage Extension.
filter	a filter to apply to select concerned files.
description	a short description of the ZooImage Extension.
pattern	a regular expression to use to select concerned files.
command	a string holding the R command to run to convert images.
author	who wrote and maintains the ZooImage Extension? Please, provide also an email address.
version	version of the ZooImage Extension as x.y-z, in a string.
date	date at which the extension was compiled as yyyy-mm-dd.
license	which are the license terms for this ZooImage Extension?
url	a link to a web page dedicated to this ZooImage Extension.
depends	which R packages does this ZooImage Extension requires?
type	type of ZooImage extension; currently, only "import" or "export".
x	a ZIE object.
...	further arguments passes to the function (currently not in use).

Value

`zieMake()`, `zieCompile()` and `zieCompileFlowCAM()` are invoked for their side-effects of importing images and metadata into a format that ZooImage can use. The first two functions return `TRUE` or `FALSE`, depending if the importation was done without errors, or not. More information is issued as warnings. The last function returns the name of the .zie file resulting from the compilation. It returns `NULL` in case it failed to compile the .zie file.

`ZIE()` creates a 'ZIE' object that describes a ZooImage Extension and allows to plug it in transparently inside the ZooImage GUI. `print.ZIE()` nicely formats the content of this objects when using the generic function `print()`.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

zimMake

Examples

```
##TODO...

### For programmers ###
# Adding a new import filter is just a mather of writing the importation function
# and making it available to users by means of a ZIEimport object.
```

zim *Manipulate .zim files (ZooImage Metadata/Measurements)*

Description

Various fonctions to manipulate ZooImage Metadata in .zim format (either '*.zim', or '*_datX.zim' files).

Usage

```
zimCreate(zimfile, template = NULL, edit = TRUE, editor =
    getOption("fileEditor"), wait = FALSE)
zimEdit(zimfile, editor = getOption("fileEditor"), wait = FALSE, ...)
zimMake(dir = ".", pattern = extensionPattern("tif"), images =
    list.files(dir, pattern))

zimExtractAll(zipdir = ".", zipfiles = zipList(zipdir), path = NULL,
    replace = FALSE)
zimUpdateAll(zipdir = ".", zipfiles = zipList(zipdir), zimdir = NULL,
    check.zim = TRUE)

isZim(zimfile)
zimVerify(zimfile, is.dat1 = hasExtension(zimfile, "_dat1.zim"),
    check.table = FALSE)

zimDatMakeFlowCAM(zimfile)
zimDatMakeFlowCAMAll(path = ".", zimfiles = NULL)
```

Arguments

zimfile	a .zim file.
zimfiles	a list of .zim files to use.
template	a .zim template to start with, if the .zim file does not exist yet.
edit	do we edit the .zim file that we just created?
editor	a program to use for editing the .zim file.
wait	do we wait that the file is edited? In this case, R is frozen until the editor is closed.
dir	a directory where .zim files will be created.
pattern	the pattern matching for automatically listed images that require a .zim file.
images	the list of images requiring a .zim file (either all image matching 'pattern' in 'dir', or provide your own listing here).
zipdir	a directory where to find .zip files.
zipfiles	a list of .zip files (by default, all .zip files in 'zipdir').

path	the path where to extract zims. If NULL, it is computed from zipdir: it is either the same path, or the parent directory if last directory is named '_raw'.
replace	do we replace existing .zim files?
zimdir	the directory where the .zim files are located.
check.zim	do we verify .zim files before refreshing metadata in .zip files?
is.dat1	is it a '_dat1.zim' file, that is a file collecting metadata AND objects measurements?
check.table	try to read the table of measurements in the '[Data]' section. Ignored if is.dat1 = FALSE.
...	further arguments passed to the fileEdit().

Details

ZooImage Metadata/Measurements ('.zim' and '_dat1.zim' files, respectively) are text files containing metadata (that is, additional information required to process the images, like sample identification, information about collection and process of the sample, digitizing hardware and software, etc.). These metadata are represented as a pair 'key' = 'value' in ANSI encoding and are organized into sections written in square brackets on a separate line. For instance, '[Subsample]' defines a 'Subsample' section. The first line of .zim files must always be 'ZI1' in the case of ZooImage version 1, 'ZI2' for version 2, and 'ZI3' for current version. This identifiant allows for making incompatible changes in future versions without taking the risk to accidentally try processing these newer versions with an old, incompatible version of ZooImage in the future. Here are the first few lines of an example .zim file: for instance).

```
ZI3
[Image]
Hardware=EPSON 4870
Software=VueScan 8.0.10  # See ZooImage.ini file for VueScan config
...
```

After 'ZI3' in the first line, there is a definition of an 'Image' section, with two keys: 'Hardware' with value 'EPSON 4870' and 'Software' with value 'VueScan 8.0.10', followed by a comment (everything after the '#' sign). Take care: since '#' defines a comment, do not use it, neither in keys, nor in values!

Take care to define **unique keys accross all sections**! The section are just there to organize your metadata into logical subunits... but they are not considered in the process. If you define a key named 'mykey' both in '[Section1]' and in '[Section2]', only the first occurence of 'mykey' will be used by ZooImage!

The ZooImage Measurements ('_dat1.zim' files) are structured the same way, but there is a special '[Data]' section at the end that contains a tab-delimited table with all measurements done on identified objects, during the image analysis (process of the images). This table starts with a header naming the colums, with two first columns being necessary '!Item' and 'Label'. 'Label' is the name of the image where the object is found and 'Item' is a unique identifier (usually a number) given to that object in the image (i.e., Label+Item is the unique identifier of each object in the whole series). The other columns define the measurements done on the objects (area, perimeter, length,

distribution of gray levels, etc.). The amount and name of measurements are not fixed. It is the particular ImageJ plugin that you use to process your image that defines them (it means that adding new measurements is very easy to do and they are automatically considered by ZooImage).

Note that these measurements are converted using calibration information, if available. That is, lengths are in microns, surfaces are in squre microns and gray levels are in OD (Optical Densities), so that, measurements are comparable from picture to picture, even if spatial resolution or distribution of gray levels (contrast, luminosity, ...) are not exactly the same in all images of the series! The table must also contains four additional columns with obligatory names being 'BX', 'BY', 'Width', 'Height'. There are the coordinates of the top-left corner of the bounding box around the object (BX, BY) and the Width and Height of this box. These fields are required to locate the object in the original image. Here is a short abstract of a [Data] section:

```
... (metadata definitions as above)

[Data]
!Item  Label    Area    Perim.  ...(other mes.)  BX     BY    Width  Height
1      Smp1+A1  0.4634  0.0582  ...              28.89  0.20  1.42   0.83
2      Smp1+A1  0.0705  0.0244  ...              72.40  0.35  2.33   32.16
3      Smp1+A2  0.0498  0.0566  ...              75.43  0.69  75.44  0.70
...
```

The reasons to choose such a simple text format for representing metadata is simplicity and flexibility. Plain text files are readable by any computer program and no sophisticated database engine, or database structure, is required to represent those data. Also, besides obligatory fields in the metadata sections, you can add as many key=value entries as you need to collect together the metadata required in your particular application. ZooImage will automatically read them and store them at the right place, available to you at any time during your analyses in ZooImage! That way, ZooImage is very flexible and capable to process many different kinds of data, even most exotic ones.

`zimCreate()` and `zimEdit()` call the associated metadata editor (by default, the one defined as `options(fileEditor = ....)`. By default, it is the same program as used by `fileEdit()` in the svMisc package. You can also use a spreadsheet, like Excel, Gnumeric, or OpenOffice Calc to edit these files. This is particularly useful for the tabular '[Data]' section, more comfortably edited in as a spreadsheed. Just save your file as dQuotetab-delimited text file when you have done and close the spreadsheet program (Excel won't allow ZooImage to access the .zim file when the file is opened as a spreadsheet). Just redefine `options(fileEditor = ...)` to use, e.g, Excel automatically with ZooImage (full path of the 'Excel.exe' file).

`zimMake()` creates one or more .zim files corresponding to the selected list of images provided in 'images', and allows for editing them one-by-one. It is the basic function for creating all .zim files manually for a set of images to be analyzed in ZooImage. See also `zieMake()` for an alternative, and automatic way to create all those .zim files.

`zimExtractAll()` and `zimUpdateAll()` work in conjonction with zipped TIFF image, as obtained by `zipImg()` and `zipImgAll()` (also done using `zidCompress`). In these .zip files, metadata is located in the zip archive comment. This comment is extracted into corresponding .zim file by `zimExtractAll()` for one or several zip archives. On the other hand, these comments are updated in the zip archive with latest information present in .zim files using `zimUpdateAll()`.

The last functions are auxiliary utilities to deal with .zim files (see also `zimList()`). `isZim()` simply checks if the file is a correct .zim file, checking first line of the file that must be 'ZI1-3' for ZooImage version 1-3. This routine returns `TRUE` or `FALSE` according to the result (the file extension is also checked if `check.ext = TRUE`).

Finally, `zimVerify()` is a very important function. It checks the validity and syntax of any .zim file. All required fields are checked. In case of an error, the function returns an explicit error message as a character string. On the other hand, if the verification process succeeds, the function returns a number corresponding to the number of objects whose measurements are recorded in the data table (for a '_dat1.zim' file), or '0' (zero, no measurements) for a '.zim' file containing only metadata.

`zimVerify()` checks for the presence of required fields. For .zim files: Section '[Image]' with 'Author', 'Hardware', 'Software' and 'ImageType' (for instance, "trans_16bits_gray" for a 16bit graylevels picture obtained by transparency, that is, using transmitted light) fields, section '[Fraction]' with 'Code' (A, B, C, ...), 'Min' and 'Max' (the minimum and maximum mesh sizes used to fraction the sample, or -1 if Min and or Max sieves are not used) and section '[Subsample]' with fields 'Subpart' (a number indicating how much of the fraction is actually digitized, for instance, 0.15 for 15% of the fraction), 'SubMethod' (volumetry, Motoda, etc.), 'CellPart' (the fraction of the digitizing cell actually covered by all images made), 'Replicates', 'VolIni' (the volume of seawater, in cubic meters or any of your favored unit, that was collected in the sea for this sample) and 'VolPrec', the precision at which 'VolIni' is measured, expressed in the same unit.

For '_dat1.zim' files, the function checks for the presence of all the previous fields, plus: '[Process]' section with fields 'Version' (version of the processing function), 'Method' (method used to process the images), 'MinSize', 'MaxSize' (the minimum and maximum ecd -equivalent circular diameter- of the particule to be considered and measured), 'Calibration' (data for gray levels calibration) and 'ProcessPixSize' (data for spatial calibration: size of one pixel in microns, or any of your favorite length unit). Column headers 'Item', 'Label', 'BX', 'BY', 'Height', 'Width', plus at least one additional measrurement are checked too. If `check.table = TRUE`, the function also tries to read the table of measurements and checks for its integrity (it takes longer for checking many large '_dat1.zim' files!).

Value

`zimCreate()`, `zimEdit()`, `zimMake()` are invoked for their side-effect of creating and/or editing .zim files on disk. `zimCreate()` returns `TRUE` invisibly, in case of successfull creation of all required .zim files, `FALSE` otherwise (details of problems are returned using warnings. The same mechanism (returning `TRUE` or `FALSE` invisibly, with detailled description of the problem in warnings) is used by `zimExtractAll()` and `zimUpdateAll()` also called for their side-effects of manipulating .zim/.zip files.

`isZim()` simply returns `TRUE` or `FALSE`.

`zimVerify()` returns an integer in case of successful verification of the .zim file. This integer represents the number of objects in the measurement table (zero, if there is no '[Data]' section in the file, see hereunder). In the case of an error during verification, the function returns a character string with explicit description of the problem.

`zimDatMakeFlowCAM()` and `zimDatMakeAllFlowCAM()` are, as you expect, special functions to transform FlowCAM metadata into dat1.zim formats. They return `TRUE` or `FALSE` invisibly, and issue warnings in case of problems.

Note

Developers have the opportunity to add custom fields (both sections and keys in these sections) to be checked by `zimVerify()`, in order to match specific uses of ZooImage. Since the verification of metadata is a critical step in the analysis, they are strongly encouraged to add such custom rules: the default checking procedure is very basic!

There are two possible ways to extend verification: adding fields in the list of required ones, or using a custom function. The second solution is more complex, but you have the possibility to check also the values associated with keys, where the first solution just check the presence of those keys, no mathers the values associated with them. You can combine both approaches.

To add required keys to be checked, just create an option(ZI.zim = list(....)) with a list of four components: "zim.required", "dat1.zim.required", "dat1.data.required" and "active". Put the list of sections (between square brackets) and keys that must be present in the '.zim' files in "zim.required" and those that must be present in the '_dat1.zim' files in "dat1.zim.required" components. Place a list of column headers that you need in the [Data] table in the "dat1.data.required" component of the list. The "active" component must be `TRUE` to activate the verification of these extra fields and column headers (otherwise, they are ignored). See the example hereunder.

To add your own verification rules, add a R function in the "verify" component of the list. That function should be defined as: `function(zimfile, ...) # Your code here` and it should return either "" (if no error found), or an explicit message in case of error(s).

Alternatively, you can completelly redefine `zimVerify()` in the "verify.all" component of the list. In this case, the other rules are completelly ignored, and you must perform the whole treatment in your function (start from the code of `zimVerify()` to make sure your own function has a similar behaviour!

Look at examples, and you will better understand how this works!

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`zieMake`, `zipImg`, `zidCompress`, , `fileEdit`

Examples

```
## Create a minimalist .zim file from current template
(zimfile <- paste(tempfile(), "zim", sep = "."))
zimCreate(zimfile, edit = FALSE)

## Display its content
if (interactive()) file.show(zimfile)

## List .zim files in the temporary directory
zimList(tempdir())

## Is this a correct .zim file?
isZim(zimfile)
zimVerify(zimfile) # Returns 0 => verification OK, with 0 records in [Data]
```

```
## The rest of this example is for programmers
## Add more required sections and keys for metadata verifications
## Add more required columns in the table of measurements
options(ZI.zim = list(active = TRUE,
    zim.required = c("[NewSection]", "requiredkey1", "requiredkey2"),
    dat1.zim.required = c("[PostProcess]", "requiredkey3"),
    dat1.data.required = c("Area", "Perim.", "Circ.", "Feret")))
try(zimVerify(zimfile)) # Of course, these new keys are missing!

## Now, inactivate these extra verifications without deleting them
oZI.zim <- getOption("ZI.zim")
oZI.zim$active <- FALSE
options(ZI.zim = oZI.zim)
rm(oZI.zim) # not needed any more
zimVerify(zimfile) # This time, extra verifications are not used any more => OK!

## Add some verification code to the existing verification procedure
options(ZI.zim = list(active = TRUE,
    verify = function (zimfile, ...) {
        # Your verification code here, for instance:
        Lines <- scan(zimfile, character(), sep = "\t", skip = 1, flush = TRUE,
      quiet = TRUE, comment.char = "#")
        ## Check if 'Code=B' or 'Code=C', using regular expression
        ## Extra spaces are allowed before and after '=', and after the value
        if (length(grep("^Code\\s*=\\s*[B|C]\\s*$", Lines)) == 0) {
            ## The condition is not matched!
            return("[Fraction] Code must be either 'B', or 'C'!")
        } else {
            ## Everything is fine: return an empty string
            return("")
        }
}))
try(zimVerify(zimfile)) # Since Code=A, verification fails!

## Reset original verification rules
options(ZI.zim = NULL)

## Erase the example .zim file
unlink(zimfile)
```

`zip` *Zip and unzip raw .tif images for ZooImage and embed metadata*

Description

Perform simple zip/unzip operations on images. Corresponding metadata from .zim files are embedded as zip comments.

Usage

```
zipImg(imagefile, zimfile = NULL, check.zim = TRUE, replace = FALSE,
    delete.source = FALSE)
zipImgAll(path = ".", images = NULL, check.zim = TRUE, replace = FALSE,
    delete.source = FALSE)
unzipImg(zipfile, replace = FALSE, delete.source = FALSE)
unzipImgAll(path = ".", zipfiles = NULL, replace = FALSE,
    delete.source = FALSE)
```

Arguments

imagefile	file path of the .tif image to compress.
zimfile	file path of the corresponding .zim file (calculated automatically if NULL, by default).
check.zim	do we verify the .zim file before zipping data?
replace	do we replace existing .zip files?
delete.source	should the original .tif file be deleted (the .zim file is never deleted)?
path	directory where .tif images to be zipped are located.
images	a list of .tif images to zip. If NULL, the list of all .tif images present in path is computed.
zipfile	a zipfile to unzip.
zipfiles	a list of zipfiles to unzip. If NULL, a list of all .zip files found is path is calculated.

Value

All these functions are designed to be run in batch mode. Problems are reported as warnings, and the function always returns TRUE or FALSE, depending if the process succeeds or fails. The xxxAll() functions are convenient wrapper around batch() to process several items in a row. Take care that, despite the functions possibly use internal R code to zip or unzip files, they still need the zip and unzip programs for injecting and extracting .zim files metadata in the .zip archive!

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

zipNoteAdd, zipNoteGet

Examples

```
## Create a fake example of two .tif images and their associated .zim files
testdir <- file.path(tempdir(), "ziptest")
dir.create(testdir)
file.copy(system.file("examples", "BIO.2000-05-05.p72.zid",
    package = "zooimage"), testdir)
curdir <-setwd(testdir)
```

```
unzip("BIO.2000-05-05.p72.zid", junkpaths = TRUE)
## Keep only first 3 image plus the .zim file
unlink("BIO.2000-05-05.p72.zid")
unlink("BIO.2000-05-05.p72_dat1.RData")
unlink(dir(pattern = "[.]jpg$")[-(1:3)])
## Rename .jpg images, pretending they are .tif images
jpgFiles <- dir(pattern = "[.]jpg$")
tifFiles <- sub("[.]jpg$", ".tif", jpgFiles)
file.rename(jpgFiles, tifFiles)
## Recreate the .zim file
zimData <- readLines("BIO.2000-05-05.p72+A_dat1.zim", n = 32)
zimFile <- "BIO.2000-05-05.p72+A_.zim"
writeLines(zimData, zimFile)
unlink("BIO.2000-05-05.p72+A_dat1.zim")
## Here is what we got...
dir()

## Zip first image...
(zipImg(tifFiles[1]))
## It is added in the _raw subdirectory
dir()
zipDir <- file.path(".", "_raw")
dir(zipDir)

## Now, zip all images in batch and delete sources
(zipImgAll(".", delete.source = TRUE))
dir() # .zim files are never deleted!
(zipFiles <- dir(zipDir, full.names = TRUE)) # All three are there
## Force delete of the .zim file
unlink(zimFile)

## Unzip first zip file...
#####(unzipImg(zipFiles[1]))
## The image and .zim file are recreated
dir()
## Check extracted .zim file
#####all(readLines(zimFile) == zimData)

## Unzip all images...
(unzipImgAll(zipDir, replace = TRUE, delete.source = TRUE))
## All original files are there...
dir()
## and the _raw subdir is now empty
dir(zipDir)

## Reset and clean up
setwd(curdir)
unlink(testdir, recursive = TRUE)
rm(testdir, curdir, jpgFiles, tifFiles, zimFile, zimData, zipDir, zipFiles)
```

ZIRes *Various function to calculate results ('ZIRes' object)*

Description

Having classified items in a 'ZIDat' object, these function calculate various statistic descriptors of whole samples (abundances, biomasses, size spectra) and they collect them together in a 'ZIRes' object.

Usage

```
processSample(x, sample, keep = NULL,
    detail = NULL, classes = "both", header = c("Abd", "Bio"),
    cells = NULL, biomass = NULL, breaks = NULL)
processSampleAll(path = ".", zidbfiles, ZIClass = NULL, keep = NULL,
    detail = NULL, classes = "both", header = c("Abd", "Bio"),
    cells = NULL, biomass = NULL, breaks = NULL)

## S3 method for class 'ZIRes'
print(x, ...)
## S3 method for class 'ZIRes'
rbind(..., deparse.level = 1)
```

Arguments

x	a 'ZIDat' object or similar data frame for `processSample()`, or a 'ZIRes' object for the other functions.
sample	the sample 'Id' to use for selecting items of one sample only, in case the object contains data for several samples. It should not be the case for 'ZIDat' objects, and you do not have to provide this argument then.
keep	a character vector with names of the levels to keep in the analysis for the classes, or `NULL` to keep them all.
detail	a character vector with names of classes for which to calculate separate statistics. The special levels `[other]` and `[total]` are also added. If `NULL`, only the total for the sample is returned.
classes	a character string with `"Class"` to use the manual classification for splitting particles in the sample, or `"Predicted"` to use automatic classification, or `"both"` (default) to use Class in priority, but falling back to Predicted for particles whose Class is not defined. One can also specify the name of another factor variable in `x`.
header	a character vector with one or two strings to use as headers for, respectively, abundances and biomasses.
cells	the path to an .rds file containing cells counting models, as used by `cellCompute()`.
biomass	a specification for biomass conversion. Can be `NULL` (by default) for turning off biomass calculation, or a numeric vector of three values P1, P2 and P3 (same biomass for all classes with Bio = P1 * ECD^P3 + P2), or a data frame with the name of classes in first column, P1, P2 and P3 in columns 2 to 4. You must also provide one line with class `"[other]"` that is used for biomass claculation for classes not otherwise specified.

breaks	either NULL (default) to turn off size spectra calculations or a numeric vector for ECD classes breaks.
path	the path containing your ZIDB or ZID files to use for samples processing.
zidbfiles	a list of ZIDB or ZID files to process in batch.
ZIClass	a 'ZIClass' object to use to classify particles during the process of your samples.
...	further arguments passed to the methods
deparse.level	integer controlling the way labels are constructed for non-matrix-like arguments (defined in the generic, but not used here).

Author(s)

Philippe Grosjean & Kevin Denis

See Also

ZIClass

Examples

```
##TODO...
```

---

zis *Manipulate .zis files (ZooImage Samples description)*

Description

Additional data concerning the samples are collected together in .zis files. These functions manipulate such .zis files.

Usage

```
zisCreate(zisfile, template = NULL, edit = TRUE, editor =
    getOption("fileEditor"), wait = FALSE)
zisEdit(zisfile, editor = getOption("fileEditor"), wait = FALSE, ...)
zisRead(zisfile = "Description.zis", expected.sections =
    c("Description", "Series", "Cruises", "Stations", "Samples"))
```

Arguments

zisfile	the name of the .zis file to manipulate (usually, "Description.zis").
template	a .zis template to start with, if the .zis file does not exist yet.
edit	do we edit the .zis file after its creation?
editor	the program to use to edit the .zis file.
wait	do we wait that edition of file is done?
expected.sections	list of the sections that must be present in the .zis file.
...	further arguments to pass to fileEdit().

Value

`zisRead()` returns a 'ZIDesc' object containing all the data in the .zis file, or, in case of fealure (detailed in a warning), it returns `NULL`. The two other functions return `TRUE` or `FALSE` invisibly, depending if the .zis file could be created/edited or not.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`zidCompress`, `fileEdit`

Examples

```
## Read content of the example zis file
zisFile <- system.file("examples", "Description.zis", package = "zooimage")
zisData <- zisRead(zisFile)
zisData  # These are data for samples
attr(zisData, "metadata")  # Further data (for the series) here

## Create a new .zis file in tempdir using the example .zis file as template
zisNew <- file.path(tempdir(), "Description.zis")
zisCreate(zisNew, template = zisFile, edit = FALSE)  # One can edit it here too

## Edit the new file, and wait edition is completed
zisEdit(zisNew, wait = TRUE)
## It contains:
if (interactive()) file.show(zisNew, title = basename(zisNew), delete.file = TRUE)

## Clean up
rm(zisFile, zisData, zisNew)
```

ZITrain *Manipulate training and testsets 'ZITrain'/'ZITest' objects*

Description

'ZITrain' contain a hierarchy of classes (taxonomic or not) and a link to a series of items belonging to these classes. It can be obtained after manual or automatic classification of various objects from .zid or .zidb files. 'ZITest' objects are almost identical, but with a constraint on the classes that must match the ones of the reference 'ZItrain' or 'ZIClass' object (a necessity to allow for comparisons)!

Usage

```
prepareTrain(traindir, zidbfiles, template = c("[Basic]", "[Detailed]",
    "[Very detailed]"), classes = NULL, ...)
addToTrain(traindir, zidbfiles, classes = NULL, ...)
```

```
getTrain(traindir, creator = NULL, desc = NULL, keep_ = FALSE, na.rm = FALSE)

prepareTest(testdir, zidbfiles, template, classes = NULL, ...)
addToTest(testdir, zidbfiles, classes = NULL, ...)
getTest(testdir, creator = NULL, desc = NULL, keep_ = NA, na.rm = FALSE)

cellModel(train, traindir, class, method = "mda")
cellCompute(data, cellModels)
cellCount(traindir, class, reset = FALSE)

template(object, ...)
## Default S3 method:
template(object, ...)

recode(object, ...)
## S3 method for class 'ZITrain'
recode(object, new.levels, depth, ...)
## S3 method for class 'ZITest'
recode(object, new.levels, depth, ...)

contextSelection()
```

Arguments

traindir	the root directory of the training set.
testdir	the root directory of the test set.
zidbfiles	.zidb files or .zid files to use for data and vignettes in the training set.
template	file containing subdirectories template to use for creating classes in the training or test set. Either a defaut template between [], or the name of a .zic file, or a template extracted from a 'ZITrain' or 'ZIClass' object using `template(object)` (with the `add.others` argument to TRUE for `prepareTest()` and to `FALSE` for `prepareTrain()`)
classes	if vignettes are already classified in the zid(b) files, should they be sorted that way in the created training or test set? If not `NULL`, indicate the name of the classification column (usually, `Class` for manual classification or `Predicted` for automatic predictions). This can also be a 'ZIClass' or 'mlearning' object that will be used for classification of the particles first, ... or it can be a function that does the classification.
creator	name of the author of this classification (or the method, if done automatically).
desc	a short description of this manual classification.
keep_	do we keep items in the '_' subdirectory (corresponding to unclassified ones)? Default to `FALSE` in `getTrain()` and to `NA` for `getTest()`, which transforms all items in the '_' or one of its subdirectories into missing data.
na.rm	do we remove item with missing data? By default, not.
train	a ZITrain file to use for building the model.
class	a character string with the name of the class to process.

method	a character string with the nazme of the predictive method to use: "lm", "lda" or "mda" (by default).
data	a sample containing the particles to count.
cellModels	the file containing the models for cells countings.
reset	do we reset excisting counts for that class? By default, no.
object	a 'ZITrain' or 'ZITest' object. For prepareTest(), a 'ZITrain' or 'ZIClass' object to use as reference to determine the classes to make.
new.levels	a character string of same length as the levels of object$Class with the labels of the new levels.
depth	the depth in the hierachy of the classes as in the "path" attribute of the object to use for recoding classes. If this argument is provided, new.levels is ignored and recomputed (and a warning is issued if both arguments are provided).
...	further arguments passed to the method. For prepareXXX() and addToXXX(), it is further arguments passed to the prediction function provided in classes, or to the predict() method for 'ZIClass' or 'mlearning' objects.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

ZIClass

Examples

```
##TODO...
```

Index