Package 'nprcgenekeepr'

July 26, 2026

Type Package

Title Genetic Tools for Colony Management

Version 2.0.0

Description Provides genetic tools for colony management and is a derivation of the work in Amanda Vinson and Michael J Raboin (2015) <https://pmc.ncbi.nlm.nih.gov/articles/PMC4671785/> ``A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques ('Macaca' 'mulatta')''. It provides a 'Shiny' application with an exposed API. The application supports five groups of functions: (1) Quality control of studbooks contained in text files or 'Excel' workbooks and of pedigrees within 'LabKey' Electronic Health Records (EHR); (2) Creation of pedigrees from a list of animals using the 'LabKey' EHR integration; (3) Creation and display of an age by sex pyramid plot of the living animals within the designated pedigree; (4) Generation of genetic value analysis reports; and (5) Creation of potential breeding groups with and without proscribed sex ratios and defined maximum kinships.

URL https://rmsharp.github.io/nprcgenekeepr/, https://github.com/rmsharp/nprcgenekeepr

BugReports https://github.com/rmsharp/nprcgenekeepr/issues

Depends R (>= 4.1.0)

Imports anytime, bslib, data.table, DT, futile.logger, ggplot2, htmlTable, lifecycle, lubridate, Matrix, openxlsx, plotrix, readxl, Rlabkey (>= 3.2.0), sessioninfo, shiny, stringi, utils

Suggests covr, dplyr, grid, shinytest2, kableExtra, knitr, markdown, mockery, pkgdown, png, rmarkdown, roxygen2 (>= 8.0.0), shinyBS, shinyWidgets, spelling, testthat, withr

Language en-US

Encoding UTF-8

License MIT + file LICENSE

LazyData TRUE

VignetteBuilder knitr

Config/renv/profiles/dev/dependencies checkhelper, devtools, httpuv, qpdf, markdown, pak, roxygen2, spelling, usethis

Config/Needs/website quarto

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Michael Raboin [aut], Terry Therneau [aut], Amanda Vinson [aut, dtc], R. Mark Sharp [aut, cre, cph, dtc] (ORCID: <https://orcid.org/0000-0002-6170-6942>), Matthew Schultz [aut] (ORCID: <https://orcid.org/0000-0001-5103-4305>), Southwest National Primate Research Center NIH grant P51 RR13986 [fnd], Oregon National Primate Research Center grant P51 OD011092 [fnd]

Maintainer R. Mark Sharp <rmsharp@me.com>

Repository CRAN

Date/Publication 2026-07-26 10:50:02 UTC

Contents

addAnimalsWithNoRelative *Add an NA value for animals with no relative*

Description

This allows kin to be used with setdiff when there are no relatives otherwise an error would occur because kin[['animal_with_no_relative']] would not be found. See the following: in **groupAddAssign**

Usage

```
addAnimalsWithNoRelative(kin, candidates)
```

Arguments

kin	named list of high-kinship relatives, as produced by getAnimalsWithHighKinship, where each name is an animal Id and each value is a character vector of the Ids sharing a kinship value at or above the threshold.
candidates	character vector of IDs of the animals available for use in the group.

Details

```
available[[i]] <- setdiff(available[[i]], kin[[id]])
```

Value

The named list of high-kinship relatives (one element per animal Id, each value a character vector of that Id's high-kinship relatives) with an added element set to NA for each candidate that has no relative.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
ped <- qcStudbook(qcPed,
  minParentAge = 2.0, reportChanges = FALSE,
  reportErrors = FALSE
)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1L)
currentGroups[[1]] <- examplePedigree$id[1:3]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
  ignore = list(c("F", "F")), minAge = 1.0
)
# Filtering out candidates related to current group members
```

```
conflicts <- unique(c(
  unlist(kin[unlist(currentGroups)]),
  unlist(currentGroups)
))
candidates <- setdiff(candidates, conflicts)
kin <- addAnimalsWithNoRelative(kin, candidates)
length(kin) # should be 259
kin[["0DAV0I"]] # should have 34 IDs
```

`addBackSecondParents` *Add back single parents trimmed pedigree*

Description

Uses the `ped` dataframe, which has full complement of parents and the `uPed` dataframe, which has all uninformative parents removed to add back single parents to the `uPed` dataframe where one parent is known. The parents are added back to the pedigree as an ID record with NA for both sire and dam of the added back ID.

Usage

```
addBackSecondParents(uPed, ped)
```

Arguments

`uPed`	a trimmed pedigree dataframe with uninformative founders removed.
`ped`	a trimmed pedigree

Value

A dataframe with pedigree with single parents added.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
probands <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
p <- addBackSecondParents(p, ped)
nrow(p)
```

addGenotype *Add genotype data to pedigree file*

Description

Assumes genotype has been opened by checkGenotypeFile

Usage

```
addGenotype(ped, genotype)
```

Arguments

ped	pedigree dataframe. ped is to be provided by qcStudbook so it is not checked.
genotype	genotype dataframe. genotype is to be provided by checkGenotypeFile so it is not checked.

Details

The two allele columns are coerced to character internally so the name-keyed allele dictionary is both built and indexed by allele label. This keeps the integer encoding consistent even when the allele columns are supplied as factors (a factor would otherwise be indexed by its integer codes).

Value

A pedigree object with genotype data added.

Examples

```
library(nprcgenekeepr)
rhesusPedigree <- nprcgenekeepr::rhesusPedigree
rhesusGenotypes <- nprcgenekeepr::rhesusGenotypes
pedWithGenotypes <- addGenotype(
  ped = rhesusPedigree,
  genotype = rhesusGenotypes
)
```

addIdRecords *Add ego records with NA parent IDs*

Description

Add ego records with NA parent IDs

Usage

```
addIdRecords(ids, fullPed, partialPed)
```

Arguments

ids	character vector of IDs to be added as Ego records having NAs for parent IDs
fullPed	a trimmed pedigree
partialPed	a trimmed pedigree dataframe with uninformative founders removed.

Value

Pedigree with Ego records added having NAs for parent IDs

Examples

```
uPedOne <- data.frame(
  id = c("d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c("s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c("d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
pedOne <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
pedOne[!pedOne$id %in% uPedOne$id, ]
newPed <- addIdRecords(ids = "s1", pedOne, uPedOne)
pedOne[!pedOne$id %in% newPed$id, ]
newPed[newPed$id == "s1", ]
```

addParents *Add parents*

Description

Pedigree curation function Given a pedigree, find any IDs listed in the "sire" or "dam" columns that lack their own line entry and generate one.

Usage

```
addParents(ped)
```

Arguments

ped	The pedigree information in data.frame format

Details

This must be run after to addUIds since the IDs made there are used by addParents

Value

An updated pedigree with entries added as necessary. Entries have the id and sex specified; all remaining columns are filled with NA.

Examples

```
pedTwo <- data.frame(
  id = c("d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, NA, NA, "s1", "s1", "s2", "s2"),
  dam = c(NA, NA, NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
newPed <- addParents(pedTwo)
newPed
```

addSexAndAgeToGroup *Build a group data frame with ID, sex, and age*

Description

Build a group data frame with ID, sex, and age

Usage

```
addSexAndAgeToGroup(ids, ped)
```

Arguments

ids	character vector of animal IDs
ped	The pedigree information in data.frame format

Details

An empty ids vector yields a zero-row data frame that still contains all three columns (ids, sex, age), with sex an empty factor, so the returned schema does not depend on the number of ids supplied.

Value

Dataframe with Id, Sex, and Current Age

Examples

```
library(nprcgenekeepr)
data("qcBreeders")
data("qcPed")
df <- addSexAndAgeToGroup(ids = qcBreeders, ped = qcPed)
head(df)
```

---

addUIds *Add placeholder IDs for unknown parents*

Description

This must be run prior to addParents since the IDs made herein are used by addParents

Usage

```
addUIds(ped, format = getAutoIdFormat())
```

Arguments

ped	datatable that is the Pedigree. It contains pedigree information. The fields sire and dam are required.
format	sprintf template for the generated placeholder IDs; defaults to getAutoIdFormat() ("U%04d").

Details

The generated placeholder IDs default to the form Unnnn (a leading "U" plus a zero-padded integer), so they are alphanumeric and never contain a period ("."), honoring the ID rule enforced at data input by qcStudbook. The format is configurable via setAutoIdFormat (default "U%04d").

Value

The updated pedigree with partial parentage removed.

Examples

```
pedTwo <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c("d0", "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
  stringsAsFactors = FALSE
)
newPed <- addUIds(pedTwo)
newPed[newPed$id == "s1", ]
pedThree <-
  data.frame(
    id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
    sire = c("s0", "s0", "s4", NA, "s1", "s1", "s2", "s2"),
    dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
    sex = c("M", "F", "M", "F", "F", "F", "F", "M"),
    stringsAsFactors = FALSE
  )
newPed <- addUIds(pedThree)
newPed[newPed$id == "s1", ]
```

---

alleleFreq *Count each allele in a vector*

Description

Part of Genetic Value Analysis

Usage

```
alleleFreq(alleles, ids = NULL)
```

Arguments

alleles	an integer vector of alleles in the population
ids	character vector of IDs indicating to which animal each allele in alleles belongs.

Details

If ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

Value

A data.frame with columns `allele` and `freq`. This is a table of allele counts within the population.

Examples

```
library(nprcgenekeepr)
data("ped1Alleles")
ids <- ped1Alleles$id
alleles <- ped1Alleles[, !(names(ped1Alleles) %in% c("id", "parent"))]
aF <- alleleFreq(alleles[[1]], ids = NULL)
aF[aF$freq >= 10, ]
```

---

applyKinshipOverrides *Apply outside-information kinship overrides to a kinship matrix*

Description

Writes pairwise kinship coefficients from outside information into a computed kinship matrix, replacing the pedigree-derived value for the named pairs. Each `(id1, id2, kinship)` row sets both `kmat[id1, id2]` and its symmetric twin `kmat[id2, id1]`; all other cells are unchanged. This is a direct cell replacement – it does not propagate to descendant rows.

Usage

```
applyKinshipOverrides(kmat, overrides)
```

Arguments

`kmat`	a dense, symmetric, id-named numeric kinship matrix.
`overrides`	data.frame of overrides (`id1`, `id2`, `kinship`); `NULL` or a zero-row frame is a no-op returning `kmat` unchanged. Validated with `checkKinshipOverrides`.

Details

`kmat` must be a dense, symmetric, id-named base R matrix (the object `kinship` returns); a sparse `Matrix` object is out of contract. The function is strict: it `stop()`s on an id absent from the matrix and on a value above the exact positive-semi-definiteness bound `sqrt(kmat[id1, id1] * kmat[id2, id2])`. Soft, run-preserving handling of ids outside the analysis set is the caller's responsibility – `reportGV` warn-drops non-member ids before calling this. `kinship()` itself is never modified (it has several callers, including two simulations that must not take current-kinship overrides).

Value

`kmat` with the override cells replaced (symmetric).

Examples

```
ped <- nprcgenekeepr::qcPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
overrides <- data.frame(
  id1 = ped$id[1], id2 = ped$id[2], kinship = 0.25,
  stringsAsFactors = FALSE
)
kmat <- applyKinshipOverrides(kmat, overrides)
```

`appServer` *Main Application Server for nprcgenekeepr*

Description

Server function for the main GeneKeepR Shiny application. Initializes all modules and manages shared reactive state between them.

Usage

```
appServer(input, output, session)
```

Arguments

`input`	Shiny input object
`output`	Shiny output object
`session`	Shiny session object

Details

The server handles:

- Configuration loading from site-specific config files
- Navigation button handlers for the home page
- Dynamic tab management for QC errors and changed columns
- Module initialization and data flow between modules

Value

No return value, called for side effects. As a 'Shiny' server function, `appServer()` is invoked by the 'Shiny' runtime to wire up the application's reactive outputs, observers, and module servers for a running GeneKeepR session.

See Also

appUI for the corresponding UI function

modInputServer for data input module

modPedigreeServer for pedigree browser module

modGeneticValueServer for genetic value analysis

shouldShowChangedColsTab for changed columns tab logic

loadSiteConfig for site configuration loading

---

appUI *Main Application UI for nprcgenekeepr*

Description

Main Application UI for nprcgenekeepr

Usage

```
appUI(siteInfo = NULL)
```

Arguments

siteInfo	Named list of site configuration as returned by getSiteInfo; defaults to NULL, in which case it is resolved internally via getSiteInfo(expectConfigFile = FALSE). A present-but-malformed site-config file makes that call fail; the failure is caught and logged (futile.logger::flog.warn) rather than propagating, and the UI falls back to hiding the ORIP Reporting tab. Its center and configFile elements gate the Oregon (ONPRC)-specific ORIP Reporting tab, which is shown only for an actual ONPRC configuration (see shouldShowOripTab).

Value

A shiny.tag.list object (as produced by shiny::tagList()) describing the complete GeneKeepR user interface; pass it as the ui argument to shiny::shinyApp() or shiny::runApp().

`assignAlleles` *Assign parent alleles randomly*

Description

Assign parent alleles randomly

Usage

```
assignAlleles(alleles, parentType, parent, id, n)
```

Arguments

`alleles`	a list with a list `alleles$alleles`, which is a list of list containing the alleles for each individual's sire and dam that have been assigned thus far and `alleles$counter` that is the counter used to track the lists of`alleles$alleles`.
`parentType`	character vector of length one with value of `"sire"` or `"dam"`.
`parent`	either `ped[id, "sire"]` or `ped[id, "dam"]`.
`id`	character vector of length one containing the animal ID
`n`	integer indicating the number of iterations to simulate.

Value

The original list `alleles` passed into the function with newly randomly assigned alleles to each `id` based on dam and sire genotypes.

Examples

```
alleles <- list(alleles = list(), counter = 1)
alleles <- assignAlleles(alleles,
  parentType = "sire", parent = NA,
  id = "o1", n = 4
)
alleles
alleles <- assignAlleles(alleles,
  parentType = "dam", parent = NA,
  id = "o1", n = 4
)
alleles
```

calcA *Count each individual's rare alleles per simulation*

Description

Part of Genetic Value Analysis

Usage

```
calcA(alleles, threshold = 1L, byID = FALSE)
```

Arguments

alleles	a matrix with {V1 ... Vn, id, parent} providing the alleles an animal received during each simulation. The first n columns provide the alleles; the final two columns provide the animal ID and the parent the allele came from.
threshold	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.
byID	logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).

Value

A matrix with named rows indicating the number of unique alleles an animal had during each round of simulation (indicated in columns).

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

MacCluer JW, et al. 1986. Pedigree analysis by computer simulation. Zoo Biology 5:147-160.

See Also

Other genetic value analysis: `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`

Examples

```
library(nprcgenekeepr)
rare <- calcA(nprcgenekeepr::ped1Alleles, threshold = 3, byID = FALSE)
```

calcAge *Calculate animal ages*

Description

Part of Pedigree Curation

Usage

```
calcAge(birth, exit)
```

Arguments

birth	Date vector of birth dates
exit	Date vector of exit dates.

Details

Given vectors of birth and exit dates, calculate an individuals age. If no exit date is provided, the calculation is based on the current date.

Value

A numeric vector (NA allowed) indicating age in decimal years from "birth" to "exit" or the current date if "exit" is NA.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
originalAge <- qcPed$age ## ages calculated at time of data collection
currentAge <- calcAge(qcPed$birth, qcPed$exit) ## assumes no changes in
## colony
```

calcFE *Calculate founder equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFE(ped)
```

Arguments

ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population).

Details

The pedigree must have no partial parentage (every animal has both parents known or both unknown); calcFE stops with an error otherwise.

Value

The founder equivalents FE = 1 / sum(p ^ 2), where p is the vector of founder mean contributions to the current descendants.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: calcA(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGUSE(), calcGeneDiversity(), calcNeSexRatio(), calcNeVariance(), calcRetention()

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
pedFactors <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = TRUE
)
pedFactors["gen"] <- findGeneration(
  pedFactors$id, pedFactors$sire,
  pedFactors$dam
)
pedFactors$population <- getGVPopulation(pedFactors, NULL)
fe <- calcFE(ped)
feFactors <- calcFE(pedFactors)
```

`calcFEFG` *Calculate founder equivalents and founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFEFG(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). The pedigree must have no partial parentage (every animal has both parents known or both unknown); `calcFEFG` stops with an error otherwise.
`alleles`	dataframe contains an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

The list containing the founder equivalents, `FE = 1 / sum(p ^ 2)`, and the founder genome equivalents, `FG = 1 / sum( (p ^ 2) / r)` where `p` is the vector of founder mean contributions to the current descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

`FE` is deterministic and always returned. `FG` is `NA` (with a warning) when a contributing founder (`p > 0`) is retained in zero of the gene-drop iterations (`r == 0`), which would otherwise collapse `FG` silently to 0; raise the number of iterations. See `calcFGSE` for the sampling standard error of `FG`.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`

Examples

```
data(lacy1989Ped)
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy

library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
```

```
alleles <- lacy1989PedAlleles
pedFactors <- data.frame(
  id = as.factor(ped$id),
  sire = as.factor(ped$sire),
  dam = as.factor(ped$dam),
  gen = ped$gen,
  population = ped$population,
  stringsAsFactors = TRUE
)
allelesFactors <- geneDrop(pedFactors$id, pedFactors$sire, pedFactors$dam,
  pedFactors$gen,
  genotype = NULL, n = 1000,
  updateProgress = NULL
)
feFg <- calcFEFG(ped, alleles)
feFgFactors <- calcFEFG(pedFactors, allelesFactors)
```

`calcFG` *Calculate founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFG(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). The pedigree must have no partial parentage (every animal has both parents known or both unknown); `calcFG` stops with an error otherwise.
`alleles`	dataframe contains an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

The founder genome equivalents, `FG = 1 / sum( (p ^ 2) / r)` where `p` is the vector of founder mean contributions to the current descendants and `r` is the mean number of founder alleles retained in the gene dropping experiment.

Returns `NA` with a warning when a contributing founder (`p > 0`) is retained in zero of the gene-drop iterations (`r == 0`): that term is `p^2 / 0 = Inf`, which would otherwise collapse `FG` silently to 0. Raise the number of iterations. See `calcFGSE` for the sampling standard error of `FG`.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFGSE(), calcGU(), calcGUSE(), calcGeneDiversity(), calcNeSexRatio(), calcNeVariance(), calcRetention()

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy

library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
pedFactors <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = TRUE
)
pedFactors["gen"] <- findGeneration(
  pedFactors$id, pedFactors$sire,
  pedFactors$dam
)
pedFactors$population <- getGVPopulation(pedFactors, NULL)
alleles <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = NULL,
  n = 1000, updateProgress = NULL
)
allelesFactors <- geneDrop(pedFactors$id, pedFactors$sire, pedFactors$dam,
  pedFactors$gen,
  genotype = NULL, n = 1000,
  updateProgress = NULL
)
fg <- calcFG(ped, alleles)
fgFactors <- calcFG(pedFactors, allelesFactors)
```

---

calcFGSE *Calculate the standard error of founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcFGSE(ped, alleles)
```

Arguments

ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). The pedigree must have no partial parentage (every animal has both parents known or both unknown); calcFGSE stops with an error otherwise.
alleles	dataframe containing an AlleleTable: one column per gene-drop iteration, followed by an id column and a parent column. Produced by geneDrop(); the same input calcFG takes.

Details

Founder genome equivalents (calcFG) is a Monte Carlo estimate: FG = 1 / sum(p^2 / r), where the founder contributions p are deterministic but the mean allelic retention values r (calcRetention) are averages over the gene-drop iterations, so FG carries sampling error that shrinks as the number of iterations grows. Unlike genome uniqueness (a mean, whose standard error is a column variance), FG is a nonlinear function of r, so its standard error is obtained by the delta method (first-order linearization).

With S = sum(p_f^2 / r_f) and FG = 1 / S, the gradient is dFG/dr_f = FG^2 * p_f^2 / r_f^2. Writing R for the founder-by- iteration retention matrix (each column an independent gene drop), the influence series y_k = sum_f (dFG/dr_f) * R[f, k] has sd(y) / sqrt(K) equal to the full delta-method standard error, including the within-iteration covariance among founders. This influence form is used because it folds in that covariance automatically and never forms the founder-by-founder covariance matrix.

Founders are matched between p and r by name (not position), so the result is correct even when the founders are not in sorted pedigree order.

A contributing founder (p > 0) that is retained in zero of the iterations (r == 0) makes FG undefined (the same degeneracy that calcFG now reports as NA); in that case this function returns NA with a warning advising more iterations. Founders that do not contribute to the current population (p == 0) are dropped, so the standard error refers to exactly the founder set FG is computed from.

Value

A single numeric value: the Monte Carlo sampling standard error of the colony founder-genome-equivalent estimate, on the same scale as calcFG. NA (with a warning) when a contributing founder has zero retention.

See Also

calcFG, calcFEFG, calcRetention, calcGUSE, reportGV

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcGU(), calcGUSE(), calcGeneDiversity(), calcNeSexRatio(), calcNeVariance(), calcRetention()

Examples

```
library(nprcgenekeepr)
data("lacy1989Ped")
data("lacy1989PedAlleles")
calcFGSE(lacy1989Ped, lacy1989PedAlleles)
```

`calcGeneDiversity` *Calculate gene diversity from founder genome equivalents*

Description

Part of the Genetic Value Analysis

Usage

```
calcGeneDiversity(fg)
```

Arguments

`fg`	Founder genome equivalents scalar, as returned by `calcFG()` or the `$FG` element of `calcFEFG()`. `NA` yields `NA`.

Details

Gene diversity is the expected heterozygosity retained relative to the founding gene pool, `GD = 1 - 1 / (2 * FG)`, where `FG` is the founder genome equivalents (see `calcFG`). It summarizes how much of the founders' allelic diversity still survives: 0 means none is retained, and it approaches (never reaches) 1 as `FG` grows.

`GD` is a diversity proportion, not a count of effective individuals, and it is computed over the same analysis set as `FG`. `NA` propagates: when `FG` is `NA` (the zero-retention degeneracy `calcFG()` reports), `GD` is `NA`.

Value

The gene diversity `GD = 1 - 1 / (2 * fg)`: a single number in `[0, 1)`, or `NA` when `fg` is `NA`.

References

Gene diversity is derived here from the founder genome equivalents (`calcFG`) of Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`

Examples

```
calcGeneDiversity(20) # 0.975
calcGeneDiversity(52.75) # gene diversity at the qcPed FG
```

`calcGU` *Calculate genome uniqueness for each population ID*

Description

Part of Genetic Value Analysis

Usage

```
calcGU(alleles, threshold = 1L, byID = FALSE, pop = NULL)
```

Arguments

`alleles`	dataframe of containing an `AlleleTable`. This is a table of allele information produced by `geneDrop()`. An AlleleTable contains information about alleles an ego has inherited. It contains the following columns: • `V1` — Unnamed integer column representing allele 1. • `V2` — Unnamed integer column representing allele 2. • `...` — Unnamed integer columns representing alleles. • `Vn` — Unnamed integer column representing the nth column. • `id` — A character vector of IDs for a set of animals. • `parent` — A factor with levels of sire and dam.
`threshold`	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.
`byID`	logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
`pop`	character vector with animal IDs to consider as the population of interest, otherwise all animals will be considered. The default is NULL.

Details

The following functions calculate genome uniqueness according to the equation described in Ballou & Lacy.

It should be noted, however that this function differs slightly in that it does not distinguish between founders and non-founders in calculating the statistic.

Ballou & Lacy describe genome uniqueness as "the proportion of simulations in which an individual receives the only copy of a founder allele." We have interpreted this as meaning that genome uniqueness should only be calculated for living, non-founder animals. Alleles possessed by living founders are not considered when calculating genome uniqueness.

We have a differing view on this, since a living founder can still contribute to the population. The function below calculates genome uniqueness for all living animals and considers all alleles. It does not ignore living founders and their alleles.

Our results for genome uniqueness will, therefore differ slightly from those returned by Pedscope. Pedscope calculates genome uniqueness only for non-founders and ignores the contribution of any founders in the population. This will cause Pedscope's genome uniqueness estimates to possibly be slightly higher for non-founders than what this function will calculate.

The estimates of genome uniqueness for founders within the population calculated by this function should match the "founder genome uniqueness" measure calculated by Pedscope.

Note that `calcGU` computes this statistic for every animal and still includes living founders' alleles; it is unchanged. The genetic value report (`reportGV`) separately applies a colony decline-to-credit policy that reports genome uniqueness as 0 for unknown-origin both-unknown ("Undetermined") animals.

Value

Dataframe `rows: id, col:` gu A single-column table of genome uniqueness values as percentages. Rownames are set to 'id' values that are part of the population.

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

MacCluer JW, et al. 1986. Pedigree analysis by computer simulation. Zoo Biology 5:147-160.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGUSE()`, `calcGeneDiversity()`, `calcNeSexRatio()`, `calcNeVariance()`, `calcRetention()`

Examples

```
library(nprcgenekeepr)
ped1Alleles <- nprcgenekeepr::ped1Alleles
gu_1 <- calcGU(ped1Alleles, threshold = 1, byID = FALSE, pop = NULL)
gu_2 <- calcGU(ped1Alleles, threshold = 3, byID = FALSE, pop = NULL)
gu_3 <- calcGU(ped1Alleles,
  threshold = 3, byID = FALSE,
  pop = ped1Alleles$id[20:60]
)
```

`calcGUSE` *Calculate the standard error of genome uniqueness*

Description

Part of Genetic Value Analysis

Usage

```
calcGUSE(alleles, threshold = 1L, byID = FALSE, pop = NULL)
```

Arguments

`alleles`	dataframe containing an `AlleleTable` (the same input `calcGU` takes): one integer column per gene-drop iteration, followed by an `id` column and a `parent` column. Produced by `geneDrop()`.
`threshold`	an integer indicating the maximum number of copies of an allele that can be present in the population for it to be considered rare. Default is 1.
`byID`	logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
`pop`	character vector with animal IDs to consider as the population of interest, otherwise all animals will be considered. The default is NULL.

Details

Genome uniqueness (`calcGU`) is a Monte Carlo estimate: it is the average, over the gene-drop iterations, of the proportion of an animal's two allele copies that are population-rare. Because it is an average over independent simulated iterations, it carries sampling error that shrinks as the number of iterations grows.

For animal `i` let `m_ik = rare[i, k] / 2` be the per-iteration value in iteration `k` (so the mean of `m_ik` over the `K` iterations equals `gu_i / 100`). This function returns the exact Monte Carlo standard error of that mean, on the same percentage scale as `calcGU`:

$$guSE_i = 100 \times \sqrt{\frac{var(m_{i.})}{K}}$$

The standard error is computed from the same per-iteration rare-allele matrix (`calcA`) that `calcGU` averages, so it is correct for any `threshold` / `byID` without a closed-form approximation. An animal whose rare-allele count does not vary across iterations has a standard error of 0.

Value

Dataframe `rows: id, col: guSE` A single-column table of genome-uniqueness standard errors as percentages. Rownames are set to 'id' values that are part of the population.

See Also

calcGU, calcA, reportGV

Other genetic value analysis: calcA(), calcFE(), calcFEFG(), calcFG(), calcFGSE(), calcGU(), calcGeneDiversity(), calcNeSexRatio(), calcNeVariance(), calcRetention()

Examples

```
library(nprcgenekeepr)
ped1Alleles <- nprcgenekeepr::ped1Alleles
guSE <- calcGUSE(ped1Alleles, threshold = 3, byID = FALSE, pop = NULL)
```

---

calcNeSexRatio *Calculate the demographic sex-ratio effective population size*

Description

Part of the Genetic Value Analysis

Usage

```
calcNeSexRatio(ped)
```

Arguments

ped	Pedigree data.frame with id, sire, dam, and sex; exit is used to identify living animals when present.

Details

The sex-ratio effective size, Ne = 4 * nMale * nFemale / (nMale + nFemale), is the effective population size implied by an unequal breeding sex ratio, where nMale and nFemale are the numbers of current living breeders that are known male and known female. It quantifies the diversity lost when many of one sex are bred to few of the other (as in a harem colony): it equals the census count when the sexes are balanced and falls toward four times the rarer sex as the ratio skews.

The breeders are the current living breeders of ped (living animals that appear as a sire or dam, excluding auto-generated unknown parents), independent of which animals are selected as probands – a different population than the analysis-set founder statistics (calcFE, calcFG, calcGeneDiversity). Only animals with a known sex ("M" or "F") are counted; unknown and hermaphrodite breeders are excluded. When either sex is absent among the living breeders (nMale == 0 or nFemale == 0, including no living breeders at all), the result is 0: a single breeding sex contributes no diversity from sex balance.

Like all effective-size estimators this idealizes a Wright-Fisher population (constant size, discrete generations, random union of gametes within each sex); a managed colony departs from those assumptions, so read the result as a sex-ratio index rather than a literal head count.

Value

The sex-ratio effective size, a single non-negative number; `0` when either breeding sex is absent among the living breeders.

References

Crow, J. F. and Kimura, M. (1970) *An Introduction to Population Genetics Theory*. Harper and Row, New York.

See Also

`calcGeneDiversity`

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcNeVariance()`, `calcRetention()`

Examples

```
ped <- data.frame(
  id = c("s1", "d1", "d2", "k1", "k2"),
  sire = c(NA, NA, NA, "s1", "s1"),
  dam = c(NA, NA, NA, "d1", "d2"),
  sex = c("M", "F", "F", "M", "F"),
  exit = c(NA, NA, NA, NA, NA),
  stringsAsFactors = FALSE
)
calcNeSexRatio(ped) # 4 * 1 * 2 / (1 + 2) = 2.666667
```

`calcNeVariance` *Calculate the variance effective population size*

Description

Part of the Genetic Value Analysis

Usage

```
calcNeVariance(ped)
```

Arguments

`ped`	Pedigree data.frame with `id`, `sire`, and `dam`; `exit` is used to identify living animals when present.

Details

The variance effective size measures the diversity lost to unequal family sizes – typically the dominant reducer of effective size in a harem colony, where a few breeders produce most of the offspring. It is the mean-adjusted Crow & Kimura (1970) form

$$N_e = \frac{N\bar{k} - 1}{\bar{k} - 1 + V_k/\bar{k}}$$

where `N` is the number of current living breeders, $\bar{k}$ the mean number of lifetime offspring among them, and V_k the variance of those offspring counts. This general form makes no constant-size assumption and reduces to the classic `(4N - 2) / (Vk + 2)` at exact replacement ($\bar{k} = 2$); it is preferred over that bare form, which assumes $\bar{k} \approx 2$ and misstates the effective size when the mean family size departs from replacement.

The breeders are the current living breeders of `ped` (living animals that appear as a sire or dam, excluding auto-generated unknown parents), independent of which animals are selected as probands – a different population than the analysis-set founder statistics (`calcFE`, `calcFG`, `calcGeneDiversity`). Unlike the sex-ratio effective size (`calcNeSexRatio`), breeders of every sex are counted. When fewer than two living breeders are present the variance is undefined and the result is `NA`.

Like all effective-size estimators this idealizes a Wright-Fisher population (constant size, discrete generations, random union of gametes); a managed colony departs from those assumptions, so read the result as a family-size-variance index rather than a literal head count.

Value

The variance effective size, a single number; `NA` when there are fewer than two living breeders.

References

Crow, J. F. and Kimura, M. (1970) *An Introduction to Population Genetics Theory*. Harper and Row, New York.

See Also

`calcNeSexRatio`, `calcGeneDiversity`

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcNeSexRatio()`, `calcRetention()`

Examples

```
ped <- data.frame(
  id = c("s1", "d1", "k1", "k2", "k3"),
  sire = c(NA, NA, "s1", "s1", "s1"),
  dam = c(NA, NA, "d1", "d1", "d1"),
  sex = c("M", "F", "M", "F", "F"),
  exit = c(NA, NA, NA, NA, NA),
  stringsAsFactors = FALSE
)
calcNeVariance(ped) # 2 breeders, equal families: (2*3-1)/(3-1) = 2.5
```

`calcRetention` *Calculate allelic retention*

Description

Part of Genetic Value Analysis

Usage

```
calcRetention(ped, alleles)
```

Arguments

`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). It is assumed that the pedigree has no partial parentage
`alleles`	dataframe of containing an `AlleleTable`. This is a table of allele information produced by `geneDrop()`.

Value

A vector of the mean number of founder alleles retained in the gene dropping simulation.

References

Lacy RC. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. Zoo Biol 8:111-123.

See Also

Other genetic value analysis: `calcA()`, `calcFE()`, `calcFEFG()`, `calcFG()`, `calcFGSE()`, `calcGU()`, `calcGUSE()`, `calcGeneDiversity()`, `calcNeSexRatio()`, `calcNeVariance()`

Examples

```
library(nprcgenekeepr)
data("lacy1989Ped")
data("lacy1989PedAlleles")
ped <- lacy1989Ped
alleles <- lacy1989PedAlleles
retention <- calcRetention(ped, alleles)
```

`calculateSexRatio` *Calculate the sex ratio of a set of animals*

Description

The Males are counted when the `ped$sex` value is `"M"`. Females are counted when the `ped$sex` value is not `"M"`. This means animals with ambiguous sex are counted with the females.

Usage

```
calculateSexRatio(ids, ped, additionalMales = 0L, additionalFemales = 0L)
```

Arguments

`ids`	character vector of animal IDs
`ped`	dataframe that is the `Pedigree`. It contains pedigree information including the IDs listed in `ids`.
`additionalMales`	Integer value of males to add to those within the group when calculating the ratio. Ignored if calculated ratio is 0 or Inf. Default is 0.
`additionalFemales`	Integer value of females to add to those within the group when calculating the ratio. Ignored if calculated ratio is 0 or Inf. Default is 0.

Value

Numeric value of sex ratio of the animals provided.

Examples

```
library(nprcgenekeepr)
data("qcBreeders")
data("pedWithGenotype")
available <- c(
  "JGPN6K", "8KM1MP", "I9TQ0T", "Q0RGP7", "VFS0XB", "CQC133",
  "2KULR3", "HOYW0S", "FHV13N", "OUM6QF", "6Z7MD9", "CFPEEU",
  "HLI95R", "RI0O7F", "7M51X5", "DR5GXB", "170ZTZ", "C1ICXL"
)
nonMales <- c(
  "JGPN6K", "8KM1MP", "I9TQ0T", "Q0RGP7", "CQC133",
  "2KULR3", "HOYW0S", "FHV13N", "OUM6QF", "6Z7MD9", "CFPEEU",
  "HLI95R", "RI0O7F", "7M51X5", "DR5GXB", "170ZTZ", "C1ICXL"
)
male <- "VFS0XB"
calculateSexRatio(ids = male, ped = pedWithGenotype)
calculateSexRatio(ids = nonMales, ped = pedWithGenotype)
calculateSexRatio(ids = available, ped = pedWithGenotype)
calculateSexRatio(
```

```
  ids = available, ped = pedWithGenotype,
  additionalMales = 1L
)
calculateSexRatio(
  ids = available, ped = pedWithGenotype,
  additionalFemales = 1L
)
calculateSexRatio(
  ids = available, ped = pedWithGenotype,
  additionalMales = 1, additionalFemales = 1L
)
calculateSexRatio(
  ids = nonMales, ped = pedWithGenotype,
  additionalMales = 1, additionalFemales = 0L
)
calculateSexRatio(
  ids = character(0), ped = pedWithGenotype,
  additionalMales = 1, additionalFemales = 0L
)
```

`checkChangedColsLst` *Check a changed-columns list for non-empty fields*

Description

Check a changed-columns list for non-empty fields

Usage

```
checkChangedColsLst(changedCols)
```

Arguments

`changedCols`	list with fields for each type of column change `qcStudbook`.

Value

Returns `TRUE` if any changed-columns field is non-empty, otherwise `FALSE`.

Examples

```
library(nprcgenekeepr)
library(lubridate)
pedOne <- data.frame(
  ego_id = c(
    "s1", "d1", "s2", "d2", "o1", "o2", "o3",
    "o4"
  ),
  `si re` = c(NA, NA, NA, NA, "s1", "s1", "s2", "s2"),
  dam_id = c(NA, NA, NA, NA, "d1", "d2", "d2", "d2"),
```

```
  sex = c("F", "M", "M", "F", "F", "F", "F", "M"),
  birth_date = mdy(
    paste0(
      sample(1:12, 8, replace = TRUE), "-",
      sample(1:28, 8, replace = TRUE), "-",
      sample(seq(0, 15, by = 3), 8, replace = TRUE) +
        2000
    )
  ),
  stringsAsFactors = FALSE, check.names = FALSE
)

errorLst <- qcStudbook(pedOne, reportErrors = TRUE, reportChanges = TRUE)
checkChangedColsLst(errorLst$changedCols)
```

---

checkErrorLst *Check an error list for non-empty fields*

Description

Check an error list for non-empty fields

Usage

```
checkErrorLst(errorLst)
```

Arguments

errorLst	list with fields for each type of error detectable by qcStudbook.

Value

Returns FALSE if all fields are empty or the list is NULL otherwise TRUE.

Examples

```
errorLst <- qcStudbook(nprcgenekeepr::pedFemaleSireMaleDam,
  reportErrors = TRUE
)
checkErrorLst(errorLst)
```

checkGenotypeFile *Check genotype file*

Description

Checks to ensure the content and structure are appropriate for a genotype file. These checks are simply based on expected columns and legal domains.

Usage

```
checkGenotypeFile(genotype)
```

Arguments

genotype	dataframe with genotype data

Value

A genotype file that has been checked to ensure the column types and number required are present. The returned genotype file has the first column name forced to "id".

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped <- ped[order(ped$id), ]
genotype <- data.frame(
  id = ped$id[50 + 1:20],
  first_name = paste0("first_name", 1:20),
  second_name = paste0("second_name", 1:20),
  stringsAsFactors = FALSE
)

## checkGenotypeFile disallows dataframe with < 3 columns
tryCatch(
  {
    checkGenotypeFile(genotype[, c("id", "first_name")])
  },
  warning = function(w) {
    cat("Warning produced")
  },
  error = function(e) {
    cat("Error produced")
  }
)
```

checkKinshipOverrides *Validate a kinship overrides table*

Description

Checks the structure and domain of an outside-information kinship override table. The table supplies pairwise kinship coefficients (`id1`, `id2`, `kinship`) that `applyKinshipOverrides` writes into a computed kinship matrix, replacing the pedigree-derived value for those pairs. It mirrors `checkGenotypeFile`: it `stop()`s on structural or domain errors and returns the (id-coerced) table when the input is acceptable.

Usage

```
checkKinshipOverrides(overrides)
```

Arguments

`overrides`	data.frame with id columns `id1` and `id2` and a numeric `kinship` column; each row is one off-diagonal pair. Any extra columns are ignored.

Details

`kinship` is the kinship coefficient f (the probability that an allele drawn at random from each of the two animals is identical by descent), **not** the coefficient of relatedness r (= $2f$ for non-inbred animals). Supplying r – e.g. 0.25 for half-sibs whose true f is 0.125 – silently corrupts the matrix, so an off-diagonal value above 0.5 (the maximum for a non-inbred pair) draws a warning here. The exact positive-semi-definiteness bound is enforced by `applyKinshipOverrides` once the matrix diagonal is known.

Value

The validated `overrides` data.frame with `id1` and `id2` coerced to character.

Examples

```
overrides <- data.frame(
  id1 = c("A1", "A3"), id2 = c("A2", "A4"),
  kinship = c(0.25, 0.125), stringsAsFactors = FALSE
)
checkKinshipOverrides(overrides)
```

checkParentAge *Check parent ages against a minimum age*

Description

Ensure parents are sufficiently older than offspring

Usage

```
checkParentAge(
  sb,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  reportErrors = FALSE
)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information.
minSireAge	numeric minimum age in years for a male to have sired an offspring. NULL (default) looks up the floor for each sire's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all sires. The check is not performed for animals with missing birth dates.
minDamAge	numeric minimum age in years for a female to have borne an offspring. NULL (default) looks up the floor for each dam's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all dams.
minParentAge	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both minSireAge and minDamAge; use those sex-specific parameters instead.
reportErrors	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Value

A dataframe containing rows for each animal where one or more parent was less than minParentAge. It contains all of the columns in the original sb dataframe with the following added columns:

1. sireBirth – sire's birth date
2. sireAge – age of sire in years on the date indicated by birth.
3. damBirth – dam's birth date
4. damAge – age of dam in years on the date indicated by birth.

Examples

```
library(nprcgenekeepr)
qcPed <- nprcgenekeepr::qcPed
checkParentAge(qcPed, minSireAge = 2L, minDamAge = 2L)
checkParentAge(qcPed, minSireAge = 3L, minDamAge = 3L)
checkParentAge(qcPed, minSireAge = 5L, minDamAge = 5L)
checkParentAge(qcPed, minSireAge = 6L, minDamAge = 6L)
head(checkParentAge(qcPed, minSireAge = 10L, minDamAge = 10L))
```

---

checkRequiredCols *Check column names for required columns*

Description

Check column names for required columns

Usage

```
checkRequiredCols(cols, reportErrors)
```

Arguments

cols	character vector of column names
reportErrors	logical value when TRUE and missing columns are found the errorLst object is updated with the names of the missing columns and returned and when FALSE and missing columns are found the program is stopped.

Details

When reportErrors = TRUE, NA entries in cols are treated as ordinary non-matching column names when building the list of missing required columns, rather than causing an error. (Earlier versions could error with "missing value where TRUE/FALSE needed" on such out-of-contract input.)

Value

NULL is returned if all required columns are present. See description of reportErrors for return values when required columns are missing.

Examples

```
library(nprcgenekeepr)
requiredCols <- getRequiredCols()
cols <-
  paste0(
    "id,sire,siretype,dam,damtype,sex,numberofparentsknown,birth,",
    "arrivalatcenter,death,departure,status,ancestry,fromcenter?,",
    "origin"
  )
all(requiredCols %in% checkRequiredCols(cols, reportErrors = TRUE))
```

chooseAlleles *Combine two allele vectors by Mendelian sampling*

Description

Combine two allele vectors by Mendelian sampling

Usage

```
chooseAlleles(a1, a2)
```

Arguments

a1	integer vector with first allele for each individual
a2	integer vector with second allele for each individual a1 and a2 are equal length vectors of alleles for one individual

Value

An integer vector with the result of sampling from a1 and a2 according to Mendelian inheritance.

Examples

```
chooseAlleles(0L:4L, 5L:9L)
```

chooseDate *Choose the earlier or later of two dates*

Description

Part of Pedigree Curation

Usage

```
chooseDate(d1, d2, earlier = TRUE)
```

Arguments

d1	Date vector with the first of two dates to compare.
d2	Date vector with the second of two dates to compare.
earlier	logical variable with TRUE if the earlier of the two dates is to be returned, otherwise the later is returned. Default is TRUE.

Details

Given two dates, one is selected to be returned based on whether it occurred earlier or later than the other. NAs are ignored if possible.

Value

Date vector of chosen dates or NA where neither is provided

Examples

```
library(nprcgenekeepr)
someDates <- lubridate::mdy(paste0(
  sample(1:12, 2, replace = TRUE), "-",
  sample(1:28, 2, replace = TRUE), "-",
  sample(seq(0, 15, by = 3), 2,
    replace = TRUE
  ) + 2000
))
someDates
chooseDate(someDates[1], someDates[2], earlier = TRUE)
chooseDate(someDates[1], someDates[2], earlier = FALSE)
```

---

convertAncestry *Convert ancestry information to a standard code*

Description

Part of Pedigree Curation

Usage

```
convertAncestry(ancestry)
```

Arguments

ancestry	character vector or NA with free-form text providing information about the geographic population of origin.

Value

A factor vector of standardized designators specifying if an animal is a Chinese rhesus, Indian rhesus, Chinese-Indian hybrid rhesus, or Japanese macaque. Levels: CHINESE, INDIAN, HYBRID, JAPANESE, OTHER, UNKNOWN.

Examples

```
original <- c("china", "india", "hybridized", NA, "human", "gorilla")
convertAncestry(original)
```

convertDate *Convert character date columns to Date type*

Description

Part of Pedigree Curation

Usage

```
convertDate(ped, timeOrigin = as.Date("1970-01-01"), reportErrors = FALSE)
```

Arguments

ped	a dataframe of pedigree information that may contain birth, death, departure, or exit dates. The fields are optional, but will be used if present.(optional fields: birth, death, departure, and exit).
timeOrigin	date object used by as.Date to set origin.
reportErrors	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Value

A dataframe with an updated table with date columns converted from character data type to Date data type. Values that do not conform to the format %Y%m%d are set to NA. NA values are left as NA.

Examples

```
library(lubridate)
set_seed(10)
someBirthDates <- paste0(
  sample(seq(0, 15, by = 3), 10,
    replace = TRUE
  ) + 2000, "-",
  sample(1:12, 10, replace = TRUE), "-",
  sample(1:28, 10, replace = TRUE)
)
someBadBirthDates <- paste0(
  sample(1:12, 10, replace = TRUE), "-",
  sample(1:28, 10, replace = TRUE), "-",
  sample(seq(0, 15, by = 3), 10,
    replace = TRUE
  ) + 2000
)
someDeathDates <- sample(someBirthDates, length(someBirthDates),
  replace = FALSE
)
```

```
someDepartureDates <- sample(someBirthDates, length(someBirthDates),
  replace = FALSE
)
ped1 <- data.frame(
  birth = someBadBirthDates, death = someDeathDates,
  departure = someDepartureDates
)
someDates <- ymd(someBirthDates)
ped2 <- data.frame(
  birth = someDates, death = someDeathDates,
  departure = someDepartureDates
)
ped3 <- data.frame(
  birth = someBirthDates, death = someDeathDates,
  departure = someDepartureDates
)
someNADeathDates <- someDeathDates
someNADeathDates[c(1, 3, 5)] <- ""
someNABirthDates <- someDates
someNABirthDates[c(2, 4, 6)] <- NA
ped4 <- data.frame(
  birth = someNABirthDates, death = someNADeathDates,
  departure = someDepartureDates
)

## convertDate identifies bad dates
result <- tryCatch(
  {
    convertDate(ped1)
  },
  warning = function(w) {
    print("Warning in date")
  },
  error = function(e) {
    print("Error in date")
  }
)

## convertDate with error flag returns error list and not an error
convertDate(ped1, reportErrors = TRUE)

## convertDate recognizes good dates
all(is.Date(convertDate(ped2)$birth))
all(is.Date(convertDate(ped3)$birth))

## convertDate handles NA and empty character string values correctly
convertDate(ped4)
```

---

convertFromCenter *Convert from-center information to a logical value*

Description

Part of Pedigree Curation

Usage

```
convertFromCenter(fromCenter)
```

Arguments

fromCenter	character or logical vector or NA indicating whether or not the animal is from the center.

Value

A logical vector specifying TRUE if an animal is from the center otherwise FALSE.

Examples

```
original <- c(
  "y", "yes", "Y", "Yes", "YES", "n", "N", "No", "NO", "no",
  "t", "T", "True", "true", "TRUE", "f", "F", "false", "False",
  "FALSE"
)
convertFromCenter(original)
```

---

convertRelationships *Convert pairwise kinship values to relationship categories*

Description

Part of Relations

Usage

```
convertRelationships(kmat, ped, ids = NULL, updateProgress = NULL)
```

Arguments

kmat	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.
ped	datatable that is the Pedigree. It contains pedigree information. The fields id, sire and dam are required.
ids	character vector of IDs or NULL to which the analysis should be restricted. If provided, only relationships between these IDs will be converted to relationships.
updateProgress	function or NULL. If this function is defined, it will be called during each iteration to update a shiny::Progress object.

Value

A dataframe with columns `id1`, `id2`, `kinship`, `relation`. It is a long-form table of pairwise kinships, with relationship categories included for each pair.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
ids <- c("A", "B", "D", "E", "F", "G", "I", "J", "L", "M", "O", "P")
relIds <- convertRelationships(kmat, ped, ids)
rel <- convertRelationships(kmat, ped, updateProgress = function() {})
head(rel)
ped <- nprcgenekeepr::qcPed
bkmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen,
  sparse = FALSE
)
relBIds <- convertRelationships(bkmat, ped, c("4LFS70", "DD1U77"))
relBIds
```

---

convertSexCodes *Convert a sex indicator to a standardized code*

Description

Part of Pedigree Curation

Usage

```
convertSexCodes(sex, ignoreHerm = TRUE)
```

Arguments

`sex`	factor with levels: "M", "F", "U". Sex specifier for an individual.
`ignoreHerm`	logical flag indicating if hermaphrodites should be treated as unknown sex ("U"), default is `TRUE`.

Details

Standard sex codes are

- `F` – replacing "FEMALE" or "2"
- `M` – replacing "MALE" or "1"
- `H` – replacing "HERMAPHRODITE" or "4", if `ignoreHerm` == FALSE
- `U` – replacing "HERMAPHRODITE" or "4", if `ignoreHerm` == TRUE
- `U` – replacing "UNKNOWN" or "3"

Value

A vector of factors representing standardized sex codes after transformation from non-standard codes.

Examples

```
library(nprcgenekeepr)
original <- c(
  "m", "male", "1", "MALE", "M", "F", "f", "female",
  "FemAle", "U", "Unknown", "H", "hermaphrodite",
  "U", "Unknown", "3", "4"
)
sexCodes <- convertSexCodes(original)
sexCodes
```

`convertStatusCodes` *Convert status indicators to a standardized code*

Description

Part of Pedigree Curation

Usage

```
convertStatusCodes(status)
```

Arguments

`status`	character vector or NA. Flag indicating an individual's status as alive, dead, sold, etc.

Value

A factor vector of the standardized status codes with levels: `ALIVE`, `DECEASED`, `SHIPPED`, and `UNKNOWN`.

Examples

```
library(nprcgenekeepr)
original <- c(
  "A", "alive", "Alive", "1", "S", "Sale", "sold", "shipped",
  "D", "d", "dead", "died", "deceased", "2",
  "shiped", "3", "U", "4", "unknown", NA,
  "Unknown", "H", "hermaphrodite", "U", "Unknown", "4"
)
convertStatusCodes(original)
```

`correctParentSex` *Correct the sex of animals listed as a sire or dam*

Description

Part of Pedigree Curation

Usage

```
correctParentSex(id, sire, dam, sex, recordStatus, reportErrors = FALSE)
```

Arguments

`id`	character vector with unique identifier for an individual
`sire`	character vector with unique identifier for an individual's father (`NA` if unknown).
`dam`	character vector with unique identifier for an individual's mother (`NA` if unknown).
`sex`	factor with levels: "M", "F", "U". Sex specifier for an individual.
`recordStatus`	character vector with value of `"added"` or `"original"`, which indicates whether an animal was added or an original animal.
`reportErrors`	logical value if TRUE will scan the entire file and make a list of all errors found. The errors will be returned in a list of list where each sublist is a type of error found.

Details

Only true female-sires (`"F"`) and male-dams (`"M"`) are corrected (to `"M"` and `"F"` respectively). Parents recorded as hermaphrodite (`"H"`) or unknown (`"U"`) sex are left unchanged, consistent with `reportErrors = TRUE` mode, which does not flag them.

Value

When `reportErrors = FALSE`, a factor (or character vector) of corrected sex codes with levels `"M"`, `"F"`, `"H"`, and `"U"` for the ids provided. When `reportErrors = TRUE`, a named list of error vectors with elements `sireAndDam`, `femaleSires`, and `maleDams` (each `NULL` when no such errors are found).

Examples

```
library(nprcgenekeepr)
pedOne <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c(NA, "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("F", "F", "M", "F", "F", "F", "F", "M"),
  recordStatus = rep("original", 8),
  stringsAsFactors = FALSE
```

```
)
pedTwo <- data.frame(
  id = c("s1", "d1", "s2", "d2", "o1", "o2", "o3", "o4"),
  sire = c(NA, "s0", "s4", NA, "s1", "s1", "s2", "s2"),
  dam = c("d0", "d0", "d4", NA, "d1", "d2", "d2", "d2"),
  sex = c("M", "M", "M", "F", "F", "F", "F", "M"),
  recordStatus = rep("original", 8),
  stringsAsFactors = FALSE
)
pedOneCorrected <- pedOne
pedOneCorrected$sex <- correctParentSex(
  pedOne$id, pedOne$sire, pedOne$dam,
  pedOne$sex, pedOne$recordStatus
)
pedOne[pedOne$sex != pedOneCorrected$sex, ]
pedOneCorrected[pedOne$sex != pedOneCorrected$sex, ]

pedTwoCorrected <- pedTwo
pedTwoCorrected$sex <- correctParentSex(
  pedTwo$id, pedTwo$sire, pedTwo$dam,
  pedTwo$sex, pedOne$recordStatus
)
pedTwo[pedTwo$sex != pedTwoCorrected$sex, ]
pedTwoCorrected[pedTwo$sex != pedTwoCorrected$sex, ]
```

`countFirstOrder` *Count first-order relatives*

Description

Part of Relations

Usage

```
countFirstOrder(ped, ids = NULL)
```

Arguments

`ped`	The pedigree information in data.frame format
`ids`	character vector of IDs or NULL These are the IDs to which the analysis should be restricted. First-order relationships will only be tallied for the listed IDs and will only consider relationships within the subset. If NULL, the analysis will include all IDs in the pedigree.

Details

Tallies the number of first-order relatives for each member of the provided pedigree. If 'ids' is provided, the analysis is restricted to only the specified subset.

Value

A dataframe with column `id`, `parents`, `offspring`, `siblings`, and `total`. A table of first-order relationship counts, broken down to indicate the number of parents, offspring, and siblings that are part of the subset under consideration.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ids <- c("B", "D", "E", "F", "G")
countIds <- countFirstOrder(ped, ids)
countIds
count <- countFirstOrder(ped, NULL)
count
```

`countKinshipValues` *Count kinship-value occurrences across simulated pedigrees*

Description

Count kinship-value occurrences across simulated pedigrees

Usage

```
countKinshipValues(kinshipValues, accummulatedKValueCounts = NULL)
```

Arguments

`kinshipValues`	matrix of kinship values from simulated pedigrees where each row represents a pair of individuals in the pedigree and each column represents the vector of kinship values generated in a simulated pedigree.
`accummulatedKValueCounts`	list object with same structure as that returned by this function.

Value

A list of three lists named `kIds` (kinship IDs), `kValues` (kinship values), and `kCounts` (kinship counts).

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
```

```
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKinship <- function(simKinships, id1, id2, simulation) {
  ids <- dimnames(simKinships[[simulation]])[[1]]
  simKinships[[simulation]][
    seq_along(ids)[ids == id1],
    seq_along(ids)[ids == id2]
  ]
}

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[
    kValue$id_1 == id1 & kValue$id_2 == id2,
    paste0("sim_", simulation)
  ]
}

n <- 10
simKinships <- createSimKinships(ped, allSimParents,
  pop = ped$id, n = n
)
kValues <- kinshipMatricesToKValues(simKinships)
extractKValue(kValues, id1 = "A", id2 = "F", simulation = 1:n)
counts <- countKinshipValues(kValues)
n <- 10
```

```
simKinships <- createSimKinships(ped, allSimParents, pop = ped$id, n = n)
kValues <- kinshipMatricesToKValues(simKinships)
extractKValue(kValues, id1 = "A", id2 = "F", simulation = 1:n)
accummulatedCounts <- countKinshipValues(kValues, counts)
```

`countLoops` *Count the number of loops in a pedigree tree*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
countLoops(loops, ptree)
```

Arguments

`loops`	a named list of logical values where each named element is named with an `id` from `ptree`. The value of the list element is set to `TRUE` if the `id` has a loop in the pedigree. Loops occur when an animal's sire and dam have a common ancestor.
`ptree`	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Uses the `loops` data structure and the list of all ancestors for each individual to calculate the number of loops for each individual.

Value

A list indexed with each ID in the pedigree tree (`ptree`) containing the number of loops for each individual.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
## You can count how many animals are in loops with the following code.
length(exampleLoops[exampleLoops == TRUE])
## You can count how many loops you have with the following code.
nLoops <- countLoops(exampleLoops, exampleTree)
```

```
sum(unlist(nLoops[nLoops > 0]))
## You can list the first 10 sets of ids, sires and dams in loops with
## the following line of code:
examplePedigree[exampleLoops == TRUE, c("id", "sire", "dam")][1:10, ]
```

`createExampleFiles` *Create example pedigree and ID-list CSV files*

Description

Creates a folder named `ExamplePedigrees` under the R session temporary directory (as returned by `tempdir()`) if it does not already exist. It then proceeds to write each example pedigree into a CSV file named based on the name of the example pedigree.

Usage

```
createExampleFiles()
```

Value

A vector of the names of the files written.

Examples

```
library(nprcgenekeepr)
files <- createExampleFiles()
```

`createPedTree` *Create a pedigree tree (PedTree)*

Description

The PedTree is a list containing sire and dam information for an individual.

Usage

```
createPedTree(ped)
```

Arguments

`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.

Details

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

This function uses only `id`, `sire`, and `dam` columns.

Value

A list of named lists forming a pedigree tree (PedTree or ptree). Each sublist represents an ID in the pedigree and contains the sire ID and the dam ID as named elements.

Examples

```
library(nprcgenekeepr)
exampleTree <- createPedTree(nprcgenekeepr::examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

---

createSimKinships *Build kinship matrices from simulated pedigrees*

Description

`createSimKinships` uses `makeSimPed` with the `ped` object and the `allSimParents` object to create a set of kinship matrices to be used in forming the *Monte Carlo* estimates for the kinship values.

Usage

```
createSimKinships(ped, allSimParents, pop = NULL, n = 10L, verbose = FALSE)
```

Arguments

`ped`	The pedigree information in data.frame format
`allSimParents`	list made up of lists where the internal list has the offspring ID, `id`, a vector of representative sires (`sires`), and a vector of representative dams (`dams`).
`pop`	Character vector with animal IDs to consider as the population of interest. This allows you to provide a pedigree in `ped` that has more animals than you want to use in the simulation by defining `pop` with the subset of interest. The default is NULL.
`n`	integer value of the number of simulated pedigrees to generate.
`verbose`	logical vector of length one that indicates whether or not to print out when an animal is missing a sire or a dam.

Value

A list of n lists with each internal list containing a kinship matrix from simulated pedigrees of possible parents for animals with unknown parents.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s2_1", "s2_2", "s2_3"),
  dams = c("d2_1", "d2_2", "d2_3", "d2_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("s3_1", "s3_2", "s3_3"),
  dams = c("d3_1", "d3_2", "d3_3", "d3_4")
)
allSimParents <- list(simParent_1, simParent_2, simParent_3)
pop <- LETTERS[1:7]
simKinships <- createSimKinships(ped, allSimParents, pop, n = 10)
```

---

create_wkbk *Create an Excel workbook with worksheets*

Description

Create an Excel workbook with worksheets

Usage

```
create_wkbk(file, df_list, sheetnames, replace = FALSE)
```

Arguments

file	filename of workbook to be created
df_list	list of data frames to be added as worksheets to workbook
sheetnames	character vector of worksheet names
replace	Specifies if the file should be replaced if it already exist (default is FALSE).

Value

TRUE if the Excel file was successfully created. FALSE if any errors occurred.

Examples

```
library(nprcgenekeepr)

make_df_list <- function(size) {
  df_list <- list(size)
  if (size <= 0) {
    return(df_list)
  }
  for (i in seq_len(size)) {
    n <- sample(2:10, 2, replace = TRUE)
    df <- data.frame(matrix(data = rnorm(n[1] * n[2]), ncol = n[1]))
    df_list[[i]] <- df
  }
  names(df_list) <- paste0("A", seq_len(size))
  df_list
}
df_list <- make_df_list(3)
sheetnames <- names(df_list)
if (any(file.exists(file.path(tempdir(), "example_excel_wkbk.xlsx")))) {
  file.remove(file.path(tempdir(), "example_excel_wkbk.xlsx"))
  create_wkbk(
    file = file.path(tempdir(), "example_excel_wkbk.xlsx"),
    df_list = df_list,
    sheetnames = sheetnames,
    replace = FALSE
  )
}
if (any(file.exists(file.path(tempdir(), "example_excel_wkbk.xlsx")))) {
  file.remove(file.path(tempdir(), "example_excel_wkbk.xlsx"))
}
```

---

cumulateSimKinships *Compute kinship summary statistics across simulations*

Description

`cumulateSimKinships` creates a named list of length 4 is generated where the first element is the mean of the simulated kinships, the second element is the standard deviation of the simulated kinships the third element is the minimum value of the kinships, and the forth element is the maximum value of the kinships.

Usage

```
cumulateSimKinships(ped, allSimParents, pop = NULL, n = 10L)
```

Arguments

`ped`	The pedigree information in data.frame format

allSimParents	list made up of lists where the internal list has the offspring ID id, a vector of representative sires (sires), and a vector of representative dams(dams).
pop	Character vector with animal IDs to consider as the population of interest. The default is NULL.
n	integer value of the number of simulated pedigrees to generate. Must be at least 1 (n < 1 is an error); the standard deviation requires n >= 2.

Value

List object containing the meanKinship, sdKinship, minKinship, and maxKinship. sdKinship is the sample standard deviation across the n simulations; it is undefined for a single simulation, so when n < 2 it is returned as NA (with a warning), as base R sd() does for a length-one vector.

Examples

```
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s2_1", "s2_2", "s2_3"),
  dams = c("d2_1", "d2_2", "d2_3", "d2_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("s3_1", "s3_2", "s3_3"),
  dams = c("d3_1", "d3_2", "d3_3", "d3_4")
)
allSimParents <- list(simParent_1, simParent_2, simParent_3)
pop <- LETTERS[1:7]
cumulativeKinships <- cumulateSimKinships(ped, allSimParents, pop, n = 10)
```

---

dataframe2string *Convert a data frame to a character vector*

Description

Adapted from print.data.frame

Usage

```
dataframe2string(object, ..., digits = NULL, addRowNames = TRUE)
```

Arguments

object	dataframe
...	optional arguments to print or plot methods.
digits	the minimum number of significant digits to be used: see print.default.
addRowNames	logical (or character vector), indicating whether (or what) row names should be printed.

Value

A character vector representation of the data.frame provided to the function.

Examples

```
library(nprcgenekeepr)
dataframe2string(nprcgenekeepr::pedOne)
```

---

exampleNprcgenekeeprConfig

Example nprcgenekeepr configuration file (loadable)

Description

A loadable version of the example configuration file `example_nprcgenekeepr_config`. It contains a working version of a **nprcgenekeepr** configuration file created at the SNPRC. Users of LabKey's EHR can adapt it to their systems and put it in their home directory. Instructions are embedded as comments within the file.

Usage

```
data(exampleNprcgenekeeprConfig)
```

Format

An object of class `character` of length 34.

Examples

```
library(nprcgenekeepr)
data("exampleNprcgenekeeprConfig")
head(exampleNprcgenekeeprConfig)
```

`examplePedigree` *Example pedigree object (from ExamplePedigree.csv)*

Description

A pedigree object created by `qcStudbook`. Represents pedigree from *ExamplePedigree.csv*.

id – character column of animal IDs

sire – the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam – the female parent of the animal indicated by the `id` column. Unknown dams are indicated with `NA`

sex – factor with levels: "F", "M", "H", "U". Sex specifier for an individual.

gen – generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth – Date vector of birth dates

exit – Date vector of exit dates

age – numerical vector of age in years

ancestry – factor with levels: INDIAN, CHINESE, HYBRID, JAPANESE, OTHER, UNKNOWN indicating the geographic population of origin.

origin – character vector or `NA` (optional) that indicates the name of the facility that the individual was imported from if other than local.

status – character vector or NA. Flag indicating an individual's status as alive, dead, sold, etc. Transformed to factor {levels: ALIVE, DECEASED, SHIPPED, UNKNOWN}. Vector of standardized status codes with the possible values ALIVE, DECEASED, SHIPPED, or UNKNOWN

recordStatus – character vector with value of `"added"` or `"original"`.

fromCenter – logical vector indicating whether the animal was born at the local center (colony-origin), derived from `origin` and `recordStatus`: `TRUE` when `origin` is blank and `recordStatus` is `"original"`; `FALSE` for animals imported from elsewhere (non-blank `origin`) or for synthetic placeholder second-parent rows (`recordStatus == "added"`) whose true origin is not confirmed. Required by `getPotentialParents` to identify in-colony candidate parents.

Usage

```
data(examplePedigree)
```

Format

An object of class `data.frame` with 3694 rows and 13 columns.

Examples

```
library(nprcgenekeepr)
data("examplePedigree")
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

`fillGroupMembersWithSexRatio` *Form breeding groups to match a target sex ratio*

Description

The sex ratio is the ratio of females to males.

Usage

```
fillGroupMembersWithSexRatio(
  candidates,
  groupMembers,
  grpNum,
  kin,
  ped,
  minAge,
  numGp,
  sexRatio
)
```

Arguments

`candidates`	character vector of IDs of the animals available for use in the group.
`groupMembers`	list initialized and ready to receive groups with the desired sex ratios that are created within this function
`grpNum`	is a list `numGp` long with each member an integer vector of `1:numGp`.
`kin`	list of animals and those animals who are related above a threshold value.
`ped`	dataframe that is the `Pedigree`. It contains pedigree information including the IDs listed in `ids`.
`minAge`	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
`numGp`	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
`sexRatio`	numeric value indicating the ratio of females to males x from 0.5 to 20 by increments of 0.5.

Value

A list containing one character vector of animal IDs such that the sex ratio of the group is as close as possible to the ratio specified by `sexRatio`.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
examplePedigree <- examplePedigree[1:300, ] # Comment out for full example
ped <- qcStudbook(examplePedigree,
  minParentAge = 2L, reportChanges = FALSE,
  reportErrors = FALSE
)

kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1)
currentGroups[[1]] <- examplePedigree$id[1L:3L]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
  ignore = list(c("F", "F")), minAge = 1L
)
# Filtering out candidates related to current group members
conflicts <- unique(c(
  unlist(kin[unlist(currentGroups)]),
  unlist(currentGroups)
))
candidates <- setdiff(candidates, conflicts)

kin <- addAnimalsWithNoRelative(kin, candidates)

ignore <- NULL
minAge <- 1.0
numGp <- 1L
harem <- FALSE
sexRatio <- 0.0
withKin <- FALSE
groupMembers <- nprcgenekeepr::makeGroupMembers(numGp,
  currentGroups,
  candidates,
  ped,
  harem = harem,
  minAge = minAge
)
groupMembersStart <- groupMembers
grpNum <- nprcgenekeepr::makeGroupNum(numGp)

groupMembers <- fillGroupMembersWithSexRatio(
  candidates, groupMembers, grpNum, kin, ped, minAge, numGp,
  sexRatio = 1.0
)
```

filterKinMatrix *Filter a kinship matrix to selected IDs*

Description

Filter a kinship matrix to selected IDs

Usage

```
filterKinMatrix(ids, kmat)
```

Arguments

ids	character vector containing the IDs of interest. The kinship matrix should be reduced to only include these rows and columns.
kmat	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.

Value

A numeric matrix that is the reduced kinship matrix with named rows and columns (row and col names are 'ids').

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen,
  sparse = FALSE
)
ids <- ped$id[c(189, 192, 194, 195)]
ncol(kmat)
nrow(kmat)
kmatFiltered <- filterKinMatrix(ids, kmat)
ncol(kmatFiltered)
nrow(kmatFiltered)
```

filterPairs *Filter kinship pairs by the animals' sexes*

Description

Part of Group Formation

Usage

```
filterPairs(
  kin,
  ped,
  ignore = list(c(sexCodes[["female"]], sexCodes[["female"]]))
)
```

Arguments

kin	a dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.
ped	Dataframe of pedigree information that must contain an id column and a sex column. The id values should include the animals referenced in kin.
ignore	a list containing zero or more character vectors of length 2 indicating which sex pairs should be ignored with regard to kinship. Defaults to list(c("F", "F")).

Value

A dataframe representing a filtered long-format kinship table.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
kin <- kinMatrix2LongForm(kmat, removeDups = FALSE)
threshold <- 0.1
kin <- filterThreshold(kin, threshold = threshold)
ped$sex <- c("M", "F", "M", "M", "F", "F", "M")
kinNull <- filterPairs(kin, ped, ignore = NULL)
kinMM <- filterPairs(kin, ped, ignore = list(c("M", "M")))
ped
kin[kin$id1 == "C", ]
kinMM[kinMM$id1 == "C", ]
```

---

filterReport *Filter a genetic value report to selected animals*

Description

Filter a genetic value report to selected animals

Usage

```
filterReport(ids, rpt)
```

Arguments

ids	character vector of animal IDs
rpt	a dataframe with required colnames id, gu, zScores, import, totalOffspring, which is a data.frame of results from a genetic value analysis.

Value

A copy of report specific to the specified animals.

Examples

```
library(nprcgenekeepr)
rpt <- nprcgenekeepr::pedWithGenotypeReport$report
rpt1 <- filterReport(c("GHH9LB", "BD41WW"), rpt)
```

---

filterThreshold *Filter out kinship pairs below a threshold*

Description

Part of Group Formation Filters kinship values less than the specified threshold from a long-format table of kinship values.

Usage

```
filterThreshold(kin, threshold = 0.015625)
```

Arguments

kin	a dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.
threshold	numeric value representing the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored.

Value

The filtered long-format kinship table (a data.frame) with all kinship relationships below the threshold value removed.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
kin <- kinMatrix2LongForm(kmat, removeDups = FALSE)
kinFiltered_0.3 <- filterThreshold(kin, threshold = 0.3)
kinFiltered_0.1 <- filterThreshold(kin, threshold = 0.1)
```

finalRpt *Genetic-value report list prior to ranking*

Description

A list object created from the list object *rpt* prepared by `reportGV`. It is created inside `orderReport`. This version is at the state just prior to calling `rankSubjects` inside `orderReport`.

Usage

```
data(finalRpt)
```

Format

An object of class `list` of length 3.

Examples

```
library(nprcgenekeepr)
data("finalRpt")
finalRpt <- rankSubjects(finalRpt)
```

findGeneration *Determine the generation number for each ID*

Description

This loops through the entire pedigree one generation at a time. It finds the zeroth generation during first loop. The first time through this loop no sire or dam is in parents. This means that the animals without a sire and without a dam are assigned to generation 0 and become the first parental generation. The second time through this loop finds all of the animals that do not have a sire or do not have a dam and at least one parent is in the vector of parents defined the first time through. The ids that were not assigned as parents in the previous loop are given the incremented generation number.

Usage

```
findGeneration(id, sire, dam)
```

Arguments

`id`	character vector with unique identifier for an individual
`sire`	character vector with unique identifier for an individual's father (`NA` if unknown).
`dam`	character vector with unique identifier for an individual's mother (`NA` if unknown).

Details

Subsequent trips in the loop repeat what was done the second time through until no further animals can be added to the `nextGen` vector.

This does not work if the pedigree does not have all parent IDs as ego IDs.

Value

An integer vector indication the generation numbers for each id, starting at 0 for individuals lacking IDs for both parents. Any id that cannot be placed — e.g. when the pedigree contains a cycle or references a parent ID that is not itself present as an ego ID — is returned as `NA` and triggers a `warning` naming the affected ids.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped[, c("id", "sire", "dam")]
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
ped
```

`findLoops` *Find loops in a pedigree tree*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
findLoops(ptree)
```

Arguments

`ptree`	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Value

A named list of logical values where each named element is named with an `id` from `ptree`. The value of the list element is set to `TRUE` if the `id` has a loop in the pedigree. Loops occur when an animal's sire and dam have a common ancestor.

Examples

```
data("examplePedigree")
exampleTree <- createPedTree(examplePedigree)
exampleLoops <- findLoops(exampleTree)
```

`findOffspring` *Count total offspring for each animal*

Description

Part of Genetic Value Analysis

Usage

```
findOffspring(probands, ped)
```

Arguments

probands	character vector of egos for which offspring should be counted and returned.
ped	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). This requires complete pedigree information.

Value

A named vector containing the offspring counts for each animal in probands. Rownames are set to the IDs from probands.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
totalOffspring <- findOffspring(probands, ped)
```

`findPedigreeNumber` *Determine the pedigree number for each ID*

Description

One of Pedigree Curation functions

Usage

```
findPedigreeNumber(id, sire, dam)
```

Arguments

`id`	character vector with unique identifier for an individual
`sire`	character vector with unique identifier for an individual's father (`NA` if unknown).
`dam`	character vector with unique identifier for an individual's mother (`NA` if unknown).

Value

Integer vector indicating the pedigree (family group) number for each id. Ids that are connected through parent-offspring links share the same number; numbering starts at 1.

Examples

```
library(nprcgenekeepr)
library(stringi)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- NULL
ped$population <- NULL
ped2 <- ped
ped2$id <- stri_c(ped$id, "2")
ped2$sire <- stri_c(ped$sire, "2")
ped2$dam <- stri_c(ped$dam, "2")
ped3 <- ped
ped3$id <- stri_c(ped$id, "3")
ped3$sire <- stri_c(ped$sire, "3")
ped3$dam <- stri_c(ped$dam, "3")
ped <- rbind(ped, ped2)
ped <- rbind(ped, ped3)
ped$pedigree <- findPedigreeNumber(ped$id, ped$sire, ped$dam)
ped$pedigree
```

fixColumnNames *Standardize pedigree column names*

Description

Standardize pedigree column names

Usage

```
fixColumnNames(orgCols, errorLst)
```

Arguments

orgCols	character vector with ordered list of column names found in a pedigree file.
errorLst	list object with places to store the various column name changes.

Value

A list object with newColNames and errorLst with a record of all changes made.

Examples

```
library(nprcgenekeepr)
fixColumnNames(c("Sire_ID", "EGO", "DAM", "Id", "birth_date"),
  errorLst = getEmptyErrorLst()
)
```

focalAnimals *Focal animal IDs from examplePedigree*

Description

A dataframe with one column (*id*) containing the animal Ids from the **examplePedigree** pedigree. They can be used to illustrate the identification of a population of interest as is shown in the example below.

Usage

```
data(focalAnimals)
```

Format

An object of class data.frame with 327 rows and 1 columns.

Examples

```
library(nprcgenekeepr)
data("focalAnimals")
data("examplePedigree")
any(names(examplePedigree) == "population")
nrow(examplePedigree)
examplePedigree <- setPopulation(
  ped = examplePedigree,
  ids = focalAnimals$id
)
any(names(examplePedigree) == "population")
nrow(examplePedigree)
nrow(examplePedigree[examplePedigree$population, ])
```

---

geneDrop *Simulate gene dropping through a pedigree*

Description

Part of Genetic Value Analysis

Usage

```
geneDrop(
  ids,
  sires,
  dams,
  gen,
  genotype = NULL,
  n = 1000L,
  updateProgress = NULL
)
```

Arguments

ids	A character vector of unique IDs for a set of animals.
sires	A character vector with IDS of the sires for the set of animals. NA is used for missing sires.
dams	A character vector with IDS of the dams for the set of animals. NA is used for missing dams.
gen	An integer vector indicating the generation number for each animal.
genotype	A dataframe containing known genotypes. It has three columns: id, first, and second. The second and third columns contain the integers indicating the observed genotypes.
n	integer indicating the number of iterations to simulate. Default is 1000.
updateProgress	function or NULL. If this function is defined, it will be called during each iteration to update a shiny::Progress object.

Details

The gene dropping method from *Pedigree analysis by computer simulation* by Jean W MacCluer, John L Vandeberg, and Oliver A Ryder (1986) doi:10.1002/zoo.1430050209 is used in the genetic value calculations.

Currently there is no means of handling knowing only one haplotype. It will be easy to add another column to handle situations where only one allele is observed and it is not known to be homozygous or heterozygous. The new fourth column could have a frequency for homozygosity that could be used in the gene dropping algorithm.

The genotypes are using indirection (integer instead of character) to indicate the genes because the manipulation of character strings was found to take 20-35 times longer to perform.

Adding additional columns to `genotype` does not significantly affect the time require. Thus, it is convenient to add the corresponding haplotype names to the dataframe using `first_name` and `second_name`.

Animal IDs (`ids`) must not contain a period ("."). A period is disallowed because it causes problems across software environments (R column-name and formula parsing, file-name extensions, programming-language namespaces, and regular expressions); `geneDrop` additionally relies on the period internally to recover the id and parent of each allele row, so a period-bearing id would silently corrupt the result. IDs containing a period are therefore rejected with an error. The same rule is enforced at data input by `qcStudbook` and honored by all automatically generated IDs.

Animal IDs (`ids`) must also be unique. `geneDrop` indexes each animal's parents and accumulates its simulated alleles by id, so duplicate ids are rejected with an error. This invariant is established upstream by `qcStudbook` (via `removeDuplicates`) and by `kinship`, both of which require unique ids.

Value

A data.frame `V1 ... Vn`, `id`, `parent` A data.frame providing the maternal and paternal alleles for an animal for each iteration. The first `n` columns indicate the allele for each iteration. These are followed by two columns: `id`, the animal's ID, and `parent`, whether the allele came from the sire or dam.

Examples

```
## We usually define `n` to be >= 1000
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
allelesNew <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = NULL, n = 50, updateProgress = NULL
)
genotype <- data.frame(
  id = ped$id,
  first_allele = c(
    NA, NA, "A001_B001", "A001_B002",
    NA, "A001_B002", "A001_B001"
  ),
  second_allele = c(
    NA, NA, "A010_B001", "A001_B001",
    NA, NA, NA
```

```
  ),
  stringsAsFactors = FALSE
)
pedWithGenotype <- addGenotype(ped, genotype)
pedGenotype <- getGVGenotype(pedWithGenotype)
allelesNewGen <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = pedGenotype,
  n = 5, updateProgress = NULL
)
```

`getAncestors` *Recursively collect an individual's ancestors*

Description

Part of Pedigree Sampling From PedigreeSampling.R 2016-01-28

Usage

```
getAncestors(id, ptree)
```

Arguments

`id`	character vector of length 1 having the ID of interest
`ptree`	a list of lists forming a pedigree tree as constructed by `createPedTree(ped)` where ped is a standard pedigree dataframe.

Details

Contains functions to build pedigrees from sub-samples of genotyped individuals.

The goal of sampling is to reduce the number of inbreeding loops in the resulting pedigree, and thus, reduce the amount of time required to perform calculations with SIMWALK2 or similar programs.

Value

A character vector of ancestors for an individual ID.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
ped <- qcStudbook(ped, minParentAge = 0)
pedTree <- createPedTree(ped)
pedLoops <- findLoops(pedTree)
ids <- names(pedTree)
allAncestors <- list()

for (i in seq_along(ids)) {
  id <- ids[[i]]
```

```
  anc <- getAncestors(id, pedTree)
  allAncestors[[id]] <- anc
}
head(allAncestors)
countOfAncestors <- unlist(lapply(allAncestors, length))
idsWithMostAncestors <-
  names(allAncestors)[countOfAncestors == max(countOfAncestors)]
allAncestors[idsWithMostAncestors]
```

`getAnimalsWithHighKinship` *List each animal's high-kinship relatives*

Description

List each animal's high-kinship relatives

Usage

```
getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups, ignore, minAge)
```

Arguments

`kmat`	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.
`ped`	The pedigree information in data.frame format
`threshold`	numeric value representing the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored.
`currentGroups`	list of character vectors of IDs of animals currently assigned to the group. Defaults to character(0) assuming no groups are existent.
`ignore`	list of character vectors representing the sex combinations to be ignored. If provided, the vectors in the list specify if pairwise kinship should be ignored between certain sexes. Default is to ignore all pairwise kinship between females.
`minAge`	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.

Value

A list of named character vectors where each name is an animal Id and the character vectors are made up of animals sharing a kinship value greater than our equal to the `threshold` value.

Examples

```
qcPed <- nprcgenekeepr::qcPed
ped <- qcStudbook(qcPed,
  minParentAge = 2L, reportChanges = FALSE,
  reportErrors = FALSE
)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen, sparse = FALSE)
currentGroups <- list(1L)
currentGroups[[1L]] <- examplePedigree$id[1L:3L]
candidates <- examplePedigree$id[examplePedigree$status == "ALIVE"]
threshold <- 0.015625
kin <- getAnimalsWithHighKinship(kmat, ped, threshold, currentGroups,
  ignore = list(c("F", "F")), minAge = 1.0
)
length(kin) # should be 259
kin[["0DAV0I"]] # should have 34 IDs
```

---

getAutoIdFormat *Get the auto-generated unknown-ID format*

Description

Returns the `sprintf` template used to mint placeholder IDs for unknown parents (see `addUIds`) and to detect them. The format is the single source of truth shared by ID *generation* and ID *detection*.

Usage

```
getAutoIdFormat()
```

Details

The value is read from `getOption("nprcgenekeepr.autoIdFormat")`, defaulting to `"U%04d"` (a leading `"U"` plus a zero-padded integer) for backward compatibility. Set it with `setAutoIdFormat`.

Value

A single character string: the auto-ID `sprintf` format.

See Also

`setAutoIdFormat`, `addUIds`

Examples

```
getAutoIdFormat()
```

getBoxWhiskerDescription *Get Box and Whisker Plot Description*

Description

Returns a description of how box and whisker plots work, suitable for use in popovers and tooltips in the Shiny application.

Usage

```
getBoxWhiskerDescription()
```

Value

Character string containing the box and whisker plot description explaining whiskers, IQR (interquartile range), and outliers.

See Also

modSummaryStatsServer which uses this for popovers

Examples

```
desc <- getBoxWhiskerDescription()
cat(desc)
```

getChangedColsTab *Build the changed-columns tab panel*

Description

Build the changed-columns tab panel

Usage

```
getChangedColsTab(errorLst, pedigreeFileName)
```

Arguments

errorLst	list of errors and changes made by qcStudbook
pedigreeFileName	name of file provided by user on Input tab

Value

HTML formatted error list

getConfigFileName *Get the configuration file name for the system*

Description

Get the configuration file name for the system

Usage

```
getConfigFileName(sysInfo)
```

Arguments

sysInfo	object returned by Sys.info()

Value

Character vector with expected configuration file

Examples

```
library(nprcgenekeepr)
sysInfo <- Sys.info()
config <- getConfigFileName(sysInfo)
```

getCurrentAge *Calculate current age in years from a birth date*

Description

Assumes current date for calculating age.

Usage

```
getCurrentAge(birth)
```

Arguments

birth	birth date(s)

Value

Age in years using the provided birthdate.

Examples

```
library(nprcgenekeepr)
age <- getCurrentAge(birth = as.Date("06/02/2000", format = "%m/%d/%Y"))
```

getDatedFilename *Prepend the date and time to a file name*

Description

Prepend the date and time to a file name

Usage

```
getDatedFilename(filename)
```

Arguments

filename	character vector with name to use in file name

Value

A character string with a file name prepended with the date and time in YYYY-MM-DD_hh_mm_ss_basename format.

Examples

```
library(nprcgenekeepr)
getDatedFilename("testName")
```

getDateErrorsAndConvertDatesInPed *Find date errors and convert dates in a pedigree*

Description

Finds date errors in columns defined in convertDate as dates and converts date strings to Date objects.

Usage

```
getDateErrorsAndConvertDatesInPed(sb, errorLst)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information.
errorLst	object with placeholders for error types found in a pedigree file by qcStudbook through the functions it calls.

Details

If there are no errors that prevent the calculation of exit dates, they are calculated and added to the pedigree otherwise the pedigree is not updated.

Value

A list with the pedigree, sb, and the `errorLst` with invalid date rows (`errorLst$invalidDateRows`)

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedInvalidDates
ped
errorLst <- getEmptyErrorLst()
colNamesAndErrors <- fixColumnNames(names(ped), errorLst)
names(ped) <- colNamesAndErrors$newColNames
pedAndErrors <- getDateErrorsAndConvertDatesInPed(ped, errorLst)
pedAndErrors$sb
pedAndErrors$errorLst
```

---

getDemographics *Get demographic data*

Description

This is a thin wrapper around `labkey.selectRows()`.

Usage

```
getDemographics(colSelect = NULL)
```

Arguments

`colSelect`	(optional) a vector of comma separated strings specifying which columns of a dataset or view to import

Value

A data.frame containing LabKey demographic data with the columns specified in the single parameter provided.

Examples

```
## Not run:
## Needs a connection to a LabKey server
library(nprcgenekeepr)
siteInfo <- getSiteInfo()
colSet <- siteInfo$lkPedColumns
source <- " generated by getDemographics: "
```

```
pedSourceDf <- tryCatch(getDemographics(colSelect = colSet),
  warning = function(wCond) {
    cat(paste0("Warning", source, wCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  },
  error = function(eCond) {
    cat(paste0("Error", source, eCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  }
)

## End(Not run)
```

`getDescendantPedigree` *Reduce a pedigree to a group and its descendants*

Description

Filters a pedigree down to only the descendants of the provided group, building the pedigree forward in time starting from a group of probands. This is the downward (descendants-only) mirror of `getProbandPedigree`: it takes the transitive closure over offspring and returns the probands together with all of their descendants. It does not include collateral relatives (siblings, cousins, or mates).

Usage

```
getDescendantPedigree(probands, ped)
```

Arguments

`probands`	a character vector with the list of animals whose descendants should be included in the final pedigree.
`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.

Value

A reduced pedigree containing the probands and all of their descendants.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
## D's descendants are F and G
getDescendantPedigree(probands = "D", ped = ped)$id
```

getEmptyErrorLst *Create an empty errorLst object*

Description

Create an empty errorLst object

Usage

```
getEmptyErrorLst()
```

Value

An errorLst object with placeholders for error types found in a pedigree file by qcStudbook.

Examples

```
library(nprcgenekeepr)
getEmptyErrorLst()
```

getErrorTab *Build the error-list tab panel*

Description

Build the error-list tab panel

Usage

```
getErrorTab(errorLst, pedigreeFileName)
```

Arguments

errorLst	list of errors and changes made by qcStudbook
pedigreeFileName	name of file provided by user on Input tab

Value

HTML formatted error list

`getFileDirectRelatives` *Get the direct relatives of selected animals from a pedigree file*

Description

File-sourced sibling of `getLkDirectRelatives`: reads a pedigree file through the internal `getPedigreeSource()` `"file"` provider (via `getPedigree`), then delegates the pedigree walk to the source-agnostic `getPedDirectRelatives`. The result is the full connected pedigree component (ancestors, descendants, and collaterals such as siblings and mates) reachable from the focal animals. It is fully offline and deterministic.

Usage

```
getFileDirectRelatives(
  ids,
  fileName = NULL,
  sep = ",",
  unrelatedParents = FALSE
)
```

Arguments

`ids`	character vector of animal IDs
`fileName`	path to a pedigree file (CSV or Excel) read via `getPedigree`; the file must provide at least `id`, `sire`, and `dam` columns.
`sep`	column separator passed to the file reader for delimited text files (default `","`); ignored for Excel files.
`unrelatedParents`	logical vector when `FALSE` the unrelated parents of offspring do not get a record as an ego; when `TRUE` a place holder record where parent (`sire`, `dam`) IDs are set to `NA`.

Details

Unlike the LabKey source, which fails soft (returns `NULL`) when its fetch fails, the file source errors loudly: a `NULL` or missing `fileName`, a file that does not exist, or a file lacking the `id`, `sire`, and `dam` columns each raises an error.

Value

A data.frame with pedigree structure containing all direct relatives – the full connected pedigree component (ancestors, descendants, and collaterals) – for the Ids provided.

See Also

Other direct relatives: `getLkDirectAncestors()`, `getLkDirectRelatives()`, `getPedDirectRelatives()`

Examples

```
library(nprcgenekeepr)
## Build a tiny pedigree file, then pull the relatives of a focal animal.
ped <- data.frame(
  id = c("A", "B", "C"), sire = c(NA, NA, "A"), dam = c(NA, NA, "B"),
  stringsAsFactors = FALSE
)
pedFile <- tempfile(fileext = ".csv")
write.csv(ped, pedFile, row.names = FALSE)
getFileDirectRelatives(ids = "C", fileName = pedFile)
unlink(pedFile)
```

`getFocalAnimalPed` *Get pedigree based on list of focal animals*

Description

Get pedigree based on list of focal animals

Usage

```
getFocalAnimalPed(fileName, sep = ",")
```

Arguments

`fileName`	character vector of temporary file path.
`sep`	column separator in CSV file

Value

A pedigree file compatible with others in this package.

Examples

```
library(nprcgenekeepr)
siteInfo <- getSiteInfo(FALSE)
source <- " generated by getFocalAnimalPed: "
tryCatch(getFocalAnimalPed(fileName = "breeding file.csv"),
  warning = function(wCond) {
    cat(paste0("Warning", source, wCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
  },
  error = function(eCond) {
    cat(paste0("Error", source, eCond),
      name = "nprcgenekeepr"
    )
    return(NULL)
```

```
  }
)
```

`getFocalAnimalPedFromFile` *Get a focal-animal pedigree from a pedigree file*

Description

File-sourced sibling of `getFocalAnimalPed`: reads a list of focal animal Ids from `fileName` (the first column, exactly as `getFocalAnimalPed` does), then builds the focal animals' full connected pedigree component from a SEPARATE pedigree file via `getFileDirectRelatives`. This lets the focal-animal workflow run entirely offline – no LabKey / EHR connection is required.

Usage

```
getFocalAnimalPedFromFile(fileName, pedigreeFileName = NULL, sep = ",")
```

Arguments

`fileName`	character path to a file (CSV, delimited text, or Excel) whose first column is the list of focal animal Ids.
`pedigreeFileName`	character path to the pedigree file (CSV, delimited text, or Excel) read via `getPedigree`; it must provide at least `id`, `sire`, and `dam` columns.
`sep`	column separator passed to the file readers for delimited text files (default `","`); ignored for Excel files.

Details

The underlying file source errors loudly on a bad pedigree file, but this function is the application boundary, so it is fail-soft: it does NOT throw. On failure it returns a classed `nprcgenekeeprFileErr` object whose `message` names the reason – a missing, not-found, or unreadable pedigree file, or one lacking the `id`, `sire`, and `dam` columns. (This mirrors how the app's other file inputs behave – the `message` surfaces as a "File Read Error" – and is distinct from the LabKey path, which returns an `nprcgenekeeprErr`.)

Value

On success, a data.frame with the focal animals' full connected pedigree component (ancestors, descendants, and collaterals), as returned by `getFileDirectRelatives`. On any failure this function does NOT throw: it returns a classed `nprcgenekeeprFileErr` object (a list with a `message` element) naming WHY the read failed – an unreadable focal-id list file; a missing, not-found, unreadable, or wrong-column pedigree file; or no focal IDs present in the pedigree. The application surfaces `message` as the "File Read Error" detail (distinct from the LabKey path, which returns an `nprcgenekeeprErr`).

Examples

```
library(nprcgenekeepr)
## A focal-id file and a pedigree file, then build the pedigree offline.
ped <- data.frame(
  id = c("A", "B", "C"), sire = c(NA, NA, "A"), dam = c(NA, NA, "B"),
  stringsAsFactors = FALSE
)
pedFile <- tempfile(fileext = ".csv")
write.csv(ped, pedFile, row.names = FALSE)
focalFile <- tempfile(fileext = ".csv")
write.csv(data.frame(id = "C"), focalFile, row.names = FALSE)
getFocalAnimalPedFromFile(focalFile, pedFile)
unlink(c(pedFile, focalFile))
```

---

getFounders *Get the founder ids from a pedigree*

Description

Part of Pedigree Curation

Usage

```
getFounders(ped)
```

Arguments

ped	datatable that is the Pedigree. It contains pedigree information. The fields id, sire and dam are required.

Details

A founder is an animal whose sire and dam are both unknown (NA). Animals with exactly one known parent (partial parentage) are **not** founders.

Value

A vector of the id values of the founders, in pedigree order. It has the same type as ped$id and is empty when there are no founders.

See Also

isFounder for the founder logical mask.

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
getFounders(ped)
```

`getGeneticDiversityStats` *Assemble breeding-group genetic diversity heat-map statistics*

Description

Assemble breeding-group genetic diversity heat-map statistics

Usage

```
getGeneticDiversityStats(
  groups,
  ped,
  geneticValues,
  kmat,
  housing = "shelter_pens",
  currentDate = Sys.Date()
)
```

Arguments

`groups`	List of character vectors of animal IDs, one per breeding group (for example the `groups` returned by `modBreedingGroupsServer`). If the list is named, the names become the group row labels; otherwise labels are `"Group 1"`, `"Group 2"`, and so on.
`ped`	Dataframe that is the `Pedigree`. The `id`, `dam`, `sex`, `birth`, and `exit` columns are required; an optional `ancestry` column enables the Origin metric.
`geneticValues`	Dataframe of the genetic value report (for example `reportGV(ped)$report`). The `id` and `value` columns are required; `value` holds the labels produced by `rankSubjects` (`"Low Value"`, `"High Value"`, `"Undetermined"`).
`kmat`	Square kinship matrix whose row and column names are animal IDs and that covers every member of every group (for example the matrix returned by `kinship`).
`housing`	Character housing type passed to `getProductionStatus`, either `"shelter_pens"` or `"corral"`. Length 1 (applied to every group) or one value per group. Defaults to `"shelter_pens"`.
`currentDate`	Date used to derive age and the production birth window. Defaults to `Sys.Date()`.

Details

For each breeding group this builds the heat-map color indices by calling the per-group providers: Value from `getProportionLow`, Origin from `getIndianOriginStatus`, Production from `getProductionStatus`, and Inbreeding from `getKinshipWithMaleStatus`. The result is the group-by-metric data frame consumed by `makeGeneticDiversityHeatmap`: the first column holds the group label and each remaining column holds a color index (1 red, 2 yellow, 3 green).

Age is derived from each member's birth date using `currentDate` rather than read from a possibly-absent `age` column, so the age filters and the production birth window share one reference date. Genetic-value labels of `"Undetermined"` are dropped before the Value proportion is taken. A group with no assessed value, and a group whose Inbreeding metric is undefined (no breeding-age females), are both scored red so that missing data is surfaced rather than shown as healthy green. When the pedigree has no `ancestry` column the Origin metric cannot be computed and its column is omitted.

Value

A data frame with one row per group: the first column `group` holds the group label and each remaining column (`Value`, `Origin` when available, `Production`, `Inbreeding`) holds an integer color index in `c(1, 2, 3)`.

`getGenotypes` *Get genotypes from file*

Description

Get genotypes from file

Usage

```
getGenotypes(fileName, sep = ",")
```

Arguments

`fileName`	character vector of temporary file path.
`sep`	column separator in CSV file

Value

A genotype file compatible with others in this package.

Examples

```
library(nprcgenekeepr)
pedCsv <- getGenotypes(fileName = system.file("testdata", "qcPed.csv",
  package = "nprcgenekeepr"
))
```

getGVGenotype *Extract genotype data for a genetic value report*

Description

Extracts genotype data if available otherwise NULL is returned.

Usage

```
getGVGenotype(ped)
```

Arguments

ped	The pedigree information in data.frame format

Value

A data.frame with the columns id, first, and second extracted from a pedigree object (a data.frame) containing genotypic data. If the pedigree object does not contain genotypic data the NULL is returned.

Examples

```
## We usually define `n` to be >= 1000
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
allelesNew <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = NULL, n = 50, updateProgress = NULL
)
genotype <- data.frame(
  id = ped$id,
  first_allele = c(
    NA, NA, "A001_B001", "A001_B002",
    NA, "A001_B002", "A001_B001"
  ),
  second_allele = c(
    NA, NA, "A010_B001", "A001_B001",
    NA, NA, NA
  ),
  stringsAsFactors = FALSE
)
pedWithGenotype <- addGenotype(ped, genotype)
pedGenotype <- getGVGenotype(pedWithGenotype)
allelesNewGen <- geneDrop(ped$id, ped$sire, ped$dam, ped$gen,
  genotype = pedGenotype,
  n = 5, updateProgress = NULL
)
```

getGVPopulation *Get the population of interest for the Genetic Value analysis*

Description

If user has limited the population of interest by defining pop, that information is incorporated via the ped$population column.

Usage

```
getGVPopulation(ped, pop)
```

Arguments

ped	The pedigree information in data.frame format
pop	character vector with animal IDs to consider as the population of interest. The default is NULL.

Value

A logical vector corresponding to the IDs in the vector of animal IDs provided to the function in pop.

Examples

```
## Example from Analysis of Founder Representation in Pedigrees: Founder
## Equivalents and Founder Genome Equivalents.
## Zoo Biology 8:111-123, (1989) by Robert C. Lacy
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
ped["gen"] <- findGeneration(ped$id, ped$sire, ped$dam)
ped$population <- getGVPopulation(ped, NULL)
```

getIdsWithOneParent *Get ids of animals with only one parent*

Description

Get ids of animals with only one parent

Usage

```
getIdsWithOneParent(uPed)
```

Arguments

uPed	a trimmed pedigree dataframe with uninformative founders removed.

Value

Character vector of all single parents

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
probands <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
p <- addBackSecondParents(p, ped)
nrow(p)
getIdsWithOneParent(p)
```

---

getIncludeColumns *Get the superset of columns that can be in a pedigree file*

Description

Part of Genetic Value Functions

Usage

```
getIncludeColumns()
```

Details

Replaces INCLUDE.COLUMNS data statement.

Value

Superset of columns that can be in a pedigree file.

Examples

```
getIncludeColumns()
```

`getLkDirectAncestors` *Get the direct ancestors of selected animals*

Description

Gets direct ancestors from labkey `study` schema and `demographics` table.

Usage

```
getLkDirectAncestors(ids)
```

Arguments

`ids`	character vector of animal IDs

Value

data.frame with pedigree structure having all of the direct ancestors for the Ids provided.

See Also

Other direct relatives: `getFileDirectRelatives()`, `getLkDirectRelatives()`, `getPedDirectRelatives()`

Examples

```
## Not run:
# Requires LabKey connection
library(nprcgenekeepr)
## Have to a vector of focal animals
focalAnimals <- c("1X2701", "1X0101")
suppressWarnings(getLkDirectAncestors(ids = focalAnimals))

## End(Not run)
```

`getLkDirectRelatives` *Get the direct relatives of selected animals from the LabKey EHR*

Description

Builds the pedigree of relatives for the provided focal animals from the LabKey `study` schema `demographics` table, obtained through the internal `getPedigreeSource()` adapter. The pedigree walk is delegated to `getPedDirectRelatives()`, so the result is the full connected pedigree component (ancestors, descendants, and collaterals such as siblings and mates) reachable from the focal animals.

Usage

```
getLkDirectRelatives(ids, unrelatedParents = FALSE)
```

Arguments

`ids`	character vector of animal IDs
`unrelatedParents`	logical vector when `FALSE` the unrelated parents of offspring do not get a record as an ego; when `TRUE` a place holder record where parent (`sire`, `dam`) IDs are set to `NA`.

Value

A data.frame with pedigree structure containing all direct relatives – the full connected pedigree component (ancestors, descendants, and collaterals) – for the Ids provided.

See Also

Other direct relatives: `getFileDirectRelatives()`, `getLkDirectAncestors()`, `getPedDirectRelatives()`

Examples

```
## Not run:
# Requires LabKey connection
library(nprcgenekeepr)
## Have to a vector of focal animals
focalAnimals <- c("1X2701", "1X0101")
suppressWarnings(getLkDirectRelatives(ids = focalAnimals))

## End(Not run)
```

getOffspring *Get offspring to corresponding animal IDs provided*

Description

Get offspring to corresponding animal IDs provided

Usage

```
getOffspring(pedSourceDf, ids)
```

Arguments

pedSourceDf	dataframe with pedigree structure having at least the columns id, sire, and dam.
ids	character vector of animal IDs

Value

A character vector containing all of the offspring IDs for all of the IDs provided in the second argument `ids`. All offspring are combined and duplicates are removed.

Examples

```
library(nprcgenekeepr)
pedOne <- nprcgenekeepr::pedOne
names(pedOne) <- c("id", "sire", "dam", "sex", "birth")
getOffspring(pedOne, c("s1", "d2"))
```

getParents *Get parents to corresponding animal IDs provided*

Description

Get parents to corresponding animal IDs provided

Usage

```
getParents(pedSourceDf, ids)
```

Arguments

pedSourceDf	dataframe with pedigree structure having at least the columns id, sire, and dam.
ids	character vector of animal IDs

Value

A character vector with the IDs of the parents of the provided ID list.

Examples

```
library(nprcgenekeepr)
pedOne <- nprcgenekeepr::pedOne
names(pedOne) <- c("id", "sire", "dam", "sex", "birth")
getParents(pedOne, c("o1", "d4"))
```

---

getPedDirectRelatives *Get the direct relatives of selected animals from a pedigree*

Description

Gets the direct relatives (ancestors and descendants) of the selected animals from the supplied pedigree (ped).

Usage

```
getPedDirectRelatives(ids, ped, unrelatedParents = FALSE)
```

Arguments

ids	character vector of animal IDs
ped	pedigree dataframe object that is used as the source of pedigree information.
unrelatedParents	logical vector when FALSE the unrelated parents of offspring do not get a record as an ego; when TRUE they get a place holder record as an ego in which the parent (sire, dam) IDs are set to NA.

Value

A data.frame of pedigree records for the selected animals and their direct relatives (ancestors and descendants) in ped.

See Also

Other direct relatives: getFileDirectRelatives(), getLkDirectAncestors(), getLkDirectRelatives()

Examples

```
library(nprcgenekeepr)
## A pedigree to search and a focal animal whose direct relatives we want
ped <- nprcgenekeepr::lacy1989Ped
getPedDirectRelatives(ids = "E", ped = ped)
```

getPedigree *Get pedigree from file*

Description

Get pedigree from file

Usage

```
getPedigree(fileName, sep = ",")
```

Arguments

fileName	character vector of temporary file path.
sep	column separator in CSV file

Value

A pedigree file compatible with others in this package.

Examples

```
library(nprcgenekeepr)
ped <- getPedigree(fileName = system.file("testdata", "qcPed.csv",
  package = "nprcgenekeepr"
))
```

getPedMaxAge *Get the maximum age of any animal in the pedigree*

Description

Returns the maximum age among all animals in the pedigree that have a non-NA age. Because ages are computed for deceased animals (age at exit) as well, the maximum can reflect a deceased animal.

Usage

```
getPedMaxAge(ped)
```

Arguments

ped	The pedigree information in data.frame format

Value

Numeric value representing the maximum age of animals in the pedigree, or NA_real_ if no animal has a non-missing age (no age column or all ages missing).

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
getPedMaxAge(ped)
```

---

getPossibleCols *Get possible column names for a studbook*

Description

Pedigree curation function

Usage

```
getPossibleCols()
```

Value

A character vector of the possible columns that can be in a studbook. The possible columns are as follows:

id	– character vector with unique identifier for an individual
sire	– character vector with unique identifier for an individual's father (NA if unknown).
dam	– character vector with unique identifier for an individual's mother (NA if unknown).
sex	– factor (levels: "M", "F", "U") Sex specifier for an individual
species	– character vector or NA (optional) naming the species of the individual (e.g. "rhesus"). Recognized as a first-class column rather than retained as an unrecognized novel column.
gen	– integer vector with the generation number of the individual
birth	– Date or NA with the individual's birth date. This is one of the required columns (see getRequiredCols); the date value itself may be NA if unknown.
exit	– Date or NA (optional) with the individual's exit date (death, or departure if applicable)

ancestry	– character vector or NA (optional) that indicates the geographic population to which the individual belongs.
age	– numeric or NA (optional) indicating the individual's current age or age at exit.
population	– an optional logical argument indicating whether or not the id is part of the extant population.
origin	– character vector or NA (optional) that indicates the name of the facility that the individual was imported from. NA indicates the individual was not imported.
status	– an optional factor indicating the status of an individual with levels ALIVE, DEAD, and SHIPPED.
condition	– character vector or NA (optional) that indicates the restricted status of an animal. "Nonrestricted" animals are generally assumed to be naive.
spf	– character vector or NA (optional) indicating the specific pathogen-free status of an individual.
vasxOvx	– character vector indicating the vasectomy/ovariectomy status of an animal where NA indicates an intact animal and all other values indicate surgical alteration.
pedNum	– integer vector indicating generation numbers for each id, starting at 0 for individuals lacking IDs for both parents.

Examples

```
library(nprcgenekeepr)
getPossibleCols()
```

`getPotentialParents` *Get potential parents for animals with unknown parents*

Description

[Experimental]

Usage

```
getPotentialParents(
  ped,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  maxGestationalPeriod = NULL,
  gestationTable = NULL
)
```

Arguments

ped	the pedigree information in data.frame format. Pedigree (req. fields: id, sire, dam, gen, population). This requires complete pedigree information.
minSireAge	numeric minimum age in years for a male to be proposed as a potential sire. NULL (default) looks up the floor for each candidate's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all male candidates. Candidates with missing birth dates are not considered.
minDamAge	numeric minimum age in years for a female to be proposed as a potential dam. NULL (default) looks up the floor for each candidate's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor for all female candidates.
minParentAge	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both minSireAge and minDamAge; use those sex-specific parameters instead.
maxGestationalPeriod	integer maximum number of days between conception and birth for the species being analyzed (a conservative upper bound, e.g. 210 for rhesus whose typical gestation is about 165 days). When NULL (the default) the window is looked up per animal from the species column of ped via getSpeciesGestation, falling back to 210 days for animals whose species is missing or unrecognized; supply an explicit integer to use one fixed window for every animal. It is used two ways: (1) a sire who exited the colony between conception (birth - maxGestationalPeriod) and birth is still retained as a candidate; and (2) a female who delivered another offspring within maxGestationalPeriod days of the focal birth is excluded as a candidate dam, because a female bears one offspring at a time. The sire check uses presence at conception while the dam check uses presence at birth; this asymmetry is intentional – a sire need only be present to conceive, whereas a dam must be present through the pregnancy to give birth.
gestationTable	optional data.frame (columns species, gestation) passed to getSpeciesGestation for the per-animal lookup when maxGestationalPeriod is NULL. Defaults to NULL, which uses the bundled speciesGestation table.

Value

a list of list with each internal list being made up of an animal id (id), a vector of possible sires (sires) and a vector of possible dams (dams). The id must be defined while the vectors sires and dams can be empty.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::rhesusPedigree
## getPotentialParents needs a logical fromCenter column flagging
## colony-born animals; add one if your pedigree lacks it.
ped$fromCenter <- TRUE
potentialParents <- getPotentialParents(
  ped = ped, minSireAge = 2, minDamAge = 2, maxGestationalPeriod = 210L
)
```

```
## Each element pairs a focal id with candidate sires and dams.
potentialParents[[1L]]
```

`getPotentialSires` *List potential sires*

Description

List potential sires

Usage

```
getPotentialSires(ids, ped, minAge = 1L)
```

Arguments

`ids`	character vector of animal IDs
`ped`	dataframe that is the `Pedigree`. It contains pedigree information including the IDs listed in `ids`.
`minAge`	integer value giving the inclusive minimum current age (in years) a male must have to be listed as a potential sire. Default is 1 year.

Value

A character vector of potential sire Ids

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedWithGenotype
ids <- nprcgenekeepr::qcBreeders
getPotentialSires(ids, ped, minAge = 1L)
```

`getProbandPedigree` *Reduce a pedigree to probands and their ancestors*

Description

Filters a pedigree down to only the ancestors of the provided group, removing unnecessary individuals from the studbook. This version builds the pedigree back in time starting from a group of probands. This will include all ancestors of the probands, even ones that might be uninformative.

Usage

```
getProbandPedigree(probands, ped)
```

Arguments

probands	a character vector with the list of animals whose ancestors should be included in the final pedigree.
ped	datatable that is the Pedigree. It contains pedigree information. The fields sire and dam are required.

Value

A reduced pedigree.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedWithGenotype
ids <- nprcgenekeepr::qcBreeders
sires <- getPotentialSires(ids, ped, minAge = 1)
head(getProbandPedigree(probands = sires, ped = ped))
```

---

getPyramidAgeDist *Get the age distribution for the pedigree*

Description

Returns the pedigree with all animals, adding a status column describing each animal as ALIVE or DECEASED and a computed age column (age at exit for deceased animals). All animals are returned, not only living ones.

Usage

```
getPyramidAgeDist(ped = NULL)
```

Arguments

ped	The pedigree information in data.frame format

Details

The lubridate package is used here because of the way the modern Gregorian calendar is constructed, there is no straightforward arithmetic method that produces a person's age, stated according to common usage — common usage meaning that a person's age should always be an integer that increases exactly on a birthday.

Value

A pedigree with status column added, which describes the animal as ALIVE or DECEASED and a age column added, which has the animal's age in years or NA if it cannot be calculated. The exit column values have been remapped to valid dates or NA.

Examples

```
library(nprcgenekeepr)
ped <- getPyramidAgeDist()
```

`getPyramidPlot` *Create an age-sex pyramid plot of a pedigree*

Description

The pedigree provided must have the following columns: `sex` and `age`. This needs to be augmented to allow pedigrees structures that are provided by the nprcgenekeepr package.

Usage

```
getPyramidPlot(
  ped = NULL,
  binWidth = 2L,
  ageUnit = "years",
  colorScheme = "default",
  showCounts = TRUE,
  ageLabelCex = 1
)
```

Arguments

`ped`	The pedigree information in data.frame format
`binWidth`	numeric bin width for age groups (default 2).
`ageUnit`	character either "years" (default) or "months".
`colorScheme`	character color scheme: "default" (blue/pink) or "viridis" (colorblind-friendly).
`showCounts`	logical whether to show count values on bars (default TRUE).
`ageLabelCex`	numeric expansion factor for age labels (default 1.0).

Value

The return value of par("mar") when the function was called.

Examples

```
library(nprcgenekeepr)
data(qcPed)
getPyramidPlot(qcPed)
getPyramidPlot(qcPed, binWidth = 5, colorScheme = "viridis")
```

getRequiredCols *Get required column names for a studbook*

Description

Pedigree curation function

Usage

```
getRequiredCols()
```

Value

A character vector of the required columns that can be in a studbook. The required columns are as follows:

- `id` – character vector with unique identifier for an individual
- `sire` – character vector with unique identifier for an individual's father (`NA` if unknown).
- `dam` – character vector with unique identifier for an individual's mother (`NA` if unknown).
- `sex` – factor (levels: "M", "F", "U") Sex specifier for an individual
- `birth` – Date or `NA` (optional) with the individual's birth date

Examples

```
library(nprcgenekeepr)
getRequiredCols()
```

getSiteInfo *Get site information*

Description

Get site information

Usage

```
getSiteInfo(expectConfigFile = TRUE)
```

Arguments

`expectConfigFile`
: logical parameter when set to `FALSE`, no configuration is looked for. Default value is `TRUE`.

Value

A list of site specific information used by the application.

Currently this returns the following character strings in a named list.

1. `center` – One of "SNPRC" or "ONPRC"
2. `baseUrl` – If `center` is "SNPRC", baseUrl is one of "https://boomer.txbiomed.local:8080/labkey" or "https://vger.txbiomed.local:8080/labkey". To allow testing, if `center` is "ONPRC" baseUrl is "https://boomer.txbiomed.local:8080/labkey".
3. `schemaName` – If `center` is "SNPRC", schemaName is "study". If `center` is "ONPRC", schemaName is "study"
4. `folderPath` – If `center` is "SNPRC", folderPath is "/SNPRC". If `center` is "ONPRC", folderPath is "/ONPRC"
5. `queryName` – is "demographics"
6. `requiredCols` – the required studbook columns, from `getRequiredCols`
7. `possibleCols` – the possible studbook columns, from `getPossibleCols`
8. `includeColumns` – the superset of report-inclusion columns, from `getIncludeColumns`

Examples

```
library(nprcgenekeepr)
## default sends warning if configuration file is missing
suppressWarnings(getSiteInfo())
getSiteInfo(expectConfigFile = FALSE)
```

---

getSpeciesGestation *Look up the maximum gestation period (days) for one or more species*

Description

Maps each supplied species name to a conservative upper bound on the number of days from conception to birth, using the `speciesGestation` lookup table (or a supplied `gestationTable`). Matching is case- and whitespace-insensitive. Any species that is missing, `NA`, an empty string, or not present in the table falls back to `default` (210 days, the conservative rhesus bound). Used by `getPotentialParents` to key its gestation window on the first-class `species` pedigree column.

Usage

```
getSpeciesGestation(species, gestationTable = NULL, default = 210L)
```

Arguments

`species`	character vector of species names (may contain `NA`).
`gestationTable`	optional data.frame with a character column `species` and an integer column `gestation` to use instead of the bundled `speciesGestation` table. Defaults to `NULL`, which uses the bundled table.
`default`	integer fallback returned for species that are missing, `NA`, empty, or not found in the table. Defaults to `210L`.

Value

an integer vector of gestation-period day bounds, the same length and order as `species`.

Examples

```
getSpeciesGestation("RHESUS")
getSpeciesGestation(c("RHESUS", "UNICORN", NA))
```

`getSpeciesMinBreedingAge` *Look up the minimum breeding age (years) for one or more species and sexes*

Description

Maps each supplied (species, sex) pair to the minimum age in years at which an animal of that species and sex can produce offspring, using the `speciesGestation` lookup table (or a supplied `breedingTable`). Matching is case- and whitespace-insensitive on both species and sex. Any species that is missing, `NA`, an empty string, or not present in the table – and any sex that is not `"M"` or `"F"` – falls back to `default` (2 years, the legacy package-wide minimum parent age). Used by the Genetic Value Analysis unknown-parent mean-kinship correction to form a focal animal's contemporaneous breeding-age peer cohort. The bundled table is populated for the common colony NHP species; the user-configurable override path is a separate feature.

Usage

```
getSpeciesMinBreedingAge(species, sex, breedingTable = NULL, default = 2)
```

Arguments

`species`	character vector of species names (may contain `NA`).
`sex`	character vector of sexes (`"M"` or `"F"`); recycled to the length of `species` (or vice versa).
`breedingTable`	optional data.frame with a character column `species` and numeric columns `minMaleBreedingAge` and `minFemaleBreedingAge` to use instead of the bundled `speciesGestation` table. Defaults to `NULL`, which uses the bundled table.
`default`	numeric fallback returned for species that are missing, `NA`, empty, or not found, and for a sex that is not `"M"`/`"F"`. Defaults to `2`.

Value

a numeric vector of minimum breeding ages in years, the same length as the longer of `species` and `sex`.

Examples

```
getSpeciesMinBreedingAge("RHESUS", "M")
getSpeciesMinBreedingAge("RHESUS", "F")
getSpeciesMinBreedingAge(c("RHESUS", "UNICORN"), c("M", "F"))
```

`getTokenList` *Get tokens from a character vector of lines*

Description

Get tokens from a character vector of lines

Usage

```
getTokenList(lines)
```

Arguments

`lines`	character vector with text from configuration file

Value

A list with two elements: `param`, a character vector of parameter names, and `tokenVec`, a list of the token vectors parsed for each parameter.

Examples

```
lines <- c(
  "center = \"SNPRC\"",
  " baseUrl = \"https://boomer.txbiomed.local:8080/labkey\"",
  " schemaName = \"study\"", " folderPath = \"/SNPRC\"",
  " queryName = \"demographics\"",
  "lkPedColumns = (\"Id\", \"gender\", \"birth\", \"death\",",
  "              \"lastDayAtCenter\", \"dam\", \"sire\")",
  "mapPedColumns = (\"id\", \"sex\", \"birth\", \"death\", ",
  "  \"exit\", \"dam\", \"sire\")"
)
lkVec <- c(
  "Id", "gender", "birth", "death",
  "lastDayAtCenter", "dam", "sire"
)
mapVec <- c("id", "sex", "birth", "death", "exit", "dam", "sire")
tokenList <- getTokenList(lines)
params <- tokenList$param
tokenVectors <- tokenList$tokenVec
```

getVersion *Get the version number of nprcgenekeepr*

Description

Get the version number of nprcgenekeepr

Usage

```
getVersion(date = TRUE)
```

Arguments

date	A logical value when TRUE (default) a date in YYYY-MM-DD (ISO) format within parentheses is appended.

Value

Current Version

Examples

```
library(nprcgenekeepr)
getVersion()
```

get_and_or_list *Join a character vector into an and/or list*

Description

Join a character vector into an and/or list

Usage

```
get_and_or_list(c_vector, conjunction = "and")
```

Arguments

c_vector	Character vector containing the list of words to be put in a list.
conjunction	The conjunction to be used as the connector. This is usually 'and' or 'or' with 'and' being the default.

Value

A character vector of length one containing the a single correctly punctuated character string that list each element in the first arguments vector with commas between if there are more than two elements with the last two elements joined by the selected conjunction.

Examples

```
get_and_or_list(c("Bob", "John")) # "Bob and John"
get_and_or_list(c("Bob", "John"), "or") # "Bob or John"
get_and_or_list(c("Bob", "John", "Sam", "Bill"), "or")
# "Bob, John, Sam, or Bill"
```

`get_elapsed_time_str` *Format the elapsed time since a start time*

Description

Taken from github.com/rmsharp/rmsutilityr

Usage

```
get_elapsed_time_str(start_time)
```

Arguments

`start_time`	a `proc_time` object as returned by `proc.time()`

Value

A character vector describing the passage of time in hours, minutes, and seconds.

Examples

```
start_time <- proc.time()
## do something
elapsed_time <- get_elapsed_time_str(start_time)
```

`groupAddAssign` *Add animals to a breeding group or form new groups*

Description

Part of Group Formation

Usage

```
groupAddAssign(
  candidates,
  kmat,
  ped,
  currentGroups = list(character(0L)),
  threshold = 0.015625,
  ignore = list(c("F", "F")),
  minAge = 1,
  iter = 1000L,
  numGp = 1L,
  harem = FALSE,
  sexRatio = 0,
  withKin = FALSE,
  updateProgress = NULL
)
```

Arguments

candidates	Character vector of IDs of the animals available for use in forming the groups. The animals that may be present in currentGroups are not included within candidates.
kmat	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.
ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.
currentGroups	List of character vectors of IDs of animals currently assigned to groups. Defaults to a list with character(0) in each sublist element (one for each group being formed) assuming no groups are prepopulated.
threshold	Numeric value indicating the minimum kinship level to be considered in group formation. Pairwise kinship below this level will be ignored. The default value is 0.015625.
ignore	List of character vectors representing the sex combinations to be ignored. If provided, the vectors in the list specify if pairwise kinship should be ignored between certain sexes. Default is to ignore all pairwise kinship between females.
minAge	Integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.
iter	Integer indicating the number of times to perform the random group formation process. Default value is 1000 iterations.
numGp	Integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
harem	Logical variable when set to TRUE, the formed groups have a single male at least minAge old.
sexRatio	Numeric value indicating the ratio of females to males x from 0.5 to 20 by increments of 0.5.

withKin	Logical variable when set to TRUE, the kinship matrix for the group is returned along with the group and score. Defaults to not return the kinship matrix. This maintains compatibility with earlier versions.
updateProgress	Function or NULL. If this function is defined, it will be called during each iteration to update a shiny::Progress object.

Details

groupAddAssign finds the largest group that can be formed by adding unrelated animals from a set of candidate IDs to an existing group, to a new group it has formed from a set of candidate IDs or if more than 1 group is desired, it finds the set of groups with the largest average size.

The function implements a maximal independent set (MIS) algorithm to find groups of unrelated animals. A set of animals may have many different MISs of varying sizes, and finding the largest would require traversing all possible combinations of animals. Since this could be very time consuming, this algorithm produces a random sample of the possible MISs, and selects from these. The size of the random sample is determined by the specified number of iterations.

Value

A list with list items group, score and optionally groupKin. The list item group contains a list of the best group(s) produced during the simulation. The list item score provides the score associated with the group(s). The list item groupKin contains the subset of the kinship matrix that is specific for each group formed.

References

Vinson, A. and Raboin, M.J. (2015) "A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques (*Macaca mulatta*)" *Journal of the American Association for Laboratory Animal Science*, 2015 Nov, Vol.54(6), pp.700-707.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
```

```
geneticValue <- reportGV(ped,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
trimmedGeneticValue <- reportGV(trimmedPed,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
candidates <- trimmedPed$id[trimmedPed$birth < as.Date("2013-01-01") &
  !is.na(trimmedPed$birth) &
  is.na(trimmedPed$exit)]
haremGrp <- groupAddAssign(
  kmat = trimmedGeneticValue[["kinship"]],
  ped = trimmedPed,
  candidates = candidates,
  iter = 10, # should be >= 1000
  numGp = 6,
  harem = TRUE
)
haremGrp$group
sexRatioGrp <- groupAddAssign(
  kmat = trimmedGeneticValue[["kinship"]],
  ped = trimmedPed,
  candidates = candidates,
  iter = 10L, # should be >= 1000L
  numGp = 6L,
  sexRatio = 9.0
)
sexRatioGrp$group
```

---

gvaConvergence *Recommend gene-drop iterations for a pedigree*

Description

Part of Genetic Value Analysis

Usage

```
gvaConvergence(
  ped,
  pop = NULL,
  nMax = 3000L,
  guThresh = 1L,
  byID = TRUE,
  grid = NULL,
```

```
  k = 20L,
  oMin = 0.9,
  rhoMin = 0.95,
  seed = NULL,
  updateProgress = NULL,
  breedingTable = NULL,
  gestationTable = NULL,
  breedingAgeDefault = NULL,
  gestationDefault = NULL,
  kinshipOverrides = NULL
)
```

Arguments

ped	The pedigree information in data.frame format (the same input reportGV takes).
pop	Character vector with animal IDs to consider as the population of interest. The default is NULL (all animals).
nMax	Integer gene-drop budget: the number of iteration columns to simulate. Reproducibility is assessed for iteration counts N with 2 * N <= nMax (each half-split needs 2 * N columns). Default 3000.
guThresh	Integer threshold number of animals for defining a rare (unique) allele, passed to calcGU / calcA. Default 1.
byID	Logical passed to alleleFreq() via calcA; if TRUE, homozygous alleles are counted once per individual. Default TRUE.
grid	Integer vector of candidate iteration counts to assess. The default builds c(25, 50, 100, 200, 400, 800, 1500), keeping only those with 2 * N <= nMax.
k	Integer size of the top-k selected set compared for overlap. Default 20.
oMin	Numeric minimum top-k overlap for reproducibility. Default 0.90.
rhoMin	Numeric minimum Kendall rank agreement for reproducibility. Default 0.95.
seed	Optional integer; when supplied, set_seed pins the gene-drop RNG so the convergence curve is reproducible. Default NULL.
updateProgress	Function or NULL passed through to geneDrop() to update a shiny::Progress object. Default NULL.
breedingTable, gestationTable, breedingAgeDefault, gestationDefault	Optional overrides for the unknown-parent mean-kinship correction, passed through to correctUnknownParentMeanKinship() exactly as reportGV passes them. NULL uses the bundled defaults.
kinshipOverrides	Optional data.frame of outside-information kinship overrides (id1, id2, kinship; the coefficient *f*, not relatedness *r*) applied to the kinship matrix before mean kinship and the unknown-parent correction, exactly as reportGV applies them, so the convergence diagnostic ranks on the same mean kinship the report uses. NULL (the default) leaves the pedigree-derived matrix unchanged. Ids outside the analysis set are warn-dropped (the run is not aborted). An override REFINES the named kinship cell; it does not suppress the + sexMean / 2 unknown-parent correction, which is kept for every animal missing one parent. See applyKinshipOverrides.

Details

Genome uniqueness (calcGU) is the only ranked Genetic Value Analysis output that carries Monte Carlo (gene-drop) sampling noise, so the number of iterations a colony actually needs is pedigree-dependent: there is no single universal "right" count. gvaConvergence answers the literal request – "define reproducible and automate finding the needed number of iterations" – on the ratified definition that the decision-relevant quantity is the *selection order* (which animals are chosen, and in what order), not the precision of the gu number itself.

Because the n gene-drop iteration columns are independent and identically distributed replicates, the whole convergence picture is recoverable from a *single* completed gene drop the user already pays for: gvaConvergence runs one gene drop at nMax, computes the per-iteration rare-allele matrix once (calcA), and for each candidate iteration count N in grid splits the columns into two disjoint halves of N columns each. The two halves are genuinely independent N-iteration estimates; each is ranked through the same ordering pipeline the report uses, and the two orderings are compared. A run is judged **reproducible at** N when both

- the top-k selected animals overlap by at least oMin (the same animals are chosen), and
- the Kendall rank agreement of the commonly-ranked animals is at least rhoMin (they come out in the same order).

The recommended iteration count is the smallest N in grid at which both criteria hold. The de-inflated gu = 0 "Undetermined" set (both parents unknown, no recorded origin) is a policy constant with rank NA; it is excluded from the order the criteria are computed on and reported separately as nUndetermined.

Because the half-split compares two N-column runs to *each other* (never to their pooled mean), it is a conservative, self-validating estimate of reproducibility at N. This changes nothing about seeding: a fixed seed already makes gu bit-identical run to run; that is reproducibility of the *process*, whereas this function reports the sampling reproducibility of the *estimate*.

Value

An object of class nprcgenekeeprGVConv: a list with

- convergence – a data.frame with one row per assessed iteration count: iterations, topOverlap (top-k selected-set overlap, from 0 to 1), and rankAgreement (Kendall rank agreement of the commonly-ranked animals, from -1 to 1).
- recommendedIter – the smallest assessed iteration count meeting both criteria, or NA if none did within grid.
- converged – TRUE if any assessed count met both criteria.
- criteria – the k, oMin, and rhoMin used.
- nRankable – the number of probands carrying a (non-NA) rank that the order metrics are computed on.
- nUndetermined – the count of the excluded Undetermined set.
- nMax – the gene-drop budget actually simulated.

See Also

reportGV, calcGU, calcGUSE

Examples

```
library(nprcgenekeepr)
## A quick, small illustration (use a larger nMax in practice).
conv <- gvaConvergence(nprcgenekeepr::qcPed, nMax = 200L, seed = 1L)
conv$convergence
conv$recommendedIter
```

`hasBothParents` *Check whether an animal has both parents*

Description

Check whether an animal has both parents

Usage

```
hasBothParents(id, ped)
```

Arguments

`id`	character vector of IDs to examine for parents
`ped`	The pedigree information in data.frame format

Value

TRUE if ID has both sire and dam identified in `ped`.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedOne
names(ped) <- c("id", "sire", "dam", "sex", "birth")
hasBothParents("o2", ped)
ped$sire[ped$id == "o2"] <- NA
hasBothParents("o2", ped)
```

hasGenotype *Check for genotype data in dataframe*

Description

Checks to ensure the content and structure are appropriate for genotype data are in the dataframe and ready for the geneDrop function by already being mapped to integers and placed in columns named first and second. These checks are simply based on expected columns and legal domains.

Usage

```
hasGenotype(genotype)
```

Arguments

genotype	dataframe with genotype data

Value

A logical value representing whether or not the data.frame passed in contains genotypic data that can be used. Non-standard column names are accepted for this assessment.

Examples

```
library(nprcgenekeepr)
rhesusPedigree <- nprcgenekeepr::rhesusPedigree
rhesusGenotypes <- nprcgenekeepr::rhesusGenotypes
pedWithGenotypes <- addGenotype(
  ped = rhesusPedigree,
  genotype = rhesusGenotypes
)
hasGenotype(pedWithGenotypes)
```

headerDisplayNames *Convert internal column names to display or header names*

Description

Converts the column names of a Pedigree or Genetic value Report to something more descriptive.

Usage

```
headerDisplayNames(headers)
```

Arguments

headers	a character vector of column (header) names

Value

Updated list of column names

Examples

```
library(nprcgenekeepr)
headerDisplayNames(headers = c("id", "sire", "dam", "sex", "birth", "age"))
```

`isFounder` *Identify the founders in a pedigree*

Description

Part of Pedigree Curation

Usage

```
isFounder(ped)
```

Arguments

`ped`	a pedigree `data.frame` with (at least) the columns `sire` and `dam`.

Details

A founder is an animal whose sire and dam are both unknown (`NA`). Animals with exactly one known parent (partial parentage) are **not** founders.

Value

A logical vector with one element per row of `ped` that is `TRUE` for each animal whose sire and dam are both `NA`. The result never contains `NA`.

See Also

`getFounders` for the founder `id` values.

Examples

```
library(nprcgenekeepr)
ped <- data.frame(
  id = c("A", "B", "C", "D", "E", "F", "G"),
  sire = c(NA, NA, "A", "A", NA, "D", "D"),
  dam = c(NA, NA, "B", "B", NA, "E", "E"),
  stringsAsFactors = FALSE
)
isFounder(ped)
```

is_valid_date_str *Test whether a string is a valid date*

Description

Taken from github.com/rmsharp/rmsutilityr

Usage

```
is_valid_date_str(
  date_str,
  format = "%d-%m-%Y %H:%M:%S",
  optional = FALSE
)
```

Arguments

date_str	character vector with 0 or more dates
format	character vector of length one. Retained for backward compatibility; the current implementation validates dates with `anytime()` and does not use `format`.
optional	logical value indicating that NA should be returned instead of `FALSE` for strings that are not valid dates. Defaults to FALSE.

Value

A logical value or `NA` indicating whether or not the provided character vector represented a valid date string.

Examples

```
is_valid_date_str(c(
  "13-21-1995", "20-13-98", "5-28-1014",
  "1-21-15", "2-13-2098", "25-28-2014"
), format = "%m-%d-%y")
```

kinMatrix2LongForm *Reformat a kinship matrix into long form*

Description

Part of Group Formation

Usage

```
kinMatrix2LongForm(kinMatrix, removeDups = FALSE)
```

Arguments

kinMatrix	numerical matrix of pairwise kinship values. The row and column names correspond to animal IDs.
removeDups	logical value indication whether or not reverse-order ID pairs be filtered out? (i.e., "ID1 ID2 kin_val" and "ID2 ID1 kin_val" will be collapsed into a single entry if removeDups = TRUE)

Value

A dataframe with columns id1, id2, and kinship. This is the kinship data reformatted from a matrix, to a long-format table.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
reformattedKmat <- kinMatrix2LongForm(kmat, removeDups = FALSE)
nrow(reformattedKmat)
reformattedNoDupsKmat <- kinMatrix2LongForm(kmat, removeDups = TRUE)
nrow(reformattedNoDupsKmat)
```

---

kinship *Generate a kinship matrix*

Description

The function previously had an internal call to the kindepth function in order to provide the parameter pdepth (the generation number). This version requires the generation number to be calculated elsewhere and passed into the function.

Usage

```
kinship(id, father.id, mother.id, pdepth, sparse = FALSE)
```

Arguments

id	character vector of IDs for a set of animals.
father.id	character vector or NA for the IDs of the sires for the set of animals.
mother.id	character vector or NA for the IDs of the dams for the set of animals.
pdepth	integer vector indicating the generation number for each animal.
sparse	logical flag. If TRUE, Matrix::Diagnol() is used to make a unit diagonal matrix. If FALSE, base::diag() is used to make a unit square matrix.

Details

The rows (cols) of founders are just 0.5 * identity matrix, no further processing is needed for them. Parents must be processed before their children, and then a child's kinship is just a sum of the kinship's for his or her parents.

The code for the kinship function was written by Terry Therneau at the Mayo clinic and taken from his website. This function is part of a package written in S (and later ported to R) for calculating kinship and other statistics.

Value

A kinship square matrix

Author(s)

Terry M. Therneau, Mayo Clinic (mayo.edu), original version

All of the code on the original S-Plus kinship function (originally hosted on Terry Therneau's Mayo Clinic software page, offline since at least 2019) was stated to be released under the GNU General Public License (version 2 or later).

The R version became the kinship2 package available on CRAN:

as modified by M Raboin, 2014-09-08 14:44:26

References

https://cran.r-project.org/package=kinship2

$Id: kinship.s,v 1.5 2003/01/04 19:07:53 therneau Exp $

Create the kinship matrix, using the algorithm of K Lange, Mathematical and Statistical Methods for Genetic Analysis, Springer, 1997, p 71-72.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
ped$gen <- findGeneration(ped$id, ped$sire, ped$dam)
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
ped
kmat
```

`kinshipMatricesToKValues` *Build a kValue table from a list of kinship matrices*

Description

A kValue matrix has one row for each pair of individuals in the kinship matrix and one column for each kinship matrix. A kValue matrix has one row for each pair of individuals in the kinship matrix and one column for each kinship matrix. Thus, in a kinship matrix with 20 individuals the kinship matrix will have 20 rows by 20 columns but only the upper or lower triangle has unique information as the diagonal values are the self-kinship coefficients, $(1 + F)/2$ (0.5 for a non-inbred animal), and the upper triangle has the same values as the lower triangle. The kValue table will have 210 rows. The calculation for the number or row in the kValue table is $20 + (20 * 19)/2$ rows with the 20 values from the kinship coeficient matrix diagonal and $(20 * 19)/2$ elements from one of either of the two triangles.

Usage

```
kinshipMatricesToKValues(kinshipMatrices)
```

Arguments

kinshipMatrices	list of square matrices of kinship values. May or may not have named rows and columns.

Details

The kValue matrix for 1 kinship matrix for 20 individuals will have 210 rows and 3 columns. The first two columns are dedicated to the ID pairs and the third column contains the pair's kinship coefficient.

Thus, the number of rows in the kValues matrix will be $n + n(n - 1)/2$ and the number of columns will be 2 plus one additional column for each kinship matrix $(2 + n)$.

Value

Dataframe object with columns id_1, id_2, and one kinship column for each kinship matrix in kinshipMatricies where the first two columns contain the IDs of the individuals in the kinship matrix provided to the function and the kinship columms contain the corresponding kinship coefficients. In contrast to the kinship matrix. Each possible pairing of IDs appears once.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
```

```
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKinship <- function(simKinships, id1, id2, simulation) {
  ids <- dimnames(simKinships[[simulation]])[[1]]
  simKinships[[simulation]][
    seq_along(ids)[ids == id1],
    seq_along(ids)[ids == id2]
  ]
}

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[kValue$id_1 == id1 & kValue$id_2 == id2, paste0(
    "sim_",
    simulation
  )]
}

n <- 10
simKinships <- createSimKinships(ped, allSimParents, pop = ped$id, n = n)
kValue <- kinshipMatricesToKValues(simKinships)
extractKValue(kValue, id1 = "A", id2 = "F", simulation = 1:n)
```

---

kinshipMatrixToKValues

Extract a kValue table from a kinship matrix

Description

A kValue matrix has one row for each pair of individuals in the kinship matrix and one column for each kinship matrix. A kValue matrix has one row for each pair of individuals in the kinship matrix and one column for each kinship matrix. Thus, in a kinship matrix with 20 individuals the kinship matrix will have 20 rows by 20 columns but only the upper or lower triangle has unique information as the diagonal values are each animal's self-kinship, $(1 + F)/2$ (0.5 when not inbred), and the upper triangle has the same values as the lower triangle. The kValue table will have 210 rows. The calculation for the number or row in the kValue table is $20 + (20 * 19)/2$ rows with the 20 values from the kinship coeficient matrix diagonal and $(20 * 19)/2$ elements from one of either of the two triangles.

Usage

```
kinshipMatrixToKValues(kinshipMatrix)
```

Arguments

kinshipMatrix	square kinship matrix. May or may not have named rows and columns.

Details

The kValue matrix for 1 kinship matrix for 20 individuals will have 210 rows and 3 columns. The first two columns are dedicated to the ID pairs and the third column contains the pair's kinship coefficient.

Thus, the number of rows in the kValues matrix will be $n + n(n - 1)/2$ and the number of columns will be 3.

Value

data.frame object with columns id_1, id_2, and kinship where the first two columns contain the IDs of the individuals in the kinship matrix provided to the function and the kinship columm contains the corresponding kinship coefficient. In contrast to the kinship matrix. Each possible pairing of IDs appears once.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
```

```
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKinship <- function(simKinships, id1, id2, simulation) {
  ids <- dimnames(simKinships[[simulation]])[[1]]
  simKinships[[simulation]][
    seq_along(ids)[ids == id1],
    seq_along(ids)[ids == id2]
  ]
}

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[
    kValue$id_1 == id1 & kValue$id_2 == id2,
    paste0("sim_", simulation)
  ]
}

simPed <- makeSimPed(ped, allSimParents)
simKinship <- kinship(
  simPed$id, simPed$sire,
  simPed$dam, simPed$gen
)
kValues <- kinshipMatrixToKValues(simKinship)
```

`lacy1989Ped` *Small hypothetical pedigree (Lacy 1989)*

Description

Small hypothetical pedigree (Lacy 1989)

Usage

```
data(lacy1989Ped)
```

Format

An object of class `data.frame` with 7 rows and 5 columns.

Source

lacy1989Ped is a dataframe containing the small hypothetical pedigree of three founders and four descendants used by Robert C. Lacy in "Analysis of Founder Representation in Pedigrees: Founder Equivalents and Founder Genome Equivalents" Zoo Biology 8:111-123 (1989).

The founders (`A`, `B`, `E`) have unknown parentages and are assumed to have independent ancestries.

id character column of animal IDs

sire the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam the female parent of the animal indicated by the `id` column. Unknown dams are indicated with `NA`

gen generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

population logical vector with all values set TRUE

`lacy1989PedAlleles` *Gene-drop alleles for lacy1989Ped (5000 iterations)*

Description

A dataframe produced by `geneDrop` on `lacy1989Ped` with 5000 iterations.

Usage

```
data(lacy1989PedAlleles)
```

Format

An object of class `data.frame` with 14 rows and 5002 columns.

Source

lacy1989Ped is a dataframe containing the small example pedigree used by Robert C. Lacy in "Analysis of Founder Representation in Pedigrees: Founder Equivalents and Founder Genome Equivalents" Zoo Biology 8:111-123 (1989).

There are 5000 columns, one for each iteration in `geneDrop` containing alleles randomly selected at each generation of the pedigree using Mendelian rules.

Column 5001 is the `id` column with two rows for each member of the pedigree (2 * 7).

Column 5002 is the `parent` column with values of `sire` and `dam` alternating.

`loadSiteConfig` *Load the site configuration for the modular Shiny application*

Description

Reads the user's site-configuration file (`~/.nprcgenekeepr_config`, or `~/_nprcgenekeepr_config` on Windows) using the tolerant `getSiteInfo` parser, which handles the documented configuration format (comment lines, blank lines, and multi-line / quoted / comma-separated values; see `inst/extdata/example_nprcgenekeepr_config`). The call is wrapped in `tryCatch` so that a missing or malformed configuration file can never crash the application on boot: in that case a warning is logged and `NULL` is returned.

Usage

```
loadSiteConfig()
```

Details

This replaces a former `read.table(sep = "=")` call in the application server that assumed a strict two-column table, could not parse the documented format, and crashed `runGeneKeepR` at startup.

Value

A named list of site information (as returned by `getSiteInfo`) when a configuration file is present and parseable; otherwise `NULL`.

See Also

`getSiteInfo`, `getConfigFileName`, `appServer`

Examples

```
library(nprcgenekeepr)
## Reads ~/.nprcgenekeepr_config (or ~/_nprcgenekeepr_config on
## Windows) if it exists; returns NULL when no config file is
## present, so this is safe to run on any machine.
config <- loadSiteConfig()
config
```

loadSpeciesOverrides *Load user-configurable species reproductive-parameter overrides*

Description

Reads the optional species-override settings from the user's site configuration file (`~/.nprcgenekeepr_config`, or `~/_nprcgenekeepr_config` on Windows) and assembles the override tables and fallbacks consumed by the Genetic Value Analysis. The configuration may carry up to three optional keys, each looked up softly (the `getConfigApiKey` pattern – absent keys are not an error and never touch the fixed-schema `getSiteInfo` parser):

- `speciesOverridesPath` – path to a CSV with the four `speciesGestation` columns (`species`, `gestation`, `minMaleBreedingAge`, `minFemaleBreedingAge`; header required, matched by name). A colony lists only the rows (species) it wants to change; the CSV is **merged onto** the bundled table, so every unlisted species keeps its bundled value (not replaced).
- `minBreedingAgeDefault` – numeric fallback (years) for a species absent from the table (bundled built-in 2.0).
- `gestationDefault` – integer fallback (days) for a species absent from the table (bundled built-in 210).

Usage

```
loadSpeciesOverrides()
```

Details

Like `loadSiteConfig`, this never crashes the application on boot: a missing configuration file, a missing override key, or a missing/malformed CSV all fall back to the bundled values (a warning is raised for an unreadable CSV).

Value

A named list with elements `breedingTable`, `gestationTable` (each the merged `speciesGestation`-shaped data.frame, or `NULL` when no CSV is configured), `breedingAgeDefault` (numeric or `NULL`), and `gestationDefault` (integer or `NULL`). A `NULL` element means "use the bundled value / built-in default".

See Also

`loadSiteConfig`, `getSpeciesMinBreedingAge`, `getSpeciesGestation`, `reportGV`

Examples

```
library(nprcgenekeepr)
## Reads optional species overrides from the user's site config
## file; with no config file every element is NULL, meaning "use
## the bundled speciesGestation values / built-in defaults".
overrides <- loadSpeciesOverrides()
str(overrides)
```

logModuleEvent *Log module events*

Description

Centralized logging function for Shiny module events. Provides consistent logging format across all modules with configurable log levels.

Usage

```
logModuleEvent(module, message, level = "INFO", ...)
```

Arguments

module	character. Name of the module generating the log message.
message	character. The log message to record.
level	character. Log level: "DEBUG", "INFO", "WARN", or "ERROR". Defaults to "INFO".
...	Additional arguments passed to the log message (for sprintf-style formatting).

Value

Invisible NULL. Called for side effect of logging.

See Also

safeExecute for error-safe execution with logging

Examples

```
logModuleEvent("modInput", "File uploaded successfully")
logModuleEvent("modPedigree", "Processing %d animals", level = "DEBUG", 100)
logModuleEvent("modGeneticValue", "Calculation failed", level = "ERROR")
```

makeCEPH *Make a CEPH-style pedigree for each id*

Description

Part of Relations

Usage

```
makeCEPH(id, sire, dam)
```

Arguments

id	character vector with unique identifier for an individual
sire	character vector with unique identifier for an individual's father (NA if unknown).
dam	character vector with unique identifier for an individual's mother (NA if unknown).

Details

Creates a CEPH-style pedigree for each id, consisting of three generations: the id, the parents, and the grandparents. Inserts NA for unknown pedigree members.

Value

List of lists: fields: id, subfields: parents, pgp, mgp. Pedigree information converted into a CEPH-style list. The top level list elements are the IDs from id. Below each ID is a list of three elements: parents (sire, dam), paternal grandparents (pgp: sire, dam), and maternal grandparents (mgp: sire, dam).

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
pedCEPH <- makeCEPH(ped$id, ped$sire, ped$dam)
head(ped)
head(pedCEPH$F)
```

---

makeExamplePedigreeFile

Write copy of nprcgenekeepr::examplePedigree into a file

Description

Uses examplePedigree data structure to create an example data file

Usage

```
makeExamplePedigreeFile(
  file = file.path(tempdir(), "examplePedigree.csv"),
  fileType = "csv"
)
```

Arguments

file	character vector of length one providing the file name
fileType	character vector of length one with possible values of "txt", "csv", or "excel". Default value is "csv".

Value

Full path name of file saved.

Examples

```
library(nprcgenekeepr)
pedigreeFile <- makeExamplePedigreeFile()
```

`makeFounderStatsTable` *Create Founder Statistics HTML Table*

Description

Generates an HTML table displaying founder statistics including counts of known founders, male founders, female founders, founder equivalents (FE), and founder genome equivalents (FG).

Usage

```
makeFounderStatsTable(founderStats)
```

Arguments

`founderStats`	list containing founder statistics with elements:

- `total` - Total number of known founders
- `nMaleFounders` - Number of male founders
- `nFemaleFounders` - Number of female founders
- `fe` - Founder equivalents
- `fg` - Founder genome equivalents
- `fgSE` - (optional) sampling standard error of `fg`; when finite it is shown inline as `FG +/- SE`

Value

Character string containing HTML table markup.

See Also

`makeGeneticSummaryTable` for genetic value summary

`calcFE` for founder equivalents calculation

`calcFG` for founder genome equivalents

Examples

```
stats <- list(
  total = 50,
  nMaleFounders = 20,
  nFemaleFounders = 30,
  fe = 25.5,
  fg = 22.3
)
html <- makeFounderStatsTable(stats)
```

`makeGeneticDiversityHeatmap` *Make a genetic diversity heat map*

Description

Renders a red/yellow/green stoplight heat map of breeding-group genetic diversity metrics. Each row is a breeding group and each column is a metric; every cell is colored by its color index, where 1 is red (the problem condition), 2 is yellow (watch), and 3 is green (healthy).

Usage

```
makeGeneticDiversityHeatmap(stats)
```

Arguments

`stats`	A data frame with one row per breeding group. The first column holds the group label; every remaining column is a metric whose values are color indices in `c(1, 2, 3)`.

Details

This function is agnostic to the number of metric columns: it draws one tile per group-by-metric cell for whatever metric columns it is handed, preserving their input order across the top of the plot.

Value

A `ggplot` object: a `geom_tile` heat map with metric headers across the top and group labels down the left, filled red/yellow/green from the color indices.

Examples

```
stats <- data.frame(
  group = c("Group_1", "Group_2"),
  Value = c(1, 3), Origin = c(2, 3),
  Production = c(3, 2), Inbreeding = c(1, 2)
)
p <- makeGeneticDiversityHeatmap(stats)
```

makeGeneticSummaryTable

Create Genetic Summary Statistics HTML Table

Description

Generates an HTML table displaying summary statistics (Min, Q1, Mean, Median, Q3, Max) for mean kinship and genome uniqueness values.

Usage

```
makeGeneticSummaryTable(geneticValues)
```

Arguments

geneticValues	data.frame containing genetic value columns, in either of two accepted vocabularies: • meanKinship / genomeUniqueness (the legacy names), or • indivMeanKin / gu (reportGV's own $report column names) Both are normalized internally, so reportGV(ped)$report may be passed directly.

Value

Character string containing HTML table markup.

See Also

makeFounderStatsTable for founder statistics

Examples

```
gv <- data.frame(
  meanKinship = c(0.1, 0.2, 0.3, 0.4, 0.5),
  genomeUniqueness = c(0.9, 0.8, 0.7, 0.6, 0.5)
)
html <- makeGeneticSummaryTable(gv)
```

makeGroupMembers *Make the initial groupMembers animal list*

Description

Make the initial groupMembers animal list

Usage

```
makeGroupMembers(numGp, currentGroups, candidates, ped, harem, minAge)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.
currentGroups	list of character vectors of IDs of animals currently assigned to the group. Defaults to character(0) assuming no groups are existent.
candidates	character vector of IDs of the animals available for use in the group.
ped	dataframe that is the Pedigree. It contains pedigree information including the IDs listed in ids.
harem	logical variable when set to TRUE, the formed groups have a single male at least minAge old.
minAge	integer value indicating the minimum age to consider in group formation. Pairwise kinships involving an animal of this age or younger will be ignored. Default is 1 year.

Value

Initial groupMembers list

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
candidates <- nprcgenekeepr::qcBreeders
## Non-harem: pre-seed group 1 with animals already assigned; a
## second, empty group is initialized ready to be filled.
currentGroups <- list(candidates[1L:3L])
groupMembers <- makeGroupMembers(
  numGp = 2L, currentGroups = currentGroups, candidates = candidates,
  ped = ped, harem = FALSE, minAge = 1L
)
groupMembers
## Harem: each group is seeded with one available male (uses sample()).
set.seed(1L)
haremMembers <- makeGroupMembers(
  numGp = 2L, currentGroups = list(), candidates = candidates,
```

```
  ped = ped, harem = TRUE, minAge = 1L
)
haremMembers
```

---

makeGroupNum *Make the initial grpNum list*

Description

Make the initial grpNum list

Usage

```
makeGroupNum(numGp)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.

Value

Initial grpNum list

Examples

```
library(nprcgenekeepr)
## Create the initial grpNum list for three groups
grpNum <- makeGroupNum(numGp = 3L)
grpNum
```

---

makeGrpNum *Deprecated alias for makeGroupNum*

Description

makeGrpNum has been renamed to makeGroupNum for consistency with makeGroupMembers. It remains as a deprecated wrapper that issues a warning and then calls makeGroupNum.

Usage

```
makeGrpNum(numGp)
```

Arguments

numGp	integer value indicating the number of groups that should be formed from the list of IDs. Default is 1.

Value

Initial grpNum list

See Also

makeGroupNum

`makeRelationClassesTable` *Make a relation classes table from kinship pairs*

Description

From Relations

Usage

```
makeRelationClassesTable(kin)
```

Arguments

`kin`	a dataframe with columns `id1`, `id2`, `kinship`, and `relation`. It is a long-form table of pairwise kinships, with relationship categories included for each pair.

Value

A data.frame with the number of instances of following relationship classes: Parent-Offspring, Full-Siblings, Half-Siblings, Grandparent-Grandchild, Full-Cousins, Cousin - Other, Full-Avuncular, Avuncular - Other, Other, and No Relation.

Examples

```
library(nprcgenekeepr)
suppressMessages(library(dplyr))

qcPed <- nprcgenekeepr::qcPed
qcPed <- qcPed[1:50, ] # Comment out for full example
bkmat <- kinship(qcPed$id, qcPed$sire, qcPed$dam, qcPed$gen,
  sparse = FALSE
)
kin <- convertRelationships(bkmat, qcPed)
relClasses <- makeRelationClassesTable(kin)
relClasses$`Relationship Class` <-
  as.character(relClasses$`Relationship Class`)
relClassTbl <- kin[!kin$relation == "Self", ] |>
  group_by(relation) |>
  summarise(count = n())
relClassTbl
```

makeSimPed *Make a simulated pedigree from representative sires and dams*

Description

For each id in allSimParents, an *unknown* (NA) sire or dam is imputed by a random draw from the supplied representative vectors (sires and dams); a *known* sire or dam is preserved unchanged. When a parent is unknown and its representative vector is empty, that parent remains NA.

Usage

```
makeSimPed(ped, allSimParents, verbose = FALSE)
```

Arguments

ped	The pedigree information in data.frame format
allSimParents	list made up of lists where the internal list has the offspring ID id, a vector of representative sires (sires), and a vector of representative dams (dams).
verbose	logical vector of length one that indicates whether or not to print out when an animal is missing a sire or a dam.

Details

The algorithm assigns parents randomly from the lists of possible sires and dams and does not prevent a dam from being selected more than once within the same breeding period. While this is probably not introducing a large error, it is not ideal.

Value

simulated pedigree in data.frame format with the id, sire, and dam.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::lacy1989Ped
## For each id below, any unknown sire/dam is replaced by a random
## draw from the supplied representative sires and dams.
allSimParents <- list(
  list(
    id = "A",
    sires = c("s1_1", "s1_2", "s1_3"),
    dams = c("d1_1", "d1_2", "d1_3", "d1_4")
  ),
  list(
    id = "B",
    sires = c("s2_1", "s2_2", "s2_3"),
    dams = c("d2_1", "d2_2", "d2_3", "d2_4")
  ),
```

```
  list(
    id = "E",
    sires = c("s3_1", "s3_2", "s3_3"),
    dams = c("d3_1", "d3_2", "d3_3", "d3_4")
  )
)
set.seed(1)
simPed <- makeSimPed(ped, allSimParents)
simPed
```

`mapIdsToObfuscated` *Map IDs to Obfuscated IDs*

Description

This is not robust as it fails if all IDs are found not within `map`.

Usage

```
mapIdsToObfuscated(ids, map)
```

Arguments

`ids`	character vector with original IDs
`map`	named character vector where the values are the obfuscated IDs and the vector of names (`names(map)`) is the vector of original names.

Value

A character vector with original IDs replaced by their obfuscated counterparts.

See Also

Other obfuscation: `obfuscateDate()`, `obfuscateId()`, `obfuscatePed()`

Examples

```
set_seed(1)
ped <- qcStudbook(nprcgenekeepr::pedSix)
obfuscated <- obfuscatePed(ped, map = TRUE)
someIds <- c("s1", "s2", "d1", "d1")
mapIdsToObfuscated(someIds, obfuscated$map)
```

`meanKinship` *Calculate mean kinship for each animal in a kinship matrix*

Description

Part of Genetic Value Analysis

Usage

```
meanKinship(kmat)
```

Arguments

`kmat`	a numeric matrix of pairwise kinship coefficients. Animal IDs are the row and column names.

Details

The mean kinship of animal *i* is

$$MK_i = \Sigma f_i j / N$$

, in which the summation is over all animals, *j*, including the kinship of animal *i* to itself.

Value

A named numeric vector of average kinship coefficients for each animal ID. Elements are named with the IDs from the columns of kmat.

References

Ballou JD, Lacy RC. 1995. Identifying genetically important individuals for management of genetic variation in pedigreed populations, p 77-111. In: Ballou JD, Gilpin M, Foose TJ, editors. Population management for survival and recovery. New York (NY): Columbia University Press.

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::qcPed
kmat <- kinship(ped$id, ped$sire, ped$dam, ped$gen)
head(meanKinship(kmat))
```

modBreedingGroupsServer

Breeding Groups Module - Server Function

Description

Server logic for breeding group formation using the groupAddAssign algorithm. This module integrates with the kinship-based maximal independent set (MIS) algorithm to form optimal breeding groups that minimize relatedness within groups while maximizing group sizes.

Usage

```
modBreedingGroupsServer(
  id,
  pedigree,
  geneticValues = NULL,
  kinshipMatrix = NULL,
  kinshipOverrides = NULL
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning pedigree data frame with columns: id, sire, dam, sex, and optionally birth, exit, gen.
geneticValues	optional reactive returning genetic value results from modGeneticValueServer, used to source the topRanked animal-source candidate list. Unrelated to kinship.
kinshipMatrix	optional reactive returning a kinship matrix, typically a full-pedigree matrix shared with modSummaryStatsServer (e.g. from appServer) rather than independently recomputed. If NULL, the module calculates kinship from the pedigree.
kinshipOverrides	optional reactive returning a validated outside-information kinship-override data frame (id1, id2, kinship); see applyKinshipOverrides. When the module recomputes kinship from the pedigree (the shared kinshipMatrix is unavailable), the overrides are applied to that matrix so group formation reflects them regardless of tab order. NULL (the default) is a no-op. A provided kinshipMatrix is expected to already carry overrides applied at its source.

Details

The module supports multiple configuration options:

- **Animal source**: Select top-ranked animals or all available
- **Kinship threshold**: Maximum allowed kinship within groups

- **Harem mode**: Form groups with exactly one male each
- **Sex ratio**: Target female-to-male ratio in groups

Value

List with reactive components:

- `groups` - List of character vectors with animal IDs per group
- `nGroups` - Number of groups formed
- `score` - Optimization score from groupAddAssign (minimum group size)
- `unassigned` - Character vector of candidate IDs not placed in groups
- `groupKinship` - List of kinship matrices per group (if withKin=TRUE)

See Also

`modBreedingGroupsUI` for the UI component

`groupAddAssign` for the underlying MIS algorithm

`modGeneticValueServer` for genetic value analysis

`kinship` for kinship matrix calculation

Other Shiny modules: `modBreedingGroupsUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

`modBreedingGroupsUI` *Breeding Groups Module - UI Function*

Description

Breeding Groups Module - UI Function

Usage

```
modBreedingGroupsUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A `div` containing breeding group formation UI.

References

Vinson, A. and Raboin, M.J. (2015) "A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques (*Macaca mulatta*)" *Journal of the American Association for Laboratory Animal Science*, 2015 Nov, Vol.54(6), pp.700-707.

See Also

modBreedingGroupsServer

groupAddAssign for group formation algorithm.

Other Shiny modules: modBreedingGroupsServer(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modGeneticDiversityServer

Genetic Diversity Module - Server Function

Description

Assembles the live breeding-group, genetic-value, and kinship reactive inputs into the per-group heat-map statistics (via getGeneticDiversityStats) and renders the red/yellow/green heat map (via makeGeneticDiversityHeatmap). When breeding groups have not been formed or the genetic value analysis has not been run, the module shows guidance instead of an empty plot.

Usage

```
modGeneticDiversityServer(
  id,
  groups,
  pedigree,
  geneticValues,
  kinshipMatrix,
  currentDate = Sys.Date()
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
groups	reactive returning a list of character vectors of animal IDs, one per breeding group (the groups returned by modBreedingGroupsServer).
pedigree	reactive returning the quality-controlled pedigree data frame.
geneticValues	reactive returning the genetic value report data frame (with id and value columns).

kinshipMatrix	reactive returning the full kinship matrix (row and column names are animal IDs).
currentDate	Date used to derive age and the production birth window. Defaults to Sys.Date().

Value

A list with two reactive elements: stats, the per-group metric data frame (or NULL when data are not ready), and heatmap, the ggplot heat map (or NULL).

See Also

modGeneticDiversityUI

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modGeneticDiversityUI *Genetic Diversity Module - UI Function*

Description

Genetic Diversity Module - UI Function

Usage

```
modGeneticDiversityUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div containing the genetic diversity heat-map UI: a housing-type selector, a guidance area, and the heat-map plot.

See Also

modGeneticDiversityServer

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

modGeneticValueServer *Genetic Value Analysis Module - Server Function*

Description

Genetic Value Analysis Module - Server Function

Usage

```
modGeneticValueServer(id, pedigree, speciesOverrides = reactive(NULL))
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning pedigree data frame.
speciesOverrides	reactive returning the user-configurable species overrides loaded at boot by loadSpeciesOverrides (a list with breedingTable, gestationTable, breedingAgeDefault, gestationDefault), or NULL. Threaded into reportGV. Defaults to reactive(NULL) so no config file means bundled behavior.

Value

List with geneticValues, topAnimals, nAnalyzed, kinshipMatrix, founderStats, maleFounders, and femaleFounders.

References

Lacy, R.C. (1989) *Zoo Biology*, **8**, 111-123.

See Also

modGeneticValueUI

modBreedingGroupsServer for using results.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

modGeneticValueUI *Genetic Value Analysis Module - UI Function*

Description

Genetic Value Analysis Module - UI Function

Usage

```
modGeneticValueUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div containing genetic value analysis UI.

References

Vinson, A. and Raboin, M.J. (2015) *Journal of the American Association for Laboratory Animal Science*, **54**(6), 700-707.

See Also

modGeneticValueServer

geneDrop for gene dropping simulation.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

modGvAndBgDescServer *Genetic Value and Breeding Group Description Module - Server Function*

Description

Server logic for genetic value and breeding group description module. This module is primarily informational and does not require reactive logic.

Usage

```
modGvAndBgDescServer(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

NULL (no reactive outputs).

See Also

modGvAndBgDescUI for the user interface.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modGvAndBgDescUI *Genetic Value and Breeding Group Description Module - UI Function*

Description

Creates user interface displaying detailed documentation about genetic value analysis and breeding group formation algorithms.

Usage

```
modGvAndBgDescUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing the description HTML.

See Also

modGvAndBgDescServer for server logic.

modGeneticValueUI for genetic value analysis.

modBreedingGroupsUI for breeding group formation.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modInputServer` *Data Input and Quality Control Module - Server Function*

Description

Server logic for data input module handling file uploads, parsing of pedigree and genotype data files, and quality control validation.

Usage

```
modInputServer(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A list with reactive components:

- `cleanedStudbook` - The QC-cleaned studbook data
- `genotypeData` - Genotype data if provided
- `qcSummary` - Summary of QC results (error/warning counts)
- `minSireAge` - The minimum sire age floor (numeric, or `NULL` to use the species+sex breeding-age table default)
- `minDamAge` - The minimum dam age floor (numeric, or `NULL` to use the species+sex breeding-age table default)
- `isReady` - Logical indicating if data is ready for next step
- `debugMode` - Logical reflecting the Input tab's "Debug on" checkbox
- `changedCols` - Renamed/changed-column diagnostics from QC
- `errorLst` - The QC error list, used for dynamic tab management
- `pedigreeFileName` - The uploaded file's name, used for dynamic tab management

Note

(Developer note) This module is the reference implementation of the package's internal Shiny module contract – see `docs/architecture/module-contract.md` in the package source (a developer-only file; it does not ship with the installed package).

See Also

modInputUI for the user interface.

modPedigreeServer for using the cleaned data.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modInputUI *Data Input and Quality Control Module - UI Function*

Description

Creates user interface for data input including file uploads for pedigree and genotype data with various format options, followed by quality control validation.

Usage

```
modInputUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing the data input UI.

See Also

modInputServer for server logic.

modPedigreeUI for pedigree browsing after QC.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

modORIPReportingServer *ORIP Reporting Module - Server Function*

Description

Server logic for ORIP reporting module. Generates summary statistics and formatted reports for Office of Research Infrastructure Programs submissions.

Usage

```
modORIPReportingServer(
  id,
  pedigree = NULL,
  geneticValues = NULL,
  siteConfig = NULL
)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigree	reactive returning pedigree data frame.
geneticValues	reactive returning genetic value analysis results.
siteConfig	reactive returning site configuration from getSiteInfo().

Value

A list with reactive components for ORIP reporting.

See Also

modORIPReportingUI for the user interface

getSiteInfo for site configuration

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modORIPReportingUI` *ORIP Reporting Module - UI Function*

Description

Creates user interface for ORIP (Office of Research Infrastructure Programs) reporting. This module will contain formatted reports suitable for submission to ORIP as part of primate center grant reporting requirements.

Usage

```
modORIPReportingUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Details

The ORIP Reporting tab provides summary statistics and formatted reports for submission to the Office of Research Infrastructure Programs. This includes:

- Colony demographics summary
- Genetic diversity metrics
- Breeding program statistics
- Founder representation analysis

Value

A `div` object containing the ORIP reporting UI.

See Also

`modORIPReportingServer` for server logic.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modORIPReportingServer()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

`modPedigreeServer` *Pedigree Browser Module - Server Function*

Description

Server logic for pedigree browser module handling focal animal selection, pedigree processing, filtering, and data export.

Usage

```
modPedigreeServer(id, studbook)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`studbook`	reactive returning the cleaned studbook data from modInput.

Details

This module processes the studbook by:

- Adding a `population` column via `setPopulation()`
- Adding a `pedNum` column via `findPedigreeNumber()`
- Ensuring a `gen` column exists via `findGeneration()`
- Optionally trimming to the ancestors and descendants of focal animals via `trimPedigree()` and `getDescendantPedigree()`

Value

A list of reactive values:

- `pedigree` - Filtered pedigree for display (respects trim/unknown settings)
- `processedPedigree` - Full pedigree with population, pedNum, gen columns
- `focalAnimals` - Character vector of focal animal IDs
- `nAnimals` - Count of animals in filtered pedigree
- `populationCount` - Count of animals marked as population
- `isReady` - Logical indicating if pedigree data is ready

See Also

modPedigreeUI for the UI component

setPopulation for population marking

trimPedigree for pedigree trimming

findPedigreeNumber for pedigree numbering

findGeneration for generation calculation

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

---

modPedigreeUI *Pedigree Browser Module - UI Function*

Description

Creates user interface for browsing and filtering pedigree data, including focal animal selection, trimming options, and export.

Usage

```
modPedigreeUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing pedigree browser UI.

See Also

modPedigreeServer for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

`modPotentialParentsServer` *Potential Parents Module - Server Function*

Description

Server logic for the Potential Parents module. On button press, it calls `getPotentialParents` against the current pedigree, flattens the result into a sortable table, and exposes it for CSV download. The surface degrades gracefully when no pedigree is loaded, when the pedigree lacks the `fromCenter` colony-origin field, or when no in-colony animal has an unknown parent.

Usage

```
modPotentialParentsServer(
  id,
  pedigree = NULL,
  minSireAge = NULL,
  minDamAge = NULL,
  gestationTable = NULL,
  gestationDefault = NULL
)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`pedigree`	reactive returning the current pedigree data.frame.
`minSireAge`	minimum age in years for a male to be proposed as a sire. May be a plain numeric, `NULL`, or a reactive returning either. `NULL` (the default) uses the species- and sex-specific breeding-age table default via `getPotentialParents`.
`minDamAge`	minimum age in years for a female to be proposed as a dam. Same forms and default as `minSireAge`, applied to females.
`gestationTable`	optional species-to-gestation lookup passed to `getSpeciesGestation` when defaulting the gestation window; `NULL` (the default) uses the bundled `speciesGestation` table. Supplied at boot from the user-configurable species overrides, so a colony's CSV values drive the prefill default.
`gestationDefault`	optional integer fallback (days) for a pedigree whose species is absent from `gestationTable`, passed through to the gestation prefill; `NULL` (the default) keeps the built-in 210. Supplied at boot from the user-configurable species overrides.

Value

A list of reactive expressions:

- `potentialParents` - the raw `getPotentialParents` result (or `NULL`).

- `tableData` - the flattened results data.frame.
- `gestationDefault` - the species-keyed default gestation window (days) used to prefill the maximum-gestational-period input.

See Also

`modPotentialParentsUI` for the user interface.

`getPotentialParents` for the underlying computation.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modPyramidUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

`modPotentialParentsUI` *Potential Parents Module - UI Function*

Description

Creates the user interface for identifying potential parents of in-colony animals that have at least one unknown parent. The user sets a maximum gestational period, presses a button to compute candidate sires and dams on the current pedigree, views a sortable results table, and downloads the results as CSV.

Usage

```
modPotentialParentsUI(id)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.

Value

A `div` object containing the Potential Parents UI.

See Also

`modPotentialParentsServer` for server logic.

`getPotentialParents` for the underlying computation.

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPyramidServer()`, `modPyramidUI()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

modPyramidServer *Age-Sex Pyramid Module - Server Function*

Description

Age-Sex Pyramid Module - Server Function

Usage

```
modPyramidServer(id, pedigreeData)
```

Arguments

id	character vector of length 1. Module namespace identifier.
pedigreeData	reactive returning pedigree data frame.

Value

A list with two elements: pedigree, a reactive returning the pedigree data frame, and animalCount, a reactive returning the number of animal rows.

See Also

modPyramidUI

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidUI(), modSummaryStatsServer(), modSummaryStatsUI()

modPyramidUI *Age-Sex Pyramid Module - UI Function*

Description

Age-Sex Pyramid Module - UI Function

Usage

```
modPyramidUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A `div` containing age-sex pyramid UI.

See Also

`modPyramidServer`

Other Shiny modules: `modBreedingGroupsServer()`, `modBreedingGroupsUI()`, `modGeneticDiversityServer()`, `modGeneticDiversityUI()`, `modGeneticValueServer()`, `modGeneticValueUI()`, `modGvAndBgDescServer()`, `modGvAndBgDescUI()`, `modInputServer()`, `modInputUI()`, `modORIPReportingServer()`, `modORIPReportingUI()`, `modPedigreeServer()`, `modPedigreeUI()`, `modPotentialParentsServer()`, `modPotentialParentsUI()`, `modPyramidServer()`, `modSummaryStatsServer()`, `modSummaryStatsUI()`

`modSummaryStatsServer` *Summary Statistics Module - Server Function*

Description

Server logic for summary statistics module displaying genetic analysis results including kinship statistics, histograms, box plots, and relationship designation analysis.

Usage

```
modSummaryStatsServer(
  id,
  geneticValues,
  pedigree,
  kinshipMatrix = NULL,
  founderStats = NULL,
  kinshipOverrides = NULL
)
```

Arguments

`id`	character vector of length 1. Module namespace identifier.
`geneticValues`	reactive returning genetic value analysis results. Must be a data frame with columns `id`, `indivMeanKin`, and `gu`. Optional `zScore` column enables z-score plots.
`pedigree`	reactive returning pedigree data frame with columns `id`, `sire`, `dam`, and `sex`. Optionally `gen`.
`kinshipMatrix`	optional reactive returning kinship matrix. If NULL, the module will calculate kinship from the pedigree.
`founderStats`	optional reactive returning a list of founder statistics (`fe`, `fg`, `total`, `nMaleFounders`, `nFemaleFounders`). When supplied, a founder summary table is rendered on the Summary Statistics tab (monolith parity). If NULL, it is omitted.

kinshipOverrides
: optional reactive returning a validated outside-information kinship-override data frame (id1, id2, kinship); see applyKinshipOverrides. When the module recomputes kinship from the pedigree (the usual path), the overrides are applied to that matrix, so the relationship table and the kinship CSV export reflect the supplied values regardless of tab order. The override moves the kinship *value* only; the relation *label* stays pedigree-derived (it is computed from pedigree structure, not from the kinship value). Overridden pairs are flagged with a logical overridden column in the relationship table. NULL (the default) is a no-op.

Details

This module provides:

- Summary statistics (counts, mean kinship, genome uniqueness)
- Histograms and box plots for genetic value distributions
- Relationship classification using convertRelationships()
- Relationship class frequency tables using makeRelationClassesTable()
- First-order relative counts using countFirstOrder()
- Export functionality for kinship matrix, founders, and relationships

Value

A list with reactive components:

- summaryData - Summary statistics (nAnimals, meanMK, meanGU)
- relationships - Pairwise relationship designations from convertRelationships(). When kinshipOverrides are supplied, a logical overridden column flags the pairs whose kinship value came from an override.
- relationClasses - Relationship class frequency table from makeRelationClassesTable()
- firstOrderCounts - First-order relative counts per animal from countFirstOrder()
- mkSummary - Six-number summary of mean kinship
- guSummary - Six-number summary of genome uniqueness

See Also

modSummaryStatsUI for the user interface

convertRelationships for relationship classification

makeRelationClassesTable for relationship class summary

countFirstOrder for first-order relative counting

kinship for kinship matrix calculation

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsUI()

modSummaryStatsUI *Summary Statistics Module - UI Function*

Description

Creates user interface for summary statistics display including histograms and box plots for mean kinship, z-scores, and genome uniqueness.

Usage

```
modSummaryStatsUI(id)
```

Arguments

id	character vector of length 1. Module namespace identifier.

Value

A div object containing summary statistics UI.

See Also

modSummaryStatsServer for server logic.

Other Shiny modules: modBreedingGroupsServer(), modBreedingGroupsUI(), modGeneticDiversityServer(), modGeneticDiversityUI(), modGeneticValueServer(), modGeneticValueUI(), modGvAndBgDescServer(), modGvAndBgDescUI(), modInputServer(), modInputUI(), modORIPReportingServer(), modORIPReportingUI(), modPedigreeServer(), modPedigreeUI(), modPotentialParentsServer(), modPotentialParentsUI(), modPyramidServer(), modPyramidUI(), modSummaryStatsServer()

obfuscateDate *Obfuscate dates with a random day offset*

Description

Get the base_date add a random number of days taken from a uniform distribution bounded by -max_delta and max_delta. Insure the resulting date is as least as large as the min_date.

Usage

```
obfuscateDate(baseDate, minDate, maxDelta = 30L)
```

Arguments

baseDate	list of Date objects with dates to be obfuscated
minDate	list object of Date objects that has the lower bound of resulting obfuscated dates
maxDelta	integer vector that is used to create min and max arguments to runif (runif(n, min = 0, max = 1))

Value

A vector of dates that have be obfuscated.

See Also

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateId()`, `obfuscatePed()`

Examples

```
library(nprcgenekeepr)
someDates <- rep(
  as.Date(c("2009-2-16", "2016-2-16"), format = "%Y-%m-%d"),
  10
)
minBirthDate <- rep(as.Date("2009-2-16", format = "%Y-%m-%d"), 20)
obfuscateDate(someDates, minBirthDate, 30L)
```

---

obfuscateId *Create ID aliases of a specified length*

Description

ID aliases are pseudorandom sequences of alphanumeric upper case characters where the letter "O" is not included for readability.. User has the option of providing a character vector of aliases to avoid using. Because aliases are alphanumeric, they never contain a period ("."), honoring the ID rule enforced at data input by qcStudbook.

Usage

```
obfuscateId(id, size = 10L, existingIds = character(0L))
```

Arguments

id	character vector of IDs to be obfuscated (alias creation).
size	character length of each alias
existingIds	character vector of existing aliases to avoid duplication.

Value

A named character vector of aliases where the name is the original ID value.

See Also

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateDate()`, `obfuscatePed()`

Examples

```
library(nprcgenekeepr)
integerIds <- 1L:10L
obfuscateId(integerIds, size = 4L)
characterIds <- paste0(paste0(sample(LETTERS, 1L, replace = FALSE)), 1L:10L)
obfuscateId(characterIds, size = 4L)
```

`obfuscatePed` *Obfuscate a pedigree by aliasing IDs and shifting dates*

Description

User provides a pedigree object (`ped`), the number of characters to be used for alias IDs (`size`), and the maximum number of days that the birthdate can be shifted (`maxDelta`).

Usage

```
obfuscatePed(
  ped,
  size = 6L,
  maxDelta = 30L,
  existingIds = character(0L),
  map = FALSE
)
```

Arguments

`ped`	The pedigree information in data.frame format
`size`	integer value indicating number of characters in alias IDs
`maxDelta`	integer value indicating maximum number of days that the birthdate can be shifted
`existingIds`	character vector of existing aliases to avoid duplication.
`map`	logical if `TRUE` a list object is returned with the new pedigree and a named character vector with the names being the original IDs and the values being the new alias values. Defaults to `FALSE`.

Value

An obfuscated pedigree

See Also

Other obfuscation: `mapIdsToObfuscated()`, `obfuscateDate()`, `obfuscateId()`

Examples

```
library(nprcgenekeepr)
ped <- qcStudbook(nprcgenekeepr::pedGood)
obfuscatedPed <- obfuscatePed(ped)
ped
obfuscatedPed
```

`offspringCounts` *Tabulate offspring counts, optionally by population*

Description

Optionally find the number that are part of the population of interest.

Usage

```
offspringCounts(probands, ped, considerPop = FALSE)
```

Arguments

`probands`	character vector of egos for which offspring should be counted.
`ped`	the pedigree information in datatable format. Pedigree (req. fields: id, sire, dam, gen, population). This is the complete pedigree.
`considerPop`	logical value indication whether or not the number of offspring that are part of the focal population are to be counted? Default is `FALSE`.

Value

A dataframe containing the column `totalOffspring` (and `livingOffspring` when `considerPop` is `TRUE`), with the animal ids as the data frame row names.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
counts <- offspringCounts(probands, ped)
```

ped1Alleles *Gene-drop alleles example (baboon pedigree)*

Description

A dataframe created by the geneDrop function.

Usage

```
data(ped1Alleles)
```

Format

A dataframe with 554 rows and 6 variables

V1 alleles assigned to the parents of the animals identified in the id column during iteration 1 of gene dropping performed by geneDrop.

V2 alleles assigned to the parents of the animals identified in the id column during iteration 2 of gene dropping performed by geneDrop.

V3 alleles assigned to the parents of the animals identified in the id column during iteration 3 of gene dropping performed by geneDrop.

V4 alleles assigned to the parents of the animals identified in the id column during iteration 4 of gene dropping performed by geneDrop.

id character vector of animal IDs provided to the gene dropping function geneDrop.

parent the parent type ("sire" or "dam") of the parent who supplied the alleles as assigned during each of the 4 gene dropping iterations performed by geneDrop.

Source

example baboon pedigree file provided by Deborah Newman, Southwest National Primate Center.

pedDuplicateIds *Example studbook with a duplicated record*

Description

A data frame with 9 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree with a duplicated record.

Usage

```
data(pedDuplicateIds)
```

Format

An object of class `data.frame` with 9 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

---

pedFemaleSireMaleDam *Example studbook with sex-mismatched parents*

Description

A data frame with 8 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree with the errors of having a sire labeled as female and a dam labeled as male.

Usage

```
data(pedFemaleSireMaleDam)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

---

pedGood *Valid example studbook (no QC errors)*

Description

A data frame with 8 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree with no errors.

Usage

```
data(pedGood)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (pedDuplicateIds, pedFemaleSireMaleDam, pedGood, pedInvalidDates, pedMissingBirth, pedSameMaleIsSireAndDam) used to demonstrate error detection by the qcStudbook function.

---

pedInvalidDates *Example studbook with invalid birth dates*

Description

A data frame with 8 rows and 5 columns (id, sire, dam, sex, birth) representing a full pedigree with values in the birth column that are not valid dates.

Usage

```
data(pedInvalidDates)
```

Format

An object of class data.frame with 8 rows and 5 columns.

Details

It is one of six pedigrees (pedDuplicateIds, pedFemaleSireMaleDam, pedGood, pedInvalidDates, pedMissingBirth, pedSameMaleIsSireAndDam) used to demonstrate error detection by the qcStudbook function.

---

pedMissingBirth *Example studbook missing the birth date column*

Description

A data frame with 8 rows and 4 columns (ego_id, sire.id, dam_id, sex) representing a full pedigree that is missing the birth_date column.

Usage

```
data(pedMissingBirth)
```

Format

An object of class data.frame with 8 rows and 4 columns.

Details

It is one of six pedigrees (pedDuplicateIds, pedFemaleSireMaleDam, pedGood, pedInvalidDates, pedMissingBirth, pedSameMaleIsSireAndDam) used to demonstrate error detection by the qcStudbook function.

pedOne *Raw pedigree-file fragment for testing (5 columns)*

Description

A loadable version of a pedigree file fragment used for testing and demonstration.

Usage

```
data(pedOne)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Examples

```
library(nprcgenekeepr)
data("pedOne")
head(pedOne)
```

pedSameMaleIsSireAndDam *Example studbook with a male as both sire and dam*

Description

A data frame with 8 rows and 5 columns (ego_id, sire.id, dam_id, sex, birth_date) representing a full pedigree in which the same male animal is listed as both a sire and a dam.

Usage

```
data(pedSameMaleIsSireAndDam)
```

Format

An object of class `data.frame` with 8 rows and 5 columns.

Details

It is one of six pedigrees (`pedDuplicateIds`, `pedFemaleSireMaleDam`, `pedGood`, `pedInvalidDates`, `pedMissingBirth`, `pedSameMaleIsSireAndDam`) used to demonstrate error detection by the qcStudbook function.

pedSix *Raw pedigree-file fragment for testing (7 columns)*

Description

A loadable version of a pedigree file fragment used for testing and demonstration.

Usage

```
data(pedSix)
```

Format

An object of class `data.frame` with 8 rows and 7 columns.

Examples

```
library(nprcgenekeepr)
data("pedSix")
head(pedSix)
```

pedWithGenotype *Pedigree with simulated genotypes (from qcPed)*

Description

A dataframe produced from qcPed by adding made up genotypes.

A dataframe containing 280 records with 12 columns: `id`, `sire`, `dam`, `sex`, `gen`, `birth`, `exit`, `age`, `first`, `second`, `first_name`, and `second_name`.

Usage

```
data(pedWithGenotype)
```

Format

An object of class `data.frame` with 280 rows and 12 columns.

pedWithGenotypeReport *Genetic-value report for pedWithGenotype*

Description

A list containing the output of `reportGV`.

Usage

```
data(pedWithGenotypeReport)
```

Format

An object of class `list` (inherits from `nprcgenekeeprGV`) of length 11.

Source

pedWithGenotypeReport was made with pedWithGenotype as input into reportGV with 10,000 iterations.

pedWithGenotypeReport is a simple example report for use in examples and unit tests. It was created using the following commands.

- set_seed(10)
- pedWithGenotypeReport <- reportGV(nprcgenekeepr::pedWithGenotype, guIter = 10000)
- save(pedWithGenotypeReport, file = "data/pedWithGenotypeReport.RData")

Examples

```
pedWithGenotypeReport <- nprcgenekeepr::pedWithGenotypeReport
```

print.summary.nprcgenekeeprErr *Print an nprcgenekeepr summary object*

Description

Print an nprcgenekeepr summary object

Usage

```
## S3 method for class 'summary.nprcgenekeeprErr'
print(x, ...)

## S3 method for class 'summary.nprcgenekeeprGV'
print(x, ...)
```

Arguments

x	object of class summary.nprcgenekeeprErr and class list
...	further arguments passed to the print() call for the suspicious-parents table (and ignored by the GV method)

Value

An object to send to the generic print function

object to send to generic print function

Examples

```
library(nprcgenekeepr)
errorLst <- qcStudbook(nprcgenekeepr::pedInvalidDates,
  reportChanges = TRUE, reportErrors = TRUE
)
summary(errorLst)
library(nprcgenekeepr)
ped <- nprcgenekeepr::pedGood
ped <- suppressWarnings(qcStudbook(ped, reportErrors = FALSE))
summary(reportGV(ped, guIter = 10))
```

---

processQcStudbookResult

Process qcStudbook Result into UI-Friendly Format

Description

Converts the errorLst object returned by qcStudbook (when reportErrors=TRUE) into a format suitable for display in the Shiny UI.

Usage

```
processQcStudbookResult(errorLst)
```

Arguments

errorLst	list object returned by qcStudbook with reportErrors = TRUE, or NULL. Expected to be of class nprcgenekeeprErr containing error fields such as femaleSires, maleDams, sireAndDam, duplicateIds, invalidIdChars, missingColumns, invalidDateRows, suspiciousParents, failedDatabaseConnection, and changedCols.

Value

A list with the following components:

- `errors` - data.frame with columns Row, Error, Details
- `warnings` - data.frame with columns Row, Warning, Details
- `changedCols` - list of changed column information
- `hasErrors` - logical indicating if any errors were found
- `hasChangedCols` - logical indicating if columns were renamed

See Also

`qcStudbook` for generating the errorLst input

`runQcStudbook` for a wrapper that uses this function

`checkErrorLst` for checking if errorLst has errors

`checkChangedColsLst` for checking column changes

Examples

```
library(nprcgenekeepr)
## Turn a qcStudbook error list into UI-friendly data frames.
errorLst <- qcStudbook(nprcgenekeepr::pedFemaleSireMaleDam,
  reportErrors = TRUE
)
result <- processQcStudbookResult(errorLst)
result$hasErrors
result$errors
```

---

qcBreeders *Potential breeder IDs (29 baboons)*

Description

A character vector of 29 baboon IDs that are potential breeders.

Usage

```
data(qcBreeders)
```

Format

An object of class `character` of length 29.

Source

qcBreeders is a character vector of 3 males and 26 females from the `qcPed` data set.

These 29 animal IDs are used for examples and unit tests. They were initially selected for having low kinship coefficients.

qcPed *Example quality-controlled baboon pedigree*

Description

A data frame with 280 rows and 8 columns.

id character column of animal IDs

sire the male parent of the animal indicated by the `id` column.

dam the female parent of the animal indicated by the `id` column.

sex sex of the animal indicated by the `id` column.

gen generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth birth date in `Date` format of the animal indicated by the `id` column.

exit exit date in `Date` format of the animal indicated by the `id` column.

age age in year (numeric) of the animal indicated by the `id` column.

Usage

```
data(qcPed)
```

Format

An object of class `data.frame` with 280 rows and 8 columns.

qcPedGvReport *Genetic-value report for qcPed*

Description

qcPedGvReport is a genetic value report for illustrative purposes only. It is used in examples and unit tests with the nprcgenekeepr package. It was created using the following commands.

- set_seed(10)
- qcPedGvReport <- reportGV(nprcgenekeepr::qcPed, guIter = 10000)
- save(qcPedGvReport, file = "data/qcPedGvReport.RData")

Usage

```
data(qcPedGvReport)
```

Format

An object of class `list` (inherits from `nprcgenekeeprGV`) of length 11.

Examples

```
qcPedGvReport <- nprcgenekeepr::qcPedGvReport
```

---

qcStudbook *Run quality control on a studbook or pedigree*

Description

Main pedigree curation function that performs basic quality control on pedigree information

Usage

```
qcStudbook(
  sb,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  reportChanges = FALSE,
  reportErrors = FALSE
)
```

Arguments

sb	A dataframe containing a table of pedigree and demographic information. The function recognizes the following columns (optional columns will be used if present, but are not required): • id — Character vector with Unique identifier for all individuals • sire — Character vector with unique identifier for the father of the current id • dam — Character vector with unique identifier for the mother of the current id • sex — Factor (levels: "M", "F", "U") Sex specifier for an individual • birth — Date or NA (optional) with the individual's birth date • departure — Date or NA (optional) an individual was sold or shipped from the colony • death — date or NA (optional) Date of death, if applicable • status — Factor (levels: ALIVE, DEAD, SHIPPED) (optional) Status of an individual • origin — Character or NA (optional) Facility an individual originated from, if other than ONPRC • ancestry — Character or NA (optional) Geographic population to which the individual belongs • spf — Character or NA (optional) Specific pathogen-free status of an individual

- vasxOvx — Character or NA (optional) Indicator of the vasectomy/ovariectomy status of an animal; NA if animal is intact, assume all other values indicate surgical alteration
- condition — Character or NA (optional) Indicator of the restricted status of an animal. "Nonrestricted" animals are generally assumed to be naive.

minSireAge — numeric minimum age in years for a male to have sired an offspring. NULL (default) looks up the floor for each sire's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor.

minDamAge — numeric minimum age in years for a female to have borne an offspring. NULL (default) looks up the floor for each dam's species via getSpeciesMinBreedingAge (falling back to 2 years when the species is missing or unknown); a supplied value overrides that floor.

minParentAge — **[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both minSireAge and minDamAge; use those sex-specific parameters instead.

reportChanges — logical value that if TRUE, the errorLst contains the list of changes made to the column names. Default is FALSE.

reportErrors — logical value if TRUE will scan the entire file and report back changes made to input and errors in a list of list where each sublist is a type of change or error found. Changes will include column names, case of categorical values (male, female, unknown), etc. Errors will include missing columns, invalid date rows, male dams, female sires, and records with one or more parents below minimum age of parents.

The following changes are made to the cols.

- Column cols are converted to all lower case
- Periods (".") within column cols are collapsed to no space ""
- egoid is converted to id
- sireid is convert to sire
- damid is converted to dam

If the dataframe (sb does not contain the five required columns (id, sire, dam, sex), and birth the function throws an error by calling stop().

Animal IDs (id, sire, dam) must be alphanumeric with no symbols; in particular a period (".") is not allowed. Periods cause problems across software environments (R column-name and formula parsing, file-name extensions, programming-language namespaces, and regular expressions), so any id, sire, or dam value containing a period is treated as an error. With reportErrors == TRUE the offending values are returned in errorLst$invalidIdChars; otherwise the function throws an error. All automatically generated IDs (see addUIds) honor this rule.

If the id field has the string *UNKNOWN* (any case) or both the fields sire or dam have NA or *UNKNOWN* (any case), the record is removed. If either of the fields sire or dam have the string *UNKNOWN* (any case), they are replaced with a unique identifier with the form Unnnn, where nnnn represents one of a series of sequential integers representing the number of missing sires and dams right justified in a pattern of 0000. See addUIds function.

The function addParents is used to add records for parents missing their own record in the pedigree.

The function convertSexCodes is used with ignoreHerm == TRUE to convert sex codes according to the following factors of standardized codes:

- F – replacing "FEMALE" or "2"
- M – replacing "MALE" or "1"
- H – replacing "HERMAPHRODITE" or "4", if ignore.herm == FALSE
- U – replacing "HERMAPHRODITE" or "4", if ignore.herm == TRUE
- U – replacing "UNKNOWN" or "3"

The function correctParentSex is used to ensure no parent is both a sire and a dam. If this error is detected, the function throws an error and halts the program.

The function convertStatusCodes converts status indicators to the following factors of standardized codes. Case of the original status value is ignored.

- "ALIVE" — replacing "alive", "A" and "1"
- "DECEASED" — replacing "deceased", "DEAD", "D", "2"
- "SHIPPED" — replacing "shipped", "sold", "sale", "s", "3"
- "UNKNOWN" — replacing is.na(status)
- "UNKNOWN" — replacing "unknown", "U", "4"

The function convertAncestry coverts ancestry indicators using regular expressions such that the following conversions are made from character strings that match selected substrings to the following factors.

- "INDIAN" — replacing "ind" and not "chin"
- "CHINESE" — replacing "chin" and not "ind"
- "HYBRID" — replacing "hyb" or "chin" and "ind"
- "JAPANESE" — replacing "jap"
- "UNKNOWN" — replacing NA
- "OTHER" — replacing not matching any of the above

The function convertDate converts character representations of dates in the columns birth, death, departure, and exit to dates using the as.Date function.

The function setExit uses heuristics and the columns death and departure to set exit if it is not already defined.

The function calcAge uses the birth and the exit columns to define the age column. The numerical values is rounded to the nearest 0.1 of a year. If exit is not defined, the current system date (Sys.Date()) is used.

The function findGeneration is used to define the generation number for each animal in the pedigree.

The function removeDuplicates checks for any duplicated records and removes the duplicates. I also throws an error and stops the program if an ID appears in more than one record where one or more of the other columns have a difference.

Columns that cannot be used subsequently are removed and the rows are ordered by generation number and then ID.

Finally the columns id sire, and dam are coerce to character.

Value

A data.frame with standardized and quality controlled pedigree information.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
ped <- qcStudbook(examplePedigree,
  minSireAge = 2.0, minDamAge = 2.0, reportChanges = FALSE,
  reportErrors = FALSE
)
names(ped)
```

---

rankSubjects *Rank animals by genetic value*

Description

Part of Genetic Value Analysis Adds a column to `rpt` containing integers from 1 to nrow, and provides a value designation for each animal of "high value" or "low value"

Usage

```
rankSubjects(rpt)
```

Arguments

`rpt`	a list of data.frame (req. colnames: value) containing genetic value data for the population. Dataframes separate out those animals that are imports, those that have high genome uniqueness (gu > 10%), those that have low mean kinship (mk < 0.25), and the remainder.

Value

A list of dataframes with value and ranking information added.

References

Vinson, A. and Raboin, M.J. (2015) "A Practical Approach for Designing Breeding Groups to Maximize Genetic Diversity in a Large Colony of Captive Rhesus Macaques (*Macaca mulatta*)" *Journal of the American Association for Laboratory Animal Science*, 2015 Nov, Vol.54(6), pp.700-707.

Examples

```
library(nprcgenekeepr)
finalRpt <- nprcgenekeepr::finalRpt
rpt <- rankSubjects(nprcgenekeepr::finalRpt)
rpt[["highGu"]][1, "value"]
rpt[["highGu"]][1, "rank"]
rpt[["lowMk"]][1, "value"]
rpt[["lowMk"]][1, "rank"]
rpt[["lowVal"]][1, "value"]
rpt[["lowVal"]][1, "rank"]
```

---

readKinshipOverrides *Read a kinship overrides table from a file*

Description

Reads an outside-information kinship override table from a user-supplied file into a data frame for `checkKinshipOverrides` and `reportGV`. The expected long form is the output of `kinMatrix2LongForm`: columns `id1`, `id2`, and `kinship`, with a header row, so a user can export the current matrix, edit a few rows, and feed it back. Excel (`.xls`/`.xlsx`) and delimited text (`.csv`/`.txt`) files are both accepted, mirroring `getGenotypes`.

Usage

```
readKinshipOverrides(fileName, sep = ",")
```

Arguments

`fileName`	character vector of length one; path to the override file (typically the temporary `datapath` from a Shiny file upload).
`sep`	column separator for delimited text files (default `","`).

Details

This reader does not validate structure or domain – that is `checkKinshipOverrides`'s job. `kinship` is the kinship coefficient *f*, **not** the coefficient of relatedness *r* (= 2*f* for non-inbred animals).

Value

A data frame of the rows read from `fileName` (typically with columns `id1`, `id2`, and `kinship`). Validate it with `checkKinshipOverrides` before use.

Examples

```
## Not run:
overrides <- readKinshipOverrides(fileName = "kinship_overrides.csv")
overrides <- checkKinshipOverrides(overrides)

## End(Not run)
```

removeAutoGenIds *Remove automatically generated IDs from pedigree*

Description

Identifies automatically generated IDs via `isGeneratedUnknownId()`, the shared detection predicate derived from the configurable auto-ID format (see `getAutoIdFormat`; default a leading "U"). Routing detection through that single predicate is the "function call" the former inline leading-"U" check was flagged to become.

Usage

```
removeAutoGenIds(ped)
```

Arguments

`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `id`, `sire` and `dam` are required.

Value

A pedigree with automatically generated IDs removed.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
length(examplePedigree$id)
ped <- removeAutoGenIds(examplePedigree)
length(ped$id)
```

removeDuplicates *Remove duplicate records from pedigree*

Description

Part of Pedigree Curation

Usage

```
removeDuplicates(ped, reportErrors = FALSE)
```

Arguments

ped	dataframe that is the Pedigree. It contains pedigree information. The id and recordStatus columns are required.
reportErrors	logical value if TRUE the function returns a character vector of duplicate id values found among original records (or NULL when none are found) instead of the de-duplicated pedigree.

Details

Returns an updated dataframe with duplicate rows removed.

Returns an error if the table has duplicate IDs with differing data.

Value

When reportErrors is FALSE, a Pedigree object with duplicate rows removed; when reportErrors is TRUE, a character vector of duplicate id values (or NULL when none are found).

Examples

```
ped <- nprcgenekeepr::smallPed
newPed <- cbind(ped, recordStatus = rep("original", nrow(ped)))
ped1 <- removeDuplicates(newPed)
nrow(newPed)
nrow(ped1)
pedWithDups <- rbind(newPed, newPed[1:3, ])
ped2 <- removeDuplicates(pedWithDups)
nrow(pedWithDups)
nrow(ped2)
```

---

removeEarlyDates *Remove dates before a specified year*

Description

Dates before a specified year are set to NA. This is often used for dates formed from malformed character representations such as a date in %m-%d-%Y format being read by %Y-%m-%d format

Usage

```
removeEarlyDates(dates, firstYear)
```

Arguments

dates	vector of dates
firstYear	integer value of first (earliest) year in the allowed date range.

Details

NA values are ignored and not changed.

Value

A vector of dates after the year indicated by the numeric value of `firstYear`.

Examples

```
dates <- structure(c(
  12361, 14400, 15413, NA, 11189, NA, 13224, 10971,
  -432000, 13262
), class = "Date")
cleanedDates <- removeEarlyDates(dates, firstYear = 1000)
dates
cleanedDates
```

`removePotentialSires` *Remove potential sires from a list of IDs*

Description

Remove potential sires from a list of IDs

Usage

```
removePotentialSires(ids, minAge, ped)
```

Arguments

`ids`	character vector of animal IDs
`minAge`	integer value giving the inclusive minimum current age (in years) a male must have to be listed as a potential sire. Default is 1 year.
`ped`	dataframe that is the `Pedigree`. It contains pedigree information including the IDs listed in `ids`.

Value

character vector of Ids with any potential sire Ids removed.

Examples

```
library(nprcgenekeepr)
qcBreeders <- nprcgenekeepr::qcBreeders
pedWithGenotype <- nprcgenekeepr::pedWithGenotype
noSires <- removePotentialSires(
  ids = qcBreeders, minAge = 2,
  ped = pedWithGenotype
)
sires <- getPotentialSires(qcBreeders, ped = pedWithGenotype, minAge = 2)
pedWithGenotype[pedWithGenotype$id %in% noSires, c("sex", "age")]
pedWithGenotype[pedWithGenotype$id %in% sires, c("sex", "age")]
```

---

removeUninformativeFounders

Remove uninformative founders

Description

Founders (having unknown sire and dam) that appear only one time in a pedigree are uninformative and can be removed from a pedigree without loss of information.

Usage

```
removeUninformativeFounders(ped)
```

Arguments

ped	datatable that is the Pedigree. It contains pedigree information. The fields sire and dam are required.

Value

A reduced pedigree.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
probands <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- getProbandPedigree(probands, breederPed)
nrow(ped)
p <- removeUninformativeFounders(ped)
nrow(p)
```

```
p <- addBackSecondParents(p, ped)
nrow(p)
```

`removeUnknownAnimals` *Remove placeholder animals added for unknown parents*

Description

Remove placeholder animals added for unknown parents

Usage

```
removeUnknownAnimals(ped)
```

Arguments

ped	The pedigree information in data.frame format

Value

Pedigree with unknown animals removed

Examples

```
library(nprcgenekeepr)
ped <- nprcgenekeepr::smallPed
addedPed <- cbind(ped,
  recordStatus = rep("original", nrow(ped)),
  stringsAsFactors = FALSE
)
addedPed[1:3, "recordStatus"] <- "added"
ped2 <- removeUnknownAnimals(addedPed)
nrow(ped)
nrow(ped2)
```

`reportGV` *Generate a genetic value report for a pedigree*

Description

This is the main function for the Genetic Value Analysis.

Usage

```
reportGV(
  ped,
  guIter = 1000L,
  guThresh = 1L,
  pop = NULL,
  byID = TRUE,
  updateProgress = NULL,
  breedingTable = NULL,
  gestationTable = NULL,
  breedingAgeDefault = NULL,
  gestationDefault = NULL,
  kinshipOverrides = NULL
)
```

Arguments

ped	The pedigree information in data.frame format
guIter	Integer indicating the number of iterations for the gene-drop analysis. Default is 1000 iterations
guThresh	Integer indicating the threshold number of animals for defining a unique allele. Default considers an allele "unique" if it is found in only 1 animal.
pop	Character vector with animal IDs to consider as the population of interest. The default is NULL.
byID	Logical variable of length 1 that is passed through to eventually be used by `alleleFreq()`, which calculates the count of each allele in the provided vector. If `byID` is TRUE and ids are provided, the function will only count the unique alleles for an individual (homozygous alleles will be counted as 1).
updateProgress	Function or NULL. If this function is defined, it will be called during each iteration to update a `shiny::Progress` object.
breedingTable	Optional data.frame overriding the bundled per-species minimum breeding ages used by the unknown-parent mean-kinship correction. `NULL` (the default) uses the bundled `speciesGestation` table.
gestationTable	Optional data.frame overriding the bundled per-species gestation windows used by the correction's conception window. `NULL` uses the bundled table.
breedingAgeDefault	Optional numeric fallback minimum breeding age (years) for species absent from the table. `NULL` uses the built-in 2 years.
gestationDefault	Optional integer fallback gestation window (days) for species absent from the table. `NULL` uses the built-in 210 days.
kinshipOverrides	Optional data.frame of outside-information kinship overrides (`id1`, `id2`, `kinship`; the coefficient *f*, not relatedness *r*) applied to the kinship matrix before mean kinship and the unknown-parent correction. `NULL` (the default) leaves the pedigree-derived matrix unchanged. Ids outside the analysis set are warn-dropped (the

run is not aborted). An override REFINES the named kinship cell; it does not suppress the + sexMean / 2 unknown-parent correction, which is kept for every animal missing one parent. See applyKinshipOverrides.

Details

Reported genome uniqueness (gu) is set to 0 for "Undetermined" animals – those with both parents unknown (U-id aware) and no recorded origin – because their apparent uniqueness is an artifact of unknown parentage (decline-to-credit policy). Imports (both parents unknown but with a recorded origin) and all other animals are unaffected, and calcGU itself is unchanged.

Value

An object of class nprcgenekeeprGV: a list with elements report (a dataframe with the genetic value report, with animals ranked in order of descending value; it carries both a gu column and a guSE column – the Monte Carlo sampling standard error of each genome uniqueness estimate, see calcGUSE), kinship (the kinship matrix), gu (a dataframe with a gu column of genome uniqueness values and a guSE column of their standard errors; gu and guSE are reported as 0 for unknown-origin both-unknown "Undetermined" animals, whose apparent uniqueness is an artifact of unknown parentage), fe (founder equivalents), fg (founder genome equivalents), fgSE (the Monte Carlo sampling standard error of fg, computed from the same gene drop; a single colony-level number, NA when a contributing founder is retained in zero gene-drop iterations – see calcFGSE), neGD (gene diversity retained in the founding gene pool, 1 - 1 / (2 * fg); a colony-level scalar beside fg, NA when fg is NA; see calcGeneDiversity), neSexRatio (the demographic sex-ratio effective size, 4 * nMale * nFemale / (nMale + nFemale), over the current living breeders – a different population than fg and neGD; 0 when a breeding sex is absent; see calcNeSexRatio), neVariance (the variance effective size, the mean-adjusted Crow-Kimura form (N * kbar - 1) / (kbar - 1 + Vk / kbar) over the same current living breeders; NA when fewer than two living breeders; see calcNeVariance), maleFounders and femaleFounders (dataframes of the known male and female founder records), nMaleFounders and nFemaleFounders (the counts of those founders), and total (the total number of known founders).

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
```

```
geneticValue <- reportGV(ped,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
trimmedGeneticValue <- reportGV(trimmedPed,
  guIter = 50, # should be >= 1000
  guThresh = 3,
  byID = TRUE,
  updateProgress = NULL
)
rpt <- trimmedGeneticValue[["report"]]
kmat <- trimmedGeneticValue[["kinship"]]
f <- trimmedGeneticValue[["total"]]
mf <- trimmedGeneticValue[["maleFounders"]]
ff <- trimmedGeneticValue[["femaleFounders"]]
nmf <- trimmedGeneticValue[["nMaleFounders"]]
nff <- trimmedGeneticValue[["nFemaleFounders"]]
fe <- trimmedGeneticValue[["fe"]]
fg <- trimmedGeneticValue[["fg"]]
```

`rhesusGenotypes` *Rhesus genotypes (two haplotypes per animal)*

Description

A dataframe with two haplotypes per animal.

Usage

```
data(rhesusGenotypes)
```

Format

An object of class `data.frame` with 31 rows and 3 columns.

Details

There are 31 rows and 3 columns.

Represents 31 animals that are also in the obfuscated `rhesusPedigree` pedigree from *obfuscated_rhesus_mhc_breeder_geno*

id – character column of animal IDs

first_name – a generic name for the first haplotype

second_name – a generic name for the second haplotype

Examples

```
library(nprcgenekeepr)
data("rhesusGenotypes")
```

rhesusPedigree *Obfuscated rhesus pedigree object*

Description

A pedigree object. Represents an obfuscated pedigree from *obfuscated_rhesus_mhc_ped.csv* where the IDs and dates have been modified to de-identify the data.

id – character column of animal IDs

sire – the male parent of the animal indicated by the `id` column. Unknown sires are indicated with `NA`

dam – the female parent of the animal indicated by the `id` column. Unknown dams are indicated with `NA`

sex – factor with levels: "F", "M". Sex specifier for an individual.

gen – generation number (integers beginning with 0 for the founder generation) of the animal indicated by the `id` column.

birth – `Date` vector of birth dates

exit – `Date` vector, all `NA` (no exit dates are recorded in this obfuscated pedigree)

age – numerical vector of age in years

Usage

```
data(rhesusPedigree)
```

Format

An object of class `data.frame` with 375 rows and 8 columns.

Examples

```
library(nprcgenekeepr)
data("rhesusPedigree")
```

runGeneKeepR *Run the GeneKeepR Shiny Application*

Description

Launches the GeneKeepR Shiny application. It uses a module-based architecture with a Home tab and improved UI components.

Usage

```
runGeneKeepR(port = 6013L, launch.browser = TRUE)
```

Arguments

`port`	Integer port number for the Shiny server (default 6013)
`launch.browser`	Logical; whether to launch browser (default TRUE)

Details

The application includes:

- Home tab with navigation buttons
- Input tab with enhanced QC display
- Dynamic error and changed columns tabs
- Enhanced Pedigree Browser with focal animal support
- Genetic Value Analysis with visualizations
- Summary Statistics with popovers
- Breeding Groups with group panels
- Age-Sex Pyramid with enhanced controls

`runModularApp` is a soft-deprecated alias for this function.

Value

Returns the error condition of the Shiny application when it terminates.

See Also

`runModularApp`, a soft-deprecated alias for this function.

Examples

```
## Not run:
library(nprcgenekeepr)
runGeneKeepR()

## End(Not run)
```

`runModularApp` *Run the Modular Version of GeneKeepR (Deprecated)*

Description

`runModularApp()` has been renamed to `runGeneKeepR`, a name that says what the function does. `runModularApp()` is now a soft-deprecated alias that launches the application via `runGeneKeepR`. Existing callers continue to work.

Usage

```
runModularApp(port = 6013L, launch.browser = TRUE)
```

Arguments

port	Integer port number for the Shiny server (default 6013)
launch.browser	Logical; whether to launch browser (default TRUE)

Value

Returns the error condition of the Shiny application when it terminates (from runGeneKeepR).

See Also

runGeneKeepR, the function this now launches.

Examples

```
## Not run:
library(nprcgenekeepr)
runModularApp()

## End(Not run)
```

---

runQcStudbook *Run Quality Control on Studbook with UI-Friendly Results*

Description

Wrapper function that runs qcStudbook and processes results into a format suitable for Shiny UI display. This function performs two passes: first to check for errors, then to get the cleaned data if no errors exist.

Usage

```
runQcStudbook(
  ped,
  minSireAge = NULL,
  minDamAge = NULL,
  minParentAge = lifecycle::deprecated(),
  reportChanges = FALSE
)
```

Arguments

ped	data.frame containing pedigree data with columns including id, sire, dam, sex, and optionally birth, death, departure, etc.
minSireAge	numeric minimum age in years for a male to have sired an offspring. NULL (default) looks up each sire's species floor via getSpeciesMinBreedingAge (2 years when species is unknown); a supplied value overrides that floor.

minDamAge	numeric minimum age in years for a female to have borne an offspring. NULL (default) looks up each dam's species floor via getSpeciesMinBreedingAge (2 years when species is unknown); a supplied value overrides that floor.
minParentAge	**[Deprecated]** Deprecated scalar minimum parent age. Supplying it sets both minSireAge and minDamAge; use those sex-specific parameters instead.
reportChanges	logical whether to report column name changes in the result (default FALSE). When TRUE, warnings about renamed columns are included in the qcResult.

Value

A list with the following components:

- cleaned - The cleaned pedigree data.frame with standardized column names, added generation numbers, etc. NULL if errors were found.
- qcResult - Result from processQcStudbookResult containing errors, warnings, changedCols, hasErrors, and hasChangedCols.
- errorLst - The raw nprcgenekeeprErr list from qcStudbook's first pass (the same object qcResult was derived from), exposed so callers that need the raw fields (e.g. femaleSires, failedDatabaseConnection) do not have to call qcStudbook a second time themselves.

See Also

qcStudbook for the underlying QC function

processQcStudbookResult for result processing

modInputServer for Shiny module integration

Examples

```
data("pedGood", package = "nprcgenekeepr")
result <- runQcStudbook(pedGood, minSireAge = 2.0, minDamAge = 2.0)
if (!result$qcResult$hasErrors) {
  cleanedPed <- result$cleaned
}
```

---

safeExecute *Execute an expression with error handling*

Description

Executes an expression with comprehensive error handling. On error, logs the error and returns a default value instead of stopping execution. This is particularly useful in Shiny reactive contexts where errors should be handled gracefully.

Usage

```
safeExecute(
  expr,
  module = "unknown",
  default = NULL,
  silent = FALSE,
  notify = FALSE
)
```

Arguments

expr	An expression to evaluate.
module	character. Name of the calling module for logging purposes.
default	The value to return if an error occurs. Defaults to NULL.
silent	logical. If TRUE, suppresses the error and warning logging. Defaults to FALSE.
notify	logical. If TRUE and in a Shiny context, shows a notification to the user. Defaults to FALSE.

Value

The result of evaluating `expr`, or `default` if an error occurs.

See Also

`logModuleEvent` for logging

Examples

```
# Returns 4
safeExecute({ 2 + 2 }, module = "test")

# Returns NULL and logs error
safeExecute({ stop("Error!") }, module = "test")

# Returns custom default on error
safeExecute({ stop("Error!") }, module = "test", default = data.frame())
```

---

saveDataframesAsFiles *Write copy of dataframes to either CSV, TXT, or Excel file*

Description

Takes a list of dataframes and creates a file based on the list name of the dataframe and the extension for the file type.

Usage

```
saveDataframesAsFiles(dfList, baseDir, fileType = "csv")
```

Arguments

dfList	list of dataframes to be stored as files.
baseDir	character vector of length on with the directory path.
fileType	character vector of length one with possible values of "txt", "csv", or "excel". Default value is "csv".

Value

Full path name of files saved.

Examples

```
library(nprcgenekeepr)
dfList <- list(
  lacy1989Ped = nprcgenekeepr::lacy1989Ped,
  pedGood = nprcgenekeepr::pedGood
)
## Write each data frame to a CSV file under a temporary directory.
files <- saveDataframesAsFiles(dfList,
  baseDir = tempdir(), fileType = "csv"
)
basename(files)
```

---

savePlotToFile *Save Plot to File*

Description

Helper function to save ggplot2 plots to files with consistent settings and error handling. Supports PNG, PDF, and SVG formats with configurable dimensions and resolution.

Usage

```
savePlotToFile(
  plot,
  file,
  format = NULL,
  width = 8L,
  height = 6L,
  dpi = 150L,
  units = "in",
  bg = "white"
)
```

Arguments

plot	A ggplot2 plot object to save. If NULL, returns FALSE.
file	character. The file path to save the plot to.
format	character. Output format: "png", "pdf", or "svg". Defaults to "png". If not specified, format is inferred from file extension.
width	numeric. Plot width in inches. Defaults to 8.
height	numeric. Plot height in inches. Defaults to 6.
dpi	numeric. Resolution in dots per inch for raster formats. Defaults to 150 for good quality web/screen use. Use 300 for print.
units	character. Units for width and height. Defaults to "in" (inches).
bg	character. Background color. Defaults to "white".

Value

Logical. TRUE if the file was saved successfully, FALSE otherwise.

See Also

ggsave for the underlying save function

Examples

```
## Not run:
library(ggplot2)
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
savePlotToFile(p, "my_plot.png")
savePlotToFile(p, "my_plot.pdf", format = "pdf")
savePlotToFile(p, "high_res.png", dpi = 300)

## End(Not run)
```

---

setAutoIdFormat *Set the auto-generated unknown-ID format*

Description

Sets the sprintf template used to mint and detect placeholder IDs for unknown parents (see addUIds). The format must have a non-empty literal prefix before its first "%" (used for detection) and must consume a single integer (used for generation), e.g. "U%04d" or "AUTO%05d". The setting is stored in options(nprcgenekeepr.autoIdFormat=) and read by getAutoIdFormat.

Usage

```
setAutoIdFormat(format)
```

Arguments

format	A single character string: the auto-ID sprintf format.

Value

The previous format, returned invisibly.

See Also

getAutoIdFormat, addUIds

Examples

```
old <- setAutoIdFormat("AUTO%05d")
getAutoIdFormat()
setAutoIdFormat(old) # restore
```

---

setExit *Set the exit date when no exit column exists*

Description

Part of Pedigree Curation

Usage

```
setExit(ped, timeOrigin = as.Date("1970-01-01"))
```

Arguments

ped	dataframe of pedigree and demographic information potentially containing columns indicating the birth and death dates of an individual. The table may also contain dates of sale (departure). Optional columns are birth, death, and departure.
timeOrigin	date object used by as.Date to set origin.

Value

A dataframe with an updated pedigree with exit dates specified based on date information that was available.

Examples

```
library(lubridate)
library(nprcgenekeepr)
death <- mdy(paste0(
  sample(1:12, 10, replace = TRUE), "-",
  sample(1:28, 10, replace = TRUE), "-",
  sample(seq(0, 15, by = 3), 10, replace = TRUE) + 2000
))
departure <- as.Date(rep(NA, 10), origin = as.Date("1970-01-01"))
departure[c(1, 3, 6)] <- as.Date(death[c(1, 3, 6)],
  origin = as.Date("1970-01-01")
)
death[c(1, 3, 5)] <- NA
death[6] <- death[6] + days(1)
ped <- data.frame(
  id = paste0(100 + 1:10),
  birth = mdy(paste0(
    sample(1:12, 10, replace = TRUE), "-",
    sample(1:28, 10, replace = TRUE), "-",
    sample(seq(0, 20, by = 3), 10, replace = TRUE) + 1980
  )),
  death = death,
  departure = departure,
  stringsAsFactors = FALSE
)
pedWithExit <- setExit(ped)
```

`setLabKeyDefaults` *Configure Rlabkey authentication for the current session*

Description

Sets up the credentials that `getDemographics` (and any other `Rlabkey` call) uses to authenticate against the LabKey EHR server. An API key, when available, is preferred; otherwise the function falls back to a `.netrc`/`_netrc` file; when neither is present it stops with an actionable error rather than letting `Rlabkey` fail later with an opaque message.

Usage

```
setLabKeyDefaults(siteInfo = getSiteInfo())
```

Arguments

`siteInfo`	list of site information as returned by `getSiteInfo`. The elements used are `baseUrl`, `configFile`, `homeDir`, and `sysname`.

Details

The API key is sourced, in order of precedence, from

1. the environment variable `NPRCGENEKEEPR_LABKEY_APIKEY`, then
2. an `apiKey` entry in the nprcgenekeepr configuration file.

When an API key is found, `labkey.setDefaults` is called with that key and `siteInfo$baseUrl`. When no API key is found, the function checks for a netrc file (the `NETRC` environment variable, then the home-directory `.netrc` on non-Windows or `_netrc` on Windows) and, if present, leaves `Rlabkey` to use it. The API key is never read from or written to the package sources; keep it in the environment, the configuration file, or the netrc file only.

Value

Invisibly, a list with elements `method` (one of `"apiKey"` or `"netrc"`) and `baseUrl`. Stops with an error when no credential can be found.

Examples

```
## Not run:
## Requires an apiKey (env var or config) or a .netrc file to succeed.
library(nprcgenekeepr)
result <- tryCatch(
  setLabKeyDefaults(getSiteInfo(expectConfigFile = FALSE)),
  error = function(e) conditionMessage(e)
)

## End(Not run)
```

`setPopulation` *Flag animals as the population of interest*

Description

Part of the pedigree filtering toolset.

Usage

```
setPopulation(ped, ids)
```

Arguments

`ped`	datatable that is the `Pedigree`. It contains pedigree information. The `id` column is required.
`ids`	character vector of IDs to be flagged as part of the population under consideration.

Value

An updated pedigree with the `population` column added or updated by being set to `TRUE` for the animal IDs in `ped$id` and `FALSE` otherwise.

Examples

```
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
breederPed <- setPopulation(ped = breederPed, ids = focalAnimals)
nrow(breederPed[breederPed$population, ])
```

---

set_seed *Set a reproducible RNG seed across R versions*

Description

The change in how `set.seed` works in R 3.6 prompted the creation of this R version agnostic replacement to get unit test code to work on multiple versions of R in a CICD test build.

Usage

```
set_seed(seed = 1L)
```

Arguments

`seed`	argument to `set.seed`

Details

It seems `RNGkind(sample.kind="Rounding")` does not work prior to version 3.6 so I resorted to using version dependent construction of the argument list to set.seed() in do.call().

Value

NULL, invisibly.

Examples

```
set_seed(1)
rnorm(5)
```

shouldShowChangedColsTab *Determine if Changed Columns tab should be displayed*

Description

Checks the changedCols list to determine if the Changed Columns tab should be inserted into the application navigation. The tab is shown when column names were modified during QC processing.

Usage

```
shouldShowChangedColsTab(changedCols)
```

Arguments

changedCols	list containing information about changed column names. Expected fields include: caseChange, spaceRemoved, periodRemoved, underScoreRemoved, egoToId, egoidToId, sireIdToSire, damIdToDam, birthdateToBirth, deathdateToDeath, recordstatusToRecordStatus, fromcenterToFromCenter.

Value

Logical. TRUE if columns were changed and tab should be shown, FALSE otherwise.

See Also

checkChangedColsLst for the original implementation

Examples

```
library(nprcgenekeepr)
## No column changes recorded: the tab stays hidden.
shouldShowChangedColsTab(list())
## A recorded case change: the tab should be shown.
shouldShowChangedColsTab(list(caseChange = c(Id = "id")))
```

shouldShowOripTab *Determine if the ORIP Reporting tab should be displayed*

Description

Determines whether the Oregon (ONPRC)-specific ORIP Reporting tab should be shown in the application navigation. ORIP (Office of Research Infrastructure Programs) grant reporting is specific to ONPRC, so the tab is shown only when an actual site configuration file is present *and* it identifies the colony as ONPRC. The getSiteInfo default fallback (center = "ONPRC" when no configuration file exists) does NOT show the tab.

Usage

```
shouldShowOripTab(center, hasConfigFile)
```

Arguments

center	Character scalar naming the colony center, as returned by `getSiteInfo()$center` (e.g. "ONPRC" or "SNPRC"). `NULL`, missing values, or any value other than "ONPRC" yield FALSE.
hasConfigFile	Logical scalar indicating whether an actual site configuration file is present (e.g. `file.exists(getSiteInfo()$configFile)`). When FALSE the colony center is the default fallback and the tab is not shown.

Value

Logical. TRUE if a real ONPRC configuration is active and the tab should be shown, FALSE otherwise.

See Also

`getSiteInfo` for the site configuration source and `shouldShowChangedColsTab` for the sibling tab-visibility predicate.

Examples

```
library(nprcgenekeepr)
shouldShowOripTab("ONPRC", TRUE)  # TRUE
shouldShowOripTab("SNPRC", TRUE)  # FALSE
shouldShowOripTab("ONPRC", FALSE) # FALSE (default fallback, no config file)
```

---

smallPed *Hypothetical 17-animal pedigree*

Description

A hypothetical pedigree. It has the following structure: structure(list(id = c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q"), sire = c("Q", NA, "A", "A", NA, "D", "D", "A", "A", NA, NA, "C", "A", NA, NA, "M", NA), dam = c(NA, NA, "B", "B", NA, "E", "E", "B", "J", NA, NA, "K", "N", NA, NA, "O", NA), sex = c("M", "F", "M", "M", "F", "F", "F", "M", "F", "F", "F", "F", "M", "F", "F", "F", "M"), gen = c(1, 1, 2, 2, 1, 3, 3, 2, 2, 1, 1, 2, 1, 1, 2, 3, 0), population = c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE)), .Names = c("id", "sire", "dam", "sex", "gen", "population"), row.names = c(NA, -17L), class = "data.frame")

Usage

```
data(smallPed)
```

Format

An object of class `data.frame` with 17 rows and 6 columns.

`smallPedTree` *Pedigree tree built from smallPed*

Description

A pedigree tree made from `smallPed`. Access it using the following commands.

Usage

```
data(smallPedTree)
```

Format

An object of class `list` of length 17.

Examples

```
library(nprcgenekeepr)
data("smallPedTree")
```

`speciesGestation` *Per-species reproductive parameters*

Description

A lookup table mapping a species name to reproductive parameters used across the package. It keys the gestation window in `getPotentialParents` through `getSpeciesGestation` and the minimum breeding ages in the Genetic Value Analysis unknown-parent mean-kinship correction through `getSpeciesMinBreedingAge`. Species names are matched case- and whitespace-insensitively; any species not present falls back to 210 days for gestation and 2 years for the breeding ages. Rhesus gestation is 210 days (the historical conservative bound; typical rhesus gestation is about 165 days, per Vinson & Raboin 2015), and rhesus minimum breeding ages are male = 4, female = 2.5. The table is populated for the common colony NHP species, with gestation values as conservative upper bounds; making the values user-configurable is a separate planned enhancement. Extend or adjust it by editing `data-raw/speciesGestation.R` and re-running that script.

species – character species name (e.g. "RHESUS").

gestation – integer maximum gestation period in days (a conservative upper bound).

minMaleBreedingAge – numeric minimum age in years at which a male of the species can sire offspring.

minFemaleBreedingAge – numeric minimum age in years at which a female of the species can bear offspring.

Usage

```
data(speciesGestation)
```

Format

An object of class `data.frame` with 14 rows and 4 columns.

Examples

```
library(nprcgenekeepr)
data("speciesGestation")
speciesGestation
```

`summarizeKinshipValues`

Summarize imputed kinship values

Description

Makes a data.frame object containing simulated kinship summary statistics using the counts of kinship values list from `countKinshipValues`.

Usage

```
summarizeKinshipValues(countedKValues)
```

Arguments

`countedKValues`	list object from countKinshipValues function that containes the lists `kIds`, `kValues`, and `kCounts`.

Value

a data.frame with one row of summary statistics for each imputed kinship value. The columns are as follows: `id_1`, `id_2`, `min`, `secondQuartile`, `mean`, `median`, `thirdQuartile`, `max`, and `sd`.

The five-number-summary columns are taken from `fivenum`: `secondQuartile` is the lower hinge (`fivenum()[2]`, approximately the first quartile) and `thirdQuartile` is the upper hinge (`fivenum()[4]`, approximately the third quartile).

Examples

```
ped <- nprcgenekeepr::smallPed
simParent_1 <- list(
  id = "A",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_2 <- list(
```

```
  id = "B",
  sires = c("s1_1", "s1_2", "s1_3"),
  dams = c("d1_1", "d1_2", "d1_3", "d1_4")
)
simParent_3 <- list(
  id = "E",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_4 <- list(
  id = "J",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_5 <- list(
  id = "K",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
simParent_6 <- list(
  id = "N",
  sires = c("A", "C", "s1_1"),
  dams = c("d3_1", "B")
)
allSimParents <- list(
  simParent_1, simParent_2, simParent_3,
  simParent_4, simParent_5, simParent_6
)

extractKinship <- function(simKinships, id1, id2, simulation) {
  ids <- dimnames(simKinships[[simulation]])[[1]]
  simKinships[[simulation]][
    seq_along(ids)[ids == id1],
    seq_along(ids)[ids == id2]
  ]
}

extractKValue <- function(kValue, id1, id2, simulation) {
  kValue[
    kValue$id_1 == id1 & kValue$id_2 == id2,
    paste0("sim_", simulation)
  ]
}

n <- 10
simKinships <- createSimKinships(ped, allSimParents, pop = ped$id, n = n)
kValues <- kinshipMatricesToKValues(simKinships)
extractKValue(kValues, id1 = "A", id2 = "F", simulation = 1:n)
counts <- countKinshipValues(kValues)
stats <- summarizeKinshipValues(counts)
```

summary.nprcgenekeeprErr *Summarize a studbook quality-control error list*

Description

Summarize a studbook quality-control error list

Usage

```
## S3 method for class 'nprcgenekeeprErr'
summary(object, ...)

## S3 method for class 'nprcgenekeeprGV'
summary(object, ...)
```

Arguments

object	object of class nprcgenekeeprErr and class list
...	additional arguments for the summary.default statement

Value

Object of class summary.nprcgenekeeprErr

object of class summary.nprcgenekeeprGV

Examples

```
errorList <- qcStudbook(nprcgenekeepr::pedOne,
  minParentAge = 0,
  reportChanges = TRUE,
  reportErrors = TRUE
)
summary(errorList)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2L,
  reportChanges = FALSE,
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
ped <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
probands <- ped$id[ped$population]
ped <- trimPedigree(probands, ped,
  removeUninformative = FALSE,
```

```
  addBackParents = FALSE
)
geneticValue <- reportGV(ped,
  guIter = 50L, # should be >= 1000L
  guThresh = 3L,
  byID = TRUE,
  updateProgress = NULL
)
trimmedGeneticValue <- reportGV(trimmedPed,
  guIter = 50L, # should be >= 1000L
  guThresh = 3L,
  byID = TRUE,
  updateProgress = NULL
)
summary(geneticValue)
summary(trimmedGeneticValue)
```

`toCharacter` *Force dataframe columns to character*

Description

Converts designated columns of a dataframe to character. Defaults to converting columns `id`, `sire`, and `dam`.

Usage

```
toCharacter(df, headers = c("id", "sire", "dam"))
```

Arguments

`df`	a dataframe where the first three columns can be coerced to character.
`headers`	character vector with the columns to be converted to character class. Defaults to `c("id", "sire", "dam")/`

Value

A dataframe with the specified columns converted to class "character" for display with xtables (in shiny)

Examples

```
library(nprcgenekeepr)
pedGood <- nprcgenekeepr::pedGood
names(pedGood) <- c("id", "sire", "dam", "sex", "birth")
class(pedGood[["id"]])
pedGood <- toCharacter(pedGood)
class(pedGood[["id"]])
```

`trimPedigree` *Trim a pedigree to a group's ancestors*

Description

Filters a pedigree down to only the ancestors of the provided group, removing unnecessary individuals from the studbook. This version builds the pedigree back in time starting from a group of probands, then moves back down the tree trimming off uninformative ancestors.

Usage

```
trimPedigree(
  probands,
  ped,
  removeUninformative = FALSE,
  addBackParents = FALSE
)
```

Arguments

`probands`	a character vector with the list of animals whose ancestors should be included in the final pedigree.
`ped`	datatable that is the `Pedigree`. It contains pedigree information. The fields `sire` and `dam` are required.
`removeUninformative`	logical defaults to `FALSE`. If set to `TRUE`, uninformative founders are removed. Founders (having unknown sire and dam) that appear only one time in a pedigree are uninformative and can be removed from a pedigree without loss of information.
`addBackParents`	logical defaults to `FALSE`. If set to `TRUE`, the function adds back single parents to the p dataframe when one parent is known. The function `addBackSecondParents` uses the ped dataframe, which has full complement of parents and the p dataframe, which has all uninformative parents removed to add back single parents to the p dataframe.

Value

A pedigree that has been trimmed, had uninformative founders removed and single parents added back.

Examples

```
library(nprcgenekeepr)
examplePedigree <- nprcgenekeepr::examplePedigree
breederPed <- qcStudbook(examplePedigree,
  minParentAge = 2,
  reportChanges = FALSE,
```

```
  reportErrors = FALSE
)
focalAnimals <- breederPed$id[!(is.na(breederPed$sire) &
  is.na(breederPed$dam)) &
  is.na(breederPed$exit)]
breederPed <- setPopulation(ped = breederPed, ids = focalAnimals)
trimmedPed <- trimPedigree(focalAnimals, breederPed)
trimmedPedInformative <- trimPedigree(focalAnimals, breederPed,
  removeUninformative = TRUE
)
nrow(breederPed)
nrow(trimmedPed)
nrow(trimmedPedInformative)
```

`withinIntegerRange` *Get integer within a range*

Description

Assures that what is returned is an integer within the specified range. Real values are truncated. Non-numerics are forced to minimum without warning.

Usage

```
withinIntegerRange(int = 0L, minimum = 0L, maximum = 0L, na = "min")
```

Arguments

`int`	value to be forced within a range
`minimum`	minimum integer value.
`maximum`	maximum integer value
`na`	if "min" then non-numerics are forced to the minimum in the range If "max" then non-numerics are forced to the maximum in the range. If not either "min" or "max" it is forced to "min".

Value

A vector of integers forced to be within the specified range.

Examples

```
library(nprcgenekeepr)
withinIntegerRange()
withinIntegerRange(, 0, 10)
withinIntegerRange(NA, 0, 10, na = "max")
withinIntegerRange(, 0, 10, na = "max") # no argument is not NA
withinIntegerRange(LETTERS, 0, 10)
withinIntegerRange(2.6, 1, 5)
```

```
withinIntegerRange(2.6, 0, 2)
withinIntegerRange(c(0, 2.6, -1), 0, 2)
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2)
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2, na = "max")
withinIntegerRange(c(0, 2.6, -1, NA), 0, 2, na = "min")
```

Index