Package 'tidygedcom'

September 1, 2026

Type Package

Title Read and Tidy 'GEDCOM' Genealogy Files

Version 0.2.0

Description Reads and parses 'GEDCOM' (Genealogical Data Communication) files, the standard interchange format exported by genealogical software, and converts them into tidy data frames. Individuals, families, life events, and parent-child links are extracted into rectangular structures suitable for pedigree and kinship analysis. Additional helpers summarize file contents, convert place coordinates, and repair malformed records. Wikipedia family tree templates can be parsed into the same tidy structure. For a discussion of these and related data structures see [Hunter et al. (2026) <doi:10.1007/s10519-026-10259-z>].

License GPL-3

URL https://github.com/R-Computing-Lab/tidygedcom/, https://r-computing-lab.github.io/tidygedcom/

BugReports https://github.com/R-Computing-Lab/tidygedcom/issues

Depends R (>= 3.5.0)

Imports BGmisc, dplyr, igraph, purrr, rlang, stringr, tidyr

Suggests discord, EasyMx, ggpedigree, ggplot2, kinship2, knitr, OpenMx, rmarkdown, testthat (>= 3.0.0), tidyverse, withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author S. Mason Garrison [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-4804-6003>), Christian Waugh [aut, dtc] (ORCID: <https://orcid.org/0000-0002-7871-5845>)

Maintainer S. Mason Garrison <garrissm@wfu.edu>

Repository CRAN

Date/Publication 2026-09-01 08:50:08 UTC

Contents

`addPersonToPed` *addPersonToPed*

Description

A function to add a new person to an existing pedigree `data.frame`.

Usage

```
addPersonToPed(
  ped,
  name = NULL,
  sex = NULL,
  momID = NA,
  dadID = NA,
  twinID = NULL,
  personID = NULL,
  zygosity = NULL,
```

```
  notes = NULL,
  url = NULL,
  overwrite = FALSE
)
```

Arguments

ped	A data.frame representing the existing pedigree.
name	Optional. A character string representing the name of the new person. If not provided, the name will be set to NA.
sex	A value representing the sex of the new person.
momID	Optional. The ID of the mother of the new person. If not provided, it will be set to NA.
dadID	Optional. The ID of the father of the new person. If not provided, it will be set to NA.
twinID	Optional. The ID of the twin of the new person. If not provided, it will be set to NA.
personID	Optional. The ID of the new person. If not provided, it will be generated as the maximum existing personID + 1.
zygosity	Optional. A character string indicating the zygosity of the new person. If not provided, it will be set to NA.
notes	Optional. A character string for notes about the new person. If not provided, it will be set to NA.
url	Optional. A URL column for the new person. If not provided, it will be set to NA.
overwrite	Logical. If TRUE, the function will overwrite an existing person with the same personID. If FALSE, it will stop if a person with the same personID already exists.

Value

A data.frame with the new person added to the existing pedigree.

---

applyTagMappings *Apply Tag Mappings to a Line*

Description

Iterates over a list of tag mappings and, if a tag matches the line, updates the record. Stops after the first match.

Usage

```
applyTagMappings(line, record, pattern_rows, tag_mappings)
```

Arguments

line	A character string from the GEDCOM file.
record	A named list representing the individual's record.
pattern_rows	A list with GEDCOM tag counts.
tag_mappings	A list of lists. Each sublist should define: - tag: the GEDCOM tag, - field: the record field to update, - mode: either "replace" or "append", - extractor: (optional) a custom extraction function.

Value

A list with the updated record (record) and a logical flag (matched).

---

convertGedcomCoords *Convert GEDCOM Coordinate Columns to Numeric*

Description

Converts all latitude and longitude columns in a parsed GEDCOM data frame from GEDCOM compass-prefix notation (e.g., ‘"N51.5074"‘, ‘"W0.1278"‘) to signed decimal degrees. By default, all columns whose names end in ‘_lat‘ or ‘_long‘ are converted.

Usage

```
convertGedcomCoords(df, lat_cols = NULL, long_cols = NULL)
```

Arguments

df	A data frame, typically returned by readGedcom().
lat_cols	Character vector of latitude column names to convert. Defaults to all columns ending in ‘"_lat"‘.
long_cols	Character vector of longitude column names to convert. Defaults to all columns ending in ‘"_long"‘.

Value

The data frame with the specified columns replaced by numeric values.

Examples

```
df <- data.frame(
  birth_lat = "N51.5074", birth_long = "W0.1278",
  stringsAsFactors = FALSE
)
convertGedcomCoords(df)
```

countPatternRows *Count GEDCOM Pattern Rows*

Description

Counts the number of lines in a file (passed as a data frame with column "X1") that match various GEDCOM patterns. Returns a list with counts for each pattern.

Usage

```
countPatternRows(file)
```

Arguments

file	A data frame with a column X1 containing GEDCOM lines.

Value

A list with counts of specific GEDCOM tag occurrences.

extractGedcomYear *Extract Year from a GEDCOM Date String*

Description

Extracts a four-digit year from a GEDCOM date string, stripping calendar escapes (e.g., '\@#DGREGORIAN\@') and common qualifiers ('ABT', 'BEF', 'AFT', 'BET'/'AND') before searching for the year. Returns 'NA_integer_' when no year is found. The function can be used to extract years from various GEDCOM date formats, including approximate dates and date ranges.

Usage

```
extractGedcomYear(x, year_len = 4)
```

Arguments

x	Character vector of GEDCOM date strings.
year_len	Integer specifying the length of the year to extract (default is 4).

Value

Integer vector of years.

Examples

```
extractGedcomYear(c("ABT 1 JAN 1900", "BEF 31 DEC 2000", "1850", NA))
```

findIDs *Find Where One or More IDs Occur Across a List of Data Frames*

Description

Searches a named list of data frames for the supplied IDs, looking in every column whose name matches ‘id_regex‘ (parent, spouse, and person-ID columns by default). Returns one row per hit, recording which dataset, which column, and which row the ID was found in, alongside any requested context columns. This is the detective step of link repair: locating every place an ID lives before deciding how to fix it.

Usage

```
findIDs(
  data_list,
  ID,
 id_regex = "(^id$|id$|^pid|pid$|pid_|dadid|momid|patid|matid|spid|spouse|sire|dame)",
  context_cols = c("sex", "byr", "dyr", "name"),
  ignore_case = TRUE,
  include_all_id_cols = FALSE
)
```

Arguments

data_list	A named list of data frames to search.
ID	A vector of one or more IDs to search for.
id_regex	Regular expression matched against column names to decide which columns are ID columns. Ignored when ‘include_all_id_cols = TRUE‘.
context_cols	Character vector of additional (non-ID) columns to carry through for context. Columns absent from a given data frame are ignored.
ignore_case	Logical. Match ‘id_regex‘ case-insensitively. Default ‘TRUE‘.
include_all_id_cols	Logical. If ‘TRUE‘, search every column rather than only those matching ‘id_regex‘. Default ‘FALSE‘.

Details

Unlike [sliceByID()], this does *not* standardize column names, because its whole purpose is to catch IDs wherever they hide, including in non-canonical columns.

Value

A data frame with one row per match, containing ‘matched_id‘, ‘dataset‘, ‘matched_column‘, ‘matched_value‘, the source ‘row_index‘, and any available ‘context_cols‘. Returns an empty data frame if nothing matches.

Examples

```
clean <- readGedcom(
  system.file("extdata", "waugh.ged", package = "tidygedcom"),
  verbose = FALSE
)
messy <- readGedcom(
  system.file("extdata", "waugh_messy.ged", package = "tidygedcom"),
  verbose = FALSE
)

# Locate every reference to W. Henderson Waugh across both parses
findIDs(list(clean = clean, messy = messy), ID = 3)
```

`gedcomLat2Numeric` *Convert GEDCOM Latitude String to Numeric*

Description

Converts GEDCOM-style latitude strings like ‘"N51.5074"‘ or ‘"S33.8688"‘ to signed decimal degrees. Returns ‘NA‘ for ‘NA‘ or unrecognized-prefix input.

Usage

```
gedcomLat2Numeric(x)
```

Arguments

x	Character vector of GEDCOM latitude values.

Value

Numeric vector of decimal degrees (positive = N, negative = S).

Examples

```
gedcomLat2Numeric(c("N51.5074", "S33.8688", NA))
```

gedcomLon2Numeric *Convert GEDCOM Longitude String to Numeric*

Description

Converts GEDCOM-style longitude strings like ‘"E151.2093"‘ or ‘"W0.1278"‘ to signed decimal degrees. Returns ‘NA‘ for ‘NA‘ or unrecognized-prefix input.

Usage

```
gedcomLon2Numeric(x)
```

Arguments

x	Character vector of GEDCOM longitude values.

Value

Numeric vector of decimal degrees (positive = E, negative = W).

Examples

```
gedcomLon2Numeric(c("E151.2093", "W0.1278", NA))
```

initializeRecord *Initialize an Empty Individual Record*

Description

Creates a named list with all GEDCOM initialized to NA_character_.

Usage

```
initializeRecord(all_var_names)
```

Arguments

all_var_names	A character vector of variable names.

Value

A named list representing an empty individual record.

mapFAMS2parents *Create a Mapping from Family IDs to Parent IDs*

Description

This function scans the data frame and creates a mapping of family IDs to the corresponding parent IDs.

Usage

```
mapFAMS2parents(df_temp, mom_sex = "F", dad_sex = "M")
```

Arguments

df_temp	A data frame produced by readGedcom().
mom_sex	Character string indicating the value of sex that corresponds to mothers (default "F").
dad_sex	Character string indicating the value of sex that corresponds to fathers (default "M").

Value

A list mapping family IDs to parent information.

processEventLine *Process Event Lines (Birth or Death)*

Description

Extracts event details (e.g., date, place, cause, latitude, longitude) from a block of GEDCOM lines. Uses level-aware sub-block parsing so fields are looked up by tag name rather than fixed offsets.

Usage

```
processEventLine(event, block, i, record, pattern_rows, event_fields)
```

Arguments

event	A character string indicating the event type ("birth", "chr", "death", or "burial").
block	A character vector of GEDCOM lines.
i	The current line index where the event tag is found.
record	A named list representing the individual's record.
pattern_rows	A list with counts of GEDCOM tag occurrences.
event_fields	A named list of field mappings as returned by make_event_fields().

Value

The updated record with parsed event information.

`readGedcom` *Read a GEDCOM File*

Description

Ingests a GEDCOM genealogy file, identifies individual records, and parses person-level identifiers, names, life events, attributes, and family relationships into a structured data frame. Optional post-processing can infer parental IDs from family relationships, reconcile redundant name fields, and remove uninformative columns from the parsed output.

Usage

```
readGedcom(
  file_path,
  verbose = FALSE,
  post_process = TRUE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE,
  parse_dates = FALSE,
  impute_partial_dates = TRUE,
  clean_names = TRUE,
  update_rate = 1000,
  ...
)

readGed(
  file_path,
  verbose = FALSE,
  post_process = TRUE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE,
  parse_dates = FALSE,
  impute_partial_dates = TRUE,
  clean_names = TRUE,
  update_rate = 1000,
  ...
)

readgedcom(
  file_path,
```

```
  verbose = FALSE,
  post_process = TRUE,
  add_parents = TRUE,
  remove_empty_cols = TRUE,
  combine_cols = TRUE,
  skinny = FALSE,
  parse_dates = FALSE,
  impute_partial_dates = TRUE,
  clean_names = TRUE,
  update_rate = 1000,
  ...
)
```

Arguments

file_path	Character string. Path to the GEDCOM file.
verbose	Logical. If 'TRUE', print progress messages.
post_process	Logical. If 'TRUE', apply post-processing steps controlled by 'add_parents', 'combine_cols', 'remove_empty_cols', 'skinny', and 'parse_dates'.
add_parents	Logical. If 'TRUE', infer 'momID' and 'dadID' from 'FAMC' and 'FAMS' mappings during post-processing.
remove_empty_cols	Logical indicating whether to remove columns that are entirely missing.
combine_cols	Logical. If 'TRUE', combine redundant name columns, such as 'name_given' with 'name_given_pieces' and 'name_surn' with 'name_surn_pieces', when their values do not conflict.
skinny	Logical. If 'TRUE', return a slimmer data frame by dropping 'FAMC', 'FAMS', and columns that are entirely 'NA' during post-processing.
parse_dates	Logical. If 'TRUE', attempt to parse date columns (e.g., 'birth_date', 'death_date') into Date objects, after removing common GEDCOM date qualifiers like "ABT", "BEF", and "AFT".
impute_partial_dates	Logical. Only used when 'parse_dates = TRUE'. If 'TRUE' (default), dates known only to the month or the year are completed with the midpoint of that interval (the 15th of a known month, or 15 June for a known year) so that they survive conversion to 'Date'. Historical records are often this imprecise, and without imputation such dates become 'NA'. Set to 'FALSE' to keep only dates that specify a day. The unmodified strings are always available by reading the file with 'parse_dates = FALSE'.
clean_names	Logical indicating whether to clean name columns by removing trailing slashes and squishing whitespace.
update_rate	Numeric. Intended rate at which progress messages should be printed. Currently unused.
...	Additional arguments. Currently unused.

Details

'readGedcom()' is a line-oriented parser tuned to common GEDCOM 5.5 and 5.5.1 structures. Individual records are identified from blocks that begin with an '@ INDI' line. Each individual block is passed to an internal parser that uses simple GEDCOM tag pattern matches to extract identifiers, names, life events, attributes, and family relationships.

Name information is parsed primarily from the GEDCOM 'NAME' tag, which often encodes given names and surnames using slash-delimited surname notation, such as 'NAME John /Smith/'. The parser extracts the given name, surname, and a cleaned full name. Additional name components are parsed when present, including name prefix, name suffix, nickname, and married surname.

Birth and death events are recognized from 'BIRT' and 'DEAT' tags. Event details are parsed by collecting all child lines whose GEDCOM level equals the event level plus one (direct children), then looking up sub-fields by tag name. 'DATE', 'PLAC', and 'CAUS' are matched as direct children of the event. Coordinates ('LATI' and 'LONG') are searched across all descendant lines, which allows them to be located whether they appear as direct children (common in some GEDCOM 5.5.x exporters), under 'PLAC' (standard GEDCOM 5.5.1), or under a 'MAP' substructure under 'PLAC' (GEDCOM 7.x). Missing sub-fields leave the corresponding output columns as 'NA'.

Attribute tags such as 'OCCU', 'EDUC', 'RELI', 'CAST', 'NCHI', 'NMR', 'NATI', 'RESI', 'PROP', 'SSN', 'TITL', 'DSCR', and 'IDNO' are parsed directly into dedicated columns prefixed with 'attribute_'.

Family relationships are parsed from 'FAMC' and 'FAMS' tags. 'FAMC' identifies the family in which an individual is a child, and 'FAMS' identifies families in which an individual is a spouse. These raw family identifiers are retained in the parsed output unless removed during post-processing. When 'add_parents = TRUE', they are also used to infer 'momID' and 'dadID'.

If 'post_process = TRUE', 'readGedcom()' applies optional cleanup steps controlled by 'add_parents', 'combine_cols', 'remove_empty_cols', and 'skinny'. These steps can infer parent IDs, collapse redundant name fields, remove columns that are entirely missing, and drop raw family relationship columns for a slimmer output.

Value

A data frame containing information about individuals, with the following potential columns:

personID Individual ID parsed from the '@ INDI' line.

momID ID of the individual's mother, if inferred.

dadID ID of the individual's father, if inferred.

sex Sex of the individual.

name Cleaned full name of the individual.

name_given Given name parsed from the 'NAME' tag.

name_given_pieces Given name parsed from a separate 'GIVN' tag, if present.

name_surn Surname parsed from the 'NAME' tag.

name_surn_pieces Surname parsed from a separate 'SURN' tag, if present.

name_marriedsurn Married surname parsed from '_MARNM', if present.

name_nick Nickname parsed from 'NICK', if present.

name_npfx Name prefix parsed from 'NPFX', if present.

name_nsfx Name suffix parsed from 'NSFX', if present.
birth_date Birth date of the individual.
birth_lat Latitude of the birthplace.
birth_long Longitude of the birthplace.
birth_place Birthplace of the individual.
chr_date Christening date of the individual ('CHR' tag).
chr_place Christening place of the individual.
death_caus Cause of death.
death_date Death date of the individual.
death_lat Latitude of the place of death.
death_long Longitude of the place of death.
death_place Place of death of the individual.
burial_date Burial date of the individual ('BURI' tag).
burial_lat Latitude of the burial place.
burial_long Longitude of the burial place.
burial_place Burial place of the individual.
attribute_caste Caste of the individual.
attribute_children Number of children of the individual.
attribute_description Description of the individual.
attribute_education Education of the individual.
attribute_idnumber Identification number of the individual.
attribute_marriages Number of marriages of the individual.
attribute_nationality Nationality of the individual.
attribute_occupation Occupation of the individual.
attribute_property Property owned by the individual.
attribute_religion Religion of the individual.
attribute_residence Residence of the individual.
attribute_ssn Social Security number of the individual.
attribute_title Title of the individual.
FAMC ID or IDs of the family in which the individual is a child.
FAMS ID or IDs of families in which the individual is a spouse.

If no individual records are found, the function returns 'NULL' with a warning.

Examples

```
# A small excerpt of the W. Henderson Waugh family tree
ged_file <- system.file("extdata", "waugh.ged", package = "tidygedcom")
ped <- readGedcom(ged_file, verbose = FALSE)
ped[, c("personID", "name", "sex", "birth_date", "momID", "dadID")]

# Keep every parsed column rather than the default slimmed output
full <- readGedcom(ged_file, verbose = FALSE, remove_empty_cols = FALSE)
ncol(full)
```

`readGedcomFamilies` *Read Family Records from a GEDCOM File*

Description

Parses ‘FAM‘ records from a GEDCOM file and returns a tidy data frame with one row per family unit. Captures husband, wife, children, marriage event, and divorce event details.

Usage

```
readGedcomFamilies(
  file_path,
  verbose = FALSE,
  parse_dates = FALSE,
  remove_empty_cols = TRUE,
  ...
)
```

Arguments

`file_path`	Character string. Path to the GEDCOM file.
`verbose`	Logical. If ‘TRUE‘, print progress messages.
`parse_dates`	Logical. If ‘TRUE‘, attempt to parse ‘marr_date‘ and ‘div_date‘ into ‘Date‘ objects, after stripping common GEDCOM qualifiers.
`remove_empty_cols`	Logical indicating whether to remove columns that are entirely missing.
`...`	Additional arguments. Currently unused.

Value

A data frame with one row per ‘FAM‘ record and the following columns:

famID Family identifier from the ‘@ FAM‘ line.

husbID Person ID of the husband (‘HUSB‘ tag).

wifeID Person ID of the wife (‘WIFE‘ tag).

children Comma-separated person IDs of children (‘CHIL‘ tags).

marr_date Marriage date (‘MARR/DATE‘).

marr_place Marriage place (‘MARR/PLAC‘).

marr_lat Marriage latitude (‘MARR/.../LATI‘).

marr_long Marriage longitude (‘MARR/.../LONG‘).

div_date Divorce date (‘DIV/DATE‘).

div_place Divorce place (‘DIV/PLAC‘).

Returns ‘NULL‘ with a warning if no family records are found.

Examples

```
fam <- readGedcomFamilies(
  system.file("extdata", "waugh.ged", package = "tidygedcom"),
  verbose = FALSE
)
fam
```

`readWikifamilytree` *Read Wiki Family Tree*

Description

Read Wiki Family Tree

Usage

```
readWikifamilytree(text = NULL, verbose = FALSE, file_path = NULL, ...)
```

Arguments

`text`	A character string containing the text of a family tree in wiki format.
`verbose`	A logical value indicating whether to print messages.
`file_path`	The path to the file containing the family tree.
`...`	Additional arguments (not used).

Value

A list containing the summary, members, structure, and relationships of the family tree.

Examples

```
tree_text <- paste(
  "{{familytree/start |summary=A three-generation example.}}",
  "{{familytree | | | | GMa |~|y|~| GPa | | GMa=Gladys|GPa=Sydney}}",
  "{{familytree | | | | | | | |)|-|-|-|.| }}",
  "{{familytree | | | MOM |y| DAD | |AUNT| MOM=Mom|DAD=Dad|AUNT=Aunt Daisy}}",
  "{{familytree | |,|-|-|-|.| | | | | }}",
  "{{familytree | JOE | | ME  | | JOE=Joe|ME=Me}}",
  "{{familytree/end}}",
  sep = "\n"
)

tree <- readWikifamilytree(text = tree_text)
tree$summary
head(tree$members)
```

`repairManually` *Apply a List of Hand-Verified Fixes to a Pedigree*

Description

Applies a set of manual corrections to a pedigree data frame, recording the provenance of each edit in ‘manual_fix‘ and ‘manual_fix_comment‘ columns and refusing to let two fixes silently touch the same row. This is the repair counterpart to BGmisc’s automated [BGmisc::repairIDs()] / [BGmisc::repairSex()]: for the cases a human has to resolve by hand, it keeps the edits reproducible and auditable.

Usage

```
repairManually(ped, fixes)
```

Arguments

`ped`	A pedigree data frame.
`fixes`	A named list of fixes, each structured as described above. The names label each fix and are stored in ‘manual_fix‘.

Details

Each element of ‘fixes‘ is itself a list with three parts:

rows A quoted expression (see [rlang::quo()]) that evaluates within ‘ped‘ to a logical row selector, e.g. ‘rlang::quo(ID == 348700)‘.

changes A named list of ‘column = new_value‘ pairs to assign to the selected rows.

comment A short human-readable note explaining the fix.

Value

‘ped‘ with the fixes applied and the ‘manual_fix‘ / ‘manual_fix_comment‘ provenance columns populated.

Examples

```
# The messy example file omits Laura Watkins's SEX line, so she cannot be
# resolved as a mother during parsing.
ped <- readGedcom(
  system.file("extdata", "waugh_messy.ged", package = "tidygedcom"),
  verbose = FALSE
)
ped$sex[ped$personID == 6]

fixes <- list(
  laura_sex = list(
    rows = rlang::quo(personID == 6),
```

```
    changes = list(sex = "F"),
    comment = "Sex absent from source export; confirmed by 1880 census."
  )
)
ped <- repairManually(ped, fixes)
ped[ped$personID == 6, c("personID", "name", "sex", "manual_fix")]
```

`royal92` *Royal pedigree data from 1992*

Description

This dataset builds on an existing dataset created by Denis Reid of the Royal Families of Europe. That data was originally published in 1992 and is available on the internet. This version has been updated to combine duplicate entries and to include additional information on birth and death dates, as well as titles. This dataset is intended for educational and illustrative use in software demonstrations involving pedigree diagrams, inheritance structures, and kinship modeling. This dataset is not intended to represent any real individuals or families beyond the original source data, and it is provided solely for educational purposes.

Usage

```
data(royal92)
```

Format

A data frame with 3010 observations

Details

The variables are as follows:

- `personID`: Person identification variable
- `momID`: ID of the mother
- `dadID`: ID of the father
- `famID`: ID of the extended family
- `twinID`: ID of the twin, if applicable
- `name`: Name of the person
- `sex`: Biological sex
- `birth_date`: Date of birth
- `death_date`: Date of death
- `title`: Title of the person

`sliceByID` *Slice a Pedigree Down to the Rows Referencing an ID*

Description

Given a pedigree data frame, returns every row in which any of the supplied IDs appears as the individual themselves, a parent, or a spouse. This is a diagnostic aid for inspecting a person's full household before and after manually repairing links in GEDCOM-derived pedigrees. The result is a subset of 'ped' with the same columns..

Usage

```
sliceByID(ped, ID, sort = TRUE)
```

Arguments

`ped`	A pedigree data frame.
`ID`	A vector of one or more IDs to slice on.
`sort`	Logical. If 'TRUE' (default), sort the result by father ID then individual ID.

Details

Incoming column names are normalized with BGmisc's internal column standardizer, so the common variants ('ID'/'personID', 'dadID'/'pid_fath', 'momID'/'pid_moth', 'spID'/'pid_spouse1', ...) are all recognized without the caller naming them. Because BGmisc's canonical schema is single-spouse but GEDCOM-derived pedigrees often record remarriages in extra columns, any additional spouse-like columns (matching 'spID2', 'pid_spouse2', ...) are detected and searched too. The returned rows keep the caller's original column names.

Value

A data frame: the subset of 'ped' rows in which any of 'ID' appears in a linking column, with the original columns and names preserved.

Examples

```
ped <- readGedcom(
  system.file("extdata", "waugh.ged", package = "tidygedcom"),
  verbose = FALSE
)

# Inspect William Pitt Waugh Sr. and everyone linked to him
sliceByID(ped, ID = 1)
```

summarizeGedcom *Summarize a Parsed GEDCOM Data Frame*

Description

Returns key counts and coverage statistics for a data frame produced by `readGedcom()`.

Usage

```
summarizeGedcom(df)
```

Arguments

`df`	A data frame returned by `readGedcom()`.

Value

An object of class `"tidygedcom_summary"` (a named list). Print the result for a human-readable overview.

Examples

```
df <- readGedcom(
  system.file("extdata", "waugh.ged", package = "tidygedcom"),
  verbose = FALSE
)
summarizeGedcom(df)

# Coverage drops on a file with missing records
messy <- readGedcom(
  system.file("extdata", "waugh_messy.ged", package = "tidygedcom"),
  verbose = FALSE
)
summarizeGedcom(messy)
```

summarizeIDs *Summarize Where One or More IDs Occur Across a List of Data Frames*

Description

A count-level companion to [findIDs()]: instead of one row per hit, returns one row per (id, dataset, column) with the number of matches. Useful for a quick census of how many places an ID appears before drilling in with [findIDs()].

Usage

```
summarizeIDs(data_list, ID, ...)

summariseIDs(data_list, ID, ...)
```

Arguments

data_list	A named list of data frames to search.
ID	A vector of one or more IDs to search for.
...	Additional arguments passed to [findIDs()].

Value

A data frame with columns ‘matched_id‘, ‘dataset‘, ‘matched_column‘, and ‘n_matches‘.

Examples

```
clean <- readGedcom(
  system.file("extdata", "waugh.ged", package = "tidygedcom"),
  verbose = FALSE
)
messy <- readGedcom(
  system.file("extdata", "waugh_messy.ged", package = "tidygedcom"),
  verbose = FALSE
)

# Count references per dataset and column
summarizeIDs(list(clean = clean, messy = messy), ID = 3)
```

---

traceTreePaths *Trace paths between individuals in a family tree grid*

Description

Trace paths between individuals in a family tree grid

Usage

```
traceTreePaths(tree_long, deduplicate = TRUE)
```

Arguments

tree_long	A data.frame with columns: Row, Column, Value, id
deduplicate	Logical, if TRUE, will remove duplicate paths

Value

A data.frame with columns: from_id, to_id, direction, path_length, intermediates

Examples

```
# Two individuals joined by a horizontal connector
tree_long <- data.frame(
  Row = rep(1, 3),
  Column = 1:3,
  Value = c("A", "+", "B"),
  id = c("A", NA, "B")
)

traceTreePaths(tree_long)
```

Index