

Package ‘correspondenceTables’

May 12, 2026

Type Package

Title Creating Correspondence Tables Between Two Statistical Classifications

Date 2026-05-12

Version 1.0.1

Description A candidate correspondence table between two classifications can be created when there are correspondence tables leading from the first classification to the second one via intermediate 'pivot' classifications.

The correspondence table between two statistical classifications can be updated when one of the classifications gets updated to a new version.

License EUPL

Encoding UTF-8

Imports data.table, httr, stringr, igraph

Suggests knitr, rmarkdown, tinytest

VignetteBuilder knitr

NeedsCompilation no

URL <https://github.com/eurostat/correspondenceTables>

BugReports <https://github.com/eurostat/correspondenceTables/issues>

Maintainer Mátyás Mészáros <matyas.meszaros@ec.europa.eu>

RoxygenNote 7.3.3

Author Vasilis Chasiotis [aut] (Department of Statistics, Athens University of Economics and Business),

Photis Stavropoulos [aut] (Quantos S.A. Statistics and Information Systems),

Martin Karlberg [aut],

Mátyás Mészáros [cre],

Martina Patone [aut],

Erkand Muraku [aut],

Clement Thomas [aut],

Loic Bienvenu [aut],

Mauro Baldacchini [aut],

Khadija Sossey-Lallemand [aut],
Mark van der Loo [aut]

Repository CRAN

Date/Publication 2026-05-12 21:00:02 UTC

Contents

| | |
|----------------------------------------|----|
| aggregateCorrespondenceTable | 2 |
| analyseCorrespondenceTable | 4 |
| classificationList | 6 |
| classificationQC | 7 |
| correspondenceTableList | 9 |
| dataStructure | 10 |
| newCorrespondenceTable | 12 |
| prefixList | 16 |
| retrieveClassificationTable | 18 |
| retrieveCorrespondenceTable | 20 |
| updateCorrespondenceTable | 22 |

Index **26**

aggregateCorrespondenceTable

Aggregate values from classification A to classification B

Description

Aggregates numeric values from a source classification A into a target classification B using a correspondence table AB. Each value in A is redistributed to one or more B codes according to weights in AB (if available), or split equally when no valid weights exist.

Usage

```
aggregateCorrespondenceTable(AB, A, B = NULL)
```

Arguments

| | |
|----|---------------------------------------------------------------------------------------------------------------------|
| AB | A data.frame containing A→B links, with optional numeric weight. |
| A | A data.frame with one (character) code column and one numeric value column. |
| B | Optional data.frame with a code column defining the output domain. Additional columns (e.g., labels) are preserved. |

Details

This function is used when data expressed in one classification must be converted into another classification using an $A \rightarrow B$ correspondence table.

The workflow is:

1. Detect and standardize column names in AB, A and B (case-insensitive).
2. For each code in A, determine all corresponding B codes.
3. Apply proportional allocation:
 - If AB contains a numeric weight column, weights are normalized per source code.
 - If weights are missing, invalid, or sum to zero, values are split equally across all mapped B codes.
4. Values are multiplied by the resulting weights and aggregated by B code.
5. If B is provided, the result is aligned to its list of codes (unmatched codes receive 0).

This function is appropriate for tasks such as converting statistical datasets from NACE to CPA, CPA to CN, CPC to HS, or any situation where values have to be re-expressed according to a different classification system.

Value

A list with:

- `result`: data.frame with aggregated values for each B code (and any additional columns from B, if provided),
- `mapping`: standardized $A \rightarrow B$ mapping used internally,
- `diagnostics`: list with total input value, mapped value, coverage ratio, and codes in A that could not be mapped.

See Also

[retrieveCorrespondenceTable](#) for downloading AB tables.

Examples

```
## Simple example (equal split)
AB <- data.frame(
  from_code = c("A1", "A1", "A2"),
  to_code   = c("B1", "B2", "B1")
)

print(AB)

A_tbl <- data.frame(
  code = c("A1", "A2"),
  value = c(100, 50)
)

print(A_tbl)
```

```

result <- aggregateCorrespondenceTable(AB, A_tbl)

print(result$result)
print(result$mapping)

```

analyseCorrespondenceTable

Perform analysis on correspondence tables

Description

Perform analysis on a correspondence table using its associated classifications to generate various statistics. It checks the validity of the input data, identifies components, calculates correspondence types, and creates summary tables.

Usage

```

analyseCorrespondenceTable(
  AB,
  Aname = NULL,
  Bname = NULL,
  A = NULL,
  B = NULL,
  longestAcodeOnly = FALSE,
  longestBcodeOnly = FALSE
)

```

Arguments

| | |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AB | A mandatory argument containing a data frame provided by the user with the correspondence table data. The table must contain at least two columns containing the two classifications A code and B code. The name of the two columns is irrelevant. |
| Aname | Optional. A character string specifying the name of the column in 'AB' that should be used as the source classification ('Acode'). If 'NULL', the function assumes that the first column of 'AB' represents the source classification. |
| Bname | Optional. A character string specifying the name of the column in 'AB' that should be used as the target classification ('Bcode'). If 'NULL', the function assumes that the second column of 'AB' represents the target classification. |
| A | A data frame containing the source classification data with an "Acode" column. Can be NULL if no source classification table is provided. |
| B | A data frame containing the target classification data with a "Bcode" column. Can be NULL if no target classification table is provided. |

longestAcodeOnly

Logical. If TRUE, source classification data are filtered based on "Acode" retaining only the maximum length, i.e. the lowest-level Acodes in the correspondence table.

longestBcodeOnly

Logical. If TRUE, target classification data are filtered based on "Bcode" retaining only the maximum length, i.e. the lowest-level Bcodes in the correspondence table.

Value

A list containing two data frames: `Inventory` and `Analysis`. `Inventory` contains statistics related to components, correspondence types, and source/target positions. `Analysis` contains statistics for each class in the correspondence table.

Examples

```
# Use data from the folder extdata/test

AB = read.csv(system.file("extdata/test", "ExempleAnnexe.csv", package = "correspondenceTables"))
head(AB)

result <- analyseCorrespondenceTable(
  AB = AB,
  Aname = NULL,
  Bname = NULL,
  A = NULL,
  B = NULL,
  longestAcodeOnly = FALSE,
  longestBcodeOnly = FALSE
)

print(result$Inventory)
print(result$Analysis)

# the name of the columns does not matter:
AB_mod = AB
colnames(AB_mod) = c("a", "b")

result_mod <- analyseCorrespondenceTable(
  AB = AB_mod,
  Aname = NULL,
  Bname = NULL,
  A = NULL,
  B = NULL,
  longestAcodeOnly = FALSE,
  longestBcodeOnly = FALSE
)

print(result_mod$Inventory)
print(result_mod$Analysis)
```

```

# examples with using Aname and Bname

# fist we add two more column to the same dataset
set.seed(123)
AB_mod2 = AB_mod

AB_mod2 <- cbind(new_first_col = sample(1:100, nrow(AB_mod2), replace = TRUE), AB_mod2)
AB_mod2 <- cbind(
  AB_mod2[1:2],
  new_middle_col = sample(1:100, nrow(AB_mod2), replace = TRUE),
  AB_mod2[3:ncol(AB_mod2)]
)
colnames(AB_mod2)

result_mod2 <- analyseCorrespondenceTable(
  AB = AB_mod2,
  Aname = "a",
  Bname = "b",
  A = NULL,
  B = NULL,
  longestAcodeOnly = FALSE,
  longestBcodeOnly = FALSE
)

print(result_mod2$Inventory)
print(result_mod2$Analysis)

# check if we obtained the same results in all the 3 examples

first = identical(result$Inventory, result_mod$Inventory)
second = identical(result_mod$Inventory, result_mod2$Inventory)
first && second
# TRUE

# the following won't work as we changed column names in AB_mod,
# and added new columns in AB_mod2 but the results are the same.
first = identical(result$Analysis, result_mod$Analysis)
second = identical(result_mod$Analysis, result_mod2$Analysis)
first && second

```

classificationList *List available classification schemes from CELLAR or FAO*

Description

List all classification schemes available on the selected repository (CELLAR, FAO, or both). Returns, for each scheme, its prefix, identifier, URI, English title, and the list of languages detected on its concepts.

Usage

```
classificationList(endpoint = "ALL", showQuery = FALSE)
```

Arguments

`endpoint` Character scalar. One of "CELLAR", "FAO" or "ALL". "ALL" queries both repositories and returns a named list.

`showQuery` Logical; if TRUE, returns also the SPARQL query used.

Value

If `endpoint` is "CELLAR" or "FAO" and `showQuery = FALSE`: a data.frame with columns Prefix, ConceptScheme, URI, Title, Languages.

If `showQuery = TRUE`: a list with "ClassificationList" and "SPARQL.query".

If `endpoint = "ALL"`: a named list with \$CELLAR and \$FAO. If `showQuery = TRUE`, each element is itself a list (ClassificationList + SPARQL.query).

Examples

```
## Not run:
head(classificationList("CELLAR"))
res <- classificationList("FAO", showQuery = TRUE)
names(res)

## End(Not run)
```

| | |
|-------------------------------|----------------------------------------------------|
| <code>classificationQC</code> | <i>Perform quality control on a classification</i> |
|-------------------------------|----------------------------------------------------|

Description

This function performs a series of structural and logical quality-control checks on a hierarchical classification. It verifies the consistency of the hierarchical levels, the presence of parent-child relationships, the uniqueness of labels, the compliance with admissible coding rules, and the sequencing of children within each level. The output is a list of data frames flagging potential issues.

Usage

```
classificationQC(
  classification,
  lengths,
  fullHierarchy = TRUE,
  labelUniqueness = TRUE,
  labelHierarchy = TRUE,
  singleChildCode = NULL,
  sequencing = NULL
)
```

Arguments

| | |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| classification | A <code>data.frame</code> containing (at least) two columns: the codes and labels of the classification to be checked. This argument is mandatory. |
| lengths | A <code>data.frame</code> with one row per hierarchical level giving the initial and final positions of the segment of the code referring to that level. The number of rows implicitly defines the number of hierarchical levels (k). This argument is mandatory. The column names should be <code>charb</code> and <code>chare</code> , if this is not the case, they will be automatically changed and a warning will appear. |
| fullHierarchy | Logical. If <code>FALSE</code> , the function checks that all positions at levels greater than 1 have a parent at the level immediately above (no orphans). If <code>TRUE</code> , it additionally checks that positions at levels strictly lower than k have children in the next level (no childless nodes). |
| labelUniqueness | Logical. When <code>TRUE</code> , the function checks whether labels are unique at each hierarchical level. Duplicates are listed in the <code>QC_duplicatesLabel</code> table. |
| labelHierarchy | Logical. When <code>TRUE</code> , the function checks that: <ul style="list-style-type: none"> • single children share the same label as their parent; • if a parent shares a label with one of its children, it must be a single-child parent. |
| singleChildCode | A <code>data.frame</code> specifying admissible codes for single children at each hierarchical level. If not <code>NULL</code> , the function checks compliance with these codes. |
| sequencing | A <code>data.frame</code> defining admissible codes and their expected order for multiple-child situations at each level. If not <code>NULL</code> , the function checks sequencing consistency. |

Value

A list of data frames flagging potential issues in the classification:

- `QC_output`: full augmented table with computed levels, code segments, and all compliance flags;
- `QC_noLevels`: rows for which hierarchical level could not be determined;
- `QC_orphan`: positions that lack an expected parent;
- `QC_childless`: positions expected to have children but do not;
- `QC_duplicatesLabel`: positions with duplicate labels within the same level;
- `QC_duplicatesCode`: duplicated codes;
- `QC_singleChildMismatch`: label inconsistencies for single-children relationships;
- `QC_singleCodeError`: single children with invalid codes (if `singleChildCode` is provided);
- `QC_multipleCodeError`: non-single children with invalid codes (if `sequencing` is provided);
- `QC_gapBefore`: missing expected codes in sequence;
- `QC_lastSibling`: flag for last sibling in sequencing rules.

Examples

```

classification <- read.csv(system.file("extdata/test", "Nace2_long.csv",
                                     package = "correspondenceTables"))
lengths <- data.frame(charb = c(1,2,3,5), chare = c(1,2,4,5))

Output <- classificationQC(
  classification = classification,
  lengths = lengths,
  fullHierarchy = TRUE,
  labelUniqueness = TRUE,
  labelHierarchy = TRUE,
  singleChildCode = NULL,
  sequencing = NULL
)

print(Output$QC_output)
print(Output$QC_orphan)
print(Output$QC_childless)
print(Output$QC_duplicatesLabel)
print(Output$QC_duplicatesCode)

```

correspondenceTableList

List available correspondence tables from online services

Description

Returns a data frame listing available correspondence tables (mappings between statistical classifications) published by one or more supported online services.

Use this function to discover valid ‘prefix’ and ‘ID_table’ values before calling ‘retrieveCorrespondenceTable()’.

Usage

```
correspondenceTableList(endpoint = "ALL", showQuery = FALSE)
```

Arguments

| | |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endpoint | Character. Which service(s) to query. One or more of the supported endpoints (e.g. "CELLAR", "FAO"), or "ALL" to query all available services. Case-insensitive. If multiple endpoints are supplied (including "ALL"), a **named list** is returned; otherwise a single ‘data.frame’. |
| showQuery | Logical. If ‘TRUE’, the SPARQL queries used are emitted via ‘message()’ (suppress with ‘suppressMessages()’). |

Details

Read-only queries; no side effects (no file writes or caching).

Value

If a single endpoint is requested, a 'data.frame' (one row per correspondence). If multiple endpoints are requested (including "ALL"), a ****named list**** of data frames, one per endpoint.

See Also

retrieveCorrespondenceTable()

Examples

```
## Not run:
# List correspondence tables available in CELLAR
ct <- correspondenceTableList(endpoint = "CELLAR")
head(ct)

# List correspondence tables available in FAO
ct_fao <- correspondenceTableList(endpoint = "FAO")
head(ct_fao)

# Get correspondence tables from both repositories
ct_all <- correspondenceTableList(endpoint = "ALL")
names(ct_all)

## End(Not run)
```

dataStructure

Retrieve a classification structure (levels and concepts) from a supported service

Description

Queries the selected service (e.g., CELLAR or FAO) to retrieve the structure of a classification scheme (levels, depth, and concept rows), with language-aware label fallback. You can return a compact summary table, a detailed concept-level table, or both.

The function first resolves the input classification using the in-package registry returned by `classificationList()`, then retrieves the structure and returns it as data frames. Network errors are handled with retry logic.

Usage

```
dataStructure(
  endpoint,
  prefix,
  conceptScheme = NULL,
  language = "en",
  showQuery = FALSE,
  return = c("summary", "details", "both"),
  timeout_sec = 60,
  retries = 3
)
```

Arguments

| | |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endpoint | Character scalar. Accepted values are "CELLAR" or "FAO". |
| prefix | Character. A classification prefix used in the registry (e.g. "NACE", "CPA", "CN2022"). Used to resolve the scheme. |
| conceptScheme | Character or NULL. Optional alternative identifier for the concept scheme. The resolver tries both prefix and conceptScheme using basic normalization (case-insensitive; dot removal). |
| language | Character scalar. Two-letter language code used to select labels (default: "en"). Fallback chain: requested language -> English ("en") -> no-language literal -> the local resource name. |
| showQuery | Logical. If TRUE, the generated queries are emitted via message() and also included in the returned list. |
| return | One of c("summary", "details", "both"). Controls which table(s) are fetched and returned. |
| timeout_sec | Numeric. HTTP timeout in seconds. Default: 60. |
| retries | Integer. Number of automatic retries on transient HTTP errors. Default: 3. |

Details**Resolution of the scheme**

The function calls `classificationList(endpoint)` and tries to match either `prefix` or `conceptScheme` using normalized keys. On success it extracts the classification namespace and scheme identifier needed to compose requests.

Language fallback

Labels for both the classification levels and concepts follow a multi-step fallback: requested language -> "en" -> no-language literal -> local name.

Value

A data.frame (for "summary" or "details"), or a list(summary, details) when return = "both".

Columns:

- All returned tables include a leading Prefix column (lower-case token).
- summary: Concept_Scheme, Depth, Level, Count.
- details: Concept, Code, Label, Depth, Level, BroaderList, BroaderCodeList.

If showQuery = TRUE, the return value is a list containing:

- resolved: metadata (endpoint, inputs, scheme_id, ns_uri, and optionally title)
- the query text (one or two elements depending on return)
- the retrieved table(s)

Errors and Warnings

- Errors if endpoint is not "CELLAR" or "FAO", or if the scheme cannot be resolved from the registry.
- Warns (and still returns rows) if no explicit levels are declared.
- Propagates HTTP errors from the service after retry policy.

See Also

[classificationList](#)

Examples

```
## Not run:
ds_cn <- dataStructure(
  endpoint      = "CELLAR",
  prefix       = "cn2022",
  conceptScheme = "cn2022",
  language     = "en",
  return       = "summary"
)
head(ds_cn)

## End(Not run)
```

newCorrespondenceTable

Correspondence table creation

Description

Create a candidate correspondence table between two classifications based on their correspondences with intermediate classifications

Usage

```
newCorrespondenceTable(
  Tables,
  Reference = "none",
  MismatchTolerance = 0.2,
  Redundancy_trim = TRUE,
  Progress = TRUE
)
```

Arguments

| | |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tables | A string of type character containing the name of a csv file which contains the names of the files that contain the classifications and the intermediate correspondence tables OR a list of vectors with the names of the dataframes classifications and the intermediate correspondence tables (see "Details" below). |
| Reference | The reference classification among A and B. If a classification is the reference to the other, and hence <i>hierarchically superior</i> to it, each code of the other classification is expected to be mapped to at most one code of the reference classification. The valid values are "none", "A", and "B". If the selected value is "A" or "B", a "Review" flag column (indicating the records violating this expectation) is included in the output (see "Explanation of the flags" below). |
| MismatchTolerance | The maximum acceptable proportion of rows in the candidate correspondence table which contain no code for classification A or no code for classification B. The default value is 0.2. The valid values are real numbers in the interval [0, 1]. |
| Redundancy_trim | An argument in the function containing the logical values TRUE or FALSE used to facilitate the trimming of the redundant records. The default value is TRUE, which removes all redundant records. The other values is FALSE, which shows redundant records together with the redundancy flag. |
| Progress | An argument in the function containing the logical values TRUE (default) or FALSE. Used to switch ON (when TRUE) or OFF (when FALSE) the progress bar that interactively display the creation process of the new table in the console output. |

Details

File and file name requirements:

- The file that corresponds to argument Tables and the files to which the contents of Tables lead, must be in *csv format with comma as delimiter*. If full paths are not provided, then these files must be available in the working directory. No two filenames provided must be identical.
- The file that corresponds to argument Tables must contain filenames, *and nothing else*, in a $(k+2) \times (k+2)$ table, where k , a positive integer, is the number of "pivot" classifications. The cells in the main diagonal of the table provide the filenames of the files which contain, with this order, the classifications A, C_1, \dots, C_k and B . The off-diagonal directly above the main diagonal contains the filenames of the files that contain, with this order, the correspondence

tables $A : C_1$, $\{C_i:C_{i+1}, 1 \leq i \leq k - 1\}$ and $B : C_k$. All other cells of the table must be empty.

Classification table requirements:

- Each of the files that contain classifications must contain at least one column and at least two rows. The first column contains the codes of the respective classification. The first row contains column headers. The header of the first column is the name of the respective classification (e.g., "CN 2021").
- The classification codes contained in a classification file (expected in its first column as mentioned above) must be unique. No two identical codes are allowed in the column.
- If any of the files that contain classifications has additional columns the first one of them is assumed to contain the labels of the respective classification codes.

Correspondence table requirements:

- The files that contain correspondence tables must contain at least two columns and at least two rows. The first column of the file that contains $A:C_1$ contains the codes of classification A. The second column contains the codes of classification C_1 . Similar requirements apply to the files that contain $C_i:C_{i+1}, 1 \leq i \leq k - 1$ and $B:C_k$. The first row of each of the files that contain correspondence tables contains column headers. The names of the first two columns are the names of the respective classifications.
- The pairs of classification codes contained in a correspondence table file (expected in its first two columns as mentioned above) must be unique. No two identical pairs of codes are allowed in the first two columns.

Interdependency requirements:

- At least one code of classification A must appear in both the file of classification A and the file of correspondence table $A:C_1$.
- At least one code of classification B must appear in both the file of classification B and the file of correspondence table $B:C_k$, where $k, k \geq 1$, is the number of pivot classifications.
- If there is only one pivot classification, C_1 , at least one code of it must appear in both the file of correspondence table $A:C_1$ and the file of correspondence table $B:C_1$.
- If the pivot classifications are k with $k \geq 2$ then at least one code of C_1 must appear in both the file of correspondence table $A:C_1$ and the file of correspondence table $C_1:C_2$, at least one code of each of the $C_i, i = 2, \dots, k - 1$ (if $k \geq 3$) must appear in both the file of correspondence table $C_{i-1}:C_i$ and the file of correspondence table $C_i:C_{i+1}$, and at least one code of C_k must appear in both the file of correspondence table $C_{k-1}:C_k$ and the file of correspondence table $B:C_k$.

Mismatch tolerance:

- The ratio that is compared with `MismatchTolerance` has as numerator the number of rows in the candidate correspondence table which contain no code for classification A or no code for classification B and as denominator the total number of rows of this table. If the ratio exceeds `MismatchTolerance` the execution of the function is halted.

If any of the conditions required from the arguments is violated an error message is produced and execution is stopped.

Value

newCorrespondenceTable() returns a list with two elements, both of which are data frames.

- The first element is the candidate correspondence table A:B, including the codes of all "pivot" classifications, augmented with flags "Review" (if applicable), "Redundancy", "Unmatched", "NoMatchFromA", "NoMatchFromB" and with all the additional columns of the classification and intermediate correspondence table files.
- The second element contains the names of classification A, the "pivot" classifications and classification B as read from the top left-hand side cell of the respective input files.

Explanation of the flags

- The "Review" flag is produced only if argument Reference has been set equal to "A" or "B". For each row of the candidate correspondence table, if Reference = "A" the value of "Review" is equal to 1 if the code of B maps to more than one code of A, and 0 otherwise. If Reference = "B" the value of "Review" is equal to 1 if the code of A maps to more than one code of B, and 0 otherwise. The value of the flag is empty if the row does not contain a code of A or a code of B.
- For each row of the candidate correspondence table, the value of "Redundancy" is equal to 1 if the row contains a combination of codes of A and B that also appears in at least one other row of the candidate correspondence table.
- When "Redundancy_Trim" is equal to FALSE the "Redundancy_keep" flag is created to identify with value 1 the records that will be kept if trimming is performed.
- For each row of the candidate correspondence table, the value of "Unmatched" is equal to 1 if the row contains a code of A but no code of B or if it contains a code of B but no code of A. The value of the flag is 0 if the row contains codes for both A and B.
- For each row of the candidate correspondence table, the value of "NoMatchFromA" is equal to 1 if the row contains a code of A that appears in the table of classification A but not in correspondence table A:C₁. The value of the flag is 0 if the row contains a code of A that appears in both the table of classification A and correspondencetable A:C₁. Finally, the value of the flag is empty if the row contains no code of A or if it contains a code of A that appears in correspondence table A:C₁ but not in the table of classification A.
- For each row of the candidate correspondence table, the value of "NoMatchFromB" is equal to 1 if the row contains a code of B that appears in the table of classification B but not in correspondence table B:C_k. The value of the flag is 0 if the row contains a code of B that appears in both the table of classification B and correspondence table B:C_k. Finally, the value of the flag is empty if the row contains no code of B or if it contains a code of B that appears in correspondence table B:C_k but not in the table of classification B.
- The argument "Redundancy_trim" is used to delete all the redundancies which are mapping correctly. The valid logical values for this argument in the candidate correspondence table are TRUE or FALSE. If the selected value is TRUE, all redundant records are removed and kept exactly one record for each unique combination. For this retained record, the codes, the label and the supplementary information of the pivot classifications are replaced with 'multiple'. If the multiple information of the pivot classifications are the same, their value will not be replaced. If the selected value is FALSE, no trimming is executed so redundant records are shown, together with the redundancy flag. If the logical values are missing the implementation of the function will stop.

Sample datasets included in the package

If the user wish to access the csv files with the sample data, they have two options:

- Option 1: Unpack into any folder of their choice the tar.gz file into which the package has arrived. All sample datasets may be found in the "inst/extdata" subfolder of this folder.
- Option 2: Go to the "extdata" subfolder of the folder in which the package has been installed in their PC's R library. All sample datasets may be found there.

Examples

```
{
## Not run:

code_in  <- data.frame(code_in = c("A","B"))
code_mid <- data.frame(code_mid = c("B","C","D"))
in_to_mid <- data.frame(code_in = c("A","B")
                        , code_out = c("C","B"))
code_out <- data.frame(code_out = c("B","C","D","E"))
mid_to_out <- data.frame(code_mid = c("B","C","D")
                        , code_out = c("B","C","D"))

write.csv(code_in,   file = "code_in.csv",   row.names = FALSE)
write.csv(code_mid,  file = "code_mid.csv",  row.names = FALSE)
write.csv(in_to_mid, file = "in_to_mid.csv", row.names = FALSE)
write.csv(mid_to_out, file = "mid_to_out.csv", row.names = FALSE)
write.csv(code_out,  file = "code_out.csv",  row.names = FALSE)

m <- matrix("",3,3)
diag(m) <- c("code_in.csv","code_mid.csv","code_out.csv")
m[1,2] <- "in_to_mid.csv"
m[2,3] <- "mid_to_out.csv"

write.table(m, file="merge_sequence.csv",row.names=FALSE,col.names=FALSE,sep=",")

out <- newCorrespondenceTable("merge_sequence.csv",MismatchTolerance = 0.9)
print(out$newCorrespondenceTable)
print(out$classificationNames)

## End(Not run) }
```

Description

Returns a character vector of 'PREFIX ...' lines used by package queries. The result always includes core vocabularies (SKOS/XKOS/RDFS/DC/RDF/XSD), and—when available—adds classification scheme namespaces discovered via 'classificationList(endpoint)'. You can filter to specific schemes with the 'prefix' argument.

Usage

```
prefixList(endpoint, prefix = NULL)
```

Arguments

| | |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endpoint | Character. Endpoint name (case-insensitive). Typically "CELLAR" or "FAO". The set of supported names is defined by the package's internal endpoint registry (see 'valid_endpoints()'). |
| prefix | Character vector of scheme tokens to include (e.g., 'c("cn2022","nace2")'). If 'NULL' (default), returns all discovered schemes. |

Details

- This function has no side effects and does not perform network I/O by itself. Discovery is delegated to 'classificationList()'.
- If 'classificationList()' is unavailable or fails, the function returns only the core vocabularies and emits a warning.

Value

A character vector of SPARQL PREFIX declarations. Use 'paste(result, collapse = "\n")' to inject into a query string.

Examples

```
## Not run:  
# All known scheme prefixes for the endpoint  
p1 <- prefixList("CELLAR")  
cat(paste(p1, collapse = "\n"))  
  
# Only include CN2022 and NACE2 if they exist  
p2 <- prefixList("CELLAR", prefix = c("cn2022", "nace2"))  
  
## End(Not run)
```

```
retrieveClassificationTable
```

Retrieve a full classification table from CELLAR or FAO

Description

Retrieves the complete structure of a statistical classification published in either CELLAR or FAO repositories based on a Prefix–ConceptScheme pair obtained using `classificationList()`.

The resulting table is suitable for classification browsing, integrity checks, hierarchical analysis, documentation, and downstream correspondence mapping.

Usage

```
retrieveClassificationTable(
  endpoint,
  prefix,
  conceptScheme,
  language = "en",
  level = "ALL",
  showQuery = FALSE,
  knownSchemes = NULL,
  preferMappingOnly = FALSE
)
```

Arguments

| | |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>endpoint</code> | Character. Repository to query. Must be either "CELLAR" or "FAO". |
| <code>prefix</code> | Character. Classification prefix used for matching and URI resolution (e.g. "cn2022", "cpc21", "isic4"). |
| <code>conceptScheme</code> | Character. Local identifier of the scheme (often identical to <code>prefix</code>). The function automatically resolves this to the canonical ConceptScheme URI published in the endpoint. |
| <code>language</code> | Character. Desired language for labels, scope notes and exclusion notes. Default: "en". |
| <code>level</code> | Character. One of: <ul style="list-style-type: none"> "ALL" (default): return all levels in the hierarchy; a specific depth value (e.g. "2") to filter concepts at that depth only. |
| <code>showQuery</code> | Logical. <ul style="list-style-type: none"> FALSE (default): returns only the classification table; TRUE: returns a list containing the SPARQL query, the resolved scheme URI, and the table itself. |
| <code>knownSchemes</code> | Optional. A data.frame supplying authoritative mappings of the form <code>Prefix, ConceptScheme, URI[, Endpoint]</code> . When provided, this overrides automatic discovery. To be obtained using <code>classificationList(endpoint)</code> . |

preferMappingOnly

Logical. If TRUE, the function never attempts SPARQL discovery and uses only information in knownSchemes or classificationList(endpoint). Default: FALSE.

Value

If showQuery = FALSE, a data.frame with one row per concept and columns:

- **Concept:** full concept URI;
- **Code:** skos:notation coerced to string;
- **Label:** preferred or alternative label in the requested language;
- **Depth:** xkos:depth if available;
- **BroaderList:** broader concept URIs (pipe-separated);
- **BroaderCodeList:** broader concept notations (pipe-separated);
- **IncludeNotes:** skos:scopeNote values (double-pipe-separated);
- **ExcludeNotes:** xkos:exclusionNote values (double-pipe-separated).

If showQuery = TRUE, returns a list with:

- SPARQL.query – the executed SPARQL string;
- scheme_uri – the resolved ConceptScheme URI;
- ClassificationTable – the data.frame described above.

Examples

```
## Not run:
# Retrieve full CN2022 classification (English)
cn <- retrieveClassificationTable(
  endpoint      = "CELLAR",
  prefix       = "cn2022",
  conceptScheme = "cn2022",
  language     = "en"
)

# Retrieve CPC v2.1 from FAO and inspect query
out <- retrieveClassificationTable(
  endpoint      = "FAO",
  prefix       = "cpc21",
  conceptScheme = "cpc21",
  showQuery    = TRUE
)
cat(out$SPARQL.query)

## End(Not run)
```

```
retrieveCorrespondenceTable
```

Download a correspondence (mapping) table between two classifications

Description

Downloads a correspondence table, a mapping of codes, from one statistical classification (A) to another (B) from an online service (CELLAR or FAO), and returns it as a plain data frame. The function has no side effects (it does not read or write files).

A correspondence table tells you which codes in classification A relate to which codes in classification B (e.g., NACE2 → CPA21).

Usage

```
retrieveCorrespondenceTable(
  endpoint,
  prefix,
  ID_table,
  language = "en",
  showQuery = FALSE
)
```

Arguments

| | |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endpoint | Character. The online service to query. Case-insensitive. Supported values are those returned by the internal endpoint registry (e.g., "CELLAR", "FAO"). |
| prefix | Character. Catalogue prefix where the correspondence is published (e.g., "nace2", "cpa21", "cn2022"). Use correspondenceTableList() to discover valid values. |
| ID_table | Character. Identifier of the correspondence, typically of the form "A_B" such as "NACE2_CPA21" or "CN2022_NACE2". Discover identifiers via correspondenceTableList() . |
| language | Character. Preferred label language as a BCP-47 code. Defaults to "en" (English). Examples: "fr", "de". |
| showQuery | Logical. If TRUE, returns a list with the SPARQL query and the result data frame; otherwise (default) returns just the data frame. |

Details

When to use this. Use `retrieveCorrespondenceTable()` when you know (or have discovered) the prefix and ID_table of a correspondence and want the mapping rows as a standard data frame.

How it works (short).

1. Validates the selected service (endpoint).
2. Optionally resolves a stable anchor via [correspondenceTableList\(\)](#).
3. Builds a SPARQL query returning source/target codes, labels, notes, and link URI.
4. Issues an HTTP request and parses the response into a data frame.

Value

A data.frame where each row represents one mapping from the source classification (**A**) to the target classification (**B**).

Columns (when available):

- CorrespondenceID: Requested identifier (e.g., "NACE2_CPA21").
- Prefix: Catalogue prefix (constant per table).
- A: Source code (A parsed from ID_table).
- B: Target code (B parsed from ID_table).
- Label_A, Label_B: Human-readable labels in language.
- Include_A, Exclude_A, Include_B, Exclude_B: Notes, if available.
- Comment: Free-text comment on the mapping, if any.
- URL: URI of the mapping association.

Attributes attached for traceability:

- endpoint, endpoint_url, prefix, ID_table
- A, B, language, normalized_codes, anchor_mode, anchor

Notes

- The function focuses on retrieval. Any code normalization is left to downstream steps.

Examples

```
## Not run:
# 1) Discover available correspondences at CELLAR
ct <- correspondenceTableList(endpoint = "CELLAR")
subset(ct, grepl("NACE2", ID) & grepl("CPA21", ID))

# 2) Retrieve one correspondence (English labels)
x <- retrieveCorrespondenceTable(
  endpoint = "CELLAR",
  prefix   = "nace2",
  ID_table = "NACE2_CPA21",
  language = "en"
)
head(x)

# 3) Save to CSV explicitly if needed (no automatic writing)
# utils::write.csv(x, "NACE2_CPA21.csv", row.names = FALSE)

# 4) Inspect the generated SPARQL query
dbg <- retrieveCorrespondenceTable(
  endpoint = "FAO",
  prefix   = "cpa21",
  ID_table = "NACE2_CPA21",
  showQuery = TRUE
)
```

```
message(substr(dbg$SPARQL.query, 1, 400), "...")
## End(Not run)
```

```
updateCorrespondenceTable
      Correspondence table creation
```

Description

Update the correspondence table between two classifications when one of them has been updated.

Usage

```
updateCorrespondenceTable(
  A,
  B,
  AStar,
  AB,
  AAStar,
  Reference = "none",
  MismatchToleranceB = 0.2,
  MismatchToleranceAStar = 0.2,
  Redundancy_trim = TRUE
)
```

Arguments

| | |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A | A data frame containing the original classification A. The first column holds the codes; optional subsequent columns may hold labels and supplementary info. |
| B | A data frame containing the target classification B. Same structure expectations as A. |
| AStar | A data frame containing the updated version A*. Same structure expectations as A. |
| AB | A data frame containing the previous correspondence table A:B (at least two columns: A-codes, B-codes). |
| AAStar | A data frame containing the concordance table A:A* (at least two columns: A-codes, A*-codes). |
| Reference | The reference classification among A and B. Valid values: "none", "A", "B". If "A" or "B", a "Review" flag column is included (see "Explanation of the flags"). |
| MismatchToleranceB | Maximum acceptable proportion of rows in the updated table with no code of B among those that have a code for A and/or A*. Default 0.2, range [0,1]. |

MismatchToleranceAStar

Maximum acceptable proportion of rows in the updated table with no code of A* among those that have a code for A and/or B. Default 0.2, range [0,1].

Redundancy_trim

Logical. If TRUE (default) trims correct redundant records by collapsing A-side fields to "Multiple" when appropriate; if FALSE, keeps redundancies and adds a Redundancy_keep indicator.

Details***Input data frame requirements***

- A, B, AStar: at least 1 column and at least 1 row. The first column contains the codes; the first row is the header.
- AB, AAStar: at least 2 columns and at least 1 row; the first two columns contain code pairs; the first row is the header.
- Codes in the first column of A, B, AStar must be unique (no duplicates).
- Pairs (first two columns) in AB and AAStar must be unique (no duplicate pairs).
- If additional columns are present in classification or correspondence tables, the second column is treated as a label.
- All columns are coerced to character.

Minimum interdependencies

- At least one code of A must appear in both AAStar and AB.
- At least one code of A* must appear in both AStar and AAStar.
- At least one code of B must appear in both B and AB.

Mismatch tolerance

- If the share of rows with NoMatchToAStar == 1 exceeds MismatchToleranceAStar, the function stops with an error.
- If the share of rows with NoMatchToB == 1 exceeds MismatchToleranceB, the function stops with an error.

Value

updateCorrespondenceTable() returns a list with two data frames:

- updateCorrespondenceTable: the updated correspondence A*:B with flags "CodeChange", "Review" (if applicable), "Redundancy", "NoMatchToAStar", "NoMatchToB", "NoMatchFromAStar", "NoMatchFromB", "LabelChange", plus any additional columns coming from A, B, AStar, AB, AAStar.
- classificationNames: the names of the classifications (A, B, A*) read from the first column headers.

Explanation of the flags

- For each row of the updated correspondence table, the value of "CodeChange" is equal to 1 if the code of A (or A*) contained in this row maps -in this or any other row of the table- to a different code of A* (or A), otherwise the "CodeChange" is equal to 0. The value of "CodeChange" is empty if either the code of A, or the code of A*, or both are missing.
- The "Review" flag is produced only if argument Reference has been set equal to "A" or "B". For each row of the updated correspondence table, if Reference = "A" the value of "Review" is equal to 1 if the code of B maps to more than one code of A*, and 0 otherwise. If Reference = "B" the value of "Review" is equal to 1 if the code of A* maps to more than one code of B, and 0 otherwise. The value of the flag is empty if either the code of A*, or the code of B, or both are missing.
- For each row of the updated correspondence table, the value of "Redundancy" is equal to 1 if the row contains a combination of codes of A* and B that also appears in at least one other row of the updated correspondence table. The value of the flag is empty if both the code of A* and the code of B are missing.
- When "Redundancy_Trim" is equal to FALSE the "Redundancy_keep" flag is created to identify with value 1 the records that will be kept if trimming is performed.
- For each row of the updated correspondence table, the value of "NoMatchToAStar" is equal to 1 if there is a code for A, for B, or for both, but no code for A*. The value of the flag is 0 if there are codes for both A and A* (regardless of whether there is a code for B or not). Finally, the value of "NoMatchToAStar" is empty if neither A nor B have a code in this row.
- For each row of the updated correspondence table, the value of "NoMatchToB" is equal to 1 if there is a code for A, for A*, or for both, but no code for B. The value of the flag is 0 if there are codes for both A and B (regardless of whether there is a code for A* or not). Finally, the value of "NoMatchToB" is empty if neither A nor A* have a code in this row.
- For each row of the updated correspondence table, the value of "NoMatchFromAStar" is equal to 1 if the row contains a code of A* that appears in the table of classification A* but not in the concordance table A:A*. The value of the flag is 0 if the row contains a code of A* that appears in both the table of classification A* and the concordance table A:A*. Finally, the value of the flag is empty if the row contains no code of A* or if it contains a code of A* that appears in the concordance table A:A* but not in the table of classification A*.
- For each row of the updated correspondence table, the value of "NoMatchFromB" is equal to 1 if the row contains a code of B that appears in the table of classification B but not in the correspondence table A:B. The value of the flag is 0 if the row contains a code of B that appears in both the table of classification B and the correspondence table A:B. Finally, the value of the flag is empty if the row contains no code of B or if it contains a code of B that appears in the correspondence table A:B but not in the table of classification B.
- For each row of the updated correspondence table, the value of "LabelChange" is equal to 1 if the labels of the codes of A and A* are different, and 0 if they are the same. Finally, the value of "LabelChange" is empty if either of the labels, or both labels, are missing. Lower and upper case are considered the same, and punctuation characters are ignored when comparing code labels.
- The argument "Redundancy_trim" is used to delete all the redundancies which are mapping correctly. If the analysis concludes that the A*code / Bcode mapping is correct for all cases involving redundancies, then an action is needed to remove the redundancies. If the selected

value is TRUE, all redundant records are removed and kept only one record for each unique combination. For this record retained, the Acodes, the Alabel and the Asupp information is replaced with 'multiple'. If the multiple A records are the same, their value will not be replaced. If the selected value is FALSE, no trimming is executed so redundant records are shown, together with the redundancy flag.

Examples

```
## Not run:

# Read CSVs outside, pass data frames in:

A_df      <- utils::read.csv(system.file("extdata/test", "NAICS2017.csv",
package = "correspondenceTables"),
sep = ",", header = TRUE, check.names = FALSE,
colClasses = "character", encoding = "UTF-8")

AStar_df  <- utils::read.csv(system.file("extdata/test", "NAICS2022.csv",
package = "correspondenceTables"),
sep = ",", header = TRUE, check.names = FALSE,
colClasses = "character", encoding = "UTF-8")

B_df      <- utils::read.csv(system.file("extdata/test", "NACE.csv",
package = "correspondenceTables"),
sep = ",", header = TRUE, check.names = FALSE,
colClasses = "character", encoding = "UTF-8")

AB_df     <- utils::read.csv(system.file("extdata/test", "NAICS2017_NACE.csv",
package = "correspondenceTables"),
sep = ",", header = TRUE, check.names = FALSE,
colClasses = "character", encoding = "UTF-8")

AAStar_df <- utils::read.csv(system.file("extdata/test", "NAICS2017_NAICS2022.csv",
package = "correspondenceTables"),
sep = ",", header = TRUE, check.names = FALSE,
colClasses = "character", encoding = "UTF-8")

UPC <- updateCorrespondenceTable(
  A = A_df, B = B_df, AStar = AStar_df, AB = AB_df, AAStar = AAStar_df,
  Reference = "none", MismatchToleranceB = 0.5, MismatchToleranceAStar = 0.3,
  Redundancy_trim = FALSE
)

summary(UPC)
head(UPC$updateCorrespondenceTable)
UPC$classificationNames

## End(Not run)
```

Index

aggregateCorrespondenceTable, [2](#)
analyseCorrespondenceTable, [4](#)

classificationList, [6](#), [12](#)
classificationQC, [7](#)
correspondenceTableList, [9](#), [20](#)

dataStructure, [10](#)

newCorrespondenceTable, [12](#)

prefixList, [16](#)

retrieveClassificationTable, [18](#)
retrieveCorrespondenceTable, [3](#), [20](#)

updateCorrespondenceTable, [22](#)