Package: correspondenceTables (via r-universe)

November 1, 2024

Type Package

Title Creating Correspondence Tables Between Two Statistical Classifications

Date 2023-04-27

Version 0.8.2

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, tidyverse, writexl

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 7.2.3

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

RemoteUrl https://github.com/eurostat/correspondencetables

RemoteRef HEAD

RemoteSha 6ad9bd17b7568918e84429d8aca2fbe4db9a5ad2

Contents

`classEndpoint` *Retrieve a list of classification tables in CELLAR, FAO or both.*

Description

Retrieve a list of classification tables in CELLAR, FAO or both.

Usage

```
classEndpoint(endpoint)
```

Arguments

`endpoint`	A string of type character containing the endpoint where the table is stored. The valid values are `"CELLAR"`, `"FAO"` and `"ALL"` for both endpoints.

Value

`classEndpoint()` returns a table with information needed to retrieve the classification table:

- Prefix name: the SPARQL instruction for a declaration of a namespace prefix
- Conceptscheme: taxonomy of the SKOS object to be retrieved
- URI: the URL from which the SPARQL query was retrieved
- Name: the name of the table retrieved

Examples

```
{
    endpoint = "ALL"
    list_data = classEndpoint(endpoint)
    }
```

`classificationEndpoint`
Retrieve a list of classification tables from CELLAR and FAO repositories or both.

Description

The purpose of this function is to provide a comprehensive summary of the data structure for each classification in CELLAR and FAO endpoint. The summary includes information such as the prefix name, URI, key, concept scheme, and title associated with each classification.

Usage

```
classificationEndpoint(endpoint = "ALL")
```

Arguments

`endpoint`	SPARQL endpoints provide a standardized way to access data sets, making it easier to retrieve specific information or perform complex queries on linked data. This is an optional parameter, which by default is set to `"ALL"`. The valid values are `"CELLAR"`, `"FAO"` and `"ALL"` for both endpoints.

Value

`classificationEndpoint()` returns a table with information needed to retrieve the classification table:

- Prefix name: the SPARQL instruction for a declaration of a namespace prefix
- Conceptscheme: taxonomy of the SKOS object to be retrieved
- URI: the URL from which the SPARQL query was retrieved
- Name: the name of the table retrieved

Examples

```
{
    endpoint = "ALL"
    list_data = classificationEndpoint(endpoint)
    }
```

classificationQC *ClassificationQC performs a quality check control checks on a given statistical classifications*

Description

The purpose of this function perform quality control checks on statistical classifications. It checks the compliance of classifications with structural rules and provides informative error messages for violations. The function requires input files containing code and label information for each classification position. It verifies the formatting requirements, uniqueness of codes, fullness of hierarchy, uniqueness of labels, hierarchical label dependencies, single child code compliance, and sequencing of codes. The function generates a QC output data frame with the classification data, hierarchical level, code segments, and test outcomes.Additionally, it allows exporting the output to a CSV file. Overall, the classificationQC function ensures the integrity and accuracy of statistical classifications.

Usage

```
classificationQC(
  classification,
  lengthsFile,
  fullHierarchy = TRUE,
  labelUniqueness = TRUE,
  labelHierarchy = TRUE,
  singleChildCode = NULL,
  sequencing = NULL,
  XLSXout = FALSE
)
```

Arguments

`classification`	Refers to a classification in csv file or an R dataframe structured with two columns, consisting of codes and labels, respectively. If the classification is provided as a csv file, it should be stored in the working directory (as defined using `getwd`). This is a mandatory argument.
`fullHierarchy`	It is used to test the fullness of hierarchy. If the parameter `fullHierarchy` is set to `FALSE`, the function will check that every position at a lower level than 1 should have parents all the way up to level 1. If it is set to `TRUE`, in addition to the previous, it will be checked that any position at a higher level than k should have children all the way down to level k.
`labelUniqueness`	It is used to test the that positions at the same hierarchical level have unique labels. If set to `TRUE`, the compliance is checked and positions with duplicate labels are marked as 1 in the "duplicateLabel" column, while positions with unique labels are marked as 0.

`labelHierarchy`	It is used to ensure that hierarchical structure of labels is respected. When set to `TRUE`, the function will check that single child have a label identical to the label of its parent and that has if a position has a label identical to the label of one of its children, then that position should only have a single child.
`singleChildCode`	It refers to CSV file with specific formatting to define valid codes for each level. If this parameter is not `NULL` then it checks compliance with coding rules for single children and non-single children, as provided in the CSV file.
`sequencing`	It refers to a CSV file to define the admissible codes for multiple children at each level. If this parameter is not `NULL`, the function checks the sequencing of multiple children codes within each level, as provided in the CSV file.
`XLSXout`	The valid values are `FALSE` or `TRUE`. In both cases the output will be returned as an R list. If output should be saved as a xlsx file, the argument should be set as `TRUE`. By default, no xlsx file is produced.
`lengthsfile`	Refers to a CSV file or a R dataframe (one record per hierarchical level) containing the initial and last position of the segment of the code specific to that level. The number of lines of this CSV file or the R dataframe will also implicitly define the number of hierarchical levels of the classification. This is a mandatory argument.

Value

`classificationQC()` returns a list of dataframes identifying possible the cases violating the formatting requirements. The databases returned depend on the rules checked. The databases produced are:

- QC_output The dataset includes all the original records in the classification. Colum "Level" refers to the hierarchical levels of each position. Each code will be parsed into segment_k (column "Segmentk") and code_k (column "Codek"), corresponding to the code and segment and hierarchical level k respectively. Additional columns are included to flag the corrected behaviour in each position. These are
  - Orphan: if fullHierarchy is set to FALSE, an "orphan" is a position at a hierarchical level (j) greater than 1 that lacks a parent at the hierarchical level (j-1) immediately above it. Orphan positions are marked with a value of 1 in the "QC output" column, indicating their orphan status. Otherwise, they are assigned a value of 0.
  - Childless: if fullHierarchy is set to TRUE, a "childless" position is one at a hierarchical level (j) less than k that lacks a child at the hierarchical level (j+1) immediately below it. Childless positions are marked with a value of 1 indicating their childless status. Otherwise, they are assigned a value of 0.
  - DuplicateLabel: new column in the output that flags positions involved in duplicate label situations (where multiple positions share the same label at the same hierarchical level) by assigning them a value of 1, while positions with unique labels are assigned a value of 0.
  - SingleChildMismatch: column in the output provides information about label hierarchy consistency in a hierarchical classification system. It indicates:c Value 1: Mismatched labels between a parent and its single child. Value 9: Parent-child pairs with matching labels, but the parent has multiple children.

  - SingleCodeError: column serves as a flag indicating whether a position is a single child and whether the corresponding "singleCode" contains the level j segment. A value of 1 signifies a mismatch, while a value of 0 indicates compliance with the coding rules
  - MultipleCodeError: column serves as a flag indicating whether a position is not a single child and whether the corresponding "multipleCodej" contains the level j segment. A value of 1 signifies a mismatch, while a value of 0 indicates compliance with the coding rules
  - GapBefore: takes the value 0 or 1 if there is a missing child in the 123456789 series.
  - LastSibling: takes the value 1 when it is the last child in the series 123456789 otherwise the value 0

- QC_noLevels A subset of the QC_output dataframe including only records for which levels is not defined. In general if this dataframe is not empty, it suggest that either the classification or the length file is not correctily specified.
- QC_orphan A subset of the QC_output dataframe including only records that have no parents at the higher hierarchical level.
- QC_childless A subset of the QC_output dataframe including only records that have no children at the lower hierarchical level.
- QC_duplicatesLabel A subset of the QC_output dataframe including only records that have duplicated label in the same hierarchical level.
- QC_duplicatesCode A subset of the QC_output dataframe including only records that have the same codes.
- QC_singleChildMismatch A subset of the QC_output dataframe including only records that are single child and have different labels from their parents or that are multiple children and have same labels to their parents.
- QC_singleCodeError A subset of the QC_output dataframe including only records that are single children and have been wrongly coded (not following the rule provided in the 'SingleChildMismatch' CSV file).
- QC_multipleCodeError A subset of the QC_output dataframe including only records that are multiple children and have been wrongly coded (not following the rule provided in the 'SingleChildMismatch' CSV file).
- QC_gapBefore A subset of the QC_output dataframe including only records that are multiple children and have gap before in the sequencing provided in the 'sequencing' CSV file.
- QC_lastSibling A subset of the QC_output dataframe including only records that are multiple and last children following the sequencing provided in the 'sequencing' CSV file.

Examples

```
{
 prefix = "nace2"
 conceptScheme = "nace2"
 endpoint = "CELLAR"
 lengthsTable = lengthsFile(endpoint, prefix, conceptScheme, correction = TRUE)
 classification = retrieveClassificationTable(prefix, endpoint, conceptScheme, level="ALL")$ClassificationTable
 classification = classification[,c(1,2)]
 classification = correctionClassification(classification)
 Output = classificationQC(classification, lengthsFile, fullHierarchy = TRUE, labelUniqueness  = TRUE, labelHierar
View(Output$QC_output)
View(Output$QC_noLevels)
View(Output$QC_orphan)
View(Output$QC_childless)
View(Output$QC_duplicatesLabel)
View(Output$QC_duplicatesCode)
View(Output$QC_singleChildMismatch)
View(Output$QC_singleCodeError)
View(Output$QC_multipleCodeError)
View(Output$QC_gapBefore)
View(Output$QC_lastSibling)
}
```

correctionClassification

Retrieve classification table from CELLAR and FAO repositories.

Description

The aim of this function is to provide a table showing the different codes and labels for each classification

Usage

```
correctionClassification(classification)
```

Arguments

`classification`	it returns a dataframe with two columns corrected according to the classification of CELLAR & FAO.

Value

`correctionClassification()` returns a table with information needed to retrieve the classification table:

- Classification Code name (e.g. nace2): the code of each object
- Classification Label: corresponding name of each object

Examples

```
{
prefix = "nace2"
conceptScheme = "nace2"
endpoint = "CELLAR"
classification = retrieveClassificationTable(prefix, endpoint, conceptScheme, level="ALL")$ClassificationTable
correct_classification = correctionClassification(classification)
View(correct_classification)
}
```

`correspondenceList`	*provides an overview of all the available correspondence classification from CELLAR and FAO repository.*

Description

provides an overview of all the available correspondence classification from CELLAR and FAO repository.

Usage

```
correspondenceList(endpoint)
```

Arguments

`endpoint`	The SPARQL Endpoint. The valid values are `"CELLAR"`, `"FAO"` or `"ALL"` for both.

Value

`correspondenceList()` returns a list of the correspondence tables available with prefix name, ID, Source classification, Target classification, Table name and URI.

Examples

```
{
    corr_list = correspondenceList("ALL")
}
```

`dataStructure`	*Retrieve information about the structure of each classification tables from CELLAR and FAO repositories.*

Description

Retrieve information, for all the classification available in the repositories (CELLAR and FAO), about the level names their hierarchy and the numbers of records the function "structureData()" can be used.

Usage

```
dataStructure(prefix, conceptScheme, endpoint, language = "en")
```

Arguments

`prefix`	Prefixes are typically defined at the beginning of a SPARQL query and are used throughout the query to make it more concise and easier to read. Multiple prefixes can be defined in a single query to cover different namespaces used in the data set. The function 'classificationEndpoint()' can be used to generate the prefixes for the selected classification table.
`conceptScheme`	Refers to a unique identifier associated to specific classification table. The conceptScheme can be obtained by utilizing the "classificationEndpoint()" function.
`endpoint`	SPARQL endpoints provide a standardized way to access data sets, making it easier to retrieve specific information or perform complex queries on linked data. The valid values are `"CELLAR"` or `"FAO"`.
`language`	Refers to the specific language used for providing label, include and exclude information in the selected classification table. By default is set to "en". This is an optional argument.

Value

`structureData()` returns the structure of a classification table from CELLAR and FAO in form a table with the following colums:

- Concept_Scheme: taxonomy of the SKOS object to be retrieved
- Level: the levels of the objects in the collection
- Depth: identify the hierarchy of each level
- Count: the number of objects retrieved in each level

Examples

```
{
   ## Obtain a list including the structure of each classification available
   ## CELLAR
   data_CELLAR = list()
   endpoint = "CELLAR"
   list_data = classificationEndpoint("ALL")

   for (i in 1:nrow(list_data$CELLAR)){
       prefix = list_data$CELLAR[i,1]
       conceptScheme = list_data$CELLAR[i,2]
      data_CELLAR[[i]] = dataStructure(prefix, conceptScheme, endpoint)
   }
   names(data_CELLAR) = list_data$CELLAR[,1]
   ## FAO
   data_FAO = list()
   endpoint = "FAO"
   for (i in 1:nrow(list_data$FAO)){
       prefix = list_data$FAO[i,1]
       conceptScheme = list_data$FAO[i,2]
       data_FAO[[i]] = dataStructure(prefix, conceptScheme, endpoint)
   }
   names(data_FAO) =  list_data$FAO[,1]
    }
```

`lengthsFile` *Retrieve correspondance tables lenghts for each level tables between classification from CELLAR and FAO repositories*

Description

The aim of this function is to provide a table showing the different levels of hierarchy for each classification and the length of each level.

Usage

```
lengthsFile(endpoint, prefix, conceptScheme, correction = TRUE)
```

Arguments

`endpoint`	SPARQL endpoints provide a standardized way to access data sets, making it easier to retrieve specific information or perform complex queries on linked data. The valid values are `"CELLAR"` or `"FAO"`.
`prefix`	Prefixes are typically defined at the beginning of a SPARQL query and are used throughout the query to make it more concise and easier to read. Multiple prefixes can be defined in a single query to cover different namespaces used in the dataset. The function 'classEndpoint()' can be used to generate the prefixes for the selected correspondence table.
`conceptScheme`	Refers to a unique identifier associated to specific classification table. The conceptScheme can be obtained by utilizing the "classEndpoint()" function.
`correction`	The valid values are `FALSE` or `TRUE`. In both cases the lengths table as an R object. If the output wants to have a correction for hierarchy levels `TRUE`. By default is set to "TRUE".

Value

`lenghtsFile()` returns a table containing the lengths for each hierarchical level of the classification.

- charb: contains the length for each code for each hierarchical level
- chare: contains the concatenated length of char b for each code for each hierarchical level

Examples

```
{
endpoint = "CELLAR"
prefix = "nace2"
conceptScheme = "nace2"

lengthsTable = lengthsFile(endpoint, prefix, conceptScheme, correction = TRUE)

#View lengthsTable
View(lengthsTable)

}
```

---

newCorrespondenceTable *Ex novo creation of candidate correspondence tables between two classifications via pivot tables*

Description

Creation of a candidate correspondence table between two classifications, A and B, when there are correspondence tables leading from the first classification to the second one via k intermediate pivot classifications $C_1, \ldots, C_k$. The correspondence tables leading from A to B are A:C_1, $\{C_i$:C_{i+1}: $1 \leq i \leq k-1\}$, B:C_k.

Usage

```
newCorrespondenceTable(
  Tables,
  CSVout = NULL,
  Reference = "none",
  MismatchTolerance = 0.2,
  Redundancy_trim = 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 (see "Details" below).
`CSVout`	The preferred name for the *output csv files* that will contain the candidate correspondence table and information about the classifications involved. The valid values are `NULL` or strings of type `character`. If the selected value is `NULL`, the default, no output file is produced. If the value is a string, then the output is exported into two csv files whose names contain the provided name (see "Value" below).
`Reference`	The reference classification among A and B. If a classification is the reference to the other, and hence *hierarchically superior* 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.

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 $(k+2)$ × $(k+2)$ 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, ..., 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.
- If any of the two files where the output will be stored is read protected (for instance because it is open elsewhere) an error message will be reported and execution will be halted.

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, $i = 2, \ldots, k-1$ (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.
- If the value of argument `CSVout` a string of type `character`, the elements of the list are exported into files of csv format. The name of the file for the first element is the value of argument `CSVout` and the name of the file for the second element is classificationNames_`CSVout`. For example, if `CSVout` = "newCorrespondenceTable.csv", the elements of the list are exported into "newCorrespondenceTable.csv" and "classificationNames_newCorrespondenceTable.csv" respectively.

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:C_1. 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:C_1. 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:C_1 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 infomration 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

Running `browseVignettes("correspondenceTables")` in the console opens an html page in the user's default browser. Selecting HTML from the menu, users can read information about the use of the sample datasets that are included in the package. If they wish to access the csv files with the sample data, users 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

```
{
  ## Application of function newCorrespondenceTable() with "example.csv" being the file
  ## that includes the names the files  and the intermediate tables in a sparse square
  ## matrix containing the 100 rows of the classifications (from ISIC v4 to CPA v2.1 through
  ## CPC v2.1). The desired name for the csv file that will contain the candidate
  ## correspondence table is "newCorrespondenceTable.csv", the reference classification is
  ## ISIC v4 ("A") and the maximum acceptable proportion of unmatched codes between
  ## ISIC v4 and CPC v2.1 is 0.56 (this is the minimum mismatch tolerance for the first 100 row
  ## as 55.5% of the code of ISIC v4 is unmatched).

    tmp_dir<-tempdir()
   A <- read.csv(system.file("extdata", "example.csv", package = "correspondenceTables"),
                header = FALSE,
                sep = ",")
    for (i in 1:nrow(A)) {
      for (j in 1:ncol(A)) {
        if (A[i,j]!="") {
          A[i, j] <- system.file("extdata", A[i, j], package = "correspondenceTables")
      }}}
    write.table(x = A,
              file = file.path(tmp_dir,"example.csv"),
              row.names = FALSE,
              col.names = FALSE,
              sep = ",")

    NCT<-newCorrespondenceTable(file.path(tmp_dir,"example.csv"),
                              file.path(tmp_dir,"newCorrespondenceTable.csv"),
                              "A",
                              0.56,
                              FALSE)

    summary(NCT)
    head(NCT$newCorrespondenceTable)
    NCT$classificationNames
    csv_files<-list.files(tmp_dir, pattern = ".csv")
    unlink(csv_files)
   }
```

`prefixList` *Create a list of prefixes for both CELLAR and FAO*

Description

Create a list of prefixes to be used when defying the SPARQL query to retrieve the tables

Usage

```
prefixList(endpoint)
```

Arguments

`endpoint`	A string of type character containing the endpoint where the table is stored. The valid values are `"CELLAR"` and `"FAO"`.

Value

`prefixList()` returns a list of prefixes to be used when defying the SPARQL query.

Examples

```
{
    endpoint = "CELLAR"
    prefix_list = prefixList(endpoint)
    }
```

`retrieveClassificationTable`
Retrieve a classification tables from CELLAR and FAO

Description

Retrieve a classification tables from CELLAR and FAO

Usage

```
retrieveClassificationTable(
  prefix,
  endpoint,
  conceptScheme,
  level = "ALL",
  language = "en",
  CSVout = FALSE
)
```

Arguments

`prefix`	The SPARQL instruction for a declaration of a namespace prefix. It can be found using the classEndpoint() function.
`endpoint`	The SPARQL Endpoint, the valid values are `"CELLAR"` or `"FAO"`.
`conceptScheme`	Taxonomy of the SKOS object to be retrieved. It can be found using the classEndpoint() function.
`level`	The levels of the objects in the collection to be retrieved, it can be found using the structureData() function. By default is set to `"ALL"`. This is an optional argument.
`language`	Language of the table. By default is set to `"en"`. This is an optional argument.
`CSVout`	The valid values are `FALSE` or `TRUE`. In both cases the correspondence table as an R object. If output should be saved as a csv file, the argument should be set as `TRUE`. By default, no csv file is produced.

Value

`retrieveClassificationTable()` returns a classification tables from CELLAR and FAO. The table includes the following variables:

- Classification name (e.g. nace2): the code of each object

- NAME: the corresponding name of each object
- Include: details on each object
- Include_Also: details on each object
- Exclude: details on each object
- URL: the URL from which the SPARQL query was retrieved

Examples

```
{
    prefix = "nace2"
    endpoint = "CELLAR"
    conceptScheme = "nace2"
    dt = retrieveClassificationTable(prefix, endpoint, conceptScheme)
    # By default retrieved all levels and only English
    head(dt)
    }
```

---

retrieveCorrespondenceTable *Retrieve a correspondence tables from CELLAR and FAO.*

Description

Retrieve a correspondence tables from CELLAR and FAO.

Usage

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

Arguments

prefix	The SPARQL instruction for a declaration of a namespace prefix. It can be found using the classEndpoint() function.
endpoint	The SPARQL Endpoint, the valid values are "CELLAR" or "FAO".
ID_table	The ID of the correspondence table which can be found using the correspondenceList() function.
language	Language of the table. By default is set to "en". This is an optional argument.
CSVout	The valid values are FALSE or TRUE. In both cases the correspondence table as an R object. If output should be saved as a csv file, the argument should be set as TRUE. By default, no csv file is produced.

Value

`retrieveCorrespondenceTable()` returns a classification tables from CELLAR and FAO. The table includes the following variables:

- Source Classification name (e.g. cn2019): the code of each object in the source classification
- Source Classification label: the corresponding label of each object
- Target Classification name (e.g. cn2021): the code of each object in the target classification
- Target Classification label: the corresponding label of each object
- Comment: details on each object, if available
- URL: the URL from which the SPARQL query was retrieved

Examples

```
{
    endpoint = "CELLAR"
    prefix = "nace2"
    ID_table = "NACE2_PRODCOM2021"
    language = "fr"
    dt = retrieveCorrespondenceTable(prefix, endpoint, ID_table, language)
    head(dt)
    }
```

---

structureData *Obtain the structure of the classification tables from CELLAR and FAO.*

Description

Obtain the structure of the classification tables from CELLAR and FAO.

Usage

```
structureData(prefix, conceptScheme, endpoint, language = "en")
```

Arguments

`prefix`	The SPARQL instruction for a declaration of a namespace prefix. It can be found using the classEndpoint() function.
`conceptScheme`	Taxonomy of the SKOS object to be retrieved. It can be found using the classEndpoint() function.
`endpoint`	The SPARQL Endpoint
`language`	Language of the table. By default is set to `"en"`. This is an optional argument.

Value

`structureData()` returns the structure of a classification table from CELLAR and FAO in form a table with the following colums:

- Concept_Scheme: taxonomy of the SKOS object to be retrieved
- Level: the levels of the objects in the collection
- Depth: identify the hierarchy of each level
- Count: the number of objects retrieved in each level

Examples

```
{
    endpoint = "CELLAR"
    prefix = "nace2"
    conceptScheme = "nace2"
    language = "en"
    structure_dt = structureData(prefix, conceptScheme, endpoint, language)
    }
```

`updateCorrespondenceTable`
*Update the correspondence table between statistical classifications A and B when A has been updated to version A**

Description

Update the correspondence table between statistical classifications A and B when A has been updated to version A*.

Usage

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

Arguments

`A`	A string of the type `character` containing the name of a csv file that contains the original classification A.
`B`	A string of the type `character` containing the name of a csv file that contains classification B.
`AStar`	A string of the type `character` containing the name of a csv file that contains the updated version A*.
`AB`	A string of the type `character` containing the name of a csv file that contains the previous correspondence table A:B.
`AAStar`	A string of the type character containing the name of a csv file that contains the *concordance table* A:A*, which contains the mapping between the codes of the two versions of the classification.
`CSVout`	The preferred name for the *output csv files* that will contain the updated correspondence table and information about the classifications involved. The valid values are `NULL` or strings of type `character`. If the selected value is `NULL`, the default, no output file is produced. If the value is a string, then the output is exported into two csv files whose names contain the provided name (see "Value" below).
`Reference`	The reference classification among A and B. If a classification is the reference to the other, and hence *hierarchically superior* 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 is included in the output (see "Explanation of the flags" below).
`MismatchToleranceB`	The maximum acceptable proportion of rows in the updated correspondence table which contain no code of the target classification B, among those which contain a code of A, of A*, or of both. The default value is `0.2`. The valid values are real numbers in the interval [0, 1].
`MismatchToleranceAStar`	The maximum acceptable proportion of rows in the updated correspondence table which contain no code of the updated classification A*, among those which contain a code of A, of B, or of both. The default value is `0.2`. The valid values are real numbers in the interval [0, 1].
`Redundancy_trim`	An argument used to facilitate the trimming of the redundant records. The valid logical values are `TRUE` or `FALSE`. The default value is `TRUE`, which removes all redundant records, replacing the values of Acode Alabel and Asupp with the value ‘Multiple’ (to indicate that multiple A records are involved). If the multiple A records are the same, their value will not be replaced. The other values is `FALSE`, which shows redundant records together with the redundancy flag.

Details

File and file name requirements:

- The files that correspond to arguments `A`, `B`, `AStar`, `AB`, `AAStar` 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.
- If any of the two files where the output will be stored is read protected (for instance because it is open elsewhere) an error message will be reported and execution will be halted.

Classification table requirements:

- The files that correspond to arguments `A`, `B` and `AStar` 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 name 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 correspond to arguments `A`, `B` and `AStar` has additional columns the first one of them is considered as containing the labels of the respective classification codes.

Correspondence and concordance table requirements:

- The files that correspond to arguments `AB` and `AAStar` must contain at least two columns and at least two rows. The first column of the file that corresponds to `AB` contains the codes of classification A. The second column contains the codes of classification B. Similar requirements apply to the file that corresponds to `AAStar`. The first row of each of these files contains column headers. The names of the first two columns are the names of the respective classifications.
- The pairs of classification codes contained in the concordance and the correspondence table files (expected in their 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 concordance table A:A* and the file of correspondence table A:B.
- At least one code of classification A* must appear in both the file of classification A* and the file of concordance table A:A*.
- At least one code of classification B must appear in both the file of classification B and the file of correspondence table A:B.

Mismatch tolerance:

- The ratio that is compared with `MismatchToleranceB` has as numerator the number of rows of the updated correspondence table which contain a code for A, for A*, or for both, but no code for B and as denominator the number of rows which contain a code for A, for A*, or for both (regardless of whether there is a code for B or not). If the ratio exceeds `MismatchToleranceB` the execution of the function is halted.
- The ratio that is compared with `MismatchToleranceAStar` has as numerator the number of rows of the updated correspondence table which contain a code for A, for B, or for both, but no code for A* and as denominator the number of rows which contain a code for A, for B*, or for both (regardless of whether there is a code for A* or not). If the ratio exceeds `MismatchToleranceAStar` the execution of the function is halted.

Value

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

- The first element is the updated correspondence table A*:B augmented with flags "CodeChange", "Review" (if applicable), "Redundancy", "NoMatchToAStar", "NoMatchToB", "NoMatchFromAStar", "NoMatchFromB", "LabelChange", and with all the additional columns of the A, B, AStar, AB and AAStar files.
- The second element contains the names of the original classification A, the target classification B, and the updated version A*, as read from the top left-hand side cell of the respective input files.
- If the value of argument CSVout is a string of type character, the elements of the list are exported into files of csv format. The name of the file for the first element is the value of argument CSVout and the name of the file for the second element is classificationNames_CSVout. For example, if CSVout = "updateCorrespondenceTable.csv", the elements of the list are exported into "updateCorrespondenceTable.csv" and "classificationNames_updateCorrespondenceTable.csv", respectively.

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.

Sample datasets included in the package

Running `browseVignettes("correspondenceTables")` in the console opens an html page in the user's default browser. Selecting HTML from the menu, users can read information about the use of the sample datasets that are included in the package. If they wish to access the csv files with the sample data, users 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

```
{
## Application of function updateCorrespondenceTable() with NAICS 2017 being the
## original classification A, NACE being the target classification B, NAICS 2022
## being the updated version A*, NAICS 2017:NACE being the previous correspondence
## table A:B, and NAICS 2017:NAICS 2022 being the A:A* concordance table. The desired
## name for the csv file that will contain the updated correspondence table is
## "updateCorrespondenceTable.csv", there is no reference classification, and the
## maximum acceptable proportions of unmatched codes between the original
## classification A and the target classification B, and between the original
## classification A and the updated classification A* are 0.5 and 0.3, respectively.

tmp_dir<-tempdir()
A <- system.file("extdata", "NAICS2017.csv", package = "correspondenceTables")
AStar <- system.file("extdata", "NAICS2022.csv", package = "correspondenceTables")
B <- system.file("extdata", "NACE.csv", package = "correspondenceTables")
AB <- system.file("extdata", "NAICS2017_NACE.csv", package = "correspondenceTables")
AAStar <- system.file("extdata", "NAICS2017_NAICS2022.csv", package = "correspondenceTables")

UPC <- updateCorrespondenceTable(A,
                                 B,
                                 AStar,
                                 AB,
                                 AAStar,
                                 file.path(tmp_dir,"updateCorrespondenceTable.csv"),
                                 "none",
                                 0.5,
                                 0.3,
                                 FALSE)

summary(UPC)
head(UPC$updateCorrespondenceTable)
UPC$classificationNames
csv_files<-list.files(tmp_dir, pattern = ".csv")
if (length(csv_files)>0) unlink(csv_files)
   }
```

Index