Package 'faoswsAupus'

March 13, 2015

Type Package

Title Package to perform the AUPUS calculation.

Version 1.0

Date 2015-02-10

Author Joshua M. Browning <joshua.browning@fao.org>, Michael C. J. Kao <michael.kao@fao.org>

Maintainer Joshua M. Browning <joshua.browning@fao.org>

Description The package is a replication and improvement to the AUPUS module in the old Statistical Working System (SWS)

License GPL (>= 3)

LazyData yes

Imports
faosws (>= 0.3.2), igraph (>= 0.7.1), data.table (>= 1.9.2),reshape2 (>= 1.4), RJSONIO (>= 1.3-0), faoswsUtil (>= 0.1.4)

ZipData no

VignetteBuilder knitr

Suggests knitr, ggplot2

R topics documented:

`faoswsAupus-package` *Package to perform the AUPUS calculation.*

Description

This package provides functions to replicate the old system methodology (AUPUS) in the new system.

Author(s)

Joshua M. Browning <joshua.browning@fao.org>, Michael. C. J. Kao <michael.kao@fao.org>

`appendSymbol` *Function to fill symbol based on calculated value*

Description

The function will take a vector of values and fill in the new symbol if the value is not missing. If the value is missing, it will insert the missing symbol.

Usage

```
appendSymbol(value, newSymbol, missingSymbol = "M")
```

Arguments

`value`	The vector of calculated values
`newSymbol`	The new symbol to be assigned
`missingSymbol`	The symbol filled in when value is missing

Value

A list of two elements: the vector of passed values and the new derived vector of symbols.

`Aupus` *AUPUS*

Description

This function performs the whole aupus procedure.

Usage

```
Aupus(aupusNetwork, aupusParam)
```

Arguments

`aupusNetwork`	A list of two data.tables. The objects should be the node and edge data.tables. Typically, this argument will be the output from the function suaToNetworkRepresentation.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

A list with two components: nodes and edges. These objects are the same as the input nodes and edges but after being processed by the AUPUS algorithm.

`aupusGroups` *AUPUS Groups*

Description

The AUPUS logic has many groupings of elements that require specialized processing. The numbers of these groupings could be hard-coded in the code, but to increase visibility of these groups (and for clearer understanding of the code) these groups have been defined in this object. This object is a dataset of the faoswsAupus package, and contains defintions for all the various groups used within this package.

Usage

```
data(aupusGroups)
```

Format

A list object containing 1 numeric vector. Each vector specifies item type groups which are special scenarios for certain elements.

Details

For an example, consider initialAsSupply. These four item types (tobacco, tea, cocoa, coffee) consider initial existence as a supply in the balance, where all other elements do not consider initial existence in the balance equation at all.

A list and short description of each element is below:

- initialAsSupply A vector of the item types which consider initial existence as a supply in the balance equation.
- reemploymentNotAsUtilization Most item types consider element 151 (Reemployment other sector) as utilization. However, some item groups do not, and those are specified in this vector.
- excludedFromBalance This vector gives item types which are not balanced.
- specialBalanceSugar Instead of computing the balance as the difference between supply and utilization, the balance is computed as 1000 times the ratio of element 161 (final existence) divided by element 171 (consumption).
- specialBalanceNegate A vector of all item types where the negative of the computed value in the balance is used.
- escrDivisionFactor: A division factor of 1000 is used in reporting production values (instead of 10000).
- sugarDivisionFactor: A division factor of 1 is used in reporting production values (instead of 10000).
- escrSugarProduction: These cases are handled differently in the calculation of element 51 (Production).
- escrSugarExistence: These cases

`balanceProductionElements`

Balance Production Elements

Description

This function takes a data.table and balances the production elements (i.e. elements 31, 41, and 51). The relationship is 51 = 31 * 41 / divisionFactor, where divisionFactor is either 1, 1000, or 10000 depending on the item type.

Usage

```
balanceProductionElements(value31, value41, value51, symb31, symb41, symb51,
  divisionFactor)
```

Arguments

`value31`	A vector of values corresponding to element 31 (Actual Producing Factor).
`value41`	A vector of values corresponding to element 41 (Productivity Element).
`value51`	A vector of values corresponding to element 51 (Output).
`symb31`	A vector of sybmols corresponding to element 31. These are provided so that they can be returned (updated, if necessary, when a balance occurs).

symb41	A vector of sybmols corresponding to element 41. These are provided so that they can be returned (updated, if necessary, when a balance occurs).
symb51	A vector of sybmols corresponding to element 51. These are provided so that they can be returned (updated, if necessary, when a balance occurs).
divisionFactor	A numeric vector taking values of 1, 1000, or 10000. This number is required for the balance equation, and is determined based on item type.

Details

If two elements are available, the third will be computed. If only one or zero elements are available, no values will be updated. divisionFactor should always be supplied.

Value

A list of 6 elements containing values 31, 41, and 51 and symbols 31, 41, and 51. All elements will be the same as what was passed except when a balance was possible, and in this case one of the three values will be updated along with it's symbol set to "C".

---

buildEdges *Build Edges*

Description

Function to merge shares and extraction rate and build the relation graph.

Usage

```
buildEdges(dataList)
```

Arguments

dataList	A list of AUPUS datasets to be analyzed, typically as produced by the function getAupusDataset. For more details on this argument, please see ?getAupusDataset.

Value

A data.table object with the same keys as dataList$inputData (typically geographicAreaFS, measuredItemParentFS, measuredItmChildFS, and timePointYearsSP). Additional columns are Value_share (shares as specified in dataList$shareData), Value_extraction (extraction rates as specified in dataList$extractionRateData), and Value_input, flagFaostat_input (as specified in dataList$inputData).

`buildNodes` *Build Nodes*

Description

This function creates the node structure by merging aupus, ratio, balance element, item info and population datasets.

Usage

```
buildNodes(dataList)
```

Arguments

`dataList`	A list of AUPUS datasets to be analyzed, typically as produced by the function getAupusDataset. For more details on this argument, please see ?getAupusDataset.

Value

A data.table with the node data. Essentially, this is a data.table containing all the variables meant to be stored in the nodes: AUPUS, input, ratio, balance element, item info, and population.

`calculateAupusElements`
Calculate AUPUS Elements

Description

This is the wrapper function that performs the full AUPUS module. It computes each individual element, then proceeds to compute utilization and balances. This function should be called several times to process all the processing levels of the items.

Usage

```
calculateAupusElements(aupusFinalData, itemTypeCol, balanceElementNum,
  aupusParam)
```

Arguments

`aupusFinalData`	The data.table object containing the nodes data, typically as created by the suaToNetworkRepresentation function.
`itemTypeCol`	The column name of aupusFinalData corresponding to the item type of the commodity item.
`balanceElementNum`	The column name of aupusFinalData corresponding to the balance element column.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

No value is returned. Instead, the passed data.table (aupusFinalData) has each of it's elements computed and is balanced.

`calculateBalance` *Calculate Balance*

Description

The function will calculate the balance (in most cases, the difference between supply and utilization) and fill this value into the balanceElement column. However, if this computed value is negative, or if the balanceElement symbol is not replaceable (see ?replaceable) then this computed value is placed in the statistical discrepancy column.

Usage

```
calculateBalance(supply, utilization, balanceElement, data, aupusParam)
```

Arguments

`supply`	The column name of data corresponding to the calculated total supply.
`utilization`	The column correspoding to calculated total utilization.
`balanceElement`	The column name of data corresponding to the balancing element.
`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

A logical vector indicating which rows were updated with the balance.

`calculateDailyNutritive`
Calculate Daily Nutritive (Calories, Proteins, and Fats)

Description

This function converts total calories/proteins/fats to daily per person calories/proteins/fats. The calculation for each case is identical, and thus the logic is implemented in this one function. The calculation is straightforward, although there is a multiplicative adjustment of 1000: E_264 = E_261 * 1000 / (365 * Population) E_274 = E_271 * 1000 / (365 * Population) E_284 = E_281 * 1000 / (365 * Population) (261/264 is calories/calories per day, 271/274 is for proteins, 281/284 is for fats).

Usage

```
calculateDailyNutritive(population11Num, population21Num, dailyElement,
  totalElement, data, aupusParam)
```

Arguments

population11Num	The column corresponds to element 11 of the population.
population21Num	The column corresponds to element 21 of the population.
dailyElement	The element number for the daily element. Currently, this should be one of 264, 274, or 284.
totalElement	The element number for the total element. Currently, this should be one of 261, 271, or 281.
data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

Population is extracted from the population commodity's element 21 (FBS population). If this element is missing, it uses the population commodity's element 11 (initial existence).

Note: this function is a replacement for calculateEle264, calculateEle274, and calculateEle284.

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 264/274/284's value and symbol updated.

---

calculateEle101 *Calculate Element 101 (Use for Animals)*

Description

The amount used for animal feed is computed using a very simple formula: (Element 101) = (Ratio 101) * (Total Supply) / 100. Essentially, the amount used for animals is computed as a simple ratio of the total supply. This function performs that calculation.

Usage

```
calculateEle101(stotal, data, aupusParam)
```

Arguments

stotal	The column name of data corresponding to the total supply.
data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 101's value and symbol updated.

`calculateEle11` *Calculate Element 11 (Initial Existence)*

Description

Initial existence (element 11) at time t is imputed with final existence (element 161) at time t-1, but imputation only occurs if data is missing. Note that not all item types are not updated in this way (see code).

Usage

```
calculateEle11(data, aupusParam)
```

Arguments

`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 11's value and symbol updated.

`calculateEle111` *Calculate Element 111 (Use for Same Product)*

Description

This function imputes element 111, which can be thought of as the amount used for seeding. The rules are as follows:

Usage

```
calculateEle111(stotal, data, aupusParam)
```

Arguments

`stotal`	The column name of data corresponding to the total supply.
`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

- If Ratio 171 (presumably a seeding ratio) is not available for the current commodity, a similar formula to 101 is used: E_111 = R_111 * (total supply) / 100.
- If Ratio 171 is available, then an estimate for total production is used to multiply by Ratio 171 to compute the amount used for seed. The preferred estimate for production value is element 21 (potential producing factor) in the next year; if this is unavailable then element 31 (actual producing factor) in the next year is used; then element 21 in the current year; and lastly element 31 in the current year. The value for element 111 is then filled in as this computed seed value times R_171 divided by 1000.

Value

This function returns a list (of length 2) of integer vectors containing the row numbers of observations which were updated (the first vector when a value was computed using ratio 111, the second for when ratio 171 was used). However, it also has a side effect: rows within the passed data.table ("data") have element 111's value and symbol updated.

---

calculateEle121 *Calculate Element 121 (Losses)*

Description

Element 121 (losses) is calculated as a simple ratio of total supply: E_121 = R_121 * (total supply) / 100. Values are always updated in this fashion unless the symbol for element 121 is not replaceable (i.e. it's an official figure).

Usage

```
calculateEle121(stotal, data, aupusParam)
```

Arguments

stotal	The column name of data corresponding to total supply
data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 121's value and symbol updated.

calculateEle131 *Calculate Element 131 (Reemployment Same Sector)*

Description

Element 131 (Reemployment Same Sector) is calculated as a simple ratio of total supply: E_131 = R_131 * (total supply) / 100. Values are always updated in this fashion unless the symbol for element 131 is not replaceable (i.e. it's an official figure).

Usage

```
calculateEle131(stotal, data, aupusParam)
```

Arguments

stotal	The column name of data corresponding to total supply.
data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 131's value and symbol updated.

calculateEle141 *Calculate Element 141 (Consumption)*

Description

This function computes element 141 (Consumption)

Usage

```
calculateEle141(stotal, data, aupusParam)
```

Arguments

stotal	The column name of data corresponding to total supply.
data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

The following AUPUS logic is used in computing this element:

- For commodity types in 58, 59, 60, and 61, element 141 is computed as E_11 + E_51 + E_61 - E_91 - E_95 - E_161.
- For commodity type 50 (Jute), the original AUPUS logic was very complex and also written in such a way as to cause potential errors. Thus, no processing is done for this commodity type: it is left unchanged by this function.
- For all other commodity types, the consumption is computed as a ratio of the total supply: E_141 = R_141 * (total supply) / 100

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 141's value and symbol updated.

`calculateEle144` *Calculate Element 144 (Consumption Per Day)*

Description

The consumption per day is the ratio of total consumption to the population. This value is computed for all replaceable symbols (i.e. when the current figure is not official). For some commodity groups, an adjustment is made by multiplying by 1000.

Usage

```
calculateEle144(population11Num, data, aupusParam)
```

Arguments

`population11Num`	The column corresponds to element 11 of population.
`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 144's value and symbol updated.

calculateEle151 *Calculate Element 151 (Reemployment other sector)*

Description

In almost all cases, element 151 (Reemployment other sector) is computed as a ratio of the total supply: E_151 = R_151 * (total supply) / 100. However, for one particular commodity (code 1687, Charcoal), element 151 is instead computed as the value of element 131 for commodity 1648 minus the value of element 51 for commodity 1687.

Usage

```
calculateEle151(stotal, data, aupusParam)
```

Arguments

stotal	The column name of data corresponding to total supply.
data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 151's value and symbol updated.

calculateEle161 *Calculate Element 161 (Final existence)*

Description

This element is only computed for commodity type 57 (sugar). This element is calculated as element 11 (Initial Existence) + element 71 (From Initial Existence).

Usage

```
calculateEle161(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 161's value and symbol updated.

`calculateEle171` *Calculate Element 171 (Consumption, Sugar only)*

Description

This element is only calculated for sugar. Moreover, it's just a summation of 5 other commodities: 101, 121, 131, 141, and 151.

Usage

```
calculateEle171(data, aupusParam)
```

Arguments

`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 171's value and symbol updated.

`calculateEle174` *Calculate Element 174 (Consumption per Caput, Sugar only)*

Description

This function computes consumption per caput for sugar. It is simply the ratio of total consumption over total population, and so the calculation is element 171 divided by population 11.

Usage

```
calculateEle174(population11Num, data, aupusParam)
```

Arguments

`population11Num`	The column corresponds to element 11 of the population.
`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 174's value and symbol updated.

`calculateEle21` *Calculate Element 21 (Potential Producing Factor)*

Description

The rule for replacement here is (element 21) = (element 111) * 1000 / (ratio 171)

Usage

```
calculateEle21(data, aupusParam)
```

Arguments

`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 21's value and symbol updated.

`calculateEle31` *Calculate Element 31*

Description

The value of this element is set to the sum of all the input values from the inputFromProcessing table (see getInputFromProcessData). Otherwise, the value (if replaceable) is set to NA.

Usage

```
calculateEle31(inputNum, data, aupusParam)
```

Arguments

`inputNum`	The column name of data corresponding to inputs from processing.
`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns a list (of length two) of integer vectors containing the row numbers of observations which were updated (first by computing as a function of inputs, then next by setting to NA). However, it also has a side effect: rows within the passed data.table ("data") have element 31's value and symbol updated.

`calculateEle314151` *Calculate Elements 31, 41, and 51*

Description

This function computes elements 31, 41, and 51 according to the AUPUS logic. The rules are as follows:

Usage

```
calculateEle314151(aupusParam, data)
```

Arguments

`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
`data`	The AUPUS node dataset, typically as produced by buildNodes.

Details

If exactly one of the three elements is missing, then that element is computed from the others using the relationship 51 = 31 * 41 / divisionFactor (i.e. "balancing").

If none of the three elements are missing, then any computed or trended values are recomputed. This is done by first attempting to balance: if only one value needs to be recomputed, it is done so by balancing. If more than one value needs to be recomputed, we first "trend" element 31 (see ?trendOnce). With element 31 recomputed, we now attempt to balance again. If we still cannot balance (i.e. 41 and 51 are still missing) we trend element 41. Now, as element 51 is the only missing value, it is computed via balancing.

Note that in one dataset, all of these different cases are possible (given different configurations for different years/countries). This function takes that into account and performs the appropriate approach for each case.

Value

This function returns a list (of length two) of integer vectors containing the row numbers of observations which were updated (first by updating missing values, second by retrending). However, it also has a side effect: rows within the passed data.table ("data") have element 31/41/51's value(s) and symbol(s) updated.

calculateEle41 *Calculate Element 41 (Productivity Element)*

Description

Calculation of element 41 is done by simply multiplying ratio 41 by 100. Values which are replaceable are updated by this function.

Usage

```
calculateEle41(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 41's value and symbol updated.

calculateEle51 *Calculate Element 51*

Description

There is no calculation of element 51 (Output) for almost all cases. Two special cases exist, however:

- If the commodity item type is in aupusGroups$escrSugarProduction and if element 58 (Production Crop Year) exists for the commodity of interest, then element 51 is set to the value of element 58.
- If the commodity code is 3183, then element 51 (Output) is set to the sum of element 51 (Output) for commodity codes 3158 and 3159.

Usage

```
calculateEle51(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 51's value and symbol updated.

`calculateEle541` *Calculate Element 541 (Final Demand)*

Description

Element 541 (Final Demand) is computed as a summation of elements 542, 543, 544, and 545. If some of those values are missing, they are ignored in the calculation. However, if all four of the values are missing, then the resulting calculation returns a 0M for element 541.

Usage

```
calculateEle541(data, aupusParam)
```

Arguments

`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 541's value and symbol updated. Additionally, columns will be added to the passed data.table if they don't currently exist: Value_measuredElementFS_54X, where X = 2, 3, 4, 5.

`calculateEle546` *Calculate Element 546 (Total Demand)*

Description

Element 546 (Total Demand) is computed as a summation of elements 541, 151, and 91 If some of those values are missing, they are ignored in the calculation. However, if all three of the values are missing, then the resulting calculation returns a 0M for element 546.

Usage

```
calculateEle546(data, aupusParam)
```

Arguments

`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 546's value and symbol updated.

---

calculateEle58 *Calculate Element 58 (Production Crop Year)*

Description

Calculation of element 58 is done by adding together element 3158 and 3159 for the source commodities.

Usage

```
calculateEle58(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 58's value and symbol updated.

---

calculateEle63 *Calculate Element 63*

Description

This function calculates element 63 (Inflow Unit Value) using element 61 (Inflow) and element 62 (Inflow Value). The calculation is simple: unit value (63) should be the total value (62) divided by the total quantity (61). However, a multiplicative constant of 1000 is also used, so the formula is 63 = 62 * 1000 / 61.

Usage

```
calculateEle63(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

Note that if 61 or 62 are not available, and if 63 is replaceable, then it is replaced with a missing value (0M).

Value

This function returns a list (of length 2) of integer vectors containing the row numbers of observations which were updated (the first vector when a value was computed, the second for when 62 or 61 was missing). However, it also has a side effect: rows within the passed data.table ("data") have element 63's value and symbol updated.

---

calculateEle66 *Calculate Element 66 (standardized inflow)*

Description

This function does not seem to be run. It contains functions which are not defined, and the documentation states that computation of this element is only for commodities that are generally not used within AUPUS, and that are only trade commodity types.

Usage

```
calculateEle66(shares, data, shareData)
```

Arguments

shares	The column name of the share variable in shareData (i.e. the column containing the proportion of the parent commodity that is passed to the child commodity).
data	The AUPUS node dataset, typically as produced by buildNodes.
shareData	The data.table containing the shares data.

Value

This function is not currently working.

---

calculateEle71 *Calculate Element 71*

Description

This function calculates element 71 (from initial existence).

Usage

```
calculateEle71(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

Element 71 is only computed for a few commodity types: 57, 58, 59, 60, and 61. For the last three types, the formula is short: element 71 = element 161 - element 11 (i.e. Final Existence minus Initial Existence). For the first two types, the computation is element 71 = element 51' + element 61 - element 91 - element 101 - element 121 - element 131 - element 141 - element 151.

Note: The documentation aupus_code_analysis.pdf has a typo. It does not mention how commodity type 57 should be handled, but after a conversation with Nick it was confirmed that commodity type 57 should be handled in the same way as commodity type 58.

Value

This function returns a list (of length 2) of integer vectors containing the row numbers of observations which were updated (the first vector when a value was computed for commodity type 58, the second for when the value was computed for commodity types 59, 60, or 61). However, it also has a side effect: rows within the passed data.table ("data") have element 63's value and symbol updated.

---

calculateEle93 *Calculate Element 93*

Description

This function calculates element 93 (outflow unit value). The logic is very similar to the calculation for element 63 (inflow unit value) in that unit value (93) should be the total value (92) divided by the total quantity (91). Also, the multiplicative constant of 1000 is still used in this case, so the formula is 93 = 92 * 1000 / 91.

Usage

```
calculateEle93(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

However, there is one difference between computation of element 93 and element 63: computation is not done for elements 42 through 52.

Value

This function returns a list (of length 2) of integer vectors containing the row numbers of observations which were updated (the first vector when a value was computed, the second for when 62 or 61 was missing). However, it also has a side effect: rows within the passed data.table ("data") have element 63's value and symbol updated.

---

calculateEle96 *Calculate Element 96 (Standardized Outflow)*

Description

This function does not seem to be run. It contains functions which are not defined, and the documentation states that computation of this element is only for commodities that are generally not used within AUPUS, and that are only trade commodity types.

Usage

```
calculateEle96(shares, data, shareData)
```

Arguments

shares	The column name of the share variable in shareData (i.e. the column containing the proportion of the parent commodity that is passed to the child commodity).
data	The AUPUS node dataset, typically as produced by buildNodes.
shareData	The data.table containing the shares data.

Value

This function is not currently working.

---

calculateTotalInput *Calculate Total Input*

Description

It's not clear what this function was intended for, as it is not called by any other functions in this package.

Usage

```
calculateTotalInput(inputData, inputNum, aupusParam)
```

Arguments

inputData	The data containing the input extracted from the function getInputFromProcess after running the updateInputFromProcess function.
inputNum	???
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

This function is not currently used.

`calculateTotalNutritive` *Calculate Total Nutritive (Calories, Proteins, Fats)*

Description

Each commodity has a ratio defined, R_261/R_271/R_281, which gives the conversion rate of the quantity of the commodity in question into calories/proteins/fats. Thus, element 261/271/281 is computed simply as E_261 = E_141 * R_261 / 100

Usage

```
calculateTotalNutritive(ratioNum, elementNum, data, aupusParam)
```

Arguments

`ratioNum`	The column name of data corresponding to the ratio for this element. Typically one of "Ratio_measuredElementFS_261" for calories, "Ratio_measuredElementFS_271" for proteins, or "Ratio_measuredElementFS_261" for fats.
`elementNum`	The element number. Currently, this should be one of 261 (calories), 271 (proteins) or 281 (fats).
`data`	The AUPUS node dataset, typically as produced by buildNodes.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

E_271 = E_141 * R_271 / 1000

E_281 = E_141 * R_281 / 1000

Value

This function returns an integer vector containing the row numbers of observations which were updated. However, it also has a side effect: rows within the passed data.table ("data") have element 261/271/281's value and symbol updated.

calculateTotalSupply *Calculate Total Supply*

Description

This function calculates total supply by summing up elements 51, 58, 61, and 66 (see arguments below to understand what these items are). Also, the itemTypeCol is required as a handful of commodities are treated differently when computing supply (see aupusGroups$initialAsSupply).

Usage

```
calculateTotalSupply(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

No value is returned, but a new column ("TOTAL_SUPPLY") is appended to the passed data.table. This column provides the total supply for the balancing.

calculateTotalUtilization *Calculate Total Utilization*

Description

This function calculates total utilization for each row by summing up all elements which correspond to utilization. These are currently elements 91 (Outflow), 96 (Standardized Outflow), 101 (Use for Animals), 111 (Use for Same Product), 121 (Losses), 131 (Reemployment), 141 (Consumption), 151 (Reemployment Other Sector), 161 (Final Existence), and 546 (Total Demand).

Usage

```
calculateTotalUtilization(data, aupusParam)
```

Arguments

data	The AUPUS node dataset, typically as produced by buildNodes.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

Also, element 151 is not considered utilization for item types defined by aupusGroups$reemploymentNotAsUtilization (currently only item type 53).

Value

No value is returned, but a new column ("TOTAL_UTILIZATION") is appended to the passed data.table.

---

checkShareUnity *This is a function checking whether the shares sums up to 100.*

Description

This is a function checking whether the shares sums up to 100.

Usage

```
checkShareUnity(shareData)
```

Arguments

shareData	The share data returned from getShare.

Value

This function is not currently used.

---

coerceColumnTypes *Coerce Column Types*

Description

Currently, the working system may not provide the correct column types for datasets, and this can cause problems during joins at later stages (as keys are sometimes numeric and sometimes character, and a failure to match leads to errors). Thus, this function coerces all column types that it can, based on the aupusParam argument. For example, aupusParam$keyNames$areaName gives the name of the area id column of data, and thus this column should always be coerced to a numeric column.

Usage

```
coerceColumnTypes(aupusParam, data)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
data	The data.table whose column types should be coerced.

Details

Note: there is currently a ticket in the SWS (issues SWS-797) that should resolve this problem. However, we need a work-around until that issue has been resolved.

Value

A data.table with updated column types.

`collapseShare` *Collapse Share*

Description

Share data is extracted from the database in three chunks: country/year specific data, country specific data, and generic data (applying to all countries/years). The most specific data available should be used when available, otherwise a more general value should be used. This function collapses the three different datasets (supplied in shareData) into one final dataset.

Usage

```
collapseShare(aupusParam, shareData, shares, verbose = FALSE)
```

Arguments

`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
`shareData`	The shares data obtained from the function getShare. This is a list of length 3 with data.tables as elements (specific, yearWildCard, and areaYearWildCard). Each data.table contains share information at a different level of specificity.
`shares`	The name of the column correspond to shares within each element of shareData (note that it must be the same for each element).
`verbose`	Whether the output should be printed.

Value

Returns a single data.table derived by condensing the shareData list into one dataset.

collapseSpecificData *Collapse Specific Data*

Description

Certain datasets are extracted from the database in three chunks: country/year specific data, country specific data, and generic data (applying to all countries/years). The most specific data available should be used when available, otherwise a more general value should be used. This function collapses the three different datasets (supplied in listData) into one final dataset. Note that all three datasets should have the same columns except for the key columns which provide the level of detail (i.e. item, item/country, and item/country/year).

Usage

```
collapseSpecificData(aupusParam, listData, verbose = FALSE)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
listData	A list of three data.tables, typically as obtained from the functions getShare, getBalanceElementData, or getRatioData. This list is of length 3 with data.tables as elements (specific, yearWildCard, and areaYearWildCard). Each data.table contains information at a different level of specificity.
verbose	Whether the output should be printed.

Value

Returns a single data.table derived by condensing the listData list into one dataset.

constructStandardizationGraph
Construct Standardization Graph

Description

This function constructs the graph/network required for standardization by generating one graph object for each year. This graph has the supplied edges and nodes as the corresponding edges and nodes on the graph.

Usage

```
constructStandardizationGraph(aupusNetwork, standardizeElement, aupusParam)
```

Arguments

aupusNetwork	A list of 2 data.tables named "nodes" and "edges". These are usually produced by suaToNetworkRepresentation and contain information about the data at the nodes and along the edges of the network.
standardizeElement	The column names of nodes that should be included in the returned graphs. These are the elements to be standardized.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

A list of the same length as the number of unique years in the aupus network.

---

defineElementVariables

Define Element Variables

Description

Much of the AUPUS processing is done by computing functions of particular elements and using those calculated variables to update another element. Thus, it's often very important to have variables which specify which columns are needed. This specification can be very tedious, as often many elements are needed to compute a new element (as well as their flags/symbols and/or ratios).

Usage

```
defineElementVariables(elements, aupusParam, envir = parent.frame(1))
```

Arguments

elements	A numeric vector containing the element numbers of interest.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
envir	An environment which specifies where the variables should be assigned. This defaults to parent.frame(1), which gives the calling environment, and thus should be sufficient for most (if not all) use cases.

Details

To avoid this complication, this function takes as an input the elements of interest and the aupusParam list (which defines how all the element value, flag, and ratio columns are named) and assigns the needed variables back to the calling environment (see example). For each element, three variables are created in the environment: elementXNum, elementXSymb, and ratioXNum (where X is the passed element number). These variables contain the column names of the nodes dataset that correspond with the value and symbol column for element X, respectively.

Value

No value is explicitly returned. However, variables are assigned in the calling environment of this function.

Examples

```
## Not run:
exists("element51Num")
exists("element51Symb")
exists("ratio51Num")
defineElementVariables(51, faoswsAupus::aupusParam)
exists("element51Num")
exists("element51Symb")
exists("ratio51Num")

## End(Not run)
```

`elementCodeDescription` *Element Code Description*

Description

The AUPUS system defines "elements" and stores information in these variables. The element code defines what the variable holds, and this dataset describes which codes correspond to what variables. The first two columns are likely to be the most useful, as they contain the code and a short description. The remaining columns give examples of what that element code/variable represents for certain commodity groups.

Usage

```
data(elementCodeDescription)
```

Format

A data.table

`ensureAupusParameter` *Ensure AUPUS parameters*

Description

This function performs several checks on the AUPUS parameters to ensure it is valid. Typically, these parameters are created by getAupusParameter and should have no issues, but these parameters could be modified by the user.

Usage

```
ensureAupusParameter(aupusParam)
```

Arguments

aupusParam	The AUPUS parameter list, typically as generated by getAupusParameter.

Value

No data is returned, but errors are raised if the aupusParam argument is invalid.

---

fbsStandardization *FBS Standardization*

Description

This function takes the graph and performs the standardization.

Usage

```
fbsStandardization(graph, standardizeElement, plot, aupusParam)
```

Arguments

graph	The standardization graph from the function constructStandardizationGraph.
standardizeElement	A character vector of the column names of the node object that should be processed. This should correspond to the elements of the FBS that need to be standardized.
plot	Whether the network should be plotted as it is processed. A plot will be generated for each year/processing level, with prompts between, so only use this option if running this function in interactive mode.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

A data.table object containing the final standardization values for each year.

---

fillBalance *Fill Balance*

Description

This function determines how the balance and statistical discrepancy columns should be filled in. There are two scenarios:

Usage

```
fillBalance(calculatedBalance, balanceNum, balanceSymb, element181Num,
  element181Symb, data)
```

Arguments

calculatedBalance	The column name of data corresponding to the balance calculated by the function calculateBalance.
balanceNum	The column name of data corresponding to the value of the balancing element.
balanceSymb	The column name of data corresponding to the symbol column for the balancing element.
element181Num	The column name of data corresponding to the value of element 181 (Statistical Discrepancies).
element181Symb	The column name of data corresponding to the symbol of element 181 (Statistical Discrepancies).
data	The AUPUS node dataset, typically as produced by buildNodes.

Details

- If the symbol currently in the balance column is "replaceable" (see ?replaceable) and if the computed balance is positive, then the computed balance is placed in the balance column and statistical discrepancy is set to 0.
- If the computed balance is negative, or if the current balance symbol is not replaceable, then the computed balance is placed in the statistical discrepancy column.

Value

A list with 5 vectors of the same length:

- originalValue: The numeric vector of the updated balance column.
- originalSymb: The symbol vector of the updated balance column.
- discrepancyValue: The numeric vector for the statistical discrepancy column.
- discrepancySymb: The symbol vector for the statistical discrepancy column.
- replaced: A logical vector which is TRUE if the computed value has been placed in the balance element column and FALSE otherwise.

---

fillMissingColumn *This function creates columns which are missing.*

Description

Some countries may not have a certain element calculated and thus when the data is extracted, the column is missing. This function will recreate the missing column.

Usage

```
fillMissingColumn(data, allColumn)
```

Arguments

data	The data
allColumn	The theoretical set of all column names.

Value

This function does not seem to be currently in use.

`findConnectedGraph` *Find Connected Graph*

Description

This function takes a graph/network and then find the subset of the network which are connected to the specified nodes/commodities.

Usage

```
findConnectedGraph(graph, commodity)
```

Arguments

`graph`	The graph object created by the function constrctGraph.
`commodity`	The commodities

Value

This function is not currently used.

`findProcessingLevel` *Find Processing Level*

Description

The function finds the processing level of the item in relation to the rest of the commodity items. The processing level is zero for all nodes which have no inputs (i.e. they can be processed in the first round). A node with inputs has it's processing level defined as 1 greater than the max processing level of all of it's input nodes. Thus, nodes at level 0 are processed first, as their processing doesn't depend on any other commodites. Then, nodes at processing level 1 are processed because all their inputs (level 0 nodes) have been computed. All nodes at a fixed processing level can thus be computed in one step.

Usage

```
findProcessingLevel(edgeData, from, to, plot = FALSE, aupusParam)
```

Arguments

`edgeData`	The edge data, typically from the function buildEdges
`from`	The column name of edgeData corresponding to the from node.
`to`	The column name of edgeData corresponding to the target node.
`plot`	Logical, indicates of the graph should be plotted.
`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

A data.table providing the processing level for each of the items.

`getAupusData` *Get AUPUS Data*

Description

This function extracts the AUPUS data from the data base.

Usage

```
getAupusData(aupusParam, database = c("new", "old"), conn)
```

Arguments

`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
`database`	Whether to use the new or the old statistical working system.
`conn`	The RJDBS connection to the old working system, only required if database = "old".

Value

A data.table object containing the AUPUS data. The first three columns contain the dimensions (geographicAreaFS, measuredItemFS, and timePointYears) and the remaining columns are values and flags for each of the elements required by AUPUS.

`getAupusDataset` *Get AUPUS Data*

Description

This function obtains all required data for the AUPUS module by running several individual functions, such as getRatioData, getShareData, etc. The datasets returned by these individual functions are then returned from this function in a list (if assignGlobal = FALSE) or are assigned to the global environment (if assignGlobal = TRUE).

Usage

```
getAupusDataset(aupusParam, assignGlobal = FALSE)
```

Arguments

`aupusParam`	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
`assignGlobal`	Whether the data should be assigned globally (TRUE) or returned as a list (FALSE). Defaults to FALSE.

Value

If assignGlobal = FALSE, then a list is returned with names "aupusData", "inputData", "ratioData", "shareData", "balanceElementData", "itemInfoData", "populationData", and "extractionRateData". Each of these elements is a data.table containing information for the AUPUS processing. If assignGlobal = TRUE, then these data.tables are assigned to the global environment and no data is returned.

---

getAupusParameter *Get AUPUS Parameters*

Description

This function gets all the parameters in order to query the data, and either assigns these parameters to the global environment of returns them as a list.

Usage

```
getAupusParameter(areaCode, assignGlobal = TRUE, yearsToUse = NULL)
```

Arguments

areaCode	A character value giving the country code of the country of interest.
assignGlobal	logical, default to FALSE, in which case a list is returned. If TRUE, then the result will be assigned globally.
yearsToUse	If NULL, then all available years (as determined by GetCodeList) will be used. Otherwise, a vector of years should be provided and these years alone will be included.

Value

If assignGlobal is TRUE, nothing is returned but 5 objects are written to the global environment: areaCode, itemCode, elementCode, year, and keyNames. The first four of these variables each define a dimension for slicing the data, and the fifth element is a list of arguments to processing functions. If assignGlobal is false, these 5 objects are returned in a named list.

---

getBalanceElementData *Get Balance Element Data*

Description

This function extracts the balance element data from the database. This data specifies, for each measured item, the final element that it will be balanced to. For example, Flour of Wheat (item = 16) is placed into Food (element = 141).

Usage

```
getBalanceElementData(aupusParam, database = c("new", "old"), conn)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
database	Whether to use the new or the old statistical working system.
conn	The RJDBS connection to the old working system. Only required if database = "old".

Value

A list of three items: specific, yearWildCard, and areaYearWildCard. Each of these elements has the same general structure: a data.table with zero to two dimension columns (area and year), the measured item, and then the element that item is balanced into.

---

getExtractionRateData *Get Extraction Rate Data*

Description

This function extracts the extraction rate data from the aupus data by grabbing the key columns of aupusData as well as the column corresponding to extraction rate (as specified by element41Num).

Usage

```
getExtractionRateData(aupusData, aupusParam)
```

Arguments

aupusData	The data returned from the function getAupusData
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

Note: the old function queried the data.base for this data, but this new, simpler function just pulls directly from a table that has already been loaded into R.

Value

A data.table with the same key columns as aupusData and additional columns "Value_extraction" and "flagFaostat_extraction". This are the columns of aupusData corresponding to element 41.

getInputFromProcessData *Get Input from Processing Data*

Description

Some data used in AUPUS will be specified by the users prior to the AUPUS routine (for example, official data provided by some countries). This function provides a way for this input data to be specified and utilized in AUPUS.

Usage

```
getInputFromProcessData(aupusParam, database = c("new", "old"), conn)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
database	Whether to use the new or the old statistical working system.
conn	The RJDBS connection to the old working system. Only required if database = "old".

Value

This function returns a data.table object containing the input from processing data.

getItemInfoData *Get Item Information Data*

Description

The name and type of the commodity corresponding to the item code is returned.

Usage

```
getItemInfoData()
```

Value

A data.table with the codes for each measured item as well as a name describing the item and an item type.

getPopulationData *Get Population Data*

Description

This is the function to obtain the population separately.

Usage

```
getPopulationData(aupusParam, database = c("new", "old"), conn)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
database	Whether to use the new or the old statistical working system.
conn	The RJDBS connection to the old working system. Only required if database = "old".

Value

A data.table containing the country and year values specified in aupusParam and value and flags for the population (elements 11 and 21).

getRatioData *Get Ratio Data*

Description

This function extracts the ratio data from the data base. The ratio data contains information on ratios that exist between certain "elements" and "items." For example, how many calories (element = "261") are in wheat flour (item = "16")? The answer according to the database is 364, but better estimates may exist for specific years and specific countries. This dataset contains the overall average rates and the specific rates.

Usage

```
getRatioData(aupusParam, database = c("new", "old"), conn)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
database	Whether to use the new or the old statistical working system.
conn	The RJDBS connection to the old working system. Only required if database = "old".

Value

A list of three items: specific, yearWildCard, and areaYearWildCard. Each of these elements has the same general structure: a data.table with one to three dimension columns (area, year, and item) and then many columns containing the ratios (or conversion factor) from that item into elements in the FBS. If such values don't make sense or don't apply, an NA is in the data.

---

getShareData *Get Share Data*

Description

This function extracts the shares data from the data base. The shares data contains information on children are shared amongst parents. For example, some children of wheat flour (item = "16") are: macaroni (18), bread (20), and pastry (22). In Algeria (geographicAreaFS = "4"), the shares are 95, 3, and 2 which means that 95 to bread and 2 wildcard, and values can vary by year (also with a wildcard of 0).

Usage

```
getShareData(aupusParam, database = c("new", "old"), conn)
```

Arguments

aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
database	Whether to use the new or the old statistical working system.
conn	The RJDBS connection to the old working system. Only required if database = "old".

Value

A list of three items: specific, yearWildCard, and areaYearWildCard. Each of these elements has the same general structure: a data.table with zero to two dimension columns (area and year), the parent and child items, and then the Value_share column (representing the proportion allocated to each child).

---

itemTree *Commodity Item Tree*

Description

This dataset provides each of the individual item codes/names for all the commodities and also specifies whether that particular commodity is a target commodity (i.e. is aggregated up to in the standardization) as well as the equivalent FBS code.

Usage

```
data(itemTree)
```

Format

A data.table object of 12 columns by 808 rows.

Details

A short description of each column is below:

- itemCode: The Supply and Utilization Account (SUA) ID number for the commodity.
- itemName: The name of the commodity
- incTot: (More documentation needed here...)
- aggCom: (More documentation needed here...)
- weight: (More documentation needed here...)
- target: Specifies if the particular commodity is a target. If so, then all the children of this commodity will be aggregated up to this commodity (i.e. expressed in this commodity) during the standardization proceedure.
- convType: If this column takes the value "(cal.)", then it is standardized only for caloric purposes. For example, when wheat (15) is processed, it generates flour wheat (16), bran wheat (17), and germ wheat (19). If each of these items is standardized to wheat in terms of quantities, we'll be overcounting. However, calories can be standardized, and so one commodity is chosen for the quantity standardization (in this case flour wheat) and all three are used for caloric standardization. The (cal.) indicates which commodities are not standardized with quantities but only with calories.
- targetCode: Each commodity is standardized to some commodity (possibly itself). The targetCode specifies the itemCode of the "target", or the commodity it is standardized to.
- targetName: The name of the target commodity, see targetCode.
- baseExtraction:
- fbsCode: The corresponding code for this commodity in the Food Balance Sheets (FBS).
- fbsName: The corresponding name for this commodity in the Food Balance Sheets (FBS).

`itemTypeDescription` *Item Type Description*

Description

Each commodity of interest has an "item type". This item type is a general grouping of items, and describes what kind of item a particular commodity is. Each commodity has an item code, and the item type can be thought of as defining logical groups of item codes.

Usage

```
data(itemTypeDescription)
```

Format

A data.table with two columns: the itemTypeCode and the corresponding name.

numberOfMissingElement

Number of Missing Elements

Description

This function takes several inputs, typically vectors, and returns a vector whose ith element specifies the number of input vectors with missing values in the ith position.

Usage

```
numberOfMissingElement(...)
```

Arguments

...	The objects

Value

A vector of the number of total inputs missing for each ith element.

Examples

```
numberOfMissingElement(c(1,2,NA,NA,5),
                       c(1,NA,2,3,NA),
                       c(1,2,NA,4,NA))
```

numberOfTrendingElement

Number of Trended Elements

Description

This function takes several inputs, typically vectors, and returns a vector whose ith element specifies the number of input vectors with symbol equal to "T" in the ith position (which denotes a trended estimate).

Usage

```
numberOfTrendingElement(...)
```

Arguments

...	The objects

Value

A vector of the number of trended symbols in the ith position.

Examples

```
numberOfTrendingElement(c("M","","","T",""),
                        c("M", "T", "M", "T", ""),
                        c("C", "C", "T", "T", "T"))
```

`plotCommodityTree` *Plot Commodity Tree*

Description

This function is useful for visualizing the relationships between different commodities as defined within the shareData dataset (see getShareData).

Usage

```
plotCommodityTree(shareData, parentName = "measuredItemParentFS",
  childName = "measuredItemChildFS", ...)
```

Arguments

`shareData`	The data.table containing the share information, typically as generated by getShareData.
`parentName`	The column name of shareData which contains the item code of the parent.
`childName`	The column name of shareData which contains the item code of the child.
`...`	Additional parameters to be passed to the plot.igraph function.

Value

No data is returned, but a plot of the commodity tree is drawn using the igraph package.

`replaceable` *Returns the index of which values can be replaced*

Description

Only data with replaceable symbols are replaced by the module. This function will return the index in which the values can be replaced.

Usage

```
replaceable(symb, newValue = 0, replaceableSymb = c("C", "T", "M"))
```

Arguments

`symb`	The vector of symbols to be tested.
`newValue`	The new value which can be written to the cell. The AUPUS logic states that the old value should not be replaced if it was trended and the new value is missing (0 in the old framework).
`replaceableSymb`	A vector specifying which symbols can be replaced.

Value

A logical vector indicating if each individual element can be replaced.

`SaveAupusData` *Save AUPUS Data*

Description

This function is designed as a wrapper around SaveData, and is specifically used to save data produced from the AUPUS analysis back to the SWS.

Usage

```
SaveAupusData(aupusData, nodes)
```

Arguments

`aupusData`	A data.table containing the AUPUS data read from the database, see ?getAupusDataset. This is provided solely for extraction of the appropriate column names, it isn't used in any other way.
`nodes`	The data after having been processed by the AUPUS proceedure. This is the data written back to the SWS.

Value

No data is returned, but the nodes data.set is written back to the SWS.

`SaveInputFromProcessingData`
Save Inputs from Processing Data

Description

Save Inputs from Processing Data

Usage

```
SaveInputFromProcessingData(inputData, edges)
```

Arguments

`inputData`	A data.table containing the input data read from the database, see ?getInputFromProcessData. This is provided solely for extraction of the appropriate column names, it isn't used in any other way.
`edges`	The edge data.table, after having been processed by the AUPUS proceedure. This is the updated data.set to write back to the SWS.

Value

No data is returned, but the edges data.set is written back to the SWS.

SavePopulationData *Save Population Data*

Description

Save Population Data

Usage

```
SavePopulationData(populationData)
```

Arguments

populationData	The population data to write back to the SWS.

Value

No data is returned, but the population data.set is written back to the SWS.

standardization *Standardization*

Description

Standardization is simply a weighted aggregation, but with multiple hierachical levels. This function contains the logic for iterating over the different hierarchical levels, but it passes the actual computation of the aggregations on to the standardizeNode function.

Usage

```
standardization(graph, standardizeElement, plot, aupusParam, ...)
```

Arguments

graph	The igraph object containing the node and edge information, typically as generated by
standardizeElement	A character vector of the column names of the node object that should be processed. This should correspond to the elements of the FBS that need to be standardized.
plot	Whether the network should be plotted as it is processed. Several plots will be generated with prompts between, so only use this option if running this function in interactive mode.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).
...	Additional arguments to be passed to the plotting function.

Value

A data.table object with one column for the item key and one column for each element of standardizeElement (containing the value after standardization has been performed).

`standardizeNode` *Standardize Node*

Description

Standardization follows the below process:

- Compute a matrix ("reverseMatrix" in the code) describing how much of a particular quantity is passed from a child to it's parent. The (i,j)-th entry of this matrix specifies a conversion factor for transferring element j quantities into element i quantities.
- Compute the matrix of required elements (column) for each commodity (row), and call this "valueMatrix".
- Compute the new value for each commodity by pre-multiplying valueMatrix by reverseMatrix and therefore aggregating children commodities into their parents.
- Lastly, save this new data back to the passed graph object, and return.

Usage

```
standardizeNode(graph, workingNode, standardizeElement)
```

Arguments

`graph`	The graph object created by the function constructGraph.
`workingNode`	The nodes to be standardized
`standardizeElement`	The attribute of the nodes to be standardized.

Value

A graph object that has been updated according to the description above.

`suaToNetworkRepresentation`
SUA to Network Representation

Description

SUA data (Supply Utilization Accounts) contains information on specific uses of commodities within the food balance sheet. Often, SUA is explained as being analogous to all the financial transactions of a company and the Food Balance Sheets are thought of as the financial summary statement.

Usage

```
suaToNetworkRepresentation(dataList, aupusParam)
```

Arguments

dataList	A list of AUPUS datasets to be analyzed, typically as produced by the function getAupusDataset. For more details on this argument, please see ?getAupusDataset.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Details

This function converts sua data to a network specification which is more natural for processing.

Value

A list of two elements: nodes and edges. Both of these objects are data.tables, and they share the same keys with one exception: edges has a parent and child item key where nodes only has an item key. All the input, share and extraction data is stored in edges and the AUPUS, ratio, balance, item, and population data is stored in the nodes.

---

subsetAupus *Subset AUPUS Data*

Description

This function takes as input the list of data.table data as generated by getAupusDataset and subsets it based on specific item (commodity) codes. This can be useful for analyzing smaller datasets and understanding what the algorithm is doing. Note that if the particular item code(s) have any children or parents, those values (and their relatives, and so on) will also be included.

Usage

```
subsetAupus(aupusData, itemKeys, aupusParam)
```

Arguments

aupusData	The AUPUS dataset, typically as generated by getAupusDataset. This contains a list of 8 data.tables with the data of interest.
itemKeys	A numeric vector containing the item code(s) to filter on.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

An object of the same structure as aupusData, but with each individual data.table filtered according to the set of keys provided.

symbolDescription *Symbol Description*

Description

A table of the symbols in use, containing the symbol/flag and the name/description of the flag. The replacable column indicates whether data with that particular symbol can be imputed.

Usage

```
data(symbolDescription)
```

Format

A data.table containing three columns, as described above.

transferSymb *Transfer Symbol*

Description

This function contains the logic for updating a symbol when it is transfered to a new cell.

Usage

```
transferSymb(symb)
```

Arguments

symb	The original symbol to be updated.

Value

The updated symbol.

trendOnce *Trend Once*

Description

"Trending" the data refers to carrying over past values. For example, trending c(1,2,NA,3) would yield c(1,2,2,3). However, missing values are only replaced if their corresponding symbols are "T" or "C".

Usage

```
trendOnce(Num, Symb, applyIndex = 1:length(Num), transfer = FALSE)
```

Arguments

Num	A vector of numeric values to be trended.
Symb	A vector of the same length as Num containing the symbols (or "flags") for the observations in Num.
applyIndex	This argument allows the user to restrict which values can be trended. By default, all values are included.
transfer	Should the symbol be converted by the function transferSymb.

Details

This function trends the passed data (Num) assuming a vector of symbols (Symb). The "transfer" argument allows the user to specify if the symbol should also be transferred when the number is copied forward.

Value

A list containing the values and symbols after the trending has occurred.

updateEdges *Update Edges*

Description

Function to update the extraction rates and input from processing after each iteration of Aupus. This function takes the quantity of a commodity that was allocated to Reemployment Same Sector in one step and passes that value on to the edges (to be passed to the children). Additionally, the extraction rates computed by the parents are passed on to the children.

Usage

```
updateEdges(nodes, edges, aupusParam)
```

Arguments

nodes	The nodes data returned by the function buildNodes
edges	The edge data returned by the function buildEdges
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

No data is returned, but the edges object is updated with the information from nodes.

---

updateInputFromProcessing

Update Input from Processing

Description

The edge dataset contains information about the quantities of a particular item flowing from a particular parent to a particular child. This function aggregates the edge dataset, grouping by child, and replaces element 31 (Actual Producing Factor) of the child with the inputs from all the parents.

Usage

```
updateInputFromProcessing(nodes, edges, aupusParam)
```

Arguments

nodes	The nodes data returned by the function buildNodes.
edges	The edge data returned by the function buildEdges.
aupusParam	A list of running parameters to be used in pulling the data. Typically, this is generated from getAupusParameter (see that function for a description of the required elements).

Value

No values are returned, but instead the passed nodes dataset is updated.

US *US Sample dataset*

Description

An example dataset of what can be created from calling the function getAupusDataset.

Usage

```
data(US)
```

Format

A list object containing 8 individual datasets. Each dataset is generated by the corresponding get* function in the faoswsAupus package. Most elements are data.tables, but some (such as the ratio and balanceElement datasets) are lists of data.tables. See the documentation for the corresponding get* functions for more details.

usAupusParam *Example AUPUS Parameters*

Description

This object is an example of the type of object created by the getAupusParameter function. It contains codes/dimensions of the data as well as column name descriptions.

Usage

```
data(usAupusParam)
```

Format

A list object with 5 elements: areaCode, itemCode, elementCode, year, and keyNames. The first four elements are numeric vectors providing the dimensions of the data to be queried (country, commodity, variable, and year, respectively). The last element is another list containing various parameters:

- areaName Column name corresponding to the variable containing the location codes.
- itemName Column name corresponding to the variable containing the item codes.
- itemParentName Used in an edge dataset (i.e. shares), this column name corresponds to the variable containing the parent item code.
- itemChildName Used in an edge dataset (i.e. shares), this column name corresponds to the variable containing the child item code.
- itemTypeName Column name corresponding to the variable containing the item type, see ?faoswsAupus::itemTypeDescription.
- elementName Column name corresponding to the variable containing the element codes.
- extractionRateName Column name corresponding to the variable containing extraction rates, see getExtractionRateData.

- balanceElementName Column name corresponding to the variable containing the balancing element, see getBalanceElementData.
- inputName Column name corresponding to the variable containing the input from processing data, see getInputFromProcessData.
- shareName Column name corresponding to the variable containing the share data, see getShareData.
- yearName Column name corresponding to the variable containing the year.
- valuePrefix To construct variables corresponding to certain elements, the valuePrefix is combined with the itemName element and the element code of interest. For example, paste0(aupusParam$keyName$val aupusParam$keyName$itemName, "_", 11) gives the variable corresponding to element 11.
- flagPrefix See above description for valuePrefix, the same applies for the flag variables.
- ratioPrefix See above description for valuePrefix, the same applies for the ratio variables.

`wildCardFill` *Wild Card Fill*

Description

The function takes wild card data to fill in original data for a particular variable.

Usage

```
wildCardFill(originalData, wildCardData, variable, verbose = FALSE)
```

Arguments

`originalData`	The data to be filled in. This is typically a data.table with all year/location pairs.
`wildCardData`	The wild card data, see ratio or shares data extracted from the getShare and getRatio function. This should be a data.table with values to be inserted into originalData.
`variable`	The column name (of both originalData and wildCardData) of the variable to be filled.
`verbose`	Whether the output should be printed.

Details

Shares, ratio and balance element have area and year specific rates, but at the same time they have wild card values which are to be applied when the specific rates are not available.

This function fills in the gap with wild card values when year specific values are not available.

Note: this function is usually called several times to fill in one main data.table (originalData) with data from other data.tables (passed as wildCardData, and these tables have varying levels of specificity). The first time this function is called, it should be given the most specific data. Then, future calls should receive more and more general tables. The rationale is that the later tables will only overwrite values which are currently missing and no more.

Value

No data is returned, but missing records in originalData are replaced by their corresponding records in wildCardData.

Index