# Package 'faosws'

April 1, 2015

**Type** Package

**Title** API providing access to the FAO Statistical Working System

**Version** 0.3.3

**Date** 2013-04-03

**Author** Engineering Ingegneria Informatica

**Maintainer** ? <rsws@fao.org>

**Description** API providing access to the FAO Statistical Working System

**License** @FAO 2013

**Depends** R (>= 3.0)

**Imports** RCurl, RJSONIO, data.table

**Collate** Dimension.R DatasetKey.R MappingTableKey.R Pivoting.R KeyDefinition.R MetadataElement.R Metadata.R BlockMetadata.R BlockMetadataSet.R GetRestCall.R PostRestCall.R PutRestCall.R GetData.R GetCodeList.R GetCodeTree.R GetHistory.R GetMapping.R GetBlockMetadata.R SaveData.R SaveValidation.R SaveBlockMetadata.R GetTestEnvironment.R zzz.R buildModifiedCells.R GetTableData.R SetTableData.R NullToNa.R

## R topics documented:

---

## `BlockMetadata-class` *Block Metadata Class*

---

### Description

This S4 object represents a single block metadata.

### Arguments

| | |
|---|---|
| `blockId` | Numeric value that holds the unique ID of the block metadata. The GetBlock-Metadata call returns this information in order to allow update operation on block metadata, by specifying an existing ID in a SaveBlockMetadata call. Omitting this information in a SaveBlockMetadata call will create a new block metadata. |
| `selection` | A list of instances of the Dimension class. |
| `metadata` | An instance of the Metadata class. |

---

## `BlockMetadataSet-class` *The Block Metadata Set Class*

---

### Description

The Block Metadata Set Class

### Arguments

| | |
|---|---|
| `keyDefinitions` | List of dimensions that characterize the underlying dataset. Each list element has to be an instance of the S4 class KeyDefinition. |
| `blockMetadata` | A list of BlockMetaData objects. |

## `DatasetKey-class` *Dataset Key Class*

### Description

Dataset Key Class

### Arguments

| | |
|---|---|
| `domain` | A character value specifying the domain of interest, such as "agriculture". |
| `dataset` | A character value specifying the dataset within the domain, such as "agriculture". |
| `dimensions` | A list of objects, each of class Dimension. |
| `sessionId` | The ID of the session from which data should be accessed. If NULL, the database is used directly. |

### Note

Many of these variables (domain, dataset, and dimensions) can be found by examining some of the swsContext objects which are created in a GetTestEnvironment call (in a debug session). Moreover, after executing GetTestEnvironment, the swsContext.datasets object will contain a list of objects of type DatasetKey.

## `Dimension-class` *Dimension Class*

### Description

A dimension is represented by its name and by the vector of selected codes/keys.

### Arguments

| | |
|---|---|
| `name` | The name of the dimension to access, some examples are "timePointYears", "measuredElement", or "geographicAreaM49". These correspond to the dimensions on the SWS, but the names are slightly different. Currently, if you want to find the name of a dimension, you need to ask CIO. |
| `keys` | The key codes specifying what values should be pulled for the dimension. |

## `GetBlockMetadata` *Get Block Metadata*

### Description

Get Block Metadata

### Usage

```
GetBlockMetadata(key)
```

### Arguments

| | |
|---|---|
| `key` | A key object, as contained in a BlockMetadata object. |

## `GetCodeList` *Get Code List*

### Description

This function returns information about certain codes for a dimension. For example, one may wish to check if a particular country has a startDate or endDate in the system before attempting to write a particular value. See example below for a query that pulls the startDates for three countries. Note that Belarus(1992-) is not defined prior to 1992.

### Usage

```
GetCodeList(domain, dataset, dimension, codes)
```

### Arguments

| | |
|---|---|
| `domain` | A character value specifying the domain for which the code list is required. |
| `dataset` | A character value specifying the dataset for which the code list is required. |
| `dimension` | A character value specifying the name of the key for which the code list is required. |
| `codes` | (optional) A list of codes for which the key data is required. |

### Value

A data table containing the key codes and matching labels.

### Examples

```
## Not run:
GetCodeList(domain = "agriculture", dataset = "agriculture",
            dimension = "geographicAreaM49", codes = c("4", "8", "112"))

## End(Not run)
```

---

## GetCodeTree *Get Code Tree*

---

### Description

Get Code Tree

### Usage

```
GetCodeTree(domain, dataset, dimension, roots)
```

### Arguments

| | |
|---|---|
| `domain` | A character value specifying the domain for which the code list is required. |
| `dataset` | A character value specifying the dataset for which the code list is required. |
| `dimension` | A character value specifying the name of the key for which the code list is required. |
| `roots` | [optional] A list of root codes for the tree. Default is all those with no parent node. |

### Value

A data table containing the parent codes and lists of child codes in the tree (see the Code Tree structure above).

### Examples

```
## Not run:
GetCodeTree(domain = "agriculture", dataset = "agriculture",
                     dimension = "geographicAreaM49", roots = "953")

## End(Not run)
```

---

## GetData *Get Data*

---

### Description

This function provides an interface between an R session and the database. Note that swsContext files must exist in your session, so you should run GetTestEnvironment before calling this function.

### Usage

```
GetData(key, flags = TRUE, normalized = TRUE, metadata = FALSE, pivoting)
```

## Arguments

| | |
|---|---|
| `key` | An object of class DatasetKey. Often, this will be one of the list elements of swsContext.datasets (if running in a debug/local session, create this object with GetTestEnvironment). |
| `flags` | Logical, indicating if flags should be returned (TRUE) otherwise not returned (FALSE). |
| `normalized` | Logical, if true then data are returned in normalized format, otherwise the format is denormalized. |
| `metadata` | Logical, if true then metadata are returned otherwise not. |
| `pivoting` | A vector, each of whose elements must be an object of type Pivoting. If omitted, no pivoting is performed on the dataset. Using this argument can allow for convenient reshaping of the data prior to pulling it into R. Note: if this argument is included, then all of the dimensions in key must be included in this vector. See ?Pivoting for a description on creating this argument and for some examples on how to use it. |

## Details

If the denormalized value is not set then the data is normalized; if set the data is denormalized along the axis specified.

If the pivoting vector is present, the dimensions are extracted in the specified order and applying the requested sort direction. This affects both normalized and denormalized extractions. For normalized extractions only the order of the dimensions is influenced, while for denormalized extraction the effect is more evident since the last dimension specified in the pivoting vector gets its values developed along the column of the generated data.table result object.

## Value

A data table containing the data matching the key (may be empty).

## Examples

```
## Not run:
# swsContext files are necessary for GetData to run (token may need to be updated)
GetTestEnvironment(
   baseUrl = "https://hqlqasws1.hq.un.fao.org:8181/sws",
   token = "7823c00b-b82e-47bc-8708-1be103ac91e4"
)

# Use GetCodeList to find all countries and commodities
areaCodes = GetCodeList("agriculture", "agriculture", "geographicAreaM49")
itemCodes = GetCodeList("agriculture", "agriculture", "measuredItemCPC")

# Pull data for one country and all commodities
dim1 = Dimension(name = "geographicAreaM49", keys = "12")
dim2 = Dimension(name = "measuredElement", keys = "5510")
dim3 = Dimension(name = "measuredItemCPC", keys = itemCodes[, code])
dim4 = Dimension(name = "timePointYears", keys = as.character(2000:2013))
key = DatasetKey(domain = "agriculture", dataset = "agriculture",
                 dimensions = list(dim1, dim2, dim3, dim4))
GetData(key)

# Pull data for all countries and one commodity
```

```
dim1 = Dimension(name = "geographicAreaM49", keys = areaCodes[, code])
dim2 = Dimension(name = "measuredElement", keys = "5510")
dim3 = Dimension(name = "measuredItemCPC", keys = "0111")
dim4 = Dimension(name = "timePointYears", keys = as.character(2000:2013))
key = DatasetKey(domain = "agriculture", dataset = "agriculture",
                 dimensions = list(dim1, dim2, dim3, dim4))
GetData(key)

## End(Not run)
```

---

`GetHistory` *Get History*

---

## Description

Get History

## Usage

```
GetHistory(key, pivoting)
```

## Arguments

| | |
|---|---|
| `key` | An object of class DatasetKey. Often, this will be one of the list elements of swsContext.datasets (if running in a debug/local session, create this object with GetTestEnvironment). |
| `pivoting` | A vector, each of whose elements must be an object of type Pivoting. If omitted, no pivoting is performed on the dataset. Using this argument can allow for convenient reshaping of the data prior to pulling it into R. |

## Value

A data table of observation objects containing the values and flags through history with the associated history metadata.

---

`GetMapping` *Get Mapping*

---

## Description

This is expected to be used only by advanced script writers. The key will be a custom key layout matching the axes of the mapping table.

## Usage

```
GetMapping(key)
```

## Arguments

| | |
|---|---|
| `key` | A DatasetKey class object. |

### Details

The key may make use of wildcards to widen the number of results matched.

Map tables are always read-only

### Value

A data table containing the map entries matching the key (may be empty).

---

## `GetRestCall` *Get Rest Call*

---

### Description

Get Rest Call

### Usage

```
GetRestCall(url)
```

### Arguments

`url`

---

## `GetTableData` *Get Table Data*

---

### Description

Invokes a SWS RESTful POST to perform a query on a specific schema/table, returning the list of records matched, or NULL if none. If the SQL request and resulting query is incorrect, SQL error details are pasted and execution blocked; if HTTPS connection fails (e.g. a runtime error on the server), HTTP status != 200 is pasted and execution blocked.

### Usage

```
GetTableData(schemaName, tableName, whereClause = NULL,
  selectColumns = NULL)
```

### Arguments

| | |
|---|---|
| `schemaName` | A string containing the name of the schema to be accessed for the query. |
| `tableName` | A string containing the name of the table to be read by the query. |
| `whereClause` | (optional) A string containing whichever SQL clauses to take place after the "FROM" section (usually WHERE conditions). Example values: - "WHERE (id IN (1,2,3,4) AND descr = 'A') OR (id = 5 AND status = 0) " - "ORDER BY descr DESC" - "WHERE descr = 'aaa' ORDER BY id LIMIT 5" If NULL, nothing is appended in the SQL query composition after the FROM section. |
| `selectColumns` | (optional) The list of columns to be returned by the query. it can be: - NULL (the query will start with "SELECT * ") - a list of strings, such as: "list('id', 'descr')" |

### Value

A data.table object containing the queried dataset.

---

## `GetTestEnvironment` *Get Test Environment*

---

### Description

Get Test Environment

### Usage

```
GetTestEnvironment(baseUrl, token)
```

### Arguments

| | |
|---|---|
| `baseUrl` | The url for the SWS server, typically "https://hqlqasws1:8181/sws" or something similar. |
| `token` | A token which tells the SWS system what dataset to access. This token can be obtained from the system by<br>• Opening a session with the relevant data.<br>• Clicking "R Plugins"<br>• Selecting the relevant script to analyze.<br>• Clicking "New debug session".<br>If the module does not currently exist on the system, you will need to upload a zipped file with an xml file specifying the dataset configurations. |

### Value

This function doesn't return any objects but creates "swsContext" objects, such as swsContext.datasets, swsContexts.token, etc.

---

## `KeyDefinition-class` *Definition for KeyDefinition Class*

---

### Description

Definition for KeyDefinition Class

### Arguments

| | |
|---|---|
| `code` | A character value holding the name of the dimension in the system (for example, "geographicAreaM49"). |
| `description` | A character value holding the description of the dimension. |
| `type` | A character value. Valid values are "normal", "measurementUnit", and "time". |

## `MappingTableKey-class` *Mapping Table Key*

### Description

Mapping Table Key

### Arguments

`mappingTable`

`dimensions`

## `Metadata-class` *Metadata Class Definition*

### Description

This object represents the metadata associated to the block of data specified by the keys held by the "selection" slot for BlockMetaData.

### Arguments

| | |
|---|---|
| `code` | A character value identifying the metadata type represented (for instance ACCURACY or GENERAL). |
| `description` | A character value of the description of the metadata type. |
| `language` | Should be one of the following character values: • "ar": Arabic • "en": English • "es": Spanish • "fr": French • "ru": Russian • "zh": Chinese |
| `elements` | A list of objects of class MetadataElement. |

## `MetadataElement-class` *MetadataElement Class Definition*

### Description

MetadataElement Class Definition

### Arguments

| | |
|---|---|
| `code` | A character value specifying the metadata element type (such as "SAMPLING_ERROR" or "COMMENT"). |
| `description` | A character value holding the extended description of the metadata element. |
| `value` | The character value associated to the metadata element. |

---

`NullToNa` *Null to NA*

---

## Description

Null to NA

## Usage

```
NullToNa(nullList)
```

## Arguments

`nullList`

---

`Pivoting-class` *Pivoting Class Definition*

---

## Description

This class represents a single dimension used for pivoting specification. It is used in data retrieval methods to indicate the requested data pivoting.

## Arguments

| | |
|---|---|
| `code` | A character value containing the descriptive string of the referred dimension (i.e. "geographicAreaM49"). |
| `ascending` | Logical, indicates the sort direction to be applied to the specified dimension. |

## Details

In order to properly specify a valid pivoting, a vector of Pivoting objects has to be passed to the data retrieval code. The vector must include all the dimensions that are defined for the target dataset. The sequence of the Pivoting objects describe the requested order of extraction of the corresponding dimensions and, in case of denormalized extractions, it puts on the columns of the returned data.table the values corresponding to the last Pivoting dimension.

## Value

An object of class Pivoting

## Examples

```
pivot1 = Pivoting(code = "geographicAreaM49", ascending = TRUE)
pivot2 = Pivoting(code = "timePointYears", ascending = FALSE)
pivot3 = Pivoting(code = "measuredElement", ascending = FALSE)
pivot4 = Pivoting(code = "measuredItemCPC", ascending = FALSE)

## Not run:
##' # swsContext files are necessary for GetData to run (token may need to be updated)
GetTestEnvironment(
   baseUrl = "https://hqlqasws1.hq.un.fao.org:8181/sws",
   token = "7823c00b-b82e-47bc-8708-1be103ac91e4"
)

# Pull data for one country and all commodities
dim1 = Dimension(name = "geographicAreaM49", keys = c("12", "40"))
dim2 = Dimension(name = "measuredElement", keys = "5510")
dim3 = Dimension(name = "measuredItemCPC", keys = "0111")
dim4 = Dimension(name = "timePointYears", keys = as.character(2000:2013))
key = DatasetKey(domain = "agriculture", dataset = "agriculture",
                 dimensions = list(dim1, dim2, dim3, dim4))

GetData(key, pivoting = c(pivot1, pivot2, pivot3, pivot4))
# Effects are more visible if normalized = FALSE
GetData(key, pivoting = c(pivot1, pivot2, pivot3, pivot4), normalized = F)
GetData(key, pivoting = c(pivot2, pivot3, pivot4, pivot1), normalized = F)

## End(Not run)
```

---

`PostRestCall` *Post Rest Call*

---

## Description

Post Rest Call

## Usage

```
PostRestCall(url, data)
```

## Arguments

`url`

`data`

## `PutRestCall` *Put Rest Call*

### Description

Put Rest Call

### Usage

```
PutRestCall(url, data)
```

### Arguments

| | |
|---|---|
| `url` | |
| `data` | |

## `SaveBlockMetadata` *Save Block Metadata*

### Description

Save Block Metadata

### Usage

```
SaveBlockMetadata(domain, dataset, blockMetadata)
```

### Arguments

| | |
|---|---|
| `domain` | A character value specifying the domain for which the code list is required. |
| `dataset` | A character value specifying the dataset for which the code list is required. |
| `blockMetadata` | An object of type BlockMetadata which should be saved back to the user's session. |

## `SaveData` *Save Data*

### Description

This is the main function used to write data back to a user's session. The data must be in a particular format, and thus it's recommended to first read data via GetData, modify the data.table, and then write it back to the database with this function.

### Usage

```
SaveData(domain, dataset, data, metadata, normalized = TRUE,
  waitMode = "wait", waitTimeout = 600, chunkSize = 50000)
```

### Arguments

| | |
|---|---|
| `domain` | A character value specifying the domain for which the code list is required. |
| `dataset` | A character value specifying the dataset for which the code list is required. |
| `data` | A data.table object containing keys, data and flags. |
| `metadata` | A data.table object containing the metadata. |
| `normalized` | Logical, indicates whether data is in a normalized or denormalized format. |
| `waitMode` | A character string indicating how to behave with respect to the backend. Should be either "wait", "forget", or "synch". "wait" allows R to routinely check for completion of the save, up to pullTimeout seconds. "forget" returns control to R immediately without verifying that the object has been saved (this can be dangerous if the object is accessed later). "synch" gives the writing process a short time period, and returns an error if the write fails. |
| `pullTimeout` | An integer giving the number of seconds to allow for the write process to the database before returning an error. |

### Details

Also, this function will throw an error if you attempt to write to invalid data. For example, some countries have restrictions on which time period they have data (as they recently became a country, or are no longer a country). See ?GetCodeList.

### Value

No object is returned, as this function just writes data to the SWS.

## `SaveValidation` *Save Validation*

### Description

Save Validation

### Usage

```
SaveValidation(domain, dataset, validation)
```

### Arguments

| | |
|---|---|
| `domain` | A character value specifying the domain for which the code list is required. |
| `dataset` | A character value specifying the dataset for which the code list is required. |
| `validation` | A data.table object containing keys and validation data. |

## SetTableData *SetTableData*

### Description

This function allows altering the content of a table for a specified schema.

### Usage

```
SetTableData(schemaName, tableName, data, replace = FALSE, purge = FALSE,
  purgeFilter = NULL)
```

### Arguments

| | |
|---|---|
| `schemaName` | A character value containing the name of the schema to be accessed for the query. |
| `tableName` | A character value containing the name of the table to be read by the query. |
| `data` | A non empty data.table containing records with values to be appended/edited on the table. No need to put all the columns that are in the DB, BUT at least the column/s that compose the primary key (if any). If no primary key is set for the table, all the records will be appended as they are |
| `replace` | Logical, if TRUE, then for a row that exists in the data.table and a corresponding record is fuond in the DB, with same key, the record is UPDATED. If FALSE, it is skipped. |
| `purge` | Logical. If true, before any inserts/updates, rows are deleted from the database table; which rows depends on the purgeFilter parameter (see below). |
| `purgeFilter` | Can be specified only for purge=TRUE. This allows the user to specify a filter for the deletion in the form of an SQL WHERE clause. Specifying purge=TRUE and purgeFilter=NULL will remove all rows from the table before the update. |

### Details

The system checks if the data passed as data.table are either conforming to the expected column type to which any cell refers, or can be parsed (e.g. a string containing "1" to the number 1, or a string containing "01-01-2001" to the corresponding date by assuming the format "dd-MM-yyyy"). If the conversion fails, or any other SQL error is arisen by the underlying DB (e.g. a column constraint for NOT NULL), the function aborts (using stop()) with the error message being placed on the R error message queue. If otherwise the function succeed, statistics on the updates are returned, indicating how many records have been:

- deleted (if purge was TRUE)
- updated (if replace was TRUE and rec found by key)
- inserted (if rec not found by key)
- skipped (if replace was FALSE and rec found by key)

## swsContext *swsContext*

### Description

These objects specify the location of important files for data access.

### Usage

```
swsContext.clientCertificate
```

### Format

```
 chr "~/.R/client.crt"
```

### Details

Most of these objects are created by a call to GetTestEnvironment.

# Index