Package 'RWDataPlyr'

August 16, 2018

Version 0.6.2

Title Read and Manipulate Data from 'RiverWare'

Description A tool to read and manipulate data generated from 'RiverWare'(TM) <http://www.riverware.org/> simulations. 'RiverWare' and 'RiverSMART' generate data in ``rdf'', ``csv'', and ``nc'' format. This package provides an interface to read, aggregate, and summarize data from one or more simulations in a 'dplyr' pipeline.

URL https://github.com/BoulderCodeHub/RWDataPlyr

BugReports https://github.com/BoulderCodeHub/RWDataPlyr/issues

Depends R (>= 3.3.0)

Imports data.table (>= 1.10.0), dplyr (>= 0.7.0), methods, stats, tibble, tidyr, tools, utils, feather, xts, zoo, Rcpp

Suggests bookdown, knitr, rmarkdown, testthat

License CC0

Copyright This software is in the public domain because it contains materials that originally came from the U.S. Bureau of Reclamation, an agency of the U.S. Department of Interior.

LazyData true

VignetteBuilder knitr

RoxygenNote 6.0.1

LinkingTo Rcpp

NeedsCompilation yes

Author Alan Butler [aut, cre], Cameron Bracken [aut]

Maintainer Alan Butler <rabutler@usbr.gov>

Repository CRAN

Date/Publication 2018-08-16 11:40:10 UTC

R topics documented:

`as_rwd_agg` *Coerce lists, matrices, and data.frames to RiverWare data aggregators*

Description

S3 generic for coercing from lists, matrices, and data.frames to rwd_agg objects.

Usage

```
as_rwd_agg(x, ...)

## S3 method for class 'data.frame'
as_rwd_agg(x, ...)

## S3 method for class 'list'
as_rwd_agg(x, ...)
```

```
## S3 method for class 'matrix'
as_rwd_agg(x, ...)

## Default S3 method:
as_rwd_agg(x, ...)
```

Arguments

x	A list. Each element of the list must have the same length.
...	Other arguments passed on to individual methods.

---

createSlotAggList *Creates a list for use by* getDataForAllScens.

Description

Deprecated: please use slot_agg_list() instead, which returns the same list, but now as a "slot_agg_list" object.

Usage

```
createSlotAggList(iData)
```

Arguments

iData	Either an N x 4 character matrix or a character with an absolute or relative path to a csv file.

---

getDataForAllScens *Get and aggregate data from an rdf file(s)*

Description

getDataForAllScens() gets slot data from multiple rdf files and/or multiple scenarios, aggregates it, and saves it as a data.frame. The slot data can be aggregated in multiple ways (see slot_agg_list).

Usage

```
getDataForAllScens(scenFolders, scenNames, slotAggList, scenPath,
  oFile = NULL, retFile = NULL, findAllSlots = TRUE)
```

Arguments

scenFolders	A string vector containing the folder names (scenarios) that the rdf files are saved in.
scenNames	A string vector containing the scenario names. This should be the same length as scenFolders. The scenario names are used as attributes to the data in the Scenario column.
slotAggList	The slot aggregation list. Either an object of class slot_agg_list or a "special" list with the keyword "all". If, it is a slot_agg_list, see that documentation for how to control the aggregation methods used in this function. If all of the slots in an entire rdf are desired, use a list of lists with each entry containing an rdf file and the keyword "all" for the slots, e.g., list(list(rdf = 'KeySlots.rdf',slots = 'all')). If this option is used, the function will return raw monthly, or annual data, i.e., no aggregation methods will be applied to the data in the rdf file.
scenPath	An absolute or relative path to the folder containing scenFolders.
oFile	If not NULL, then an absolute or relative path with the file name of the location the table will be saved to. Valid file types are .csv, .txt, or .feather.
retFile	Deprecated. Data are always returned invisibly.
findAllSlots	Boolean; if TRUE (default), then the function will abort if it cannot find a particular slot. If FALSE, then the function will continue, even if a slot cannot be found. If a slot is not found, then the function will return -99 for the Trace, Year, and Value.

Value

A data.frame returned invisibly.

See Also

slot_agg_list()

Examples

```
# get a specified set of slots and apply some aggregation method to them
# get the data from two scenarios
scenFolders <- c('ISM1988_2014,2007Dems,IG,Most',
  'ISM1988_2014,2007Dems,IG,2002')
# slotAggTable.csv lists the slots to obtain, and the aggregation method to
# apply to them
slotAggList <- slot_agg_list(
  system.file('extdata','SlotAggTable.csv',package = 'RWDataPlyr')
)
scenPath <- system.file('extdata','Scenario/',package = 'RWDataPlyr')
# expect Deprecated warning
testthat::expect_warning(
  keyData <- getDataForAllScens(
    scenFolders,
    scenNames = scenFolders,
    slotAggList = slotAggList,
```

```
    scenPath = scenPath
  )
)

# get all of the data from the KeySlots rdf file
scenFolders <- scenFolders[1] # only one scenario
slotAggList <- list(list(rdf = 'KeySlots.rdf', slots = 'all'))
# will return monthly data for all slots in KeySlots.rdf
# expect Deprecated warning
testthat::expect_warning(
  allData <- getDataForAllScens(
    scenFolders,
    scenNames = scenFolders,
    slotAggList = slotAggList,
    scenPath = scenPath
  )
)
```

`is_rdf` *Test if the object is an rdf*

Description

Test if the object is an rdf

Usage

```
is_rdf(x)

is.rdf(x)
```

Arguments

`x`	An object

Value

`TRUE` if the object inherits from the `rdf` class.

is_rwd_agg *Test if the object is a rwd_agg*

Description

Test if the object is a rwd_agg

Usage

```
is_rwd_agg(x)

is.rwd_agg(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the rwd_agg class.

is_slot_agg_list *Test if the object is a slot_agg_list*

Description

Test if the object is a slot_agg_list

Usage

```
is_slot_agg_list(x)

is.slot_agg_list(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the slot_agg_list class.

keyRdf *Example rdf file with monthly data.*

Description

An example of an rdf file that has already been read into R via `read.rdf()`. This example contains 39 slots, at the monthly timestep for 11 years and 25 runs. Slots include pool elevation, flow, and flags. Use this with `rdf_slot_names()` or `rdf_get_slot()` to use the data.

Usage

```
keyRdf
```

Format

A multi level list. `keyRdf$meta` provides a description of the RiverWare run used to generate this data.

Source

Bureau of Reclamation, 2016

rbind.rwd_agg *Combine RiverWare data aggregators*

Description

Take a sequence of `rwd_agg` arguments (or vector, matrix, or data.frames) and combine by rows. If the objects are not `rwd_agg` objects they will be combined through the default `rbind()` method, and then verified that they meet all constraints to be a valid `rwd_agg` object. `cbind()` will fail for `rwd_agg` objects.

Usage

```
## S3 method for class 'rwd_agg'
rbind(..., deparse.level = 1)

## S3 method for class 'rwd_agg'
cbind(..., deparse.level = 1)
```

Arguments

`...`	(generalized) vectors or matrices. These can be given as named arguments. Other R objects may be coerced as appropriate, or S4 methods may be used: see sections 'Details' and 'Value'. (For the `"data.frame"` method of `cbind` these can be further arguments to `data.frame` such as `stringsAsFactors`.)
`deparse.level`	integer controlling the construction of labels in the case of non-matrix-like arguments (for the default method): `deparse.level = 0` constructs no labels; the default, `deparse.level = 1 or 2` constructs labels from the argument names, see the 'Value' section below.

Examples

```
ra1 <- rwd_agg(data.frame(
  file = "KeySlots.rdf",
  slot = "Powell.Pool Elevation",
  period = "wy",
  summary = "min",
  eval = "<",
  t_s = 3550,
  variable = "powellLt3550",
  stringsAsFactors = FALSE
))

ra2 <- rwd_agg(read.csv(
  system.file(
    "extdata/rwd_agg_files/passing_aggs.csv",
    package = "RWDataPlyr"
  ),
  stringsAsFactors = FALSE
))

rbind(ra1, ra2)

## Not run:
# will fail because you cannot have repeating variable names
rbind(ra1, ra1)

# will also fail
cbind(ra1, ra2)

## End(Not run)
```

`rdf_aggregate` *Aggregate RiverWare output for one or more scenarios*

Description

Process the user specified rwd_agg object for one or more scenarios to aggregate and summarize RiverWare output data.

Usage

```
rdf_aggregate(agg, rdf_dir = ".", scenario = NULL, keep_cols = FALSE,
  nans_are = "0", find_all_slots = TRUE, cpp = TRUE, verbose = TRUE)

rw_scen_aggregate(scenarios, agg, scen_dir = ".", nans_are = "0",
  keep_cols = FALSE, file = NULL, scen_names = NULL,
  find_all_slots = TRUE, cpp = TRUE, verbose = TRUE)
```

Arguments

agg	A rwd_agg object specifying the rdfs, slots, and aggregation methods to use.
rdf_dir	The top level directory that contains the rdf files. See **Directory Structure**.
scenario	An optional parameter, that if it is not NULL or NA (default) will be added to the tibble as another variable. Coerced to a character if it is not already a character.
keep_cols	Either boolean, or a character vector of column names to keep in the returned tibble. The values of keep_cols work as follows: • FALSE (default) only includes the defaults columns: TraceNumber, ObjectSlot, and Value. Scenario is also returned if scenario is specified. • TRUE, all columns are returned. • A character vector, e.g., c("ObjectName", "Units"), allows the user to include other columns that are not always required, in addition to the "default" set of columns. If any of the values in keep_cols are not found, a warning will post, but all other columns will be returned.
nans_are	Either "0" or "error". If "0", then NaNs in the rwtbl are treated as 0s. If "error", then any NaNs will cause an error in this function.
find_all_slots	Boolean; if TRUE (default), then the function will abort if it cannot find a particular slot. If FALSE, then the function will continue, even if a slot cannot be found. If a slot is not found, then the function will return -99 for the Trace, and NaN for Year, and Value.
cpp	Boolean; if TRUE (default), then use rdf_to_rwtbl2, which relies on C++, otherwise, use original rdf_to_rwtbl function.
verbose	Boolean; if TRUE (default), then print out status of processing the scenario(s) and the slots in each scenario.
scenarios	A character vector of scenario foders. This is usually a vector of folder names, where each folder name contains one scenario worth of data. scenarios can be named or unnamed. The names are used as the scenario name in the returned tbl_df. Scenario names can also be specified through the scen_names argument. If scen_names is specified, scenarios should not already have names. If scen_names is not specified and, scenarios is not already named, then the scenario folders will also be used as the scenario names. See **Directory Structure**.

scen_dir	File path to the directory that contains the scenario folders. **Directory Structure**.
file	Optionally save the tbl_df of aggregated scenario data as a .txt, .csv, or .feather file. If file is specified, then the data are saved in the specified output format.
scen_names	An alternative way to specify scenario names.

Details

rdf_aggregate() aggregates a single scenario of data by processing a rwd_agg object.

In both cases, the user specifies the rwd_agg, which determines the slots that are aggregated, and how they are aggregated. See rwd_agg for more details on how it should be specified.

See the **Directory Structure** section for how to specify scenarios, scen_dir, and rdf_dir.

rw_scen_aggregate() aggregates multiple scenarios of data. It processes the rwd_agg object (agg) for each single scenario, and then binds all of the individual scenario data together into a single tbl_df.

Value

A tbl_df containing all aggregated and summarized data for all of the specified scenarios.

Directory Structure

RiverWare and RiverSMART typically write data into an expected directory structure. The below shows an example directory structure and corresponding variable names for rw_scen_aggregate() and rdf_aggregate(). (In the example below, C:/user/crss/CRSS.Jan2017/Scenario is the more complete directory setup for the data included in system.file("extdata/Scenario/").)

```
C:/user/crss
|
|- CRSS.Jan2017
|    - model
|    - ruleset
|    - Scenario
|         - ISM1988_2014,2007Dems,IG,Most
|         - ISM1988_2014,2007Dems,IG,2002
|    - ...
|- CRSS.Jan2018
|    - model
|    - ... (same general setup as CRSS.Jan2017)
```

To get one scenario's data, rdf_aggregate() can be called with rdf_dir set to "C:/user/crss/CRSS.Jan2017/Scenario/ISM19
(scenario can optionally be specified to git a scenario name.)

To aggregate multiple scenarios of data together, rw_scen_aggregate() should be called with scen_dir set to "C:/user/CRSS/CRSS.Jan2017/Scenario" and scenarios set to c("ISM1988_2014,2007Dems,IG,Most", "
(Optionally, scenarios can be named, or scen_names specified to use scenario names that are different from the above scenario folders.)

Finally, to aggregate scenario data from both CRSS.Jan2017 and CRSS.Jan2018, `rw_scen_aggregate()` should be called with `scen_dir` set to "C:/users/crss/". `scenarios` can then be set to `c("CRSS.Jan2017/Scenario/ISM1988_` assuming the same scenario exists in both folders. In this case it is advisable to also specify `scen_names` or name `scenarios`.

Examples

```
# rdf_aggregate() ----------

rdfPath <- system.file(
  "extdata/Scenario/ISM1988_2014,2007Dems,IG,Most",
  package = "RWDataPlyr"
)

rwa <- rwd_agg(read.csv(
  system.file(
    "extdata/rwd_agg_files/passing_aggs.csv",
    package = "RWDataPlyr"
  ),
 stringsAsFactors = FALSE
))

x <- rdf_aggregate(rwa[1,], rdf_dir = rdfPath, scenario = "Most")

# rw_scen_aggregate() ----------

scens <- c("ISM1988_2014,2007Dems,IG,2002", "ISM1988_2014,2007Dems,IG,Most")
scenNames <- c("2002", "Most")
namedScens <- scens
names(namedScens) <- scenNames

scenPath <- system.file("extdata/Scenario", package = "RWDataPlyr")

rwa <- rwd_agg(read.csv(
  system.file(
    "extdata/rwd_agg_files/passing_aggs.csv",
    package = "RWDataPlyr"
  ),
 stringsAsFactors = FALSE
))

x <- rw_scen_aggregate(namedScens, agg = rwa[1,], scen_dir = scenPath)

# y will be identical to x

y <- rw_scen_aggregate(
  scens,
  agg = rwa[1,],
  scen_dir = scenPath,
  scen_names = scenNames
)
```

```
identical(x, y) # is TRUE
```

`rdf_get_slot` *Get a slot out of an rdf object*

Description

`rdf_get_slot()` gets a slot from an `rdf` object and creates a matrix with rows indexing through time and columns indexing over traces.

Usage

```
rdf_get_slot(rdf, slot)

rdfSlotToMatrix(rdf, slot)
```

Arguments

`rdf`	An `rdf` object.
`slot`	Character slot name that exists in the `rdf`.

Value

A matrix with traces as columns and timesteps as rows.

Functions

- `rdfSlotToMatrix`: Deprecated version of `rdf_get_slot()`

Examples

```
pe <- rdf_get_slot(keyRdf, "Mead.Pool Elevation")
```

rdf_get_timespan *Returns the simulation timespan from an rdf*

Description

`rdf_get_timespan()` gets the simulation timespan from an `rdf` object.

Usage

```
rdf_get_timespan(rdf)
```

Arguments

`rdf`	An `rdf` object (likely from `read.rdf()`).

Value

A named character vector with two elements. The first element, named `"start"`, includes the start date of the simulation. The second element, named `"end"`, includes the end date of the simulation.

Examples

```
rdf_get_timespan(keyRdf)
```

rdf_slot_names *Returns all slots contained in an rdf file.*

Description

`rdf_slot_names()` returns a character vector of all slots contained within an rdf object.

Usage

```
rdf_slot_names(rdf)

getSlotsInRdf(rdf)
```

Arguments

`rdf`	An `rdf` object.

Value

A character vector.

Functions

- `getSlotsInRdf`: Deprecated version of `rdf_slot_names()`

See Also

`read.rdf()`

Examples

```
rdf_slot_names(keyRdf)
```

`rdf_to_rwtbl` *Convert an rdf to a tibble*

Description

`rdf_to_rwtbl()` converts an rdf list to a tibble.

`rdf_to_rwtbl2()` converts an rdf file into a tibble, but is faster than `rdf_to_rwtbl()` since it uses c++. It also reads the rdf file in, while `rdf_to_rwtbl()` needs an `rdf` object.

Usage

```
rdf_to_rwtbl(rdf, scenario = NULL, keep_cols = FALSE, add_ym = TRUE)

rdf_to_rwtbl2(file, scenario = NA_character_, keep_cols = FALSE,
  add_ym = TRUE)
```

Arguments

`rdf`	An rdf object (from `read_rdf()`).
`scenario`	An optional parameter, that if it is not `NULL` or `NA` (default) will be added to the tibble as another variable. Coerced to a character if it is not already a character.
`keep_cols`	Either boolean, or a character vector of column names to keep in the returned tibble. The values of `keep_cols` work as follows: • `FALSE` (default) only includes the defaults columns: `Timestep`, `TraceNumber`, `ObjectSlot`, and `Value`. `Scenario` is also returned if `scenario` is specified. • `TRUE`, all columns are returned. • A character vector, e.g., `c("ObjectName", "Units")`, allows the user to include other columns that are not always required, in addition to the "default" set of columns. If any of the values in `keep_cols` are not found, a warning will post, but all other columns will be returned.
`add_ym`	Boolean that controls whether or not `Year` and `Month` columns are added to the returned tibble. If `TRUE` (default), they will be added, and if `FALSE` they will not be added. They are constructed from the dates in the `Timestep` column.
`file`	The relative or absolute rdf filename.

Details

The rdf object is converted to a data frame, and then converted to a `tibble::tibble()`. All of the `meta` entries in the rdf object are stored as attributes in the returned tibble. These attributes are: `mrm_config_name`, `owner`, `description`, `create_date`, and `n_traces`.

If the rdf contains a scalar slot(s), the scalar slot value(s) will be repeated for every timestep.

Value

A tbl_df with additional attributes from the rdf object.

See Also

`read_rdf()`

Examples

```
rdftbl <- rdf_to_rwtbl(keyRdf)
# same as previous, except you do not want "Year" and "Month" columns
rdftbl <- rdf_to_rwtbl(keyRdf, add_ym = FALSE)
# but you do want to keep the object name seperately:
rdftbl <- rdf_to_rwtbl(keyRdf, add_ym = FALSE, keep_cols = "Object")
rdftbl <- rdf_to_rwtbl(sysRdf, scenario = "ISM1988_2014,2007Dems,IG,2002")

# rdf_to_rwtbl2 wants a file path instead of an rdf object
rdfPath <- system.file(
  "extdata/Scenario/ISM1988_2014,2007Dems,IG,Most/KeySlots.rdf",
  package = "RWDataPlyr"
)
rdftbl <- rdf_to_rwtbl2(rdfPath)
```

`read.rdf` *Read an rdf file into R.*

Description

`read.rdf()` reads an rdf file into R and formats it as a multi-level list containing all of the metadata included in the rdf file. rdf files are generated by RiverWare and are documented in the RiverWare documentation.

Usage

```
read.rdf(iFile, rdf = TRUE)

read.rdf2(iFile)

read_rdf(iFile, rdf = TRUE)
```

Arguments

iFile	The input rdf file that will be read into R.
rdf	Boolean; if TRUE, then an rdf object is returned. If FALSE, then a character vector is returned.

Details

read.rdf()uses data.table::fread() to read in the file, which provides performance benefits as compared to earlier versions of the function.

read.rdf2() is deprecated and will be removed in a future release.

Value

An rdf object or character vector.

Functions

- read.rdf2: Deprecated version of read.rdf()

Examples

```
zz <- read_rdf(system.file(
  'extdata/Scenario/ISM1988_2014,2007Dems,IG,Most',
  "KeySlots.rdf",
  package = "RWDataPlyr"
))
```

---

read_rwd_agg *Read in a rwd_agg file*

Description

read_rwd_agg() reads in a csv file and creates a rwd_agg object. Therefore, if the csv file is not properly formatted to contain the correct information for a rwd_agg object, it will fail. rwd_agg_template() will create a blank template file for the user to fill in, which has the correct headers.

Usage

```
read_rwd_agg(file)
```

Arguments

file	The csv file to be read in and converted

See Also

rwd_agg_template()

Examples

```
read_rwd_agg(
  system.file(
    "extdata/rwd_agg_files/passing_aggs.csv",
    package = "RWDataPlyr"
  )
)
```

`read_rw_csv` *Read RiverWare/RiverSMART produced csv files*

Description

`read_rw_csv()` reads in a CSV file created from RiverWare. If the CSV file does not contain column names that RiverWare always uses (see Details), then it assumes that the CSV file was not created from RiverWare and throws an error. It also removes spaces from the column names, and adjusts the `Object.Slot` and `Slot Value` columns to be `ObjectSlot` and `Value`, respectively.

Usage

```
read_rw_csv(file)
```

Arguments

`file`	The name of the file which the data are to be read from. Either an absolute or relative path.

Details

The required column names are: `Run Number`, `Trace Number`, `Object.Slot`, `Timestep`, `Slot Value`. See the CSV output section of the RiverWare documentation for more information on the other optional column names.

This function uses `data.table::fread()` to read in the CSV file, and forces it to expect a CSV file, expect headers, and return `data.frame`.

Value

A tibble (data frame) containing the data in the csv.

See Also

`read.rdf()`

Examples

```
zz <- read_rw_csv(system.file(
  "extdata/Scenario/ISM1988_2014,2007Dems,IG,Most",
  "KeySlots.csv",
  package = "RWDataPlyr"
))
```

`rwd_agg` *Class to specify how to aggregate RiverWare data*

Description

`rwd_agg()` creates a RiverWare data aggregator (`rwd_agg`) object, which lets users specify how specific RiverWare slots should be aggregated.

Usage

```
rwd_agg(x = NULL, rdfs = NULL)
```

Arguments

`x`	A data.frame with required column names and valid entries; see *Details* and *Aggregation Specification* sections.
`rdfs`	A vector of rdf names; see *Details* and *Aggregation Specification* sections.

Details

`rwd_agg` objects can be created in three ways:

1. By providing a data.frame, with the following expected columns `file`, `slot`, `period`, `summary`, `eval`, `t_s`, and `variable`. Each row in the data.frame should include all of the information for how each individual slot will be aggregated. See the *Aggregation Specification* section below for details on how to specify each column.
2. By providing a vector of rdf files. If specified in this manor, all of the slots in each rdf file will be read in to a `rwtbl`, but will not be aggregated/summarized.

   In this case, the `variable` names are automatically constructed from the `ObjectSlot` names. The variable names are constructed as the all lower case version of the object_slot name. If the slot name is "Pool Elevation", it is shortened to "pe", otherwise the full object and slot name are used. If there are any spaces, they are replaced with underscores.
3. By reading in a csv file with `read_rwd_agg()`. This csv file must have the correct column names and meet other requirements described below. To ensure it has the correct column names, `rwd_agg_template()` can be used to create a blank csv file for the user to fill in.

Aggregation Specification

In order to specify how each slot should be aggregated, each column should include specific keywords, which are described below. It is up to the user to specify which rdf file contains each slot. In a general case, the user specifies the `slot` that is found in a specific rdf file (`file`). A `summary` function is applied to a subset `period` of the `slot`, and then compared (`eval`) to a threshold (`t_s`) and saved as the `variable`.

- *file:* specifies the rdf file that contains the slot.
- *slot:* the full RiverWare slot name, i.e., "Object.Slot".
- *period:* the period that the slot should be summarized over. This should either be a function name, a full month name (found in month.name), or the keyword "asis".
  - *function name:* Specifying a function name allows for pre-specified or custom functions to group together several months in the *period*. This package provides the following functions: `cy()`, `wy()`, `eocy()`, and `eowy()`. `cy()` indicates the data will be summarized over the calendar year, i.e., January - December, while `wy()` summarizes over the water year, i.e., October - September. `eocy()` selects the end of the calendar year, and `eowy()` selects the end of the water year. When specified in the `slot_agg` object, leave off the parenthesis, i.e., only specify the function name. If `wy()` is specified, the function will remove data for any water years that have less than 7 months of data. This "tolerance" is specified by the `rwdataplyr.wy_month_tol` option, and can be modified by updating this option to another number. For standard monthly data that starts in January and ends in December, this results in keeping the first water year, since it includes 9 months of data, and removing the final water year, since it includes only three months of data. Setting this option to 0 will result in keeping any water year data that has at least one month of data; setting this option to 11, ensures that there must be a full water year of data for that year to be kept.
    This can also be a user specified custom function; see the *Custom Period Functions* section for details on constructing the custom functions.
  - *full month name:* When the full month name is specified, data will be filtered to only include data for that particular month. To select multiple months of data, use a function as described above. If the month specified is not found in month.name, an error will occur.
  - *asis:* If the keyword "asis" is specified, the data is returned for its native timestep, i.e, monthly data will return monthly data and annual data will return annual.
- *summary:* the summary function that should be applied to the period specified as a function name, or `NA`. If the `period` specified is "asis" or returns only one month, e.g., `eocy()`, then the summary should be `NA`. The summary function should only return one value; for that reason, most of the `Summary` S4groupGenerics work. Notably, `range()` will not since it returns two values. There is no reason that a custom function will not work here, but it has not been tested.
- *eval:* the comparison operator to use (see the `Compare` S4groupGenerics). If no comparison is desired, then `NA` should be used. If `eval` is specified the value returned from applying the `summary` to the `period` will be compared to the threshold specified by `t_s`. The results of the comparison are returned as 0 and 1 instead of `TRUE` and `FALSE`.
- *t_s:* either the threshold to be compared to if `eval` is not `NA` or a value to scale the result by, e.g,. 0.001 to convert from acre-ft to thousand acre-ft. `NA` can also be specified to not scale the data.

- *variable:* the variable name that will be used to identify the results of applying the period, summary, comparison/scaling to. All variable names should be unique.

For example, to determine if the minimum water year elevation at Lake Powell is below elevation 3550 feet, the following would be specified:

```
data.frame(
  file = "KeySlots.rdf",
  slot = "Powell.Pool Elevation",
  period = "wy",
  summary = "min",
  eval = "<",
  t_s = 3550,
  variable = "powellLt3550",
  stringsAsFactors = FALSE
)
```

Custom Period Functions

Users can specify custom period functions to make it easier to group months together in custom ways. For example a function could return all of the summer months, or the more complicated case groups months across different calendar years together. In fact, `wy()` is an example of a function that does this; another example might be grouping December - February together for winter months.

The custom period function should return a list with three elements:

- `fun` - a function that will modify a rwtbl and properly determine the new `Years` based on the custom period.
- `filter_months` - the months that should be grouped together.
- `group_tbl` - how to group the returned rwtbl; likely either `c("Year")` or `c("Year", "Month")`

See the "RWDataPlyr Workflow" vignette for example implementations of both the summer and winter custom functions described above.

See Also

`rwd_agg_template()`, `read_rwd_agg()`

Examples

```
# determine if Powell's minimum water year elevation is < 3550'
rwd_agg(
  data.frame(
    file = "KeySlots.rdf",
    slot = "Powell.Pool Elevation",
    period = "wy",
    summary = "min",
    eval = "<",
    t_s = 3550,
    variable = "powellLt3550",
    stringsAsFactors = FALSE
```

```
  )
)

# get all the monthly slots in KeySlots.rdf
rwd_agg(rdfs = "KeySlots.rdf")
```

`rwd_agg_template` *Create a* `rwd_agg` *template*

Description

`rwd_agg_template()` creates a template csv file to use to create a RiverWare data aggregator (rwd_agg).

Usage

```
rwd_agg_template(file, path = ".", examples = FALSE)
```

Arguments

`file`	The file name to use for the template
`path`	The path to create the template at
`examples`	Boolean; When `FALSE` (default), the template includes only headers. When `TRUE`, the template will include several examples of specifying how each slot should be summarized.

See Also

`read_rwd_agg()`

Examples

```
rwd_agg_template(file = "rwa_slots.csv", path = tempdir())
rwd_agg_template(file = "rwa_slots.csv", path = tempdir(), examples = TRUE)
```

`rwslot_annual_sum` *Simple aggregation functions for monthly matrix data*

Description

`rwslot_annual_sum()` takes a matrix containing monthly data (months by traces), and sums the monthly data into annual data. Returns a years by traces matrix.

`rwslot_annual_min()` finds the minimum annual value for all years and traces.

`rwslot_annual_max()` finds the maximum annual value for all years and traces.

`rwslot_fwaac()` calculates the flow-weighted average annual concentration (fwaac). Given mass and flow at the monthly basis, the flow-weighted average annual concentration is computed. `mass` and `flow` should be monthly data. `rwslot_fwaac()` expects flow to be in acre-ft/month and mass to be in tons; however, there are no checks to ensure this is true. Return value will be in mg/L.

Usage

```
rwslot_annual_sum(rwslot, multFactor = 1)

sumMonth2Annual(rwslot, multFactor = 1)

rwslot_annual_min(rwslot)

getMinAnnValue(rwslot)

rwslot_annual_max(rwslot)

getMaxAnnValue(rwslot)

rwslot_fwaac(mass, flow)
```

Arguments

`rwslot`	A matrix (months by traces) such as that returned by `rdf_get_slot()`. Function will error if the `rwslot` does not contain "regular" monthly data, i.e., the data must start in January and end in December, or start in October and end in September (water year).
`multFactor`	A factor the annual sum will be multiplied by. Can be used to convert from flow to volume, or to scale all results in another manor.
`mass`	A matrix (months by traces), such as that returned by `rdf_get_slot()`, of mass in tons.
`flow`	A matrix (months by traces), such as that returned by `rdf_get_slot()`, of flow in acre-ft/month.

Value

`rwslot_annual_sum()` returns the annual sum as a matrix (years by traces).

`rwslot_annual_min()` returns a matrix (years by traces) with the maximum annual value for each year and trace.

`rwslot_annual_max()` returns a matrix (years by traces) with the maximum annual value for each year and trace.

`rwslot_fwaac()` returns a matrix of yearly data of the flow-weighted average annual concentration. Units are mg/L.

Functions

- `sumMonth2Annual`: Deprecated version of `rwslot_annual_sum()`.

See Also

`rdf_get_slot()`

Examples

```
zz <- rdf_get_slot(keyRdf, 'Powell.Outflow')

# returns in original units, e.g., acre-ft
annualTotVal <- rwslot_annual_sum(zz)

# returns in scaled units, e.g., kaf
annualTotVal <- rwslot_annual_sum(zz, 0.001)

pe <- rdf_get_slot(keyRdf,'Mead.Pool Elevation')
peMax <- rwslot_annual_min(pe)

pe <- rdf_get_slot(keyRdf,'Mead.Pool Elevation')
peMax <- rwslot_annual_max(pe)

flow <- rdf_get_slot(keyRdf,'Powell.Outflow')
# make up mass, since it's not stored in the example data
rr <- matrix(
  rnorm((nrow(flow) * ncol(flow)), mean = 1000, sd = 200),
  nrow = nrow(flow),
  ncol = ncol(flow)
)
mass <- flow / 1000000 * rr^2 - rr + 1500
fwaac <- rwslot_fwaac(mass, flow)
```

rwtbl_get_scen_folder *Map a scenario name to the original scenario folder*

Description

`rwtbl_get_scen_folder()` provides the original file path to the scenario folder for the specified scenario name(s) (`scenarios`). If `scenarios` are not found in `rwtblsmmry`, a warning message is posted.

Usage

```
rwtbl_get_scen_folder(rwtblsmmry, scenarios)
```

Arguments

`rwtblsmmry`	A tbl_df of summarized RiverWare data; likely output from `rw_scen_aggregate()`.
`scenarios`	A vector of scenario names to map to scenario folders.

Value

A vector of scenario folders; `character(0)` if none of the `scenarios` are found.

Examples

```
rwtbl_get_scen_folder(scen_data, "Most")
rwtbl_get_scen_folder(scen_data, c("Most", "2002"))
```

rwtbl_slot_names *List the slot names in a tbl_df*

Description

`rwtbl_slot_names()` lists all of the slot names found in a `tbl_df` object containing RiverWare output data.

Usage

```
rwtbl_slot_names(rwtbl)
```

Arguments

`rwtbl`	A `tbl_df` object with RiverWare output. Must contain the `ObjectSlot` column.

Details

Given a tbl_df object that is returned by rdf_to_rwtbl() or read_rw_csv(), return all of the Object.Slot names found in the data frame. These are the unique full slot names found in the ObjectSlot column.

See Also

rdf_to_rwtbl(), read_rw_csv()

Examples

```
rwtbl <- rdf_to_rwtbl(keyRdf)
rwtbl_slot_names(rwtbl)
```

---

rwtbl_var_to_slot *Map a variable name to the RiverWare slot name*

Description

rwtbl_var_to_slot() provides the RiverWare slot name that was used to create the specified variable name (varname). If varname is not found in rwtblsmmry, a warning message is posted.

Usage

```
rwtbl_var_to_slot(rwtblsmmry, varname)
```

Arguments

rwtblsmmry	A tbl_df of summarized RiverWare data; likely output from rw_scen_aggregate().
varname	A vector of variable names to map to slot names.

Value

A character vector of the found slot names. character(0) if no variable names were found.

Examples

```
rwtbl_var_to_slot(scen_data, "peLt1000")
rwtbl_var_to_slot(scen_data, c("peLt1000", "peEocy"))
```

`rw_scen_gen_names` *Create a vector of scenarios from different dimensions*

Description

`rw_scen_gen_names()` creates a vector of full scenario names by combining multiple dimensions together.

Usage

```
rw_scen_gen_names(dim1, dim2, ..., sep = ",")

makeAllScenNames(dim1, dim2, ..., sep = ",")
```

Arguments

`dim1`	A character vector with all of the first dimension's names.
`dim2`	A character vector with all of the second dimension's names.
`...`	As many individual character vectors as necessary for the remaining dimension's names.
`sep`	The character used to separate the different dimension names. Defaults to`","`.

Details

Many RiverWare runs are specified by multiple dimensions (or assumptions), and RiverSMART creates folder names by combining the dimension names together for a full scenario name. `rw_scen_gen_names()` makes it quick to create all of the full scenario names by passing in the names of the individual dimensions and creating all possible combinations of all dimensions.

For example, the RiverWare run might consist of a supply dimension and a demand dimension, each consisting of two scenarios. This would result in four total scenarios.

The function will work with two or more dimensions, as there is no need for this function if there is only one dimension.

Value

A character vector of all possible combinations of the dimensions.

Functions

- `makeAllScenNames`: Deprecated version of `rw_scen_gen_names()`

Examples

```
rw_scen_gen_names("DNF", "CT", c("IG", "NA"), c("MTOM", "24-MS"))
rw_scen_gen_names("DNF", "CT", c("IG", "NA"), sep = "_")
```

scen_data *Example aggregated scenario data*

Description

An example of the tbl_df returned by `rw_scen_aggregate()` containing two scenarios of data.

Usage

```
scen_data
```

Format

An object of class `grouped_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 720 rows and 6 columns.

slot_agg_list *A class to control how RiverWare data are aggregated*

Description

`"slot_agg_list"` is a class that contains a set of RiverWare slots, which rdf file they are found in, and a set of keywords that are used to control how `getDataForAllScens()` aggregates RiverWare data.

Usage

```
slot_agg_list(x)
```

Arguments

x	Either an Nx4 character matrix or a character with an absolute or relative path to a csv file.

Details

The `slot_agg_list` class, contains a list that includes: which rdf file to find each slot in, how to aggregate and process the slot data, and any thresholds or scaling factors. The function can either read in a csv file or start from an N x 4 or N x 5 string matrix (the 5th column is optional).

The csv file and the matrix should be in the form of an Nx4 or Nx5 matrix. Each row is a single slot, aggregation, and threshold combination. If you want to compare a single slot value to multiple thresholds, it needs to have one row for each threshold. The first column is the rdf the slot is found in. The second column is the slot name. The third column is the aggregation method that will be applied to the slot (see below for a list of the aggregation methods). The fourth column is a scaling factor or threshold to compare the slot data to. The fifth column is an optional column; if specified, the 5th column will be used for the variable name for the data.frame created by

getDataForAllScens(). If it is not specified the variable name will be created by concatenating the slot, aggregation method, and threshold/scaling factor using '_' to separate the columns. Below is an example table. All values should be strings except for NA, if specified as a matrix in R.

rdf	Slot	Aggregation Method	Threshold or Scaling Factor	Variable Name (optiona
'KeySlots.rdf'	'Mead.Pool Elevation'	'EOCY'	NA	Mead EOCY Elevation
'KeySlots.rdf'	'Mead.Pool Elevation'	'AnnMinLTE'	'1100'	Mead < 1,100
'KeySlots.rdf'	'Mead.Pool Elevation'	'AnnMinLTE'	'1060'	Mead < 1,060
'Other.rdf'	'Powell.Outflow'	'AnnualSum'	'0.001'	Powell Annual Release

The above table lists each slot, the rdf the slot is saved in, the summary function, the threshold to be used to scale the data by or compare the data to, and an optionally specified variable name. The threshold and scaling factors are described in more detail below. For example, the first row will result in compiling all end-of-December values for Mead's pool elevation. The data will not be scaled, and getDataForAllScens() will look in KeySlots.rdf for the "Mead.Pool Elevation" slot. The second row will find the annual minimum Mead pool elevation and see if it is less than or equal to 1,100' feet in the second line and less than or equal to 1,060' feet in the third row.

To scale the data by a value less than 1, use decimals rather than fractions, as shown in the fourth row. If the Variable Name column was not specified, the variable name for the first row would be `Mead.Pool Elevation_EOCY_1` NA is replaced with a 1 when constructing the variable names.

See the **Aggregation Methods** section for available aggregation methods.

Value

A `slot_agg_list` object.

Aggregation Methods

The available aggregation methods are as follows. The behavior of the "Threshold or scaling factor" are described and a bold **"Threshold"** or **"Scaled"** indicates which is used by the aggregation method. For scaling factors, a value of NA will not scale the data.

'AnnMin' Returns the minimum annual **scaled** value.

'AnnMax' Returns the maximum annual **scaled** value.

'AnnualSum' Returns the annual **scaled** sum.

'AnnMinLTE' Checks to see if the annual minimum value is less than or equal to a **threshold.** Returns 1 if it is less than or equal to the **threshold** and 0 otherwise.

'AnnualRaw' Returns the annual **scaled** data. This aggregation method should only be used if the rdf file contains only annual data. For rdf files that include monthly data and only an annual value is desired, the EOCY aggregation method should be used. This differs from the Monthly aggregation method, only in the timestep naming.

'BOCY' Beginning-of-calendar year values are reported and **scaled**. Any values that are NaNs are changed to 0s.

'EOCY' End-of-calendar year values are reported and **scaled**. Any values that are NaNs are changed to 0s.

'EOCYGTE' Checks to see if the end-of-calendar year values are greater than or equal to a **threshold**. Returns 1 if it is greater than or equal to the **threshold** and 0 otherwise.

'EOCYLTE' Checks to see if the end-of-calendar year values are less than or equal to a **threshold**. Returns 1 if it is less than or equal to the **threshold** and 0 otherwise.

'EOWY' End-of-water year values are reported and **scaled**. Any values that are NaNs are changed to 0s.

'Monthly' Returns the monthly **scaled** data.

'WYMaxLTE' Checks to see if the maximum water year value is less than or equal to a **threshold.** Returns 1 if it is less than or equal to the **threshold** and 0 otherwise. This can be used to determine if an entire water year is below a **threshold**. The water year is defined as October through September of the next year. For the first year, only January through September are evaluated as RiverWare does not typically export pre-simulation data.

'WYMinLTE' Checks to see if the minimum water year value is less than or equal to a **threshold.** Returns 1 if it is less than or equal to the **threshold** and 0 otherwise. The water year is defined as October through September of the next year. For the first year, only January through September are evaluated as RiverWare does not typically export pre-simulation data.

See Also

`getDataForAllScens()`

Examples

```
# read in a csv file that contains the data
slot_agg_list(
  system.file('extdata','SlotAggTable.csv',package = 'RWDataPlyr')
)

# or specify as a matrix
slot_agg_matrix <- matrix(
  c("KeySlots.rdf", "Powell.Outflow", "AnnualSum", ".001", "powellAnnRel",
  "KeySlots.rdf", "Mead.Pool Elevatoin", "AnnMinLTE", "1050", "meadLt1050"),
  nrow = 2,
  byrow = TRUE
)
slot_agg_list(slot_agg_matrix)
```

---

sysRdf *Example rdf file with annual data.*

Description

An example of an rdf file that has already been read into R via `read.rdf()`. This example contains 23 slots, at the annual timestep for 11 years and 25 runs. Slots only include flags. Use this with `rdf_slot_names()` or `rdf_get_slot()` to use the data.

Usage

```
sysRdf
```

Format

A multi level list. `sysRdf$meta` provides a description of the RiverWare run used to generate this data.

Source

Bureau of Reclamation, 2016

`ym_get_wateryear` *Get the water year from a year-month (yearmon) value*

Description

`ym_get_wateryear()` returns the water year (assumed to be October - September) from a zoo::yearmon object.

Usage

```
ym_get_wateryear(ym)

getWYFromYearmon(ym)
```

Arguments

`ym`	An object of class zoo::yearmon, or something that can be successfully converted to zoo::yearmon.

Details

If the argument is not already a yearmon object, it will attempt to convert it to a zoo::yearmon. This may result in unexpected results. For example, the string `"12-1-1906"` can be converted to a zoo::yearmon, however, it will not convert to `"Dec 1906"` as you might desire. It will convert to `"Jan 0012"` since it is not a format expected by `zoo::as.yearmon()`. Therefore, a warning is posted when the function attempts to convert to zoo::yearmon, and it is safer to ensure `ym` is already a zoo::yearmon.

Value

The water year as a numeric.

Examples

```
ym_get_wateryear(zoo::as.yearmon(c("Dec 1906", "Oct 1945", "Jul 1955")))
ym_get_wateryear("2000-11")
```

Index