Package 'crunch'

November 16, 2022

Type Package

Title Crunch.io Data Tools

Description The Crunch.io service <https://crunch.io/> provides a cloud-based data store and analytic engine, as well as an intuitive web interface. Using this package, analysts can interact with and manipulate Crunch datasets from within R. Importantly, this allows technical researchers to collaborate naturally with team members, managers, and clients who prefer a point-and-click interface.

Version 1.30.1

URL https://crunch.io/r/crunch/, https://github.com/Crunch-io/rcrunch

BugReports https://github.com/Crunch-io/rcrunch/issues

License LGPL (>= 3)

Depends R (>= 3.0.0)

Imports abind, crayon, curl, grDevices, httr (>= 1.0.0), httpcache (>= 0.1.4), jsonlite (>= 0.9.15), methods

Suggests base64enc, covr, crunchy, devtools, haven, httptest (>= 3.0.0), knitr, lintr, magrittr, mime, rmarkdown, rstudioapi, spelling, styler, testthat (>= 2.0.0), withr

VignetteBuilder knitr

Language en-US

Encoding UTF-8

RoxygenNote 7.2.2

LazyData true

Collate 'AllClasses.R' 'AllGenerics.R' 'variable-definition.R' 'R-to-variable.R' 'SO-survey.R' 'abstract-categories.R' 'abstract-category.R' 'add-subvariable.R' 'add-variable.R' 'api.R' 'append-dataset.R' 'archive-and-publish.R' 'as-data-frame.R' 'as-vector.R' 'auth.R' 'automation.R' 'batches.R' 'case-variables.R' 'case-when-variable.R' 'categories.R' 'category.R' 'change-category-id.R' 'combine-categories.R' 'compare-categories.R'

'compare-datasets.R' 'compare-subvariables.R'
'compare-variables.R' 'compare.R' 'conditional-transform.R'
'consent.R' 'context-manager.R' 'copy.R' 'crunch-data-frame.R'
'crunch.R' 'crunchbox.R' 'cube-collapse-dimensions.R'
'cube-dims.R' 'cube-index-table.R' 'cube-query.R'
'cube-residuals.R' 'cube-result.R' 'cube-subset.R'
'transform.R' 'cube-transforms.R' 'cube-variables.R' 'cut.R'
'dataset-catalog.R' 'dataset-extract.R' 'dataset-order.R'
'dataset-reference.R' 'dataset-stream.R' 'dataset-update.R'
'dataset.R' 'datetime.R' 'shoji-catalog.R' 'decks.R' 'delete.R'
'dichotomize.R' 'dimension-transforms.R' 'drop-rows.R'
'edit-lock.R' 'export-dataset.R' 'expressions.R'
'fill-variable.R' 'filters.R' 'folders.R' 'fork-and-merge.R'
'formula.R' 'gadgets.R' 'geo.R' 'get-datasets.R' 'github.R'
'hide-variables.R' 'insertions.R' 'interact.R' 'join-dataset.R'
'json.R' 'length.R' 'make-array.R' 'members.R'
'merge-crunch-data-frame.R' 'misc.R' 'multitables.R'
'new-dataset.R' 'ordering.R' 'palettes.R'
'permission-catalog.R' 'permissions.R' 'show.R'
'pretty-print.R' 'private-variables.R' 'progress.R'
'project-folder.R' 'projects.R' 'rowwise.R' 'search.R'
'session.R' 'share.R' 'shoji-entity.R' 'shoji-folder.R'
'shoji-order-slots.R' 'shoji-order.R' 'shoji.R' 'slides.R'
'sliding-categories.R' 'subtotals-and-headings.R'
'subvariables.R' 'summary-insertions.R' 'tab-book.R' 'teams.R'
'transform-calculate.R' 'tuple.R' 'univariate.R' 'user-info.R'
'user.R' 'variable-as-methods.R' 'variable-catalog.R'
'variable-derivation.R' 'variable-folder.R'
'variable-metadata.R' 'variable-order.R' 'variable-summary.R'
'variable-type.R' 'variable-update.R' 'variable.R'
'variables.R' 'versions.R' 'weight.R' 'zcl.R'

NeedsCompilation no

Author Greg Freedman Ellis [aut, cre],
Jonathan Keane [aut],
Mike Malecki [aut],
Neal Richardson [aut],
Gordon Shotwell [aut]

Maintainer Greg Freedman Ellis <greg@crunch.io>

Repository CRAN

Date/Publication 2022-11-16 14:00:05 UTC

R topics documented:

addGeoMetadata *Add geodata metadata to a crunch variable*

Description

If the variable matches a single geographic shapefile hosted by crunch, addGeoMetadata will make the appropriate CrunchGeography to add to a variable's geo() metadata. It matches based on how well the contents of the variable match the feature properties that are in each shapefile.

Usage

```
addGeoMetadata(variable, ...)
```

Arguments

variable	a Crunch variable to use for matching. This must be either a text or a categorical variable.
...	arguments passed on to matchCatToFeat() for example a set of available geographic features as all_features if you want to limit the number of features to be considered.

Details

If more than one property of the same geographic shapefile has the same highest matching score, the first one will be used.

If more than one geographic shapefile has the same highest matching score, an error will be printed listing the geographic shapefiles that matched. Information from this error can be used to setup an appropriate CrunchGeography by hand to connect a variable with the metadata needed.

Value

a CrunchGeography object that can be assigned into geo(variable)

Examples

```
## Not run:
geo(ds$state) <- addGeoMetadata(ds$state)

## End(Not run)
```

addSubvariable *Add subvariable to an array*

Description

Add subvariable to an array

Usage

```
addSubvariable(variable, subvariable)

addSubvariables(variable, subvariable)
```

Arguments

variable	the array variable to modify
subvariable	the subvariable to add, or a list of those to add, or a dataset subset. You can supply variables, variable definitions or lists of variables and variable definitions.

Value

`variable` with the indicated subvariables added.

See Also

`subvariables()`

Examples

```
## Not run:
ds$allpets <- addSubvariable(ds$allpets, ds$allpets_4)
ds$petloc <- addSubvariables(ds$petloc, ds[c("petloc_school", "petloc_daycare")])

## End(Not run)
```

addSummaryStat *Add summary statistics to a CrunchCube*

Description

Use `addSummaryStat()` to add a summary statistic to a CrunchCube object. If not otherwise specified, the summary statistic will be `mean` and be placed at the bottom of the cube. You can change those defaults by passing any value you can use with `SummaryStat()` (e.g. `position`, `categories`, `after`).

Usage

```
addSummaryStat(cube, stat = c("mean", "median"), var, margin, ...)
```

Arguments

cube	a CrunchCube to add stats to
stat	a character with the summary statistic to include (default: "mean")
var	a character with the name of the dimension variable to add the summary statistic for generally the alias of the variable in Crunch, but might include Crunch functions like `rollup()`, `bin()`, etc.
margin	which margin should the summary statistic be applied for (used in the cases of categorical arrays where a variable might contribute more than one margin)
...	options to pass to `SummaryStat()` (e.g., position, after, etc.)

Value

a CrunchCube with the summary statistic Insertion added to the transforms of the variable specified

See Also

SummaryStat

Examples

```
## Not run:
pet_feelings
#                   animals
# feelings            cats dogs
#  extremely happy      9    5
#  somewhat happy      12   12
#  neutral             12    7
#  somewhat unhappy    10   10
#  extremely unhappy   11   12

# add a mean summary statistic to a CrunchCube
addSummaryStat(pet_feelings, stat = "mean", var = "feelings")
#                 animals
# feelings                      cats             dogs
#  extremely happy               9                5
#   somewhat happy              12               12
#          neutral              12                7
# somewhat unhappy              10               10
# extremely unhappy              11               12
#             mean 4.90740740740741 4.34782608695652

# we can also store the CrunchCube for use elsewhere
pet_feelings <- addSummaryStat(pet_feelings, stat = "mean", var = "feelings")
pet_feelings
#                 animals
# feelings                      cats             dogs
```

```
#  extremely happy               9               5
#   somewhat happy              12              12
#          neutral              12               7
# somewhat unhappy              10              10
# extremely unhappy              11              12
#             mean 4.90740740740741 4.34782608695652

# `addSummaryStat` returns a CrunchCube that has had the summary statistic
# added to it, so that you can still use the Crunch logic for multiple
# response variables, missingness, etc.
class(pet_feelings)
# [1] "CrunchCube"
# attr(,"package")
# [1] "crunch"

# Since `pet_feelings` is a CrunchCube, although it has similar properties
# and behaviors to arrays, it is not a R array:
is.array(pet_feelings)
# [1] FALSE

# cleanup transforms
transforms(pet_feelings) <- NULL
# add a median summary statistic to a CrunchCube
pet_feelings <- addSummaryStat(pet_feelings, stat = "median", var = "feelings")
pet_feelings
#                 animals
# feelings             cats    dogs
#  extremely happy       9       5
#   somewhat happy      12      12
#          neutral      12       7
# somewhat unhappy      10      10
# extremely unhappy      11      12
#           median       5       5

# additionally, if you want a true matrix object from the CrunchCube, rather
# than the CrunchCube object itself, `applyTransforms()` will return the
# array with the summary statistics (just like subtotals and headings)
pet_feelings_array <- applyTransforms(pet_feelings)
pet_feelings_array
#                 animals
# feelings             cats    dogs
#  extremely happy       9       5
#   somewhat happy      12      12
#          neutral      12       7
# somewhat unhappy      10      10
# extremely unhappy      11      12
#           median       5       5

# and we can see that this is an array and no longer a CrunchCube
is.array(pet_feelings_array)
# [1] TRUE

## End(Not run)
```

addVariables *Add multiple variables to a dataset*

Description

This function lets you add more than one variable at a time to a dataset. If you have multiple variables to add, this function will be faster than doing `ds$var <- value` assignment because it doesn't refresh the dataset's state in between variable POST requests.

Usage

```
addVariables(dataset, ...)
```

Arguments

`dataset`	a CrunchDataset
`...`	VariableDefinitions or a list of VariableDefinitions.

Value

`dataset` with the new variables added (invisibly)

aliases *Get and set names, aliases on Catalog-type objects*

Description

These methods let you get and set names and aliases for variables in a Dataset's catalog, or within Subvariables in an array variable. They work like the base R names methods.

Usage

```
aliases(x)

aliases(x) <- value

descriptions(x)

descriptions(x) <- value

emails(x)

types(x)
```

```
timestamps(x)

ids(x)

ids(x) <- value

values(x)

values(x) <- value

scriptBody(x)

dates(x)

dates(x) <- value

## S4 method for signature 'AbstractCategories'
names(x)

## S4 replacement method for signature 'AbstractCategories'
names(x) <- value

## S4 method for signature 'AbstractCategories'
ids(x)

## S4 method for signature 'ScriptCatalog'
timestamps(x)

## S4 method for signature 'Script'
timestamps(x)

## S4 method for signature 'ScriptCatalog'
scriptBody(x)

## S4 method for signature 'Script'
scriptBody(x)

## S4 method for signature 'BatchCatalog'
names(x)

## S4 replacement method for signature 'Categories'
ids(x) <- value

## S4 method for signature 'Categories'
values(x)

## S4 replacement method for signature 'Categories'
values(x) <- value
```

```
## S4 method for signature 'Categories'
dates(x)

## S4 replacement method for signature 'Categories'
dates(x) <- value

## S3 method for class 'CrunchDataFrame'
names(x)

## S4 method for signature 'CrunchCube'
names(x)

## S4 method for signature 'CrunchCube'
aliases(x)

## S4 method for signature 'CrunchCube'
descriptions(x)

## S4 method for signature 'CrunchCube'
types(x)

## S4 method for signature 'CrunchCube'
notes(x)

## S4 method for signature 'CrunchDataset'
names(x)

## S4 method for signature 'ShojiCatalog'
names(x)

## S4 replacement method for signature 'ShojiCatalog'
names(x) <- value

## S4 method for signature 'ShojiCatalog'
emails(x)

## S4 method for signature 'CrunchDeck'
names(x)

## S4 replacement method for signature 'CrunchDeck'
names(x) <- value

## S4 method for signature 'CrunchDeck'
types(x)

## S4 replacement method for signature 'MultitableCatalog'
names(x) <- value
```

```
## S4 method for signature 'ShojiFolder'
types(x)

## S4 method for signature 'ShojiOrder'
names(x)

## S4 method for signature 'OrderGroup'
names(x)

## S4 method for signature 'SlideCatalog'
names(x)

## S4 replacement method for signature 'SlideCatalog'
names(x) <- value

## S4 method for signature 'SlideCatalog'
types(x)

## S4 method for signature 'ArrayVariable'
names(x)

## S4 method for signature 'TabBookResult'
names(x)

## S4 method for signature 'TabBookResult'
aliases(x)

## S4 method for signature 'TabBookResult'
descriptions(x)

## S4 method for signature 'MultitableResult'
names(x)

## S4 method for signature 'MultitableResult'
aliases(x)

## S4 method for signature 'MultitableResult'
descriptions(x)

## S4 method for signature 'VariableCatalog'
aliases(x)

## S4 replacement method for signature 'VariableCatalog'
aliases(x) <- value

## S4 method for signature 'VariableCatalog'
notes(x)
```

```
## S4 replacement method for signature 'VariableCatalog'
notes(x) <- value

## S4 method for signature 'VariableCatalog'
descriptions(x)

## S4 replacement method for signature 'VariableCatalog'
descriptions(x) <- value

## S4 method for signature 'VariableCatalog'
types(x)

## S4 method for signature 'VariableCatalog'
ids(x)

## S4 method for signature 'VariableFolder'
aliases(x)

## S4 method for signature 'list'
types(x)

## S4 method for signature 'VersionCatalog'
names(x)

## S4 method for signature 'VersionCatalog'
descriptions(x)

## S4 method for signature 'VersionCatalog'
timestamps(x)
```

Arguments

x	a VariableCatalog, Subvariables, or similar object
value	For the setters, an appropriate-length character vector to assign

Details

Note that the Dataset names method returns the aliases of its variables by default. This behavior is controlled by envOrOption("crunch.namekey.dataset"). Set options(crunch.namekey.dataset="name") if you wish to use variable names. See the variables vignette for more information.

Value

Getters return the character object in the specified slot; setters return x duly modified.

See Also

Subvariables Categories base::names() vignette("variables", package="crunch")

appendDataset *Append one Crunch dataset to another*

Description

With Crunch, you can add additional rows to a dataset by appending a second dataset to the bottom of the original dataset. Crunch makes intelligent guesses to align the variables between the two datasets and to harmonize the categories and subvariables of variables, as appropriate.

Usage

```
appendDataset(dataset1, dataset2, upsert = FALSE)
```

Arguments

dataset1	a CrunchDataset
dataset2	another CrunchDataset, or possibly a data.frame. If dataset2 is not a Crunch dataset, it will be uploaded as a new dataset before appending. If it is a CrunchDataset, it may be subsetted with a filter expression on the rows and a selection of variables on the columns.
upsert	Logical: should the append instead "update" rows based on the primary key variable and "insert" (append) where the primary key values are new? Default is FALSE. Note that this upserting behavior requires a primary key variable to have been set previously; see pk().

Details

Variables are matched between datasets based on their aliases. Variables present in only one of the two datasets are fine; they're handled by filling in with missing values for the rows corresponding to the dataset where they don't exist. For variables present in both datasets, you will have best results if you ensure that the two datasets have the same variable names and types, and that their categorical and array variables have consistent categories. To preview how datasets will align when appended, see compareDatasets().

Particularly if you're appending to datasets that are already shared with others, you may want to use the fork-edit-merge workflow when appending datasets. This allows you to verify your changes before releasing them to the other viewers of the dataset. To do this fork the dataset with forkDataset(), append the new data to the fork, ensure that the append worked as expected, and then merge the fork back to the original dataset with mergeFork(). For more, see vignette("fork-and-merge", package = "crunch").

Value

dataset1, updated with dataset2, potentially filtered on rows and variables, appended to it.

Examples

```
## Not run:
ds <- loadDataset("Survey, 2016")
new_wave <- loadDataset("Survey, 2017")
ds <- appendDataset(ds, new_wave)

## End(Not run)
```

---

appendStream *Manually trigger a pending append to a dataset*

Description

Crunch allows you to stream data to a dataset. Streaming data is useful for datasets which have frequent updates (see the Crunch API documentation for more information). Crunch automatically appends streamed data periodically; however, if you would like to trigger appending pending streamed data to a dataset, you can call `appendStream()`.

Usage

```
appendStream(ds)
```

Arguments

`ds`	a CrunchDataset

Value

the dataset with pending stream data appended.

---

archive-and-publish *Get and set "archived" and "published" status of a dataset*

Description

"Archived" datasets are excluded from some views. "Draft" datasets are visible only to editors, while published datasets are available to all viewers. A dataset can either be published or in draft, but not both. These properties are accessed and set with the "is" methods. You can also set the properties by assigning into the function. The verb functions `archive` and `publish` are alternate versions of the setters.

Usage

```
is.archived(x)

is.archived(x) <- value

is.draft(x)

is.draft(x) <- value

is.published(x)

is.published(x) <- value

## S4 method for signature 'CrunchDataset'
is.archived(x)

## S4 method for signature 'CrunchDataset'
is.draft(x)

## S4 method for signature 'CrunchDataset'
is.published(x)

## S4 replacement method for signature 'CrunchDataset,logical'
is.archived(x) <- value

archive(x)

## S4 replacement method for signature 'CrunchDataset,logical'
is.draft(x) <- value

## S4 replacement method for signature 'CrunchDataset,logical'
is.published(x) <- value

publish(x)

## S4 method for signature 'DatasetCatalog'
is.archived(x)

## S4 method for signature 'DatasetCatalog'
is.draft(x)

## S4 method for signature 'DatasetCatalog'
is.published(x)

## S4 replacement method for signature 'DatasetCatalog,logical'
is.archived(x) <- value

## S4 replacement method for signature 'DatasetCatalog,logical'
```

```
is.draft(x) <- value

## S4 replacement method for signature 'DatasetCatalog,logical'
is.published(x) <- value
```

Arguments

x	CrunchDataset
value	logical

Value

For the getters, the logical value of whether the dataset is archived, in draft mode, or published, where draft and published are inverses. The setters return the dataset.

Examples

```
## Not run:
ds <- loadDataset("mtcars")
is.draft(ds) # FALSE
is.published(ds) # TRUE
identical(is.draft(ds), !is.published(ds))
# Can make a dataset a "draft" by:
is.draft(ds) <- TRUE
is.published(ds) # FALSE
# Could also have set is.published(ds) <- FALSE
# Now, can go the other way by setting is.draft, is.published, or:
ds <- publish(ds)
is.published(ds) # TRUE

is.archived(ds) # FALSE
is.archived(ds) <- TRUE
is.archived(ds) # TRUE
# Could have achieved the same effect by:
ds <- archive(ds)

## End(Not run)
```

---

as-vector *Convert Variables to local R objects*

Description

Crunch Variables reside on the server, allowing you to work with datasets that are too big to bring into memory on your machine. Many functions, such as `max`, `mean`, and `crtabs()`, translate your commands into API queries and return only the result. But, not every operation you'll want to perform has been implemented on the Crunch servers. If you need to do something beyond what is currently supported, you can bring a variable's data into R with `as.vector(ds$var)` and work with it like any other R vector.

Usage

```
## S4 method for signature 'CrunchVariable'
as.vector(x, mode = "any")

## S4 method for signature 'CrunchExpr'
as.vector(x, mode = "any")
```

Arguments

x	a CrunchVariable
mode	for Categorical variables, one of either "factor" (default, which returns the values as factor); "numeric" (which returns the numeric values); or "id" (which returns the category ids). If "id", values corresponding to missing categories will return as the underlying integer codes; i.e., the R representation will not have any NA elements. Otherwise, missing categories will all be returned NA. For non-Categorical variables, the mode argument is ignored.

Details

`as.vector` transfers data from Crunch to a local R session. Note: `as.vector` returns the vector in the row order of the dataset. If filters are set that specify an order that is different from the row order of the dataset, the results will ignore that order. If you need the vector ordered in that way, use syntax like `as.vector(ds$var)[c(10, 5, 2)]` instead.

Value

an R vector of the type corresponding to the Variable. E.g. CategoricalVariable yields type factor by default, NumericVariable yields numeric, etc.

See Also

as.data.frame for another interface for (lazily) fetching data from the server as needed; `exportDataset()` for pulling all of the data from a dataset.

`as.environment,CrunchDataset-method`

as.environment method for CrunchDataset

Description

This method allows you to `eval` within a Dataset.

Usage

```
## S4 method for signature 'CrunchDataset'
as.environment(x)
```

Arguments

x	CrunchDataset

Value

an environment in which named objects are (promises that return) CrunchVariables.

---

as.Text *as.* methods for variables*

Description

Use the as.* family of functions to make a derived copy of a variable that has been converted into a new type.

Usage

```
as.Text(x, ...)

as.Numeric(x)

as.Categorical(x, ...)

as.Datetime(x, format = "%Y-%m-%d %H:%M:%S", resolution, offset)

## S4 method for signature 'CrunchVariable'
as.Numeric(x)

## S4 method for signature 'CrunchVariable'
as.Text(x, format)

## S4 method for signature 'CrunchVariable'
as.Categorical(x, format)

## S4 method for signature 'CrunchVariable'
as.Datetime(x, format = "%Y-%m-%d %H:%M:%S", resolution, offset)

## S3 method for class 'CrunchVariable'
as.double(x, ...)

## S3 method for class 'CrunchVariable'
as.character(x, ...)

## S4 method for signature 'CrunchExpr'
as.Numeric(x)

## S4 method for signature 'CrunchExpr'
```

```
as.Text(x, format)

## S4 method for signature 'CrunchExpr'
as.Categorical(x, format)

## S4 method for signature 'CrunchExpr'
as.Datetime(x, format = "%Y-%m-%d %H:%M:%S", resolution, offset)

## S3 method for class 'CrunchExpr'
as.double(x, ...)

## S3 method for class 'CrunchExpr'
as.character(x, ...)
```

Arguments

x	a Crunch variable to derive and convert to a new type
...	additional arguments for as.character and as.numeric, ignored when used with Crunch variables
format	for as.Datetime, when the variable in x is a text or categorical variable, format is the typographical format that the datetime is already formatted in that needs to be parse from (default: "%Y-%m-%d %H:%M:%S"); for as.Text and as.Categorical, is the typographical format that the datetime is to be formatted as (e.g. "%Y-%m-%d %H:%M:%S" for "2018-01-08 12:39:57", the default if no rollup resolution is specified on the source variable. If a rollup resolution is specified, a reasonable default will be used.).
resolution	for as.Datetime, when the variable in x is a numeric variable, the resolution of the number (e.g. "ms" for milliseconds, "s" for seconds, etc. see expressions for more information about valid values.)
offset	for as.Datetime, when the variable in x is a numeric the, a character of the offset to count from in the shape "2018-01-08 12:39:57". If not supplied, Crunch's default of 1970-01-01 00:00:00 will be used.

Details

Each type of Crunch variable (text, numeric, categorical, etc.) has an as.* function (as.Text, as.Numeric, and as.Categorical respectively) that takes the input given as x, and makes a new derived variable that is now of the type specified. See below for detailed examples.

For as.Text and as.Numeric, aliases to the R-native functions as.character and as.numeric are provided for convenience.

Value

a CrunchExpr to be used as the derivation

Examples

```
## Not run:
# ds$v1 is of type Text
is.Text(ds$v1)
# [1] TRUE

# that has strings of numbers
as.vector(ds$v1)
# [1] "32"   "8"    "4096" "1024"

# convert this to a numeric variable with the alias `v1_numeric`
ds$v1_numeric <- as.Numeric(ds$v1)

# the values are the same, but are now numerics and the type is Numeric
as.vector(ds$v1_numeric)
# [1]   32    8 4096 1024
is.Numeric(ds$v1_numeric)
# [1] TRUE

# this new variable is derived, so if new data is appended or streamed, the
# new rows of data will be updated.
is.derived(ds$v1_numeric)
# [1] TRUE

## End(Not run)
```

`automation-undo` *Undo behavior of a Crunch Automation Script*

Description

There are two ways to revert the output of a script:

- `undoScript()` - A "softer" delete of a script's created artifacts and variables, or
- `revertScript()` - A "harder" revert that returns the dataset to the state it was before running such script.

Usage

```
undoScript(dataset, x)

revertScript(dataset, x)

scriptSavepoint(x)

## S4 method for signature 'CrunchDataset,Script'
undoScript(dataset, x)
```

```
## S4 method for signature 'CrunchDataset,ANY'
undoScript(dataset, x)

## S4 method for signature 'CrunchDataset,Script'
revertScript(dataset, x)

## S4 method for signature 'CrunchDataset,ANY'
revertScript(dataset, x)

## S4 method for signature 'Script'
scriptSavepoint(x)
```

Arguments

dataset	A CrunchDataset
x	A Script or index for a ScriptCatalog (generally a number)

Details

The difference between both is that a hard revert restores the dataset, as it drops all ensuing scripts and their output (artifacts and variables), while an undo only deletes the artifacts and variables created by this script, but changes made by other scripts and this script's record will remain in place.

The function scriptSavepoint() gets the version object

Value

For undoScript() and revertSctipt(), invisibly return the updated dataset. For scriptSavePoint() a version list object that can be used in restoreVersion().

See Also

runCrunchAutomation() & script-catalog

---

availableGeodataFeatures

Get the property features for available geographies

Description

Get the property features for available geographies

Usage

```
availableGeodataFeatures(
  x = getAPIRoot(),
  geodatum_fields = c("name", "description", "location")
)
```

Arguments

x	an API root address (default: the R-session default)
geodatum_fields	character, what pieces of information about each geodatum should be retained? (default: ‘c("name", "description", "location")‘)

Value

a dataframe with all of the available features and geographies for matching

---

batches *See the appended batches of this dataset*

Description

See the appended batches of this dataset

Usage

```
batches(x)
```

Arguments

x	a CrunchDataset

Value

a BatchCatalog

---

c-categories *S3 method to concatenate Categories and Category objects*

Description

S3 method to concatenate Categories and Category objects

Usage

```
## S3 method for class 'Categories'
c(...)

## S3 method for class 'Category'
c(...)
```

Arguments

...	see c

Value

An object of class Categories

Examples

```
cat.a <- Category(name = "First", id = 1, numeric_value = 1, missing = FALSE)
cat.b <- Category(name = "Second", id = 2)
cat.c <- Category(name = "Third", id = 3, missing = TRUE)
cats.1 <- Categories(cat.a, cat.b)
identical(cats.1, c(cat.a, cat.b))
identical(c(cats.1, cat.c), Categories(cat.a, cat.b, cat.c))
```

---

catalog-dataframes *as.data.frame method for catalog objects*

Description

This method gives you a view of a catalog, such as a VariableCatalog, as a data.frame in order to facilitate further exploration.

Usage

```
## S3 method for class 'VariableCatalog'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  keys = c("alias", "name", "type"),
  ...
)

## S3 method for class 'ShojiCatalog'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'BatchCatalog'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  keys = c("id", "status"),
  ...
)
```

```
## S3 method for class 'FilterCatalog'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  keys = c("name", "id", "is_public"),
  ...
)

## S3 method for class 'UserCatalog'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  keys = c("name", "email", "teams", "collaborator"),
  ...
)
```

Arguments

x	A catalog object
row.names	A character vector of elements to use as row labels for the resulting data.frame, or NULL, the default, adds no row labels.
optional	part of as.data.frame signature. Ignored.
keys	A character vector of the catalog attributes that you would like included in the data.frame. To include all attributes, set keys to TRUE, which is the default for some catalogs. Other catalog classes specify a narrower default: • VariableCatalog: c("alias", "name", "type") • BatchCatalog: c("id", "status") • FilterCatalog: c("name", "id", "is_public")
...	Additional arguments passed to data.frame

Details

Modifying the data.frame produced by this function will not update the objects on the Crunch server. Other methods exist for updating the metadata in the variable catalog, for example. See vingette("variables", package = "crunch").

Value

A data.frame including metadata about each entity contained in the catalog. The fields in the data.frame match the keys argument provided to the function, and each row represents a entity.

Examples

```
## Not run:
ds <- loadDataset("iris")
vars <- variables(ds)
```

```
var_df <- as.data.frame(vars, keys = TRUE)
# With row names
as.data.frame(vars, row.names = urls(vars))

## End(Not run)
```

`Categories-class` *Categories in CategoricalVariables*

Description

CategoricalVariables, as well as the array types composed from Categoricals, contain Categories. Categories are a subclass of list that contains only Category objects. Category objects are themselves subclasses of lists and contain the following fields:

- "name": The name of the category, must be unique within a set of categories
- "id": An integer that uniquely identifies the category
- "numeric_value": A numeric value associated with the category (defaults to NA meaning that no value is associated, *not* that the category is missing)
- "missing": Logical indicating whether the category should be considered missing (defaults to `FALSE`)
- "selected": Logical indicating whether the category is selected or not (defaults to `FALSE`)
- "date": A string indicating a day or range of days that should be associated with the category. Accepted formats are "YYYY-MM-DD" ("2020-01-01") for a day, "YYYY-WXX" ("2020-W01") for an ISO week (a week that starts on a Monday, with the first week of the year being the first week with more than 4 days in it), "YYYY-MM" ("2020-01") for a month, "YYYY" ("2020") for a year, or "YYYY-MM-DD,YYYY-MM-DD" ("2020-01-01,2020-01-10") for a range of days.

Usage

```
Categories(..., data = NULL)

Category(..., data = NULL)
```

Arguments

`...`	Category attributes
`data`	For the constructor functions `Category` and `Categories`, you can either pass in attributes via `...` or you can create the objects with a fully defined `list` representation of the objects via the `data` argument. See the examples.

Examples

```
cat.a <- Category(name = "First", id = 1, numeric_value = 1, missing = FALSE)
cat.b <- Category(data = list(name = "First", id = 1, numeric_value = 1, missing = FALSE))
identical(cat.a, cat.b)
cat.c <- Category(name = "Second", id = 2)
cats.1 <- Categories(cat.a, cat.c)
cats.2 <- Categories(data = list(cat.a, cat.c))
identical(cats.1, cats.2)
```

---

categoriesFromLevels *Convert a factor's levels into Crunch categories.*

Description

Crunch categorical variables have slightly richer metadata than R's factor variables. This function generates a list of category data from a factor's levels which can then be further manipulated in R before being imported into Crunch.

Usage

```
categoriesFromLevels(level_vect)
```

Arguments

level_vect	A character vector containing the levels of a factor. Usually obtained by running base::levels()

Value

A list with each category levels id, name, numeric_value, and missingness.

Examples

```
categoriesFromLevels(levels(iris$Species))
```

---

cd *Change to different folder*

Description

Like cd in a file system, this function takes you to a different folder, given a relative path specification.

Usage

```
cd(x, path, create = FALSE)
```

Arguments

x	A CrunchDataset or Folder (VariableFolder or ProjectFolder)
path	A character "path" to the folder: either a vector of nested folder names or a single string with nested folders separated by a delimiter ("/" default, configurable via options(crunch.delimiter)). The path is interpreted as relative to the location of the folder x (when x is a dataset, that means the root, top-level folder). path may also be a Folder object.
create	logical: if the folder indicated by path does not exist, should it be created? Default is FALSE. Argument mainly exists for the convenience of mv(), which moves entities to a folder and ensures that the folder exists. You can call cd directly with create=TRUE, though that seems unnatural.

Value

A Folder (VariableFolder or ProjectFolder)

See Also

mv() to move entities to a folder; rmdir() to delete a folder; base::setwd() if you literally want to change your working directory in your local file system, which cd() does not do

Examples

```
## Not run:
ds <- loadDataset("Example survey")
demo <- cd(ds, "Demographics")
names(demo)
# Or with %>%
require(magrittr)
ds <- ds %>%
    cd("Demographics") %>%
    names()
# Can combine with mv() and move things with relative paths
ds %>%
    cd("Key Performance Indicators/Brand X") %>%
    mv("nps_x", "../Net Promoters")

## End(Not run)
```

---

changeCategoryID *Change the id of a category for a categorical variable*

Description

Changes the id of a category from an existing value to a new one. The variable can be a categorical, categorical array, or multiple response variable. The category changed will have the same numeric value and missing status as before. The one exception to this is if the numeric value is the same as the id, then the new numeric value will be the same as the new id.

Usage

```
changeCategoryID(variable, from, to)
```

Arguments

variable	the variable in a crunch dataset that will be changed (note: the variable must be categorical, categorical array, or multiple response)
from	the (old) id identifying the category you want to change
to	the (new) id for the category

Details

It is highly recommended to disable any exclusion filter before using changeCategoryID, especially if it is being called multiple times in quick succession (e.g. as part of an automated script). If a problematic exclusion is encountered changeCategoryID will attempt to disable and re-enable the exclusion, but that process will be repeated for every call made which could have adverse consequences (not to mention slow down processing time).

Value

variable with category from and all associated data values mapped to id to

Examples

```
## Not run:
ds$country <- changeCategoryID(ds$country, 2, 6)

## End(Not run)
```

---

cleanseBatches *Remove batches from a dataset*

Description

Sometimes append operations do not succeed, whether due to conflicts between the two datasets or other server-side issues. Failed appends can leave behind "error" status batch records, which can cause confusion. This function lets you delete batches that don't match the status or statuses you want to keep.

Usage

```
cleanseBatches(dataset, keep = c("imported", "appended"))
```

Arguments

dataset	CrunchDataset
keep	character the statuses that you want to keep. By default, batches that don't have either "imported" or "appended" status will be deleted.

Value

`dataset` with the specified batches removed.

---

collapseCategories *Combine Categories in place*

Description

This function allows you to combine the categories of a variable without making a copy of the variable.

Usage

```
collapseCategories(var, from, to)
```

Arguments

`var`	A categorical Crunch variable
`from`	A character vector of categories you want to combine.
`to`	A character string with the destination category.

Value

the variable duly modified

See Also

`combine()`

---

combine *Combine categories or responses*

Description

Crunch allows you to create a new categorical variable by combining the categories of another variable. For instance, you might want to recode a categorical variable with three categories small, medium, and large to one that has just small and large.

Usage

```
combine(variable, combinations = list(), ...)

combineCategories(variable, combinations = list(), ...)

combineResponses(variable, combinations = list(), ...)
```

Arguments

variable	Categorical, Categorical Array, or Multiple Response variable
combinations	list of named lists containing 1. "categories": category ids or names for categorical types, or for multiple response, "responses": subvariable names, aliases, or positional indices; 2. a "name" for the new category or response; and 3. optionally, other category ("missing", "numeric_value") or subvariable ("alias", "description") attributes. If combinations is omitted, the resulting variable will essentially be a copy (but see copy() for a more natural way to copy variables.
...	Additional variable metadata for the new derived variable

Details

Categorical and categorical array variables can have their categories combined (by specifying categories in the combinations argument). Multiple response variables can only have their responses (or items) combined (by specifying responses in the combinations argument). Categorical array items are not able to be combined together (even by specifying responses).

dplyr users may experience a name conflict between crunch::combine() and dplyr:: combine(). To avoid this, you can either explicitly use the crunch:: prefix, or you can call combineCategories() and combineResponses(), provided for disambiguation.

Value

A VariableDefinition that will create the new combined-category or -response derived variable. Categories/responses not referenced in combinations will be appended to the end of the combinations.

Examples

```
## Not run:
ds$fav_pet2 <- combine(ds$fav_pet,
    name = "Pets (combined)",
    combinations = list(
        list(name = "Mammals", categories = c("Cat", "Dog")),
        list(name = "Reptiles", categories = c("Snake", "Lizard"))
    )
)
ds$pets_owned2 <- combine(ds$allpets,
    name = "Pets owned (collapsed)",
    combinations = list(list(name = "Mammals", responses = c("Cat", "Dog")))
)

## End(Not run)
```

`compareDatasets` *Compare two datasets to see how they will append*

Description

When one dataset is appended to another, variables and subvariables are matched on their aliases, and then categories for variables that have them are matched on category name. This function lines up the metadata between two datasets as the append operation will so that you can inspect how well the datasets will align before you do the append.

Usage

```
compareDatasets(A, B)
```

Arguments

`A`	CrunchDataset
`B`	CrunchDataset

Details

Calling `summary` on the return of this function will print an overview of places where the matching on variable alias and category name may lead to undesired outcomes, enabling you to alter one or both datasets to result in better alignment.

Value

An object of class 'compareDatasets', a list of three elements: (1) 'variables', a data.frame of variable metadata joined on alias; (2) 'categories', a list of data.frames of category metadata joined on category name, one for each variable with categories; and (3) 'subvariables', a list of data.frames of subvariable metadata joined on alias, one for each array variable.

Summary output reports on (1) variables that, when matched across datasets by alias, have different types; (2) variables that have the same name but don't match on alias; (3) for variables that match and have categories, any categories that have the same id but don't match on name; (4) for array variables that match, any subvariables that have the same name but don't match on alias; and (5) array variables that, after assembling the union of their subvariables, point to subvariables that belong to other arrays.

Examples

```
## Not run:
comp <- compareDataset(ds1, ds2)
summary(comp)

## End(Not run)
```

`conditionalTransform` *Conditional transformation*

Description

Create a new variable that has values when specific conditions are met. Conditions are specified using a series of formulas: the left-hand side is the condition that must be true (a `CrunchLogicalExpr`) and the right-hand side is where to get the value if the condition on the left-hand side is true. This is commonly a Crunch variable but may be a string or numeric value, depending on the type of variable you're constructing.

Usage

```
conditionalTransform(
  ...,
  data,
  else_condition = NA,
  type = NULL,
  categories = NULL,
  formulas = NULL
)
```

Arguments

`...`	a list of conditions to evaluate (as formulas, see Details) as well as other properties to pass to the new conditional variable (i.e. alias, description)
`data`	a Crunch dataset object to use
`else_condition`	a default value to use if none of the conditions are true (default: `NA`)
`type`	a character that is either "categorical", "text", "numeric" what type of output should be returned? If `NULL`, the type of the source variable will be used. (default: `NULL`) The source variables will be converted to this type if necessary.
`categories`	a vector of characters if `type="categorical"`, these are all of the categories that should be in the resulting variable, in the order they should be in the resulting variable or a set of Crunch categories.
`formulas`	a list of conditions to evaluate (as formulas, see Details). If specified, `...` must not contain other formulas specifying conditions.

Details

The type of the new variable can depend on the type(s) of the source variable(s). By default (`type=NULL`), the type of the new variable will be the type of all of the source variables (that is, if all of the source variables are text, the new variable type will be text, if all of the source variables are categorical, the new variable will be categorical). If there are multiple types in the source variables, the result will be a text variable. The default behavior can be overridden by specifying `type = "categorical"`, `"text"`, or `"numeric"`.

conditionalTransform is similar to makeCaseVariable; however, conditionalTransform can use other Crunch variables as a source of a variable, whereas, makeCaseVariable can only use characters. This additional power comes at a cost: makeCaseVariable can be executed entirely on Crunch servers, so no data needs to be downloaded or uploaded to/from the local R session. conditionalTransform on the other hand will download the data necessary to construct the new variable.

Value

a Crunch VariableDefinition

Examples

```
## Not run:

ds$cat_opinion <- conditionalTransform(pet1 == "Cat" ~ Opinion1,
    pet2 == "Cat" ~ Opinion2,
    pet3 == "Cat" ~ Opinion3,
    data = ds,
    name = "Opinion of Cats"
)

## End(Not run)
```

---

consent *Give consent to do things that require permission*

Description

Potentially destructive actions require that you confirm that you really want to do them. If you're running a script and you know that you want to perform those actions, you can preemptively provide consent.

Usage

```
consent()

with_consent(expr)
```

Arguments

expr	Code to evaluate with consent

Value

consent returns an S3 class "contextManager" object, which you can use with with. with_consent evaluates its arguments inside the consent context.

See Also

with-context-manager ContextManager

Examples

```
## Not run:
with(consent(), delete(ds))
# Equivalent to:
with_consent(delete(ds))

## End(Not run)
```

`ContextManager` *Context managers*

Description

Context managers

Usage

```
ContextManager(
  enter = function() {
 },
  exit = function() {
 },
  error = NULL,
  as = NULL
)
```

Arguments

`enter`	function to run before taking actions
`exit`	function to run after taking actions
`error`	optional function to run if an error is thrown
`as`	character optional way to specify a default name for assigning the return of the enter function.

Value

an S3 class "contextManager" object

See Also

`with-context-manager`

copyFolders *Copy the folder structure from one dataset to another.*

Description

Copy the folder structure from one dataset to another.

Usage

```
copyFolders(source, target)
```

Arguments

source	the dataset you want to copy the order from
target	the dataset you want to copy the order to

Value

returns the target dataset with source's folder structure

Examples

```
## Not run:
ds <- copyFolders(ds1, ds)

## End(Not run)
```

copyOrder *Copy the variable order from one dataset to another.*

Description

`copyOrder` is deprecated and will be removed in a future version. Instead, you should use the `copyFolders` function.

Usage

```
copyOrder(source, target)
```

Arguments

source	the dataset you wan to copy the order from
target	the dataset you want to copy the order to

Value

returns an object of class `VariableOrder` (which can be assigned to a dataset with `ordering`)

Examples

```
## Not run:
ordering(ds) <- copyOrder(ds1, ds)

## End(Not run)
```

`copyVariable` *Copy a variable*

Description

Makes a copy of a Crunch variable on the server.

Usage

```
copyVariable(x, deep = FALSE, ...)

copy(x, deep = FALSE, ...)
```

Arguments

`x`	a CrunchVariable to copy
`deep`	logical: should this be a deep copy, in which there is no dependence on the original variable, or a shallow one, in which the copy is more of a symbolic link? Default is `FALSE`, meaning symlink.
`...`	Additional metadata to give to the new variable. If not given, the new variable will have a name that is the same as the original but with " (copy)" appended, and its alias will be the old alias with "_copy" appended.

Details

Copies can be shallow (linked) or deep. Shallow copying is faster and is preferable unless a true hard copy is required. Shallow copies are effectively pointers to the original variable, and then you append data to the original variable or otherwise alter its values, the values in the copy automatically update. This linking may be desirable, but it comes with some limitations. First, you cannot edit the values of the copy independently of the original. Second, some attributes of the copy are immutable: of note, properties of categories cannot be altered independently in the copy, but you can alter Subvariable names and ordering within arrays.

Value

a VariableDefinition for the copied variable. Assign into a Dataset to make the copy happen.

createWithPreparedData *Upload a prepared data.frame with metadata to Crunch*

Description

If you have manually created a Crunch dataset object with `prepareDataForCrunch()` this function allows you to upload it to the app.

Usage

```
createWithPreparedData(data, metadata = attr(data, "metadata"))
```

Arguments

`data`	a data.frame that meets the Crunch API specification, as returned by `prepareDataForCrunch()`, or a character path to a file or URL where such data has been written as CSV.
`metadata`	list of Crunch metadata that corresponds to `data`. Default is the "metadata" attribute of `data`, as returned by `prepareDataForCrunch`, or a character path to a file where such metadata has been written as JSON.

Value

A CrunchDataset.

crtabs *Crunch xtabs: Crosstab and otherwise aggregate variables in a Crunch Dataset*

Description

Create a contingency table or other aggregation from cross-classifying variables in a CrunchDataset, expanding on the notation allowed in `stats::xtabs()` to tailor to the kinds of calculations available in crunch.

Usage

```
crtabs(
  formula,
  data,
  weight = crunch::weight(data),
  useNA = c("no", "ifany", "always")
)
```

Arguments

formula	a stats::formula object that specifies that query to calculate. See Details for more information.
data	an object of class CrunchDataset
weight	a CrunchVariable that has been designated as a potential weight variable for data, or NULL for unweighted results. Default is the currently applied weight().
useNA	whether to include missing values in tabular results. See base::table().

Details

There are 3 types of queries supported:

- Crosstabs: Share the most in common with stats::xtabs(), are defined by a formula with only a right hand side, with each dimension specified on the right-hand side, separated by a +. A dimension are generally variables, but categorical array variables contribute 2 dimensions, "categories" and "subvariables". If you just use the categorical array variable directly, the subvariables dimensions will be added first and the categories second, but you can choose their order by specifying both categories(var) and subvariables(var) (where var is a Categorical Array CrunchVariable).
- Aggregations: An extension to 'Crosstabs' where you can select one or more measures by putting them in the left-hand side of the formula. Multiple measures can be placed in a list to calculate them together. The currently supported measures are mean(var), n() (the same as a crosstab), min(var), max(var), sd(var), sum(var) and median(var) (where var is a CrunchVariable).
- Scorecards: When you want to compare multiple MR variables with the same subvariables, you can use a scorecard to create a tabulation where they are lined up. Scorecard queries cannot be combined with the other types. Use the scorecard(..., vars = NULL) (where ... is a set of MR variables or vars is a list of them).

Value

an object of class CrunchCube

See Also

weight()

Examples

```
## Not run:
# Crosstab of people by `age_cat`:
crtabs(~age_cat, ds)

# Aggregation of means of income by `age_cat`
crtabs(mean(income) ~ age_cat, ds)

# Scorecard of multiple MRs with aligned subvariables
crtabs(~scorecard(trust_mr, value_mr, quality_mr), ds)
```

```
# Can also pre-define the variables in a scorecard with
mr_list <- list(ds$trust_mr, ds$value_mr, ds$quality_mr)
crtabs(~scorecard(vars = mr_list), ds)

# Crosstab of people by `age_cat` and the reasons for enjoying a brand (cat array)
crtabs(~age_cat + enjoy_array, ds)

# Crosstab of people by `age_cat` and the `enjoy_array` (cat array)
# But manually choosing the order of the dimensions
crtabs(~subvariables(enjoy_array) + age_cat + categories(enjoy_array), ds)

# Aggregation of means & standard deviations of income by `age_cat`
crtabs(list(mean = mean(income), sd = sd(income)) ~ age_cat, ds)

## End(Not run)
```

`crunch` *Crunch.io: instant, visual, collaborative data analysis*

Description

Crunch.io provides a cloud-based data store and analytic engine. It has a web client for interactive data exploration and visualization. The crunch package for R allows analysts to interact with and manipulate Crunch datasets from within R. Importantly, this allows technical researchers to collaborate naturally with team members, managers, and clients who prefer a point-and-click interface: because all connect to the same dataset in the cloud, there is no need to email files back and forth continually to share results.

See Also

To learn more about using the package, see `vignette("crunch")`. To sign up for a Crunch.io account, visit `https://app.crunch.io/`.

`crunch-api-key` *Crunch API Keys*

Description

The rcrunch package recommends using API keys for authentication.

Details

To get an API key for your account, follow the instructions in the crunch help desk

The rcrunch package looks for the key in the environmental variable "R_CRUNCH_API_KEY" or the option "crunch.api.key" (see `envOrOption()` for details).

One way to establish your key is to add it to your ".Renviron" file. This file is located in your home directory (you can use `usethis::edit_r_environ()` to open the file if you have the `usethis` package installed). The .Renviron file has the name of the environment varia ble, followed by an equal sign and then the value. It is good practice to set the API host too, (usually equal to "https://app.crunch.io/api/").

```
R_CRUNCH_API=https://app.crunch.io/api/
R_CRUNCH_API_KEY=YOUR_SECRET_KEY
```

You can either restart your session, or run `readRenviron("~/.Renviron")` and then rcrunch will know to use your key going forward.

`crunch-uni` *Univariate statistics on Crunch objects*

Description

Univariate statistics on Crunch objects

Usage

```
mean(x, ...)

sd(x, na.rm = FALSE)

median(x, na.rm = FALSE, ...)

## S4 method for signature 'CrunchVariable'
mean(x, ...)

## S4 method for signature 'NumericVariable'
mean(x, ...)

## S4 method for signature 'CrunchVariable'
sd(x, na.rm = FALSE)

## S4 method for signature 'NumericVariable'
sd(x, na.rm = FALSE)

## S4 method for signature 'CrunchVariable'
min(x, na.rm)
```

```
## S4 method for signature 'NumericVariable'
min(x, na.rm = FALSE)

## S4 method for signature 'DatetimeVariable'
min(x, na.rm = FALSE)

## S4 method for signature 'CrunchVariable'
max(x, na.rm)

## S4 method for signature 'NumericVariable'
max(x, na.rm = FALSE)

## S4 method for signature 'DatetimeVariable'
max(x, na.rm = FALSE)
```

Arguments

x	a NumericVariable, or for min and max, a NumericVariable or DatetimeVariable
...	additional arguments to summary statistic function
na.rm	logical: exclude missings?

See Also

base::mean() stats::sd() stats::median() base::min() base::max()

---

crunchBox *Make a CrunchBox*

Description

CrunchBoxes allow you to publish results to the world.

Usage

```
crunchBox(
  dataset,
  filters = crunch::filters(dataset),
  weight = crunch::weight(dataset),
  brand_colors,
  static_colors,
  category_color_lookup,
  ...
)

CrunchBox(
  dataset,
```

```
  filters = crunch::filters(dataset),
  weight = crunch::weight(dataset),
  brand_colors,
  static_colors,
  category_color_lookup,
  ...
)
```

Arguments

dataset	A CrunchDataset, potentially a selection of variables from it
filters	FilterCatalog, or NULL for no filters. Default all filters in your catalog, filters(dataset).
weight	a CrunchVariable that has been designated as a potential weight variable for dataset, or NULL for unweighted results. Default is the currently applied weight().
brand_colors	an optional color vector of length 3 or less, or a named list with names 'primary', 'secondary', and 'message'. See "Details" for more about color specification.
static_colors	an optional vector of colors to use for categorical plots. Bars and lines are colored in the order of static_colors. See "Details" for more about color specification.
category_color_lookup	an optional list of category names to colors to use for that category, wherever it appears in the data. This allows you to always see a category displayed in a specific color. See "Details" for more about color specification.
...	additional metadata for the box, such as "title", "header", etc.

Details

In addition to specifying the variables and filters to include in your CrunchBox, you can provide custom color palettes. The arguments brand_colors, static_colors, and category_color_lookup allow you to provide color lists to use. Colors should be either a valid hexadecimal string representation, like "#fa1af1", or they may also be an R named color, such as "darkgreen".

Value

The URL to the newly created box.

See Also

preCrunchBoxCheck() to provide guidance on what you're including in the CrunchBox

Examples

```
## Not run:
# Creating a CrunchBox with three variables
crunchBox(ds[c("var1", "var2", "var3")], title = "New CrunchBox")

# Creating a CrunchBox changing primary, secondary, and message brand colors
crunchBox(ds[c("var1", "var2", "var3")],
```

```
    title = "Branded CrunchBox",
    brand_colors = c("#ff0aa4", "#af17ff", "#260aff")
)

# Creating a CrunchBox changing category-specific colors
crunchBox(ds[c("var1", "var2", "var3")],
    title = "CrunchBox with category colors",
    category_color_lookup = list(
        "agree" = "#ff0aa4",
        "disagree" = "#af17ff",
        "don't know" = "#260aff"
    )
)

## End(Not run)
```

`CrunchDataFrame` *CrunchDataFrame*

Description

CrunchDataFrames are designed to mimic the ways that `data.frame`s are used. They should be a drop-in replacement in many places where data.frames are used.

Usage

```
## S3 method for class 'CrunchDataFrame'
dim(x)
```

Arguments

`x`	a CrunchDataFrame

Details

CrunchDataFrames are generated not by downloading all of the variables from a dataset, but rather only the variables that are needed by subsequent functions. So, if you create a CrunchDataFrame, and then run a linear model using `lm()`, only the variables used by the linear model will be downloaded.

CrunchDataFrames can be altered (that is: adding more columns, removing columns, subsetting rows, etc.) with the same `[`, `[[`, and `$` syntax as data.frames.

`CrunchDataset-class` *Crunch Datasets*

Description

Crunch Datasets

CrunchGeography-class *Geography properties for crunch variables*

Description

Crunch stores geographic data as variable metadata. There are a number of functions that help access and change this metadata.

Usage

```
CrunchGeography(..., data = NULL)

geo(x)

geo(x) <- value

## S4 method for signature 'CrunchVariable'
geo(x)

## S4 replacement method for signature 'CrunchVariable,CrunchGeography'
geo(x) <- value

## S4 replacement method for signature 'CrunchVariable,`NULL`'
geo(x) <- value

availableGeodata(x = getAPIRoot())
```

Arguments

...	for CrunchGeography, named arguments from which to construct a CrunchGeography: geodatum, feature_key, and match_field
data	for CrunchGeography, list of named arguments from which to construct a CrunchGeography: geodatum, feature_key, and match_field
x	a crunch variable
value	value of the geography property to set

Details

geo retrieves the geographic information associate with a variable. If there is geographic information it returns an object of class CrunchGeography otherwise it returns NULL.

CrunchGeography objects store geography metadata from a variable. There are three slots:

- geodatum an object of class CrunchGeodata which stores references to the Crunch-hosted (geoltopo)json to use
- feature_key a character string representing the feature inside of the (geoltopo)json which is used to match match_field (e.g. properties.name)

- `match_field` a character string representing the variable metadata information which is used to match `feature_key` to (e.g. name)

Value

geographic information of class `CrunchGeography` (`NULL` if there is none)

Examples

```
## Not run:
geo(ds$location)

geo(ds$location)$feature_key <- "properties.name"
geo(ds$location)$match_field <- "name"

## End(Not run)
```

CrunchVariable-class *Variables in Crunch*

Description

Variables are S4 objects. All inherit from the base class `CrunchVariable`.

Slots

`filter` either `NULL` or `CrunchLogicalExpr`

`tuple` `VariableTuple`

---

crunch_sitrep *Crunch situation report*

Description

Get a situation report on how R will connect to crunch.io

Usage

```
crunch_sitrep(redact = TRUE, verbose = TRUE)
```

Arguments

`redact`	Whether to redact the API key found (default TRUE)
`verbose`	Whether to print information to the console (default TRUE)

Value

Invisibly, a list with information about the API

Examples

```
## Not run:
crunch_sitrep()

## End(Not run)
```

`cube-computing` *Work with CrunchCubes, MultitableResults, and TabBookResults*

Description

These functions provide an interface like `base::margin.table()` and `base::prop.table()` for the CrunchCube object. CrunchCubes contain richer metadata than standard R `array` objects, and they also conceal certain complexity in the data structures from the user. In particular, multiple-response variables are generally represented as single dimensions in result tables, but in the actual data, they may comprise two dimensions. These methods understand the subtleties in the Crunch data types and correctly compute margins and percentages off of them.

Usage

```
margin.table(x, margin = NULL)

prop.table(x, margin = NULL)

bases(x, margin = NULL)

## S4 method for signature 'CrunchCube'
prop.table(x, margin = NULL)

## S4 method for signature 'CrunchCube'
round(x, digits = 0)

## S4 method for signature 'CrunchCube'
bases(x, margin = NULL)

## S4 method for signature 'CrunchCube'
margin.table(x, margin = NULL)

## S4 method for signature 'MultitableResult'
prop.table(x, margin = NULL)

## S4 method for signature 'TabBookResult'
prop.table(x, margin = NULL)
```

```
## S4 method for signature 'TabBookResult'
bases(x, margin = NULL)

## S4 method for signature 'MultitableResult'
bases(x, margin = NULL)
```

Arguments

x	a CrunchCube
margin	index, or vector of indices to generate margin for. See base::prop.table(). bases() accepts 0 as an additional valid value for margin, which yields the unweighted counts for the query.
digits	For round, the number of decimal places to round to. See base::round()

Details

These functions also generalize to MultitableResults and TabBookResults, which are returned from a tabBook() request. When called on one of those objects, they effectively apply over each CrunchCube contained in them.

bases is an additional method for CrunchCubes. When making weighted requests, bases allows you to access the unweighted counts for every cell in the resulting table (array). The bases function takes a "margin" argument to work like margin.table, or with margin=0 gives all cell counts.

Value

When called on CrunchCubes, these functions return an array. Calling prop.table on a MultitableResult returns a list of prop.tables of the CrunchCubes it contains. Likewise, prop.table on a TabBookResult returns a list of lists of prop.tables.

See Also

margin.table() prop.table()

---

cube-residuals *Calculate standardized residuals from a CrunchCube*

Description

Standardized residuals, (observed - expected) / sqrt(V), where V is the residual cell variance (Agresti, 2007, section 2.4.5). Special care is taken for multiple-response variables which are in effect a series of separate tables where 'not selected' cells for each item are are hidden.

Usage

```
zScores(x)

## S4 method for signature 'CrunchCube'
zScores(x)

rstandard(model)
```

Arguments

x	A CrunchCube representing a contingency table
model	A CrunchCube representing a contingency table (for rstandard() only)

Value

an array of standardized residuals or Z-scores from the hypothesis being tested. The default method is that the joint distributions of (weighted) counts are equal to the marginal distributions of the table.

References

Agresti, A. (2007) *An Introduction to Categorical Data Analysis, 2nd ed.*, New York: John Wiley & Sons. Page 38.

See Also

stats::chisq.test

---

cubeMeasureType *Get measure type of cube result*

Description

Returns a string describing the measure type of the cube result, such as "count", "mean", "sd", etc.

Usage

```
cubeMeasureType(x, measure = NULL)

## S4 method for signature 'CrunchCube'
cubeMeasureType(x, measure = 1)
```

Arguments

x	A CrunchCube
measure	Which measure in the cube to check, can index by position with numbers or by name. NULL, the default, will select a "sum" type measure first, "mean" if no sum is available, and will use the cube's names in alphabetic order if there are no "sum" or "mean" measures (or if a tie breaker between two measure types is needed).

Value

A string describing the cube's measure type

Examples

```
## Not run:
cube1 <- crtabs(~allpets, ds)
cubeMeasureType(cube1)
#> "count"

cube2 <- crtabs(list(a = n(), b = mean(age)) ~ allpets, ds)
cubeMeasureType(cube2)
#> "count"
cubeMeasureType(cube2, "b")
#> "mean"

## End(Not run)
```

---

cut,DatetimeVariable-method

Cut a Datetime Crunch variable

Description

`crunch::cut()` is equivalent to `base::cut()` except that it operates on Crunch variables instead of in-memory R objects. The function takes a Datetime variable and derives a new categorical variable from it based on the `breaks` argument. You can either break the variable into evenly spaced categories by specifying an interval using a string that defines a period or a vector containing the start and end point of each category. For example, specifying `breaks = "2 weeks"` will break the datetime data into 2 week size bins while `breaks = as.Date(c("2020-01-01", "2020-01-15" "2020-02-01"))` will recode the data into two groups based on whether the numeric vector falls between January 1 and 14 or January 15 and 31

Usage

```
## S4 method for signature 'DatetimeVariable'
cut(x, breaks, labels = NULL, dates = NULL, name, right = FALSE, ...)
```

Arguments

`x`	A Crunch `DatetimeVariable`
`breaks`	Either a numeric vector of two or more unique cut point datetimes or a single string giving the interval size into which `x` is to be cut with a number optionally at the beginning nd "day", "weeks", "months", a "quarters" or "years". If specifying cut points, values that are less than the smallest value in `breaks` or greater than the largest value in `breaks` will be marked missing in the resulting categorical variable.

labels	A character vector representing the labels for the levels of the resulting categories. The length of the labels argument should be the same as the number of categories, which is one fewer than the number of breaks. If not specified, labels are constructed with a formatting like "YYYY/MM/DD - YYYY/MM/DD" (for example ("2020/01/01 - 2020/01/14"))
dates	(Optionally) A character vector with the date strings that should be associated with the resulting categories. These dates can have the form "YYYY-MM-DD", "YYYY-MM", "YYYY", "YYYY-WXX" (where "XX" is the ISO week number) or "YYYY-MM-DD,YYYY-MM-DD". If left NULL, it will be created from the categories.
name	The name of the resulting Crunch variable as a character string.
right	logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. This only applies if giving a vector of break points.
...	further arguments passed to makeCaseVariable

Value

a Crunch VariableDefinition. Assign it into the dataset to create it as a derived variable on the server.

Examples

```
## Not run:
ds <- loadDataset("example")
ds$month_cat <- cut(ds$date, breaks = "month", name = "monthly")
ds$four_weeks_cat <- cut(ds$date, breaks = "4 weeks", name = "four week categorical date")

ds$wave_cat <- cut(
    ds$date,
    as.Date(c("2020-01-01", "2020-02-15", "2020-04-01", "2020-05-15")),
    labels = c("wave1", "wave2", "wave3"),
    name = "wave var"
  )

## End(Not run)
```

---

cut,NumericVariable-method

Cut a numeric Crunch variable

Description

crunch::cut() is equivalent to base::cut() except that it operates on Crunch variables instead of in-memory R objects. The function takes a numeric variable and derives a new categorical variable from it based on the breaks argument. You can either break the variable into evenly spaced categories by specifying the number of breaks, or specify a numeric vector identifying the start and end point of each category. For example, specifying breaks = 5 will break the numeric data into five evenly spaced portions while breaks = c(1, 5, 10) will recode the data into two groups based on whether the numeric vector falls between 1 and 5 or 5 and 10.

Usage

```
## S4 method for signature 'NumericVariable'
cut(
  x,
  breaks,
  labels = NULL,
  name,
  include.lowest = FALSE,
  right = TRUE,
  dig.lab = 3,
  ordered_result = FALSE,
  ...
)
```

Arguments

x	A Crunch NumericVariable
breaks	Either a numeric vector of two or more unique cut points or a single number giving the number of intervals into which x is to be cut. If specifying cut points, values that are less than the smallest value in breaks or greater than the largest value in breaks will be marked missing in the resulting categorical variable.
labels	A character vector representing the labels for the levels of the resulting categories. The length of the labels argument should be the same as the number of categories, which is one fewer than the number of breaks. If not specified, labels are constructed using interval notation. For example, [1, 5) indicates that the category goes from 1 to 5. The bracket shape indicates whether the boundary value is included in the category, i.e. whether it is "closed". [1, 5) indicates that the interval includes (is closed on) 1 but does not include (is open on) 5. If labels = FALSE, simple integer codes are returned instead of a factor.
name	The name of the resulting Crunch variable as a character string.
include.lowest	logical, indicating if an x[i] equal to the lowest (or highest, for right = FALSE) breaks value should be included.
right	logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa.
dig.lab	integer which is used when labels are not given. It determines the number of digits used in formatting the break numbers.
ordered_result	Ignored.
...	further arguments passed to makeCaseVariable

Value

a Crunch VariableDefinition. Assign it into the dataset to create it as a derived variable on the server.

Examples

```
## Not run:
ds <- loadDataset("mtcars")
ds$cat_var <- cut(ds$mpg,
    breaks = c(10, 15, 20),
    labels = c("small", "medium"), name = "Fuel efficiency"
)
ds$age <- sample(1:100, 32)
ds$age4 <- cut(df$age, c(0, 30, 45, 65, 200),
    c("Youth", "Adult", "Middle-aged", "Elderly"),
    name = "Age (4 category)"
)

## End(Not run)
```

---

dashboard *View or set a dashboard URL*

Description

You can designate a dashboard that will show when the dataset is loaded in the Crunch web app. This dashboard could be a Crunch Shiny ("Crunchy") app, a CrunchBox, an RMarkdown website or something else.

Usage

```
dashboard(x)

setDashboardURL(x, value)

dashboard(x) <- value
```

Arguments

x	CrunchDataset
value	For the setter, a URL (character) or NULL to unset the dashboard.

Value

The getter returns a URL (character) or NULL. The setter returns the dataset (x).

Examples

```
## Not run:
dashboard(ds) <- "https://shiny.crunch.io/example/"

## End(Not run)
```

dataset-to-R *as.data.frame method for CrunchDataset*

Description

This method is defined principally so that you can use a CrunchDataset as a data argument to other R functions (such as stats::lm()) without needing to download the whole dataset. You can, however, choose to download a true data.frame.

Usage

```
## S3 method for class 'CrunchDataset'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  force = FALSE,
  categorical.mode = "factor",
  row.order = NULL,
  include.hidden = TRUE,
  ...
)

## S3 method for class 'CrunchDataFrame'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  include.hidden = attr(x, "include.hidden"),
  ...
)
```

Arguments

x	a CrunchDataset or CrunchDataFrame
row.names	part of as.data.frame signature. Ignored.
optional	part of as.data.frame signature. Ignored.
force	logical: actually coerce the dataset to data.frame, or leave the columns as unevaluated promises. Default is FALSE.
categorical.mode	what mode should categoricals be pulled as? One of factor, numeric, id (default: factor)
row.order	vector of indices. Which, and their order, of the rows of the dataset should be presented as (default: NULL). If NULL, then the Crunch Dataset order will be used.
include.hidden	logical: should hidden variables be included? (default: TRUE)
...	additional arguments passed to as.data.frame (default method).

Details

By default, the `as.data.frame` method for `CrunchDataset` does not return a `data.frame` but instead `CrunchDataFrame`, which behaves like a `data.frame` without bringing the whole dataset into memory. When you access the variables of a `CrunchDataFrame`, you get an R vector, rather than a `CrunchVariable`. This allows modeling functions that require select columns of a dataset to retrieve only those variables from the remote server, rather than pulling the entire dataset into local memory.

If you call `as.data.frame()` on a `CrunchDataset` with `force = TRUE`, you will instead get a true `data.frame`. You can also get this `data.frame` by calling `as.data.frame` on a `CrunchDataFrame` (effectively calling `as.data.frame` on the dataset twice)

When a `data.frame` is returned, the function coerces Crunch Variable values into their R equivalents using the following rules:

- Numeric variables become numeric vectors
- Text variables become character vectors
- Datetime variables become either `Date` or `POSIXt` vectors
- Categorical variables become either factors with levels matching the Crunch Variable's categories (the default), or, if `categorical.mode` is specified as "id" or "numeric", a numeric vector of category ids or numeric values, respectively
- Array variables (Categorical Array, Multiple Response) are decomposed into their constituent categorical subvariables. An array with three subvariables, for example, will result in three columns in the `data.frame`.

Column names in the `data.frame` are the variable/subvariable aliases.

Value

When called on a `CrunchDataset`, the method returns an object of class `CrunchDataFrame` unless `force = TRUE`, in which case the return is a `data.frame`. For `CrunchDataFrame`, the method returns a `data.frame`.

See Also

`as.vector()`

---

datasets *Get a catalog of datasets*

Description

Crunch datasets are collected in folders called "projects". `datasets()` can be used to filter a project's contents to see only datasets (and not other projects). You can also use it to pull a catalog of datasets from search results.

Usage

```
datasets(x = getAPIRoot())

datasets(x) <- value
```

Arguments

x	a ProjectFolder or SearchResults that may contain datasets
value	For assignment, a CrunchDataset to move

Details

The datasets()<- assignment function provides an alternative method for moving a dataset into a project. This may be more convenient in some cases than using mv().

Value

When x is a ProjectFolder, datasets() returns the folder with its "index" filtered to contain only datasets; otherwise, it returns an object of class DatasetCatalog. The assignment function returns the project x with the given dataset added to it.

Examples

```
## Not run:
# Get the names of the datasets contained in a project
projects() %>%
    cd("Important Clients") %>%
    datasets() %>%
    names()
# The assignment method lets you move a dataset to a project
proj <- cd(projects(), "Important Clients")
ds <- loadDataset("New important client survey")
datasets(proj) <- ds

## End(Not run)
```

---

decks *Get a dataset's DeckCatalog*

Description

Crunch decks are stored in catalogs. This function returns those catalogs so that you can access and manipulate decks in R.

Usage

```
decks(x)

decks(x) <- value

## S4 method for signature 'CrunchDataset'
decks(x)
```

Arguments

x	a Crunch Dataset
value	a CrunchDeck to add

Value

a DeckCatalog

---

delete *Delete a Crunch object from the server*

Description

These methods delete entities, notably Datasets and Variables within them, from the server. This action is permanent and cannot be undone, so it should not be done lightly. Consider instead using archive for datasets and hide for variables.

Usage

```
delete(x, ...)

## S4 method for signature 'CrunchDataset'
delete(x, ...)

## S4 method for signature 'DatasetTuple'
delete(x, ...)

## S4 method for signature 'CrunchDeck'
delete(x, ...)

## S4 method for signature 'CrunchSlide'
delete(x, ...)

## S4 method for signature 'Multitable'
delete(x, ...)

## S4 method for signature 'CrunchTeam'
```

```
delete(x, ...)

## S4 method for signature 'CrunchVariable'
delete(x, ...)

## S4 method for signature 'VariableTuple'
delete(x, ...)

## S4 method for signature 'ShojiFolder'
delete(x, ...)

## S4 method for signature 'ShojiTuple'
delete(x, ...)

## S4 method for signature 'ShojiObject'
delete(x, ...)

## S4 method for signature 'ANY'
delete(x, ...)
```

Arguments

x	a Crunch object
...	additional arguments, generally ignored

Details

Deleting requires confirmation. In an interactive session, you will be asked to confirm. To avoid that prompt, or to delete objects from a non-interactive session, wrap the call in `with_consent()` to give your permission to delete.

See Also

`hide()` `deleteDataset()` `deleteVariables()` `deleteSubvariables()`

---

deleteDataset *Delete a dataset from the dataset list*

Description

This function lets you delete a dataset without first loading it, which is faster.

Usage

```
deleteDataset(x, ...)
```

Arguments

x	The name (character) of a dataset, a path to a dataset, or a CrunchDataset. Unless x is a parsed folder path, it can only be of length 1–for your protection, this function is not vectorized.
...	additional parameters passed to delete()

Details

The function also works on CrunchDataset objects, just like delete(), which may be useful if you have loaded another package that masks the crunch::delete() method.

Value

(Invisibly) the API response from deleting the dataset

See Also

delete(); cd() for details of parsing and walking dataset folder/project paths.

---

deleteSubvariables *Delete subvariables from an array*

Description

Deleting variables requires confirmation. In an interactive session, you will be asked to confirm. To avoid that prompt, or to delete subvariables from a non-interactive session, wrap the call in with_consent() to give your permission to delete.

Usage

```
deleteSubvariables(variable, to.delete)

deleteSubvariable(variable, to.delete)
```

Arguments

variable	the array variable
to.delete	aliases (following crunch.namekey.dataset) or indices of variables to delete.

Details

To delete the subvariables the function unbinds the array, deletes the subvariable, and then binds the remaining subvariables into a new array.

Value

a new version of variable without the indicated subvariables

See Also

deleteVariable() delete()

---

deleteVariables *Delete Variables Within a Dataset*

Description

This function permanently deletes a variable from a dataset.

Usage

```
deleteVariables(dataset, variables)

deleteVariable(dataset, variables)
```

Arguments

dataset	the Dataset to modify
variables	aliases (following crunch.namekey.dataset) or indices of variables to delete.

Details

In an interactive session, you will be prompted to confirm that you wish to delete the variable. To avoid that prompt, or to delete variables from a non-interactive session, wrap the call in with_consent() to give your permission to delete.

Value

(invisibly) dataset with the specified variables deleted

See Also

delete(); deleteSubvariable(); For a non-destructive alternative, see hide().

`derivation` *Get or set a derived variable's expression*

Description

Get a derived variable's derivation formula as a CrunchExpr with `derivation(variable)`. Set (change) a derived variable's derivation with `derivation(variable) <- expression`.

Usage

```
derivation(x)

derivation(x) <- value

is.derived(x)

is.derived(x) <- value

## S4 method for signature 'CrunchVariable'
derivation(x)

## S4 replacement method for signature 'CrunchVariable,ANY'
derivation(x) <- value

## S4 replacement method for signature 'CrunchVariable,`NULL`'
derivation(x) <- value

## S4 method for signature 'CrunchVariable'
is.derived(x)

## S4 replacement method for signature 'CrunchVariable,logical'
is.derived(x) <- value
```

Arguments

`x`	a variable
`value`	a `CrunchExpr` to be used as the derivation (for the setter only) or `NULL` to integrate a derived variable. For `is.derived`, `FALSE` can be used to integrate a derived variable.

Details

To break a derivation link between a derived variable and the originating variable, set the derivation value of the derived variable to `NULL` with `derivation(variable) <- NULL`

`is.derived` can be used to see if a variable is derived or not. Additionally setting a derived variable's `is.derived` to `FALSE` will break the derivation link between two variables.

Value

a CrunchExpr of the derivation for derivation; a logical for is.derived; the variable given in x for is.derived<- returns

Examples

```
## Not run:

ds$derived_v1 <- ds$v1 + 5

derivation(ds$derived_v1)
# Crunch expression: v1 + 5

derivation(ds$derived_v1) <- ds$v1 + 10
derivation(ds$derived_v1)
# Crunch expression: v1 + 10

is.derived(ds$derived_v1)
# TRUE

# to integrate or instantiate the variable in place (remove the link between
# variable v1 and the derivation) you can:
derivation(ds$derived_v1) <- NULL

# after integrating, the derived variable is no longer derived.
is.derived(ds$derived_v1)
# FALSE

# Derivations can be updated with arbitrary expressions.
# Consider a numeric case variable that combines weights
# calculated separately in a separate variable
# for each of several waves:
ds$weight <- makeCaseWhenVariable(
    ds$wave == 1 ~ ds$weight_wave1,
    ds$wave == 2 ~ ds$weight_wave2,
    ds$wave == 3 ~ ds$weight_wave3,
    name = "Weight"
)

# When a new wave is added, update the derivation
# of the weight to add the new condition and source
# column.
derivation(ds$weight) <- caseWhenExpr(
    ds$wave == 1 ~ ds$weight_wave1,
    ds$wave == 2 ~ ds$weight_wave2,
    ds$wave == 3 ~ ds$weight_wave3,
    ds$wave == 4 ~ ds$weight_wave4
)

## End(Not run)
```

deriveArray *Make a Categorical Array or Multiple Response variable*

Description

In most situations we recommend using `deriveArray` which leaves your subvariables in the dataset. `makeArray` *removes* component subvariables from your dataset. Array variables are composed of a set of "subvariables" bound together for display in the app. For example, you might have a set of survey questions that ask how the respondent would rate a TV show from 1-5. Array variables allow you to display all of their ratings in a compact table rather than a set of distinct variables.

Usage

```
deriveArray(subvariables, name, selections, numeric = NULL, ...)

makeArray(subvariables, name, ...)

makeMR(subvariables, name, selections, ...)
```

Arguments

`subvariables`	a list of Variable objects to bind together, or a Dataset subset which contains only the Variables to bind.
`name`	character, the name that the new Categorical Array variable should have.
`selections`	character (preferred, indicating the names of the categories), or numeric (indicating the IDs of the categories in the combined array, which may not be the same as in the original variables - also note that a category's ID is not the same thing as its `numeric_value`). Required for `makeMR`; optional for `deriveArray`; ignored in `makeArray`.
`numeric`	Logical indicating whether the array should be a numeric array or categorical array. `NULL` the default will guess numeric if all variables are known to be numeric and categorical if all are categorical. If any subvariables are created from expressions, then their type cannot be guessed and so `numeric` must be specified.
`...`	Optional additional attributes to set on the new variable.

Value

A VariableDefinition that when added to a Dataset will create the categorical-array or multiple-response variable. `deriveArray` will make a derived array expression (or a derived multiple response expression if `selections` are supplied), while `makeArray` and `makeMR` return an expression that "binds" variables together, removing them from independent existence.

Examples

```
## Not run:
# Categorical Array - Variables from list of variables
ds$enjoy_cat2 <- deriveArray(
    list(ds$enjoy1, ds$enjoy2),
    "Enjoy activities"
)

# Categorical Array - Variables from var catalog
# (result is the same as `ds$enjoy_cat1` above)
ds$enjoy_cat2 <- deriveArray(
    ds[c("enjoy1", "enjoy2")],
    "Enjoy activities v2"
)

# Multiple Response (selections as character names)
ds$enjoy_mr1 <- deriveArray(
    list(ds$enjoy1, ds$enjoy2),
    "Enjoy activities very much or a little",
    selections = c("Very much", "A little")
)

# Numeric Array
ds$rating_numa <- deriveArray(
    list(ds$rating1, ds$rating2),
    "Activity Rating"
)

# Using VarDef to specify metadata (and thus needing to specify type)
ds$enjoy_mr <- deriveArray(
    list(
        VarDef(ds$enjoy1 == "Very much", name = "enjoy brand 1"),
        VarDef(ds$enjoy2 == "Very much", name = "enjoy brand 2")
    ),
    "Enjoy activities with custom names"
)

# Multiple Response (selections as ids, same as ds$enjoy_mr1)
# Be careful `ids(categories(ds$enjoy1))` is not necessarily the same as
# `values(categories(ds$enjoy1))`
ds$enjoy_mr1 <- deriveArray(
    list(ds$enjoy1, ds$enjoy2),
    "Enjoy activities very much or a little v2",
    selections = c(1, 2)
)

## End(Not run)
```

`describe-entity` *Name, alias, and description for Crunch objects*

Description

Name, alias, and description for Crunch objects

Usage

```
name(x)

name(x) <- value

id(x)

value(x)

value(x) <- value

description(x)

description(x) <- value

startDate(x)

startDate(x) <- value

endDate(x)

endDate(x) <- value

alias(object, ...)

alias(x) <- value

digits(x)

digits(x) <- value

uniformBasis(x)

uniformBasis(x) <- value

notes(x)

notes(x) <- value

## S4 method for signature 'AbstractCategory'
name(x)

## S4 replacement method for signature 'AbstractCategory'
name(x) <- value
```

```
## S4 replacement method for signature '`NULL`'
name(x) <- value

## S4 method for signature 'AbstractCategory'
id(x)

## S4 method for signature 'Category'
value(x)

## S4 replacement method for signature 'Category'
value(x) <- value

## S4 method for signature 'Category'
dates(x)

## S4 replacement method for signature 'Category'
dates(x) <- value

## S4 method for signature 'CrunchDataset'
name(x)

## S4 replacement method for signature 'CrunchDataset'
name(x) <- value

## S4 method for signature 'CrunchDataset'
description(x)

## S4 replacement method for signature 'CrunchDataset'
description(x) <- value

## S4 method for signature 'CrunchDataset'
startDate(x)

## S4 replacement method for signature 'CrunchDataset'
startDate(x) <- value

## S4 method for signature 'CrunchDataset'
endDate(x)

## S4 replacement method for signature 'CrunchDataset'
endDate(x) <- value

## S4 method for signature 'CrunchDataset'
id(x)

## S4 method for signature 'CrunchDataset'
notes(x)
```

```
## S4 replacement method for signature 'CrunchDataset'
notes(x) <- value

## S4 replacement method for signature 'CrunchDeck'
name(x) <- value

## S4 method for signature 'CrunchDeck'
description(x)

## S4 replacement method for signature 'CrunchDeck'
description(x) <- value

## S4 method for signature 'Geodata'
description(x)

## S4 replacement method for signature 'Multitable'
name(x) <- value

## S4 replacement method for signature 'ProjectFolder'
name(x) <- value

## S4 method for signature 'ProjectFolder'
name(x)

## S4 method for signature 'ShojiObject'
name(x)

## S4 replacement method for signature 'VariableFolder'
name(x) <- value

## S4 method for signature 'VariableTuple'
alias(object)

## S4 method for signature 'VariableTuple'
description(x)

## S4 method for signature 'VariableTuple'
notes(x)

## S4 method for signature 'CrunchVariable'
name(x)

## S4 replacement method for signature 'CrunchVariable'
name(x) <- value

## S4 method for signature 'CrunchVariable'
id(x)
```

```
## S4 method for signature 'CrunchVariable'
description(x)

## S4 replacement method for signature 'CrunchVariable'
description(x) <- value

## S4 method for signature 'CrunchVariable'
alias(object)

## S4 replacement method for signature 'CrunchVariable'
alias(x) <- value

## S4 method for signature 'CrunchVariable'
notes(x)

## S4 replacement method for signature 'CrunchVariable'
notes(x) <- value

## S4 method for signature 'CrunchVariable'
digits(x)

## S4 replacement method for signature 'NumericVariable'
digits(x) <- value

## S4 replacement method for signature 'CrunchVariable'
digits(x) <- value

## S4 method for signature 'MultipleResponseVariable'
uniformBasis(x)

## S4 replacement method for signature 'MultipleResponseVariable'
uniformBasis(x) <- value
```

Arguments

x	a Dataset or Variable.
value	For the setters, a length-1 character vector to assign
object	Same as x but for the alias method, in order to match the generic from another package. Note that alias and digits are only defined for Variables.
...	additional arguments in the alias generic, ignored.

Value

Getters return the character object in the specified slot; setters return x duly modified.

See Also

Categories describe-catalog

dichotomize *Indicate how categories represent a dichotomized value*

Description

Multiple Response variables are Categorical Arrays in which one or more categories are set as "selected". These methods allow you to view and set that attribute.

Usage

```
is.dichotomized(x)

dichotomize(x, i)

undichotomize(x)

is.selected(x)

is.selected(x) <- value

## S4 method for signature 'Categories'
is.dichotomized(x)

## S4 method for signature 'Categories,numeric'
dichotomize(x, i)

## S4 method for signature 'Categories,logical'
dichotomize(x, i)

## S4 method for signature 'Categories,character'
dichotomize(x, i)

## S4 method for signature 'Categories'
undichotomize(x)

## S4 method for signature 'CategoricalVariable,ANY'
dichotomize(x, i)

## S4 method for signature 'CategoricalArrayVariable,ANY'
dichotomize(x, i)

## S4 method for signature 'CategoricalVariable'
undichotomize(x)

## S4 method for signature 'CategoricalArrayVariable'
undichotomize(x)
```

```
## S4 method for signature 'Categories'
is.selected(x)

## S4 replacement method for signature 'Categories'
is.selected(x) <- value

## S4 method for signature 'Category'
is.selected(x)

## S4 replacement method for signature 'Category'
is.selected(x) <- value
```

Arguments

x	Categories or a Variable subclass that has Categories
i	For the dichotomize methods, the numeric or logical indices of the categories to mark as "selected", or if character, the Category "names". Note that unlike some other categorical variable methods, numeric indices are positional, not with reference to category ids.
value	For is.selected<-, A logical vector indicating whether the category should be selected. For a single category the value should be either TRUE or FALSE. To change the selection status for a Categories object, supply a logical vector which is the same length as the number of categories.

Details

dichotomize lets you specify which categories are "selected", while undichotomize strips that selection information. Dichotomize converts a Categorical Array to a Multiple Response, and undichotomize does the reverse. is.dichotomized reports whether categories have any selected values.

is.selected is lower level and maps more directly onto the "selected" attributes of categories. The best illustration of this difference is that is.selected(categories(var)) returns a logical vector, a value for each category, while is.dichotomized(categories(var)) returns a single TRUE/FALSE value.

Value

Categories or the Variable, (un)dichotomized accordingly

See Also

describe-entity

Examples

```
## Not run:
ds <- newExampleDataset()
is.MR(ds$allpets)
is.dichotomized(categories(ds$allpets))
is.selected(categories(ds$allpets))
```

```
ds$allpets <- undichotomize(ds$allpets)
is.CA(ds$allpets)
ds$allpets <- dichotomize(ds$allpets, "selected")
is.MR(ds$allpets)

## End(Not run)
```

`dimension-comparison` *Column and row comparison*

Description

Comparing a column or row with a baseline column or row. This calculates the z-score for the cells when comparing `x` to the baseline columns

Usage

```
compareCols(cube, ...)

compareRows(cube, ...)

compareDims(cube, dim = c("cols", "rows"), baseline, x)
```

Arguments

`cube`	a cube to calculate the comparison on
`...`	arguments passed from `compareRows()` or `compareCols()` to `compareDims()` (i.e. `baseline` and `x`)
`dim`	which dimension is being compared (`rows` or `cols`, only valid for `compareDims()`)
`baseline`	a character, the column to use as a baseline to compare `x` against
`x`	a character, the column to compare against the baseline

Value

the z-score for the column or row given in `x`

`dimension-comparison-pairwise`

Pairwise column and row comparison

Description

Given a single baseline column compare each other row or column against this baseline. Internally this function uses `compareDims()` iteratively.

Usage

```
compareColsPairwise(cube, ...)

compareRowsPairwise(cube, ...)

compareDimsPairwise(cube, dim = c("cols", "rows"), baseline)
```

Arguments

`cube`	a cube to calculate the comparison on
`...`	arguments passed from `compareRowsPairwise()` or `compareColsPairwise()` to `compareDimsPairwise()` (i.e. `baseline`)
`dim`	which dimension is being compared (`rows` or `cols`, only valid for `compareDims()`)
`baseline`	a character, the column to use as a baseline to compare against all other columns

Details

Warning since there is more than one comparison being made against each baseline the z-scores, and especially the p-values derived from these z-scores should be interpreted with caution. Using standard p-value cutoffs will result in anti-conservative interpretations because of the multiple comparisons problem. Adjustments to p-value cut offs (e.g. Bonferonni correction) should be used when interpreting z-scores from the `compare[Rows|Cols|Dims]Pairwise()` family of functions.

Value

an array of z-score for the all the columns or rows compared to `baseline`. The `baseline` column is all 0s

dimensions *Methods on Cube objects*

Description

These methods provide an array-like interface to the CrunchCube object.

Usage

```
dimensions(x)

dimensions(x) <- value

measures(x)

## S4 method for signature 'CubeDims'
dimnames(x)

## S4 method for signature 'CubeDims'
dim(x)

## S4 method for signature 'CubeDims'
is.na(x)

## S4 method for signature 'CrunchCube'
dimensions(x)

## S4 replacement method for signature 'CrunchCube,CubeDims'
dimensions(x) <- value

## S4 method for signature 'CrunchCube'
dim(x)

## S4 method for signature 'CrunchCube'
dimnames(x)

## S4 method for signature 'CrunchCube'
measures(x)
```

Arguments

x	a CrunchCube or its CubeDims component.
value	for dimensions<- a CubeDims object to overwrite a CrunchCube dimensions

Value

Generally, the same shape of result that each of these functions return when applied to an array object.

See Also

cube-computing base::array

---

dropRows *Permanently delete rows from a dataset*

Description

Permanently delete rows from a dataset

Usage

```
dropRows(dataset, expr)
```

Arguments

dataset	a CrunchDataset
expr	a CrunchLogicalExpr

Value

dataset without the rows indicated by expr

See Also

exclusion for a non-destructive way to suppress rows

Examples

```
## Not run:
ds <- dropRows(ds, ds$gender == "Male")

## End(Not run)
```

duplicated *"duplicated" method for Crunch objects*

Description

"duplicated" method for Crunch objects

Usage

```
duplicated(x, incomparables = FALSE, ...)

## S4 method for signature 'CrunchVariable'
duplicated(x, incomparables = FALSE, ...)

## S4 method for signature 'CrunchExpr'
duplicated(x, incomparables = FALSE, ...)
```

Arguments

x	CrunchVariable or CrunchExpr
incomparables	Ignored
...	Ignored

Value

A CrunchLogicalExpr that evaluates TRUE for all repeated entries after the first occurrence of a value.

See Also

base::duplicated()

email *Extract the email from a User Entity*

Description

Extract the email from a User Entity

Usage

```
email(x)

## S4 method for signature 'UserEntity'
email(x)
```

Arguments

x	a UserEntity returned from me()

Value

a character string of the user's email

---

embedCrunchBox *Get HTML for embedding a CrunchBox*

Description

crunchBox() returns a URL to the box data that it generates, but in order to view it in a CrunchBox or to embed it on a website, you'll need to translate that to the Box's public URL and wrap it in some HTML. This function takes a CrunchBox and returns the HTML which you can embed in a website.

Usage

```
embedCrunchBox(box, title = NULL, logo = NULL, ...)
```

Arguments

box	character URL of the box data, as returned by crunchBox()
title	character title for the Box, to appear above the iframe. Default is NULL, meaning no title shown
logo	character URL of a logo to show instead of a title. Default is NULL, meaning no logo shown. If both logo and title are provided, only the logo will be shown. Note also that logo must be a URL of a hosted image: it cannot be a path to a local file.
...	Additional arguments, not currently used.

Value

Prints the HTML markup to the screen and also returns it invisibly.

See Also

crunchBox()

Examples

```
## Not run:
box <- crunchBox(ds)
embedCrunchBox(box, logo = "//myco.example/img/logo_200px.png")

## End(Not run)
```

exclusion *View and set exclusion filters*

Description

Exclusion filters express logic that defines a set of rows that should be dropped from the dataset. The rows aren't permanently deleted—you can recover them at any time by removing the exclusion filter—but they are omitted from all views and calculations, as if they had been deleted.

Usage

```
exclusion(x)

exclusion(x) <- value
```

Arguments

x	a Dataset
value	an object of class CrunchLogicalExpr, or NULL

Details

Note that exclusion filters work opposite from how "normal" filters work. That is, a regular filter expression defines the subset of rows to operate on: it says "keep these rows." An exclusion filter defines which rows to omit. Applying a filter expression as a query filter will have the opposite effect if applied as an exclusion. Indeed, applying it as both query filter and exclusion at the same time will result in 0 rows.

Value

exclusion returns a CrunchFilter if there is one, else NULL. The setter returns x with the filter set.

exportDataset *Export a dataset to a file*

Description

This function allows you to write a CrunchDataset to a .csv or SPSS .sav file.

Usage

```
exportDataset(
  dataset,
  file,
  format = c("csv", "spss", "parquet"),
  categorical = c("name", "id"),
  na = NULL,
  varlabel = c("name", "description"),
  include.hidden = FALSE,
  ...
)

## S4 method for signature 'CrunchDataset'
write.csv(x, ...)
```

Arguments

dataset	CrunchDataset, which may have been subsetted with a filter expression on the rows and a selection of variables on the columns.
file	character local filename to write to
format	character export format: currently supported values are "csv" and "spss" (and experimental support for "parquet").
categorical	character: export categorical values to CSV as category "name" (default) or "id". Ignored by the SPSS exporter.
na	Similar to the argument in utils::write.table(), 'na' lets you control how missing values are written into the CSV file. Supported values are: 1. NULL, the default, which means that categorical variables will have the category name or id as the value, and numeric, text, and datetime variables will have the missing reason string; 2. A string to use for missing values. 3. "" means that empty cells will be written for missing values for all types.
varlabel	For SPSS export, which Crunch metadata field should be used as variable labels? Default is "name", but "description" is another valid value.
include.hidden	logical: should hidden variables be included? (default: FALSE)
...	additional options. See the API documentation. Currently supported boolean options include 'include_personal' for personal variables (default: FALSE) and 'prefix_subvariables' for SPSS format: whether to include the array variable's name in each of its subvariables "varlabels" (default: FALSE).
x	(for write.csv) CrunchDataset, which may have been subsetted with a filter expression on the rows and a selection of variables on the columns.

Value

Invisibly, file.

Examples

```
## Not run:
csv_file <- exportDataset(ds, "data.csv")
data <- read.csv(csv_file)

# parquet will likely read more quickly and be a smaller download size
parquet_file <- exportDataset(ds, "data.parquet")
# data <- arrow::read_parquet(parquet_file) # The arrow package can read parquet files

## End(Not run)
```

---

exportDeck *Export a Crunch Deck*

Description

Crunch decks can be exported as excel or json files.

Usage

```
exportDeck(deck, file, format = c("xlsx", "pptx", "json"), ...)
```

Arguments

deck	A CrunchDeck
file	The file path to save the exported deck
format	Either "xlsx", "pptx", or "json"
...	Further options to be passed on to the API

Value

the filename (file, if specified, or the the autogenerated file name).

---

expressions *Construct Crunch Expressions from Crunch Database Functions*

Description

Crunch Expressions, i.e. CrunchExpr and CrunchLogicalExpr, encapsulate derivations of Crunch variables, possibly composed of other functions which are only evaluated when sent to the server when creating a variable using VarDef() or using as.vector() to get data. The crunch database functions can be found in the Help Center, and can be called directly via crunchdbFunc()m but many have also been wrapped in native R functions, and are described in the details section below.

Usage

```
crunchdbFunc(fun, x, ...)
```

Arguments

fun	The name of the crunch database function to call
x	An input, a crunch variable, expression or R object
...	Other arguments passed to the database function

Details

Logical expressions

- These logical operators ==, !=, &, |, !,%in% work the same way as their base R counterparts
- is.selected(x) return CrunchLogicalExpr whether a value is in a selected category
- rowAny(x) and rowAll(x) work row-wise on MultipleResponse Variables (and expressions), though na.rm is not implemented for all(x). %ornm% is similar to |, but where "not selected" beats "missing" (so FALSE %ornm% NA is FALSE instead of NA as it would be with FALSE | NA)

Comparisons

- Comparison operators <, <=, >, >= work the same way as their base R counterparts.
- crunchBetween(x, lower, upper, inclusive) to provide lower and upper bounds in a single expression.

Missing data expressions

- is.na(x), is.valid(x) return CrunchLogicalExpr whether a single variable (or expression that creates one) is missing (or not missing).
- rowAnyNA(x), rowAllNA(x) return CrunchLogicalExpr whether any/all values in an array variable (or expression that creates one) are missing.
- complete.cases(x) returns an expression that is "selected" if all cases are non-missing, "missing" if they are all missing, and "other" otherwise.

Selection expressions

- selectCategories(x, selections, collapse = TRUE) takes a categorical variable (or array) and marks categories as selected. selections should be a list of category names or values. If collapse is TRUE, (the default), it collapses the categories to "selected", "other" and "missing", but it is FALSE, then the old categories are preserved.
- asSelected(x) returns an expression that condenses a categorical into 3 categories ("selected", "other" or "missing")
- selectedDepth(x) returns an expression that creates a numeric variable that counts the number of selections across rows of an array variable (or expression that creates one)
- arraySelections(x) returns an expression that takes an array and creates an array with each variable condensed to "selected", "other" or "missing" and an extra subvariable "**any**" that indicates whether any is selected.

- `alterCategoriesExpr(x, categories = NULL, category_order = NULL, subvariables = NULL)` Change the category names, order, or subvariable names of categorical or Array variables (can only modify existing ones, not add or remove categories or subvariables). `categories` is a `Categories` object or a list of lists, each with a `name` indicating the new name, as well as an `id` or `old_name` to identify which category to modify. `category_order` is either a numeric vector indicating category ids or a character vector indicating the names of the categories in the order they should be displayed (note that all categories must be specified). `subvariables` is a list of lists, each with a `name` to rename the subvariable and an `alias`, `old_nam` or `id` to identify the subvariable. When `x` is an expression, all categories and subvariables must be identified by `id`.

Array expressions

- `makeFrame(x, numeric = NULL)` an expression that creates an array from existing variables or expressions, see `deriveArray()` for more details
- `arraySubsetExpr(x, subvars, subvar_id = c("alias", "name", "id"))` Take a subset of an existing array variable, identifying the subvariables by alias, name, or id (if `x` is an expression, you must use id).
- ```
  alterArrayExpr(
    x,
    add = NULL,
    order = NULL,
    order_id = c("alias", "name", "id"),
    remove = NULL,
    remove_id = c("alias", "name", "id"),
    subreferences = NULL,
    subreferences_id = c("alias", "name", "id")
  )
  ```
  Add, reorder, remove or rename subvariables on an an array variable `x`. The `add` argument is a list of variables or expressions, optionally named with the id they should have. `order` and `remove` are vectors of aliases, names or ids (specify which with `order_id`/`remove_id`). The `subreferences` object is a list of lists that are named the alias, name, or id (again specify which with `subreferences_id`) with metadata information like name and alias in the list.

Miscellaneous expressions

- `caseExpr(..., cases)` Create a categorical variable from a set of logical expressions that when met are assigned to a category. See `makeCaseVariable()` for more details.
- `bin(x)` returns a column's values binned into equidistant bins.
- `nchar(x)` returns a numeric value indicating the length of a string (or missing reason) in a `TextVariable` (or expression that creates one)
- `unmissing(x)` for a `NumericVariable` (or expression that creates one) return the values of the data, ignoring the ones set to missing.
- `trim(x, min, max)` for a `NumericVariable` (or expression that creates one) return values that where all values less than `min` have been replaced with `min` and all values greater than `max` have been
- `crunchDifftime(e1, e2, resolution)` Gets the difference between two datetimes as a number with specified resolution units (one of `c("Y", "Q", "M", "W", "D", "h", "m", "s", "ms")`).
  ```

- `datetimeFromCols(year, month, day, hours, minutes, seconds)` create a `Datetime` variable from numeric variables or expressions (`year`, `month`, and `day` are required, but `hours`, `minutes`, and `seconds` are optional)
- `rollup(x, resolution)` sets the resolution of a datetime variable or expression, see `resolution()`

`filter` *Get and set slide analyses*

Description

Slides are composed of analyses, which are effectively `CrunchCubes` with some additional metadata. You can get and set a slide's Analysis Catalog with the `analyses` method, and access an individual analysis with `analysis`. There are also helpers to get and set the components of the analysis such as `filter()`, `weight()`, `transforms()`, `displaySettings()` and `vizSpecs()`. You can also get the `CrunchCube` from an analysis using `cube()`.

Usage

```
filter(x, ...)

filter(x) <- value

## S4 replacement method for signature 'CrunchDeck,ANY'
weight(x) <- value

## S4 replacement method for signature 'CrunchDeck'
filter(x) <- value

## S4 replacement method for signature 'CrunchDeck,ANY'
filters(x) <- value

## S4 method for signature 'CrunchAnalysisSlide'
transforms(x)

## S4 method for signature 'AnalysisCatalog'
transforms(x)

## S4 method for signature 'Analysis'
transforms(x)

## S4 replacement method for signature 'CrunchAnalysisSlide,ANY'
transforms(x) <- value

## S4 replacement method for signature 'AnalysisCatalog,ANY'
transforms(x) <- value

## S4 replacement method for signature 'Analysis,ANY'
```

```
transforms(x) <- value

analyses(x)

analysis(x)

analysis(x) <- value

query(x) <- value

cube(x)

cubes(x)

displaySettings(x)

displaySettings(x) <- value

vizSpecs(x)

vizSpecs(x) <- value

## S4 method for signature 'CrunchSlide'
type(x)

## S4 method for signature 'CrunchAnalysisSlide'
analyses(x)

## S4 method for signature 'CrunchAnalysisSlide'
analysis(x)

## S4 replacement method for signature 'CrunchAnalysisSlide,formula'
analysis(x) <- value

## S4 replacement method for signature 'CrunchAnalysisSlide,Analysis'
analysis(x) <- value

## S4 replacement method for signature 'CrunchAnalysisSlide,list'
analysis(x) <- value

## S4 method for signature 'CrunchAnalysisSlide'
filter(x, ...)

## S4 method for signature 'CrunchAnalysisSlide'
filters(x)

## S4 replacement method for signature 'CrunchAnalysisSlide,ANY'
filters(x) <- value
```

```
## S4 replacement method for signature 'CrunchAnalysisSlide,ANY'
query(x) <- value

## S4 method for signature 'CrunchAnalysisSlide'
cubes(x)

## S4 method for signature 'CrunchAnalysisSlide'
cube(x)

## S4 method for signature 'CrunchAnalysisSlide'
displaySettings(x)

## S4 replacement method for signature 'CrunchAnalysisSlide,ANY'
displaySettings(x) <- value

## S4 method for signature 'CrunchAnalysisSlide'
vizSpecs(x)

## S4 replacement method for signature 'CrunchAnalysisSlide,ANY'
vizSpecs(x) <- value

## S4 method for signature 'AnalysisCatalog'
cubes(x)

## S4 method for signature 'AnalysisCatalog'
displaySettings(x)

## S4 replacement method for signature 'AnalysisCatalog,list'
displaySettings(x) <- value

## S4 method for signature 'AnalysisCatalog'
vizSpecs(x)

## S4 replacement method for signature 'AnalysisCatalog,list'
vizSpecs(x) <- value

## S4 replacement method for signature 'Analysis,formula'
query(x) <- value

formulaToSlideQuery(query, dataset)

## S4 method for signature 'Analysis'
cube(x)

## S4 method for signature 'Analysis'
displaySettings(x)
```

```
## S4 replacement method for signature 'Analysis,ANY'
displaySettings(x) <- value

## S4 method for signature 'Analysis'
vizSpecs(x)

## S4 replacement method for signature 'Analysis,ANY'
vizSpecs(x) <- value

## S4 method for signature 'Analysis'
filter(x, ...)

## S4 method for signature 'Analysis'
filters(x)

## S4 method for signature 'ANY'
filter(x, ...)

## S4 replacement method for signature 'CrunchAnalysisSlide'
filter(x) <- value

## S4 replacement method for signature 'Analysis'
filter(x) <- value

## S4 replacement method for signature 'Analysis,CrunchLogicalExpr'
filters(x) <- value

## S4 replacement method for signature 'Analysis,CrunchFilter'
filters(x) <- value

## S4 replacement method for signature 'Analysis,`NULL`'
filters(x) <- value

## S4 replacement method for signature 'Analysis,list'
filters(x) <- value

slideQueryEnv(weight, filter)

## S4 method for signature 'CrunchDeck'
cubes(x)

## S4 method for signature 'CrunchAnalysisSlide'
weight(x)

## S4 replacement method for signature 'CrunchAnalysisSlide,ANY'
weight(x) <- value

## S4 method for signature 'Analysis'
```

```
weight(x)
```

Arguments

x	a CrunchSlide, AnalysisCatalog, or Analysis
...	ignored
value	for the setter, an object to set it
query	For formulaToSlideQuery(), a formula that specifies the query, as in newSlide(). See Details of crtabs() for more information.
dataset	For formulaToSlideQuery(), a CrunchDataset that the variables in query refer to.
weight	For slideQueryEnv() a crunch variable to use as a weight or NULL to indicate no weight should be used.
filter	for slideQueryEnv(), a CrunchFilter or CrunchExpression to filter the slide.

Details

For more complex objects like displaySettings(), vizSpecs() and transforms(), the API documentation provides more details.

Advanced users of the API can assign a list to analysis<- to specify settings on the analyses that are not otherwise available in rcrunch. The helpers formulaToSlideQuery() and slideQueryEnv() help you create objects for the query and query_environment.

Examples

```
## Not run:
# Examples of setting analysis details (in general these setters work on
# the slide, analysis catalog and analysis, but for brevity the examples only
# show on the slide)

# Change the filter
filters(slide) <- NULL # to remove a filter
filters(slide) <- filters(ds)[["My filter"]]
filters(slide) <- list( # Can set multiple filter
    filters(ds)[["My filter"]],
    ds$age_grp == "18-35"
)
filters(deck) <- filters(ds)[["My filter"]] # Can set the same filter on a whole deck too

# Change the weight
weight(slide) <- NULL # to remove
weight(slide) <- ds$weight
weight(deck) <- ds$weight # Can set the same weight on a whole deck too

# Change the transforms
transforms(slide) <- list(rows_dimension = makeDimTransform(hide = "Neutral"))

# Change the displaySettings
displaySettings(slide) <- list(vizType = "groupedBarPlot")
```

```
# Change the vizSpecs
# viz_specs can get quite long, see
# https://crunch.io/api/reference/#post-/datasets/-dataset_id-/decks/-deck_id-/slides/
vizSpecs(slide) <- viz_specs

# Change the query
#' query(slide) <- ~ cyl + wt

## End(Not run)
```

`filters` *Get or set a dataset's filters*

Description

You can build and save filters in the Crunch web app, and these filters are stored in a `FilterCatalog`. This function allows you to retrieve and modify those filters.

Usage

```
filters(x)

filters(x) <- value

## S4 method for signature 'CrunchDataset'
filters(x)

## S4 replacement method for signature 'CrunchDataset,ANY'
filters(x) <- value
```

Arguments

`x`	a CrunchDataset
`value`	for the setter, a FilterCatalog

Value

an object of class FilterCatalog containing references to Filter entities usable in the web application. (Setter returns the Dataset.)

flipArrays *Rearrange array subvariables*

Description

Sometimes it is useful to group subvariables across arrays in order to compare them more easily. This function generates a set of derived views of common subvariables across arrays. Because they are derived, they share data with the underlying array variables, and they are thus automatically updated when new data is appended.

Usage

```
flipArrays(variables, suffix = ", flipped")
```

Arguments

variables	List of variables, a variable catalog, or a dataset subset containing the categorical array or multiple response variables you want to rearrange.
suffix	character string to append to the new variable names. Pass "" if you don't want it to append anything.

Value

A list of derived VariableDefinitions, one per unique subvariable name across all variables. Each variable in variables that contains this subvariable will appear as a subvariable in these new derived array definitions. Use addVariables to add these to your dataset.

Examples

```
## Not run:
ds <- addVariables(ds, flipArrays(ds[c("petloc", "petloc2")], suffix = ", rearranged"))

## End(Not run)
```

folder *Find and move entities to a new folder*

Description

Find and move entities to a new folder

Usage

```
folder(x)

folder(x) <- value
```

Arguments

x	For `folder`, a Variable to find. For folder assignment, a Variable, selection of variables in a Dataset, or any other object that can be moved to a folder.
value	For assignment, a character "path" to the folder: either a vector of nested folder names or a single string with nested folders separated by a delimiter ("/" default)

Value

`folder` returns the parent folder of `x`, or `NULL` if the `x` is the root level. `folder<-` returns the `x` input, having been moved to the requested location.

See Also

`mv()` `cd()`

Examples

```
## Not run:
ds <- loadDataset("Example survey")
folder(ds$income) <- "Demographics/Economic"
folder(ds$income)
## [1] "Demographics"    "Economic"

## End(Not run)
```

---

forceVariableCatalog *Force variables catalog to be loaded*

Description

Variables catalogs are generally loaded lazily, but this function allows you to force them to be loaded once.

Usage

```
forceVariableCatalog(x)
```

Arguments

x	A crunch dataset

Details

The `forceVariableCatalog()` function is probably most useful when writing tests because it allows you to be more certain about when API calls are made.

Another situation where you may care about when API calls for loading the variables are made is when you are loading many datasets at the same time (~15+) and referring to their variables later. In this situation, it can be faster to turn off the variables catalog with the option `crunch.lazy.variable.catalog` because there is a limit to the number of datasets your user can hold open at the same time and so at some point the server will have to unload and then reload the datasets. However, it's probably even faster if you are able to alter your code so that it operates on datasets sequentially.

Value

A dataset with it's variable catalogs filled in

`forkDataset` *Create a fork of a dataset*

Description

Forking a dataset makes a copy of the data that is linked by Crunch's version control system to the original dataset. When you make edits to a fork, users of the original dataset do not see the changes.

Usage

```
forkDataset(dataset, name = defaultForkName(dataset), draft = FALSE, ...)
```

Arguments

`dataset`	The `CrunchDataset` to fork
`name`	character name to give the fork. If omitted, one will be provided for you
`draft`	logical: Should the dataset be a draft, visible only to those with edit permissions? Default is `FALSE`.
`...`	Additional dataset metadata to provide to the fork

Details

A common strategy for revising a dataset that has been shared with others is to fork it, make changes to the fork, and then merge those changes back into the original dataset. This workflow allows you to edit a dataset and review changes before publishing them, so that you don't accidentally send your clients incorrect data. For more on this workflow, see `vignette("fork-and-merge", package = "crunch")`.

Value

The new fork, a `CrunchDataset`.

See Also

mergeFork()

---

getTeams *Retrieve your teams*

Description

Teams contain a list of users. You can grant access to a group of users by inviting the team. You can also share a set of datasets with a user all at once by adding the user to a team that contains those datasets.

Usage

```
getTeams()
```

Details

`getTeams()` returns your `TeamCatalog`. You can extract an individual team by name, or create a team by assigning into the function. To create a team by assignment, assign a list to `teams("myteam") <- value_list`. The `value_list` can either empty (to just create a team with that name), or can contain a "members" element with the emails or URLs of users to add to the team. Users can be also be added later with the `members<-` method.

Value

A `TeamCatalog`. Extract an individual team by name. Create a team by assigning in with a new name.

See Also

members

---

http-methods *HTTP methods for communicating with the Crunch API*

Description

These methods let you communicate with the Crunch API, for more background see Crunch Internals.

Usage

```
crGET(...)

crPUT(...)

crPATCH(...)

crPOST(...)

crDELETE(...)
```

Arguments

...	see crunchAPI for details. url is the first named argument and is required; body is also required for PUT, PATCH, and POST.

Value

Depends on the response status of the HTTP request and any custom handlers.

---

index *Get the body of a Catalog*

Description

The core of Catalog data is in its "index". These methods get and set that slot.

Usage

```
index(x)

index(x) <- value

## S4 method for signature 'ShojiCatalog'
index(x)

## S4 replacement method for signature 'ShojiCatalog'
index(x) <- value
```

Arguments

x	a Catalog (VariableCatalog, Subvariables, or similar object)
value	For the setters, an appropriate-length list to assign

Value

Getters return the list object in the "index" slot; setters return x duly modified.

index.table *Calculate an index table for a CrunchCube*

Description

Index tables are percentages of percentages. They take the percentage from `prop.table(cube, margin)` and, by default, divide that by the proportions of the other margin. The `baseline` argument can be used to provide baseline proportions to compare against.

Usage

```
index.table(x, margin, baseline)
```

Arguments

`x`	A CrunchCube to calculate index table for
`margin`	which margin to index against (1 for rows, 2 for columns)
`baseline`	an arbitrary set of proportions to compare the table given in `x` to. Useful for comparing two separate cubes. `baseline` must have the same length as the extent of the dimension given in `margin`.

Details

`index.table()` is only implemented for 2 dimensional cubes. If you need to calculate indexes for a higher dimension Cube, please slice the cube first.

Value

an array of percentages indexed to the margin provided

Examples

```
## Not run:
cube_object
#    v7
# v4  C E
#   B 5 2
#   C 5 3
index.table(cube_object, 1)
#    v7
# v4         C         E
#   B 107.1429  85.71429
#   C  93.7500 112.50000
index.table(cube_object, 2)
#    v7
# v4    C   E
#   B 100  80
#   C 100 120
```

```
index.table(cube_object, 2, c(0.6, 0.4))
#    v7
# v4          C         E
#   B  83.33333  66.66667
#   C 125.00000 150.00000

## End(Not run)
```

`Insertions-class` *Insert categories in transformations*

Description

Insertions allow you to insert new categories into a categorical-like response on a variable's transformations.

Usage

```
Insertions(..., data = NULL)

Insertion(...)

.Insertion(..., data = NULL)

anchor(x, ...)

anchors(x)

anchor(x) <- value

arguments(x, ...)

arguments(x) <- value

func(x)

funcs(x)

## S4 replacement method for signature 'Insertion'
anchor(x) <- value

## S4 replacement method for signature 'Subtotal'
anchor(x) <- value

## S4 replacement method for signature 'Heading'
anchor(x) <- value
```

```
## S4 replacement method for signature 'SummaryStat'
anchor(x) <- value

## S4 replacement method for signature 'Insertion,ANY'
subtotals(x) <- value

## S4 replacement method for signature 'Insertion'
arguments(x) <- value

## S4 replacement method for signature 'Subtotal'
arguments(x) <- value

## S4 replacement method for signature 'Heading'
arguments(x) <- value

## S4 replacement method for signature 'SummaryStat'
arguments(x) <- value

## S4 method for signature 'Insertion'
arguments(x)

## S4 method for signature 'Subtotal'
arguments(x, var_items)

## S4 method for signature 'Heading'
arguments(x)

## S4 method for signature 'SummaryStat'
arguments(x, var_items)

## S4 method for signature 'Insertion'
anchor(x, ...)

## S4 method for signature 'Subtotal'
anchor(x, var_items)

## S4 method for signature 'Heading'
anchor(x, var_items)

## S4 method for signature 'SummaryStat'
anchor(x, var_items)

## S4 method for signature 'Insertion'
func(x)

## S4 method for signature 'Subtotal'
func(x)
```

```
## S4 method for signature 'Heading'
func(x)

## S4 method for signature 'SummaryStat'
func(x)

## S4 method for signature 'Insertions'
anchors(x)

## S4 method for signature 'Insertions'
funcs(x)
```

Arguments

...	additional arguments to [, ignored
data	For the constructor functions Insertion and Insertions, you can either pass in attributes via ... or you can create the objects with a fully defined list representation of the objects via the data argument. See the examples.
x	For the attribute getters and setters, an object of class Insertion or Insertions
value	For [<-, the replacement Insertion to insert
var_items	categories (from categories()) or subvariables (from subvariables() to used by the arguments and anchor methods when needed to translate between category/subvariable names and category ids/aliases.

Working with Insertions

Insertions are used to add information about a variable or CrunchCube that extends the data in the dataset but does not alter it. This new data includes: aggregations like subtotals that sum the count of more than on category together or headings which can be added between categories.

Insertions objects are containers for individual Insertion objects. The individual Insertions contain all the information needed to calculate, apply, and display insertions to CrunchCubes and categorical variables.

An Insertion must have two properties:

- anchor - which is the id of the category the insertion should follow
- name - the string to display

Additionally, Insertions may also have the following two properties (though if they have one, they must have the other):

- function - the function to use to aggregate (e.g. "subtotal")
- args - the category ids to use as operands to the function above.

Although it is possible to make both subtotals and headings using Insertion alone, it is much easier and safer to use the functions Subtotal() and Heading() instead. Not only are they more transparent, they also are quicker to type, accept both category names as well as ids, and have easier to remember argument names.

interactVariables *Create a variable by interacting categorical variables*

Description

interactVariables takes two or more variables and creates a new one that is the cartesian product expansion of their unique values. For example, if we cross ethnicity (with 2 categories) and race (with 4 categories), the new variable would have 8 valid categories (e.g. black:hispanic, white:hispanic, black:non-hispanic, etc.) and 7 categories where at least one of the variables is missing (e.g. white:No Data).

Usage

```
interactVariables(..., name, collapse_missings = FALSE)
```

Arguments

...	a sequence of categorical variables to make an interaction from as well as other properties to pass about the case variable (i.e. alias, description)
name	a character to use as the name for the interaction variable
collapse_missings	a logical indicating whether to combine all new categories that are formed from existing missing categories into a single one (defaults to FALSE).

Value

A VariableDefinition that creates the new interaction variable.

Examples

```
## Not run:
ds$ethn_race <- interactVariables(
  ds$ethnicity, ds$race, name = "Interaction of ethnicity and race"
)

## End(Not run)
```

is-na-categories *is.na for Categories*

Description

Crunch categorical variables allow you to set multiple categories as missing. For instance, you might have "not answered" and "doesn't know" both coded as missing. This function returns a logical vector of all dataset entries that fall into any of the missing categories. It also allows you to append additional categories to the list of missing categories using the setter.

Usage

```
## S4 method for signature 'Categories'
is.na(x)

## S4 replacement method for signature 'Categories,character'
is.na(x) <- value

## S4 replacement method for signature 'Categories,logical'
is.na(x) <- value

## S4 method for signature 'Category'
is.na(x)

## S4 replacement method for signature 'Category,logical'
is.na(x) <- value
```

Arguments

x	Categories or a single Category
value	To change the missingness of categories, supply either: 1. a logical vector of equal length of the categories (or length 1 for the Category method); or 2. the names of the categories to mark as missing. If supplying the latter, any categories already indicated as missing will remain missing.

Value

Getters return logical, a named vector in the case of the Categories method; setters return x duly modified.

---

is-public *View and modify the "public" attribute of artifacts*

Description

View and modify whether all dataset viewers have access to the dataset. This will return FALSE if the dataset is in draft.

Usage

```
is.public(x)

is.public(x) <- value

## S4 method for signature 'CrunchFilter'
is.public(x)
```

```
## S4 replacement method for signature 'CrunchFilter'
is.public(x) <- value

## S4 method for signature 'CrunchDeck'
is.public(x)

## S4 replacement method for signature 'CrunchDeck'
is.public(x) <- value

## S4 method for signature 'MultitableCatalog'
is.public(x)

## S4 replacement method for signature 'MultitableCatalog'
is.public(x) <- value

## S4 method for signature 'Multitable'
is.public(x)

## S4 replacement method for signature 'Multitable'
is.public(x) <- value
```

Arguments

x	a Crunch object
value	an attribute to set

Value

For `is.public`, a logical value for whether the object is flagged as shared with all dataset viewers. (Its setter thus takes a logical value as well.) Catalogs of datasets return a vector of logicals corresponding to the length of the catalog, while entities return a single value.

---

is.editor *Read and set edit privileges*

Description

Read and set edit privileges

Usage

```
is.editor(x)

is.editor(x) <- value

## S4 method for signature 'MemberCatalog'
```

```
is.editor(x)

## S4 replacement method for signature 'MemberCatalog,logical'
is.editor(x) <- value

## S4 method for signature 'PermissionCatalog'
is.editor(x)

## S4 method for signature 'PermissionTuple'
is.editor(x)
```

Arguments

x	PermissionCatalog or MemberCatalog
value	For the setter, logical: should the indicated users be allowed to edit the associated object?

Value

`is.editor` returns a logical vector corresponding to whether the users in the catalog can edit or not.`is.editor<-` returns the catalog, modified.

---

is.VariableDefinition *Test whether a Crunch object belongs to a class*

Description

Test whether a Crunch object belongs to a class

Usage

```
is.VariableDefinition(x)

is.VarDef(x)

is.script(x)

is.dataset(x)

is.CrunchExpr(x)

is.Expr(x)

is.Geodata(x)

is.shoji(x)
```

```
is.variable(x)

is.Numeric(x)

is.Categorical(x)

is.Text(x)

is.Datetime(x)

is.Multiple(x)

is.MR(x)

is.MultipleResponse(x)

is.CA(x)

is.CategoricalArray(x)

is.NumericArray(x)

is.Array(x)
```

Arguments

x	an object

Value

logical

---

is.weight<- *Dataset weights*

Description

`weight` lets you view and set your user's currently applied weight on the server. `weightVariables` lets you view all of the variables that have been designated as valid to use as weights.

Usage

```
is.weight(x) <- value

weight(x)

weight(x) <- value
```

```
## S4 replacement method for signature 'Analysis,CrunchVariable'
weight(x) <- value

## S4 replacement method for signature 'Analysis,`NULL`'
weight(x) <- value

## S4 method for signature 'CrunchDataset'
weight(x)

## S4 replacement method for signature 'CrunchDataset,ANY'
weight(x) <- value

is.weight(x)

## S4 replacement method for signature 'NumericVariable'
is.weight(x) <- value
```

Arguments

x	a Dataset
value	a Variable, VariableDefinition, or NULL. If a VariableDefinition is passed, the variable will first be created and then set as the datasets weight. Set to NULL to remove existing weights from the dataset.

Value

For the weight getter, a Variable if there is a weight, else NULL. For the setter, x, modified accordingly. weightVariables returns the aliases (or names, according to options(crunch.namekey.dataset)), of the variables designated as weights.

See Also

weightVariables() makeWeight()

---

joinDatasets *Add columns from one dataset to another, joining on a key*

Description

As base::merge() does for data.frames, this function takes two datasets, matches rows based on a specified key variable, and adds columns from one to the other.

Usage

```
joinDatasets(
  x,
  y,
  by = intersect(names(x), names(y)),
  by.x = by,
  by.y = by,
  all = FALSE,
  all.x = TRUE,
  all.y = FALSE,
  copy = TRUE
)

extendDataset(
  x,
  y,
  by = intersect(names(x), names(y)),
  by.x = by,
  by.y = by,
  all = FALSE,
  all.x = TRUE,
  all.y = FALSE,
  ...
)

## S3 method for class 'CrunchDataset'
merge(
  x,
  y,
  by = intersect(names(x), names(y)),
  by.x = by,
  by.y = by,
  all = FALSE,
  all.x = TRUE,
  all.y = FALSE,
  ...
)
```

Arguments

x	CrunchDataset to add data to
y	CrunchDataset to copy data from. May be filtered by rows and/or columns.
by	character, optional shortcut for specifying by.x and by.y by alias if the key variables have the same alias in both datasets.
by.x	CrunchVariable in x on which to join, or the alias (following crunch.namekey.dataset of a variable. Must be type numeric or text and have all unique, non-missing values.

by.y	CrunchVariable in y on which to join, or the alias (following crunch.namekey.dataset of a variable. Must be type numeric or text and have all unique, non-missing values.
all	logical: should all rows in x and y be kept, i.e. a "full outer" join? Only FALSE is currently supported.
all.x	logical: should all rows in x be kept, i.e. a "left outer" join? Only TRUE is currently supported.
all.y	logical: should all rows in y be kept, i.e. a "right outer" join? Only FALSE is currently supported.
copy	logical: make a virtual or materialized join. Default is TRUE, which means materialized. Virtual joins are in fact not currently implemented, so the default is the only valid value.
...	additional arguments, ignored

Details

Since joining two datasets can sometimes produce unexpected results if the keys differ between the two datasets, you may want to follow the fork-edit-merge workflow for this operation. To do this, fork the dataset with forkDataset(), join the new data to the fork, ensure that the resulting dataset is correct, and merge it back to the original dataset with mergeFork(). For more, see vignette("fork-and-merge", package = "crunch").

Value

x extended by the columns of y, matched on the "by" variables.

---

listDatasets *Get the names of datasets in a project*

Description

listDatasets() is a convenience function for quickly seeing what datasets are in a project. It is equivalent to names(datasets(proj)), with some additional optional arguments.

Usage

```
listDatasets(
  kind = c("active", "all", "archived"),
  project = NULL,
  refresh = FALSE,
  shiny = FALSE
)
```

Arguments

kind	character specifying whether to look in active, archived, or all datasets. Default is "active", i.e. non-archived.
project	ProjectFolder entity, character name of a project, or NULL, the default. If a Project entity or reference is supplied, the function will display datasets from that Project's datasets. If NULL, your personal folder will be used.
refresh	logical: should the function check the Crunch API for new datasets? Default is FALSE.
shiny	logical: launch a Shiny gadget to help select the right dataset. The gadget will return a valid loadDataset() call which loads the selected dataset. The gadget requires RStudio, as well as the crunchy package.

Details

Specifying listDatasets(shiny = TRUE) will, instead of printing dataset names, load a Shiny gadget that provides a GUI for navigating the project tree to find a dataset, if you're running in RStudio.

Value

A character vector of dataset names, each of which would be a valid input for loadDataset()

---

loadDataset *Load a Crunch Dataset*

Description

This function gives you a Dataset object, which refers to a dataset hosted on the Crunch platform. With this Dataset, you can perform lots of data cleaning and analysis as if the dataset were fully resident on your computer, without having to pull data locally.

Usage

```
loadDataset(
  dataset,
  kind = c("active", "all", "archived"),
  project = NULL,
  refresh = FALSE
)
```

Arguments

dataset	character, the name or path to a Crunch dataset to load, or a dataset URL. If dataset is a path to a dataset in a project, the path will be be parsed and walked, relative to project if specified, and the function will look for the dataset inside that project. If no path is specified and no project provided, the function will call a search API to do an exact string match on dataset names.

kind	character specifying whether to look in active, archived, or all datasets. Default is "active", i.e. non-archived.
project	ProjectFolder entity, character name (path) to a project, or NULL, the default. If a Project entity or reference is supplied, either here or as a path in dataset, the dataset lookup will be limited to that project only.
refresh	logical: should the function check the Crunch API for new datasets? Default is FALSE.

Details

You can specify a dataset to load by its human-friendly "name", possibly also by indicating a project (folder) to find it in. This makes code more readable, but it does mean that if the dataset is renamed or moved to a different folder, your code may no longer work. The fastest, most reliable way to use loadDataset() is to provide a URL to the dataset–the dataset's URL will never change.

Value

An object of class CrunchDataset.

See Also

See cd() for details of parsing and walking dataset folder/project paths.

Examples

```
## Not run:
ds <- loadDatasets("A special dataset")
ds2 <- loadDatasets("~/My dataset")
ds3 <- loadDataset("My dataset", project = "~") # Same as ds2
ds4 <- loadDataset("https://app.crunch.io/api/datasets/bd3ad2/")

## End(Not run)
```

---

lock *Lock and unlock a dataset for editing*

Description

Crunch allows a single active editor. If you have edit privileges but are not currently editing the dataset, you must unlock the dataset before making changes. You may then lock the dataset when you're done editing.

Usage

```
lock(dataset)

unlock(dataset)
```

Arguments

dataset	a CrunchDataset

Value

dataset, invisibly, after having set the current editor.

---

logout *DEPRECATED! Authenticate with the Crunch API*

Description

A deprecated method to authenticate to the crunch.io API. See crunch-api-key for the currently supported method, as login(), logout() and resetPassword() no longer work.

Usage

```
logout()

login(...)

resetPassword(...)
```

Arguments

...	Ignored

---

makeArrayGadget *Array builder*

Description

Launch array builder gadget

Usage

```
makeArrayGadget()
```

Details

Categorical Array and Multiple Response variables can be difficult to construct without being able to investigate the available variables, and their categories. This shiny gadget lets you select subvariables from the dataset list, and ensures that those variables have consistent categories. To use the gadget you must have at least one CrunchDataset loaded into the global environment.

Value

a valid call to makeArray() or makeMR()

`makeCaseVariable` *Make a case variable*

Description

The `makeCaseVariable` function derives a variable using values from other variables. These are evaluated in the order they are supplied in the list as the `cases` argument (they proceed in an IF, ELSE IF, ELSE IF, ..., ELSE fashion); the first one that matches selects the corresponding value from the case list. `caseExpr()` is a version that returns an expression that could be used when creating complex variables, see `expressions` for more details.

Usage

```
makeCaseVariable(..., cases, data = NULL, name)

caseExpr(..., cases)
```

Arguments

`...`	a sequence of named expressions to use as cases as well as other properties to pass about the case variable (i.e. alias, description)
`cases`	a list of lists with each case condition to use each must include at least a `name` and an `expression` element. Cases may also include `missing` (logical) and `numeric_value` (numeric).
`data`	(optional) a crunch dataset to use. Specifying this means you don't have to put `dataset$` in front of each variable name.
`name`	a character to use as the name of the case variable to create

Details

There are two ways to specify cases, but you must pick only one (note these two will produce the same case variable):

1. When you just want to specify conditions, you can use named conditions: `makeCaseVariable(case1=ds$v1 == 1, case2=ds$v2 == 2, name="new case")`
2. You can also use the `cases` argument, which is useful when you want to provide category ids, numeric values, or missingness: `makeCaseVariable( cases=list( list(expression=ds$v1 == 1, name="case1"), list(expression=ds$v2 == 2, name="case2") ), name="new case" )`

Rows in the dataset that do not match any of the provided "cases" will be assigned to an "else" category. By default, Crunch will use the system missing "No Data" category. Alternatively, you can provide an else case definition for these rows by including as the last "case" you provide one with its `expression` set to the string "else". See the examples for details.

Value

A `VariableDefinition` that will create the new case variable when assigned into the Dataset.

Examples

```
## Not run:
makeCaseVariable(case1 = ds$v1 == 1, case2 = ds$v2 == 2, name = "new case")
makeCaseVariable(
    cases = list(
        list(expression = ds$v1 == 1, name = "case1"),
        list(expression = ds$v2 == 2, name = "case2")
    ),
    name = "new case"
)

# different ways to specify else cases
makeCaseVariable(
    cases = list(
        list(expression = ds$v1 == 1, name = "case1"),
        list(expression = ds$v2 == 2, name = "case2"),
        list(expression = "else", name = "other")
    ),
    name = "new case"
)
makeCaseVariable(case1 = ds$v1 == 1, case2 = ds$v2 == 2, other = "else", name = "new case")

# the dataset can be specified with data=
makeCaseVariable(case1 = v1 == 1, case2 = v2 == 2, data = ds, name = "new case")

## End(Not run)
```

---

makeCaseWhenVariable *Create a categorical or numeric variable based on conditions*

Description

Conditions are specified using a series of formulas: the left-hand side is the condition that must be true (a `CrunchLogicalExpr`) and the right-hand side is where to get the value if the condition on the left-hand side is true. When creating a categorical variable, the right-hand side must be a `Category` or a categorical `CrunchVariable` or `CrunchExpression`, while for numeric variables it is a single number or variable or expression.

Usage

```
makeCaseWhenVariable(..., data = NULL, cases = NULL, name, type = NULL)

caseWhenExpr(..., data = NULL, cases = NULL, type = NULL)
```

Arguments

`...`	formulas where the left hand side is a `CrunchLogicalExpression` (or `TRUE` to indicate the "else" case that will be met if all the other expression are not met) and the right hand side is a CrunchVariable that should be filled in, a `Category`

object describing the Category it should be used, a string which will be the name of the Category or NA to indicate that it should be replaced with the system missing value. For makeCaseWhenVariable() non-formula arguments will be passed to [VarDef()]

data — A CrunchDataset to use if variable aliases are left bare in the formulas.

cases — A list of formulas that match the description in ... or a list of lists with named items, "expression" (like the left-hand side of the formulas above), "fill" for a variable to fill in, or "name", "id", and other items that describe a category.

name — For makeCaseWhenVariable() the name of the variable to create.

type — The type of the variable to output (either "categorical" or "numeric"), only required if all fills are expressions and so their type cannot be guessed automatically.

Value

makeCaseWhenVariable() returns a VariableDefinition and caseWhenExpr() returns an expression

Examples

```
## Not run:
# Creating categorical variables
ds$new_var <- makeCaseWhenVariable(
   ds$x %in% c("a", "b") ~ ds$y, # can fill with a variable
   ds$x %in% c("c", "d") ~ Category(name = "c or d", numeric_value = 10), # or a Category
   # If none of the categories match, will be set to missing unless you
   # specify an "else" case with `TRUE` in the left hand side
   TRUE ~ Category(name = "catch all"),
   name = "combined x and y"
)

ds$brand_x_pref <- makeCaseWhenVariable(
   ds$brand[[1]] == "Brand X" ~ ds$pref[[1]],
   ds$brand[[2]] == "Brand X" ~ ds$pref[[2]],
   ds$brand[[3]] == "Brand X" ~ ds$pref[[3]],
   name = "brand x preference"
)

ds$x_among_aware <- makeCaseWhenVariable(
   ds$aware_x == "Yes" ~ ds$x,
   TRUE ~ Category(name = "(Not aware)", missing = TRUE),
   name = "x (among respondents aware of x)"
)

ds$new_num_var <- makeCaseWhenVariable(
  ds$x %in% c("a", "b") ~ ds$z, # LHS as before, RHS can be numeric variables,
  ds$x == "c" ~ ds$z * 10, # expressions,
  ds$x == "d" ~ 100, # or numbers
  name = "New numeric variable"
)
```

```
ds$capped_z <- makeCaseWhenVariable(
  ds$z > 10 ~ 10,
  TRUE ~ ds$z,
  name = "Capped z"
)

# caseWhenExpr can be used inside other expressions
ds$brand_x_prefer_high <- VarDef(
   selectCategories(
       caseWhenExpr(
           ds$brand_shown[[1]] == "Brand X" ~ ds$ratings[[1]],
           ds$brand_shown[[2]] == "Brand X" ~ ds$ratings[[2]],
           ds$brand_shown[[3]] == "Brand X" ~ ds$ratings[[3]]
       ),
       c("Best", "Very Good")
   ),
   name = "Rate X highly"
)

# Using lists in `cases` argument can be helpful when working programmatically
source_var <- ds$x
inclusion_condition <- ds$skipped_x != "Yes"

ds$x2_among_aware <- makeCaseWhenVariable(
   cases = list(list(fill = source_var, expression = inclusion_condition)),
   name = "x2 among aware"
)

## End(Not run)
```

`makeDimTransform` *Helper for creating slide dimension transformations for dashboards and exports*

Description

When displayed in a Crunch Dashboard or exported, crunch slides can have transformations that customize their display. This is a helper to form the correct data structure for the functions `newSlide()` for setting the transformation directly. For more details see the API documentation

Usage

```
makeDimTransform(
  colors = NULL,
  hide = NULL,
  rename = NULL,
  order = NULL,
  name = NULL,
```

```
  description = NULL,
  ...
)
```

Arguments

colors	A crunch AnalyticPalettes (palettes()) or a vector of color RGB hex color codes that will be used for the color of graphs in the dashboard (used in the order of appearance of categories/subvariables).
hide	A vector of category names/ids or subvariable names/aliases to hide from display
rename	A named vector of category names/ids or subvariable names/aliases to override their default values
order	A vector of category names/ids or subvariable names/aliases to override the default ordering of the dimension.
name	A name for the dimension, overrides the variable's name
description	A description for the dimension, overrides the variable's description
...	Other arguments, passed directly to the API for future expansion

Examples

```
## Not run:
# Hiding an element
transforms(slide) <- list(rows_dimension = makeDimTransform(hide = "Neutral"))

# Using an existing saved palette
transforms(slide) <- list(rows_dimension = makeDimTransform(
    colors = defaultPalette(ds)
))

# Setting specific colors
transform(slide) <- list(rows_dimension = makeDimTransform(
     colors = c("#af8dc3", "#f7f7f7", "#7fbf7b")
))

# Reordering & renaming elements
transforms(slide) <- list(
     rows_dimension = makeDimTransform(
         rename = c("V. Good" = "Very Good", "V. Bad" = "Very Bad"),
         order = 5:1
     ),
     columns_dimension = makeDimTransform(order = c("Brand X", "Brand A", "Brand B"))
)

## End(Not run)
```

makeMRFromText *Create Multiple Response Variable from Delimited lists*

Description

Surveys often record multiple response questions in delimited lists where each respondent's selections are separated by a delimiter like ; or |. This function breaks the delimited responses into subvariables, uploads those subvariables to Crunch, and finally creates a multiple response variable from them.

Usage

```
makeMRFromText(
  var,
  delim,
  name,
  selected = "selected",
  not_selected = "not_selected",
  unanswered = NA,
  ...
)
```

Arguments

var	The variable containing the delimited responses
delim	The delimiter separating the responses
name	The name of the resulting MR variable
selected	A character string used to indicate a selection, defaults to "selected"
not_selected	Character string identifying non-selection, defaults to "not_selected"
unanswered	Character string indicating non-response, defaults to NA.
...	Other arguments to be passed on to makeMR()

Value

a Multiple response variable definition

makeWeight *Generate a weight variable*

Description

This function allows you to generate a weight variable by supplying a set of categorical variables and the target distribution for each of the variables' categories. Weights are computed by iteratively 'raking' conditional 'cells' to the provided marginal targets.

Usage

```
makeWeight(..., name)
```

Arguments

...	A series of expressions of the form `variable ~ target_weights`. The variable must be a categorical Crunch variable, and the target weights must be a numeric vector whose length should be equal to the number of categories contained in the variable, and whose sum is equal to 100 or 1. If you supply fewer target weights than there are categories `makeWeight` will pad the target weight vector with 0s.
name	The name of the resulting variable

Details

For instance, if you wanted to create a weight variable which equally weighted four categories stored in `ds$var` you would call `ds$weight1 <- makeWeight(ds$var ~ c(25, 25, 25, 25), name = "weight1")`. Note that `makeWeight` returns a `VariableDefinition`, an expression that when assigned into your Dataset becomes a derived variable. This does not on its own set the new variable as "the weight" for your dataset. To set that attribute, use `weight()`. Alternatively, you can also create the variable and set the weight attribute in one step with `weight(ds) <- makeWeight(ds$var ~ c(25, 25, 25, 25), name = "weight1")`.

Value

A crunch `VariableDefinition()` of the weight variable

See Also

`weight<-()`; `settings()` for the "default weight" for other dataset viewers.

Examples

```
## Not run:
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)
ds <- newDataset(mtcars)
# Create a new "raked" variable
```

```
ds$weight <- makeWeight(ds$cyl ~ c(30, 30, 40, 0),
    ds$gear ~ c(20, 20, 60, 0),
    name = "weight"
)
summary(ds$weight)
# ds$weight is not "the weight" for the dataset unless you set it:
weight(ds) <- ds$weight
# Or, you can create the variable and set as weight in one step:
weight(ds) <- makeWeight(ds$var ~ c(25, 25, 25, 25), name = "weight2")

## End(Not run)
```

`matchCatToFeat` *Match categories with features from geodata*

Description

Match categories with features from geodata

Usage

```
matchCatToFeat(categories, all_features = availableGeodataFeatures())
```

Arguments

`categories`	a vector of categories to match
`all_features`	a dataframe of all available geodata features. (default: downloaded from Crunch servers)

Value

geodatum to associate with the variable that produced categories

---

`me` *My user entity*

Description

Get the user entity of the currently authenticated user.

Usage

```
me()
```

Value

A UserEntity

members *Manage access to datasets and other objects*

Description

These methods allow you to work with teams.

Usage

```
members(x)

members(x) <- value

permissions(x)

## S4 method for signature 'ProjectFolder'
members(x)

## S4 replacement method for signature 'ProjectFolder,MemberCatalog'
members(x) <- value

## S4 method for signature 'CrunchTeam'
members(x)

## S4 replacement method for signature 'ProjectFolder,character'
members(x) <- value

## S4 replacement method for signature 'CrunchTeam,MemberCatalog'
members(x) <- value

## S4 replacement method for signature 'CrunchTeam,character'
members(x) <- value
```

Arguments

x	CrunchDataset, ProjectFolder, or CrunchTeam
value	for members<-, a character vector of emails or URLs of users to add to the team.

Value

members() returns a MemberCatalog, which has references to the users that are members of the team. members<- returns x with the given users added to the members catalog. permissions() returns a PermissionCatalog with similar semantics.

See Also

users()

merge *Merge a CrunchDataFrame*

Description

mergeing a CrunchDataFrame with a local dataframe is useful in situations where you have new information in your local R session that you want to connect with Crunch data. For example, for making plots with Crunch and non-Crunch data. It produces a hybrid CrunchDataFrame that has the local data attached to it, but like normal CrunchDataFrames it is still judicious about downloading data from the server only when it is needed.

Usage

```
## S3 method for class 'CrunchDataFrame'
merge(
  x,
  y,
  by = intersect(names(x), names(y)),
  by.x = by,
  by.y = by,
  sort = c("x", "y"),
  ...
)
```

Arguments

x	a CrunchDataFrame
y	a standard data.frame
by	name of the variable to match in both data sources (default: the intersection of the names of x and y)
by.x	name of the variable to match in x
by.y	name of the variable to match in y
sort	character, either "x" or "y" (default: "x"). Which of the inputs should be used for the output order. Unlike merge.data.frame, merge.CrunchDataFrame will not re-sort the order of the output. It will use the order of either x or y.
...	ignored

Details

Merging a CrunchDataFrame with a local dataframe does not allow specifying all rows from both sources. Instead, the resulting CrunchDataFrame will include all of the rows in whichever source is used for sorting (x or y). So if you specify sort="x" (the default) all rows of x will be present but rows in y that do not match with rows in x will not be present.

Merging a CrunchDataFrame with a local dataframe is experimental and might result in unexpected results. One known issue is that using merge on a CrunchDataFrame will change the both the CrunchDataFrame used as input as well as create a new CrunchDataFrame.

Value

a CrunchDataFrame with columns from both `x` and `y`

`mergeFork` *Merge changes to a dataset from a fork*

Description

Crunch datasets include information about the dataset's revision history. This function takes the changes made on a dataset fork and adds them to the revision history of the parent dataset, like a merge of branches in a version control system.

Usage

```
mergeFork(dataset, fork, autorollback = TRUE, force = FALSE)
```

Arguments

`dataset`	The `CrunchDataset` to merge to
`fork`	The `CrunchDataset`, which must be a fork from `dataset`, that is to be merged in.
`autorollback`	logical If the merge fails, should `dataset` be restored to its state prior to the merge, or should it be left in its partially merged state for debugging and manual fixing? Default is `TRUE`.
`force`	logical Attempt to push through merge conflicts by dropping all changes to `dataset` that occurred after `fork` diverged from and take only the changes from `fork`? Default is `FALSE`. You should only use `force=TRUE` after first attempting and failing to merge without forcing.

Details

All modifications of a dataset record actions in its revision history. For example, if you add a variable to the dataset, that action is recorded. The sum of these records is a dataset's revision history, and it is possible to merge in the revision history of a dataset that has been forked.

This function is most often used in conjunction with `forkDataset()` to create a copy of a dataset, make some changes to that copy, and then merge the changes back into the original dataset. For more on this workflow, see `vignette("fork-and-merge", package = "crunch")`.

Value

`dataset` with changes from `fork` merged to it.

See Also

`forkDataset()`

Examples

```
## Not run:
ds <- loadDataset("My survey")
fork <- forkDataset(ds)
# Do stuff to fork
ds <- mergeFork(ds, fork)
# Now the changes you did to fork are also on ds

## End(Not run)
```

`multitables` *Multitable entities for a dataset*

Description

Multitable entities for a dataset

Usage

```
multitables(x)

multitables(x) <- value

## S4 method for signature 'CrunchDataset'
multitables(x)

## S4 replacement method for signature 'CrunchDataset'
multitables(x) <- value
```

Arguments

`x`	a CrunchDataset
`value`	for the assignment method, a MultitableCatalog

Value

an object of class MultitableCatalog containing references to Multitable entities. (Setter returns the Dataset.)

mv *Functions to manipulate variables' or project's folder structure*

Description

Variables in Crunch datasets are organized into folders, like in a file system. Datasets are similarly organized into hierarchical Projects. These functions allow you to create new folders and move objects into folders. Their names, `mv` and `mkdir`, suggest their Unix file utility inspiration.

Usage

```
mv(x, what, path)

mkdir(x, path)
```

Arguments

`x`	A `CrunchDataset` or `Folder` (`VariableFolder` or `ProjectFolder`)
`what`	A Variable, selection of variables from `dataset`, or any other object that can be moved to a folder (e.g. a dataset when organizing projects).
`path`	A character "path" to the folder: either a vector of nested folder names or a single string with nested folders separated by a delimiter ("/" default, configurable via `options(crunch.delimiter)`). The path is interpreted as relative to the location of the folder `x` (when `x` is a dataset, that means the root, top-level folder). `path` may also be a `Folder` object.

Details

The functions have some differences from the strict behavior of their Unix ancestors. For one, they work recursively, without additional arguments: `mkdir` will make every directory necessary to construct the requested path, even if all parent directories didn't already exist; and `mv` doesn't require that the directory to move to already exist—it will effectively call `mkdir` along the way.

Value

`x`, with the folder at `path` guaranteed to be created, and for `mv`, containing `what` moved into it.

See Also

`cd()` to select a folder by path; `rmdir()` to delete a folder; `folder()` to identify and set an object's parent folder; `base::dir.create()` if you literally want to create a directory in your local file system, which `mkdir()` does not do

Examples

```
## Not run:
ds <- loadDataset("Example survey")
ds <- mv(ds, c("gender", "age", "educ"), "Demographics")
ds <- mkdir(ds, "Key Performance Indicators/Brand X")
# These can also be chained together
require(magrittr)
ds <- ds %>%
    mv(c("aware_x", "nps_x"), "Key Performance Indicators/Brand X") %>%
    mv(c("aware_y", "nps_y"), "Key Performance Indicators/Brand Y")
# Can combine with cd() and move things with relative paths
ds %>%
    cd("Key Performance Indicators/Brand X") %>%
    mv("nps_x", "../Net Promoters")
# Can combine with folder() to move objects to the same place as something else
ds %>% mv("nps_y", folder(ds$nps_x))
# Now let's put ds in a Project
projects() %>%
    mv(ds, "Brand Tracking Studies")

## End(Not run)
```

---

na.omit *Omit missing categories*

Description

Omit missing categories

Usage

```
na.omit(object, ...)

## S4 method for signature 'Categories'
na.omit(object, ...)
```

Arguments

object	Categories
...	additional arguments, ignored

Value

object with any categories that have missing: TRUE excluded

ncol *Dataset dimensions*

Description

Dataset dimensions

Usage

```
ncol(x)

## S4 method for signature 'CrunchDataset'
dim(x)

## S4 method for signature 'CrunchDataset'
ncol(x)
```

Arguments

x	a Dataset

Value

integer vector of length 2, indicating the number of rows and non-hidden variables in the dataset. Array subvariables are excluded from the column count.

See Also

`base::dim()`

newDataset *Upload data to Crunch to make a new dataset*

Description

This function creates a new dataset on the Crunch server with either a data.frame or similar object in your R session, a file, or a URL to a file. It captures available metadata from your R object and translates it into Crunch types.

Usage

```
newDataset(x, name = NULL, ...)
```

Arguments

x	a data.frame or other rectangular R data object, or a string file name or URL to upload to create a dataset. The file may be a compressed Zip file containing a single file in CSV or SPSS format.
name	character name to give the new Crunch dataset. By default the function uses the name of the R object, or, if passing a file, the file name.
...	additional arguments passed to createDataset(), or schema if you're upload Triple-S

Details

If you have an SPSS file, it is better specify the file name directly rather than first reading it into R. Uploading SPSS files directly to Crunch will preserve metadata that is stripped by the R import, regardless of the library used to read it into R.

If you have Triple-S files, you can import those directly to Crunch like you can with SPSS files. You should use the filename to the data file (ending in .asc or .dat) as the x argument and use the metadata file (ending in .sss or .xml) as the schema argument.

Value

If successful, an object of class CrunchDataset.

See Also

newDatasetFromFile(); newDatasetByColumn() for an alternate upload method.

Examples

```
## Not run:
ds <- newDataset(mtcars, "cars")
ds <- newDataset("mysurvey.sav")

## End(Not run)
```

---

newDeck *Create an empty Crunch Deck*

Description

Create an empty Crunch Deck

Usage

```
newDeck(dataset, name, ...)
```

Arguments

dataset	A Crunch Dataset
name	The name of the Deck
...	Further attributes of the deck such as the description, see API docs for options.

Value

The CrunchDeck that was created.

---

newExampleDataset *Import a fixture dataset for testing*

Description

The crunch package includes some data for you to explore the features of the platform. Use this function to upload one to create a demo dataset.

Usage

```
newExampleDataset(name = "pets")
```

Arguments

name	string name of the fixture dataset. Currently "pets" is the only one available.

Value

A new CrunchDataset entity.

---

newFilter *Create a new filter*

Description

This function creates a new filter for a CrunchDataset. You can achieve the same results by assigning into a dataset's filters catalog using`filters()`, but this may be a more natural way to think of the action, particularly when you want to do something with the filter entity after you create it.

Usage

```
newFilter(name, expression, catalog = NULL, ...)
```

Arguments

name	character name for the filter
expression	CrunchLogicalExpr with which to make a filter entity
catalog	FilterCatalog in which to create the new filter. May also provide a dataset entity. If omitted, the function will attempt to infer the dataset (and thus its FilterCatalog) from the contents of expression.
...	Additional filter attributes to set, such as is_public.

Value

A CrunchFilter object.

---

newMultitable *Create a new Multitable*

Description

Multitables, or "banners" or "crossbreaks", define a set of variables or or query expressions to crosstab with as a unit. They are used in the Crunch web app to display tables side by side, as well as to define one dimension of a tab book.

Usage

```
newMultitable(formula, data, name, ...)
```

Arguments

formula	an object of class 'formula' object with the cross-classifying variables separated by '+' on the right-hand side. Following how stats::formula() works in R, it should start with "~". Variables on left-hand side of the formula have no meaning in this function.
data	an object of class CrunchDataset in which to create the multitable, and to which the variables referenced in formula belong.
name	character name to give the new multitable object. If omitted, a default name will be derived from formula.
...	Additional multitable attributes to set. Options include is_public.

Value

An object of class Multitable

See Also

stats::formula

Examples

```
## Not run:
m <- newMultitable(~ gender + age4 + marstat, data = ds)
name(m) # [1] "gender + age4 + marstat"

## End(Not run)
```

`newProject` *Create a new folder*

Description

This function creates a new project. You can achieve the same results by assigning into the projects catalog, but this may be a more natural way to think of the action, particularly when you want to do something with the project entity after you create it.

Usage

```
newProject(name, members = NULL, catalog = projects(), ...)
```

Arguments

`name`	character name for the project
`members`	Optional character vector of emails or user URLs to add as project members.
`catalog`	ProjectFolder in which to create the new project. There is only one project catalog currently, `projects()`, but this is left here so that all `new*` functions follow the same pattern.
`...`	Additional project attributes to set

Value

A `ProjectFolder` object.

See Also

`mkdir()`

Examples

```
## Not run:
proj <- newProject("A project name")
# That is equivalent to doing:
p <- projects()
p[["A project name"]] <- list()
proj <- p[["A project name"]]

proj2 <- newProject("Another project", members = "you@yourco.com")
```

```
# That is equivalent to doing:
p[["Another project"]] <- list(members = "you@yourco.com")
proj <- p[["Another project"]]

## End(Not run)
```

`newSlide` *Append a new slide to a Crunch Deck*

Description

Append a new slide to a Crunch Deck

Usage

```
newSlide(
  deck,
  query = NULL,
  display_settings = list(),
  title = "",
  subtitle = "",
  filter = NULL,
  weight = NULL,
  viz_specs = NULL,
  transform = NULL,
  ...
)
```

Arguments

`deck`	A Crunch Deck
`query`	A formula definition of a query to be used by the slide. See Details of `crtabs()` for more information about making queries.
`display_settings`	(optional) A list of display settings. If omitted, slide will be a table of column percentages with hypothesis test highlighting enabled. The most common setting used is `vizType`, which can be: `table`, `groupedBarPlot`, `stackedBarPlot`, `horizontalBarPlot`, `horizontalStackedBarPlot`, `donut`, and (if the second variable in the query formula is a wave variable) `timeplot`. In addition, `showValueLabels` (logical) controls whether the web app and exports show labels on bars or arcs of donuts.
`title`	The slide's title
`subtitle`	The slide's subtitle
`filter`	a `CrunchLogicalExpression`, a crunch `filter` object or a vector of names of `filters` defined in the dataset (defaults to `NULL`, using all data).
`weight`	A weight variable (defaults to NULL, meaning no weight)

viz_specs	Another set of options for the display of the slide, see the API documentation for more information.
transform	A list of slide transformations, usually created using the function makeDimTransform().
...	Further options to be passed on to the API

Value

CrunchSlide object

See Also

newMarkdownSlide for creating a markdown slide

Examples

```
## Not run:
newSlide(
    main_deck,
    ~ cyl + wt,
    title = "Cyl and Weight",
    subtitle = "2017 Data"
)

# Grouped bar plot
newSlide(
    main_deck,
    ~ approval + age4,
    title = "Approval by age group",
    display_settings = list(
        vizType = "groupedBarPlot",
        showValueLabels = TRUE
    ),
    subtitle = "2017 Data"
)

# Horizontal stacked bars
newSlide(
    main_deck,
    ~ approval + age4,
    title = "Approval by age group",
    display_settings = list(
        vizType = "horizontalStackedBarPlot"
    ),
    subtitle = "2017 Data"
)

# A donut is only suitable for a single variable
newSlide(
    main_deck,
    ~ approval,
    title = "Approval of new feature",
    display_settings = list(
```

```
        vizType = "donut",
        showValueLabels = FALSE
    ),
    subtitle = "2017 Data"
)

# A Grouped bar plot with slide transformations to hide a category
newSlide(
    main_deck,
    ~ approval + age4,
    title = "Approval by age group",
    display_settings = list(
        vizType = "groupedBarPlot",
        showValueLabels = TRUE
    ),
    transform = list(rows_dimension = makeDimTransform(hide = "Neutral")),
    subtitle = "2017 Data"
)

# Example of advanced options being set:
# viz_specs can get quite long, see
# https://crunch.io/api/reference/#post-/datasets/-dataset_id-/decks/-deck_id-/slides/
viz_specs <- list(
    default = list(
        format = list(
            decimal_places = list(percentages = 0L, other = 2L),
            show_empty = FALSE
        )
    ),
    table = list(
        measures = c("col_percent", "pairwise_t_test"),
        page_layout = list(
            rows = list(
                top = list(),
                bottom = c("base_unweighted", "scale_mean", "significant_columns")
            ),
            measure_layout = "long"
        ),
        pairwise_comparison = list(sig_threshold = c(0.05, 0.01)),
        format = list(pval_colors = FALSE)
    )
)

newSlide(
    main_deck,
    ~categories(fav_array)+subvariables(fav_array),
    display_settings = list(viz_type = list(value = "table")),
    title = "custom slide",
    filter = filters(ds)[[1]],
    weight = ds$weight,
    viz_specs = viz_specs
)
```

```
# Can also specify `analyses` directly, which allows for very advanced use.
# `formulaToSlideQuery()` and `slideQueryEnv()` help describe the API
newSlide(
    main_deck,
    title = "custom slide",
    analyses = list(list(
        query = formulaToSlideQuery(~categories(fav_array)+subvariables(fav_array), ds),
        query_environment = slideQueryEnv(filter = filters(ds)[[1]]),
        display_settings = list(viz_type = list(value = "table")),
        viz_specs = viz_specs
    ))
)

## End(Not run)
```

---

noTransforms *Remove transformations from a CrunchCube*

Description

Remove transformations from a CrunchCube

Usage

```
noTransforms(cube)
```

Arguments

cube	a CrunchCube

Value

the CrunchCube with no transformations

Removing transforms

`noTransforms()` is useful if you don't want to see or use any transformations like Subtotals and Headings. This action only applies to the CrunchCube object in R: it doesn't actually change the variables on Crunch servers or the query that generated the CrunchCube.

Examples

```
## Not run:
# A CrunchCube with a heading and subtotals
crtabs(~opinion, ds)
#               All opinions
#             Strongly Agree 23
#             Somewhat Agree 24
#                      Agree 47
# Neither Agree nor Disagree 18
```

```
#          Somewhat Disagree 16
#          Strongly Disagree 19
#                   Disagree 35

noTransforms(crtabs(~opinion, ds))
#             Strongly Agree             Somewhat Agree Neither Agree nor Disagree
#                         23                         24                         18
#          Somewhat Disagree          Strongly Disagree
#                         16                         19

## End(Not run)
```

---

owner *Get and set the owner of a dataset*

Description

Get and set the owner of a dataset

Usage

```
owner(x)

owner(x) <- value

## S4 method for signature 'CrunchDataset'
owner(x)

## S4 replacement method for signature 'CrunchDataset'
owner(x) <- value
```

Arguments

x	CrunchDataset
value	For the setter, either a URL (character) or a Crunch object with a `self` method. Users and Projects are valid objects to assign as dataset owners.

Value

The dataset.

owners *See who owns these datasets*

Description

See who owns these datasets

Usage

```
owners(x)

ownerNames(x)
```

Arguments

x	DatasetCatalog

Value

For owners, the URLs of the users or projects that own these datasets. For ownerNames, their names.

palettes *Get the palettes from a crunch object*

Description

CrunchDatasets have color palettes associated with them that can be used as default colors for dashboard tiles. One of them can be assigned the "default".

Usage

```
palettes(x)

defaultPalette(x, ...)

## S4 method for signature 'CrunchDataset'
palettes(x)

## S4 method for signature 'CrunchDataset'
defaultPalette(x, ...)

## S4 method for signature 'AnalyticPalettes'
defaultPalette(x, ...)
```

Arguments

x	A crunch object, like a CrunchDataset
...	ignored (reserved for future expansion)

---

pendingStream *Get the pending streams for a dataset*

Description

Retrieves the number of pending messages. Use appendStream() to append all pending streamed rows to the dataset.

Usage

```
pendingStream(ds)
```

Arguments

ds	a CrunchDataset

Value

number of pending messages in the stream for the dataset

---

pk *Get and set the primary key for a Crunch dataset*

Description

A primary key is a variable in a dataset that has a unique value for every row. A variable must be either numeric or text type and have no duplicate or missing values. A primary key on a dataset causes appends to that dataset that have the rows with the same primary key value(s) as the first dataset to update the existing rows rather than inserting new ones.

Usage

```
pk(x)

pk(x) <- value
```

Arguments

x	a Dataset
value	For the setter, a single Variable to use as the primary key or NULL to remove the primary key.

Value

Getter returns the Variable object that is used as the primary key (NULL if there is no primary key); setter returns x duly modified.

---

pollProgress *Check a Crunch progress URL until it finishes*

Description

You'll probably only call this function if progress polling times out and its error message tells you to call pollProgress to resume.

Usage

```
pollProgress(progress_url, wait = 0.5, error_handler = NULL)
```

Arguments

progress_url	A Crunch progress URL
wait	Number of seconds to wait between polling. This time is increased 20 percent on each poll.
error_handler	An optional function that takes the status object when the progress is less than 0 (meaning the request failed)

Value

The percent completed of the progress. Assuming the options(crunch.timeout) (default: 15 minutes) hasn't been reached, this will be 100. If the timeout is reached, it will be the last reported progress value.

---

popSize *Get and set the market size for Crunch datasets*

Description

Crunch Datasets allow you to set a target population size in order to extrapolate population estimates from survey percentages. These functions let you work with the population size and magnitude.

Usage

```
popSize(x)

popMagnitude(x)

popSize(x) <- value

popMagnitude(x) <- value

setPopulation(x, size, magnitude)

## S4 method for signature 'CrunchDataset'
popSize(x)

## S4 replacement method for signature 'CrunchDataset'
popSize(x) <- value

## S4 method for signature 'CrunchDataset'
popMagnitude(x)

## S4 replacement method for signature 'CrunchDataset'
popMagnitude(x) <- value

## S4 method for signature 'CrunchDataset'
setPopulation(x, size, magnitude)
```

Arguments

x	a Crunch Dataset
value	For the setters, the size or magnitude to be set
size	the target population size, to remove a population set to NULL
magnitude	the order of magnitude with which to display the population size. Must be either 3, 6, or 9 for thousands, millions, and billions respectively.

Value

popSize and popMagnitude return the population size or magnitude. setPopulation returns the modified dataset.

---

preCrunchBoxCheck *Check if a dataset will make a good CrunchBox*

Description

CrunchBoxes allows you to share data with the world in a simple, easy to embed format. However, not all datasets naturally translate to the CrunchBox format. This function checks your dataset to see if it has variable & category definitions that will work well with the CrunchBox format.

Usage

```
preCrunchBoxCheck(dataset)
```

Arguments

dataset	CrunchDataset, potentially subsetted on variables

Value

Invisibly, the dataset. Called for side-effect of printing things.

See Also

CrunchBox

---

prepareDataForCrunch *Translate a data.frame to Crunch format*

Description

This is called within newDataset to extract the Crunch metadata from the data and to transform the data to match the extracted metadata. You can call this directly in order to tailor the data import flow more finely.

Usage

```
prepareDataForCrunch(data, ...)
```

Arguments

data	A data.frame or other rectangular R object
...	additional arguments passed to createDataset. "name" will be required by the Crunch server but is not required by this function.

Value

A data.frame that is a transformation of data suitable for uploading to Crunch, also containing a "metadata" attribute that is the associated Crunch metadata.

See Also

createWithPreparedData writePreparedData

projects *List project folders*

Description

List project folders

Usage

```
projects(x = getAPIRoot())
```

Arguments

x	a ShojiObject that has an associated catalog. If omitted, the default value for x means that you will load the user's primary folder. (Currently, there are no other folders to load.)

Value

An object of class ProjectFolder.

Examples

```
## Not run:
myprojects <- projects()
proj <- myprojects[["Project name"]]

## End(Not run)
```

publicFolder *Hide/Unhide or Privatize/Deprivatize Variables*

Description

The public folder is the top level folder of all regular public variables. Both hidden and private are hidden from most views in crunch by default. Hidden variables can be accessed by an user, while private variables (and all variables derived from them) are only accessible by users granted "editor" access to the dataset and so can be used to secure personally identifiable information from non-editors of a dataset.

Usage

```
publicFolder(x)

hiddenFolder(x)

privateFolder(x)

hide(x)

unhide(x)

privatize(x)

deprivatize(x)

## S4 method for signature 'CrunchDataset'
publicFolder(x)

## S4 method for signature 'VariableCatalog'
publicFolder(x)

## S4 method for signature 'VariableFolder'
publicFolder(x)

## S4 method for signature 'CrunchDataset'
hiddenFolder(x)

## S4 method for signature 'VariableCatalog'
hiddenFolder(x)

## S4 method for signature 'VariableFolder'
hiddenFolder(x)

## S4 method for signature 'CrunchVariable'
hide(x)

## S4 method for signature 'VariableCatalog'
hide(x)

## S4 method for signature 'CrunchVariable'
unhide(x)

## S4 method for signature 'VariableCatalog'
unhide(x)

hideVariables(dataset, variables)

hiddenVariables(x) <- value
```

```
unhideVariables(dataset, variables)

hiddenVariables(dataset, key = namekey(dataset))

## S4 method for signature 'CrunchDataset'
privateFolder(x)

## S4 method for signature 'VariableCatalog'
privateFolder(x)

## S4 method for signature 'VariableFolder'
privateFolder(x)

## S4 method for signature 'CrunchVariable'
privatize(x)

## S4 method for signature 'VariableCatalog'
privatize(x)

## S4 method for signature 'CrunchVariable'
deprivatize(x)

## S4 method for signature 'VariableCatalog'
deprivatize(x)

privatise(x)

deprivatise(x)

privatizeVariables(dataset, variables)

privatiseVariables(dataset, variables)

privateVariables(x) <- value

deprivatizeVariables(dataset, variables)

deprivatiseVariables(dataset, variables)

privateVariables(dataset, key = namekey(dataset))
```

Arguments

x	a Variable, VariableCatalog, or dataset to hide/unhide/privatize/deprivatize
dataset	A dataset
variables	Variables to change status of
value	Replacement values for assignment methods.

key (for hiddenVariables() / privateVariables() the Variable attribute to return. Default is "alias", following envOrOption("crunch.namekey.dataset").

Details

There are several ways to assign variables into these categories and access them:

- hideVariables() / privatizeVariables() - take a character vector of variable aliases and makes them hidden/private. (unhideVariables() / deprivatizeVariables() put them back in the main variable catalog).
- hide() / privatize() - take a CrunchVariable or VariableCatalog and make them hidden/private. (unhide() / deprivatize() put them back in the main variable catalog).
- hiddenFolder() / privateFolder() / publicFolder() - take a dataset and return a folder that contains the public/hidden/private variables. This folder is like other CrunchFolders and so you can use mkdir() to create subfolders and mv() to move them in/out.
- hiddenVariables() / privateVariabiles() - return a character vector of variables that are hidden/private. You can assign into the catalog to add variables or assign to NULL to remove all of them.

---

reassignUser *Reassign all Crunch objects from a user*

Description

If you want to transfer all teams, projects, and datasets owned by one user to another you can with reassignUser. To have permission to use reassignUser you must be an account admin and be from the same account as the user who is being reassigned. This is useful if a user leaves your organization and you want to transfer all of the teams, projects, and datasets they own to someone else.

Usage

```
reassignUser(from, to)
```

Arguments

from	a character of the email address of the user to reassign from
to	a character of the email address of the user who should be the new owner

Details

The user given in to will become the owner of all of the teams, projects, and datasets that were previously owned by the user given in from.

Reassigning requires confirmation. In an interactive session, you will be asked to confirm. To avoid that prompt, or to reassign datasets from a non-interactive session, wrap the call in with_consent() to give your permission to reassign

Value

NULL if successful

---

refresh *Get a fresh copy from the server*

Description

Crunch objects generally keep themselves in sync with the server when you manipulate them, but some operations cause the local version to diverge from the version on the server. For instance, someone else may have modified the dataset you're working on, or maybe you have modified a variable outside of the context of its dataset. `refresh()` allows you to get back in sync.

Usage

```
refresh(x)

## S4 method for signature 'CrunchDataset'
refresh(x)

## S4 method for signature 'ShojiObject'
refresh(x)

## S4 method for signature 'CrunchVariable'
refresh(x)
```

Arguments

x	pretty much any Crunch object

Value

a new version of x

---

reorderSlides *Reorder slides in a CrunchDeck*

Description

Reorder slides in a CrunchDeck

Usage

```
reorderSlides(x, order)
```

Arguments

x	A SlideCatalog
order	The numeric order for slides to be reordered to.

Value

A SlideCatalog

---

resolution *Methods for Datetime variable resolutions*

Description

Datetime data has a "resolution", the units of the values. `resolution()` exposes that property and `resolution<-` lets you set it. "Rollups" are a way of binning datetime data into meaningful units. `rollup()` lets you create an expression that you can query with. Datetime variables also have a `rollupResolution()` attribute that is the default resolution they will roll-up to, if not specified in `rollup()`; `rollupResolution<-` lets you set that.

Usage

```
resolution(x)

resolution(x) <- value

rollup(x, resolution = rollupResolution(x))

rollupResolution(x)

rollupResolution(x) <- value
```

Arguments

x	a Datetime variable
value	a resolution string. Valid resolutions in Crunch are `c("Y", "Q", "M", "W", "D", "h", "m", "s", "ms")`. `NULL` is also valid for `rollupResolution<-` but not for `resolution<-`.
resolution	Same as `value`, in `rollup()`. This may be `NULL`, in which case the server will determine an appropriate resolution based on the range of the data.

Details

Note that `resolution` is a property of the data while `rollupResolution` is metadata. Setting `resolution` alters the column data, and if setting a more coarse resolution (e.g. going from "s" to "m"), it cannot be reversed. Setting `rollupResolution` is non-destructive.

Value

`resolution()` and `rollupResolution()` return the resolution string for datetime variables, `NULL` otherwise. The setters return the variable entity after modifying the state on the server. `rollup()` returns a `CrunchExpr` expression.

Examples

```
## Not run:
resolution(ds$starttime)
## [1] "ms"
resolution(ds$starttime) <- "s"
rollup(ds$starttime)
rollup(ds$starttime, "D")
rollupResolution(ds$starttime) <- "D"
crtabs(~ rollup(starttime), data = ds)

## End(Not run)
```

---

restoreVersion *Restore a dataset to a previously saved version*

Description

You can save a version of a dataset using `saveVersion()`. Savepoints are also created automatically by certain Crunch functions that make major changes to the dataset. You can get the list of saved versions with the `versions()` function.

Usage

```
restoreVersion(dataset, version)
```

Arguments

`dataset`	a `CrunchDataset`
`version`	either the name ("description") of the version to restore to or the integer index of the version, as given by `versions(dataset)`

Value

`dataset`, rolled back to `version`.

See Also

versions saveVersion

retry *Retry*

Description

Retry an expression. This is useful for situations where a web resource is not yet available. You can set `options("crunch_retry_wait" = X)` some number larger than the default 0.1 in your script if you are working with large exports.

Usage

```
retry(
  expr,
  wait = envOrOption("crunch_retry_wait", default = 0.1, expect_num = TRUE),
  max.tries = 10
)
```

Arguments

`expr`	An expression
`wait`	The time in seconds to wait before retrying the expression. Defaults to 0.1.
`max.tries`	The number of times to retry the expression

rmdir *Delete a folder*

Description

Like `rmdir` in a file system, this function removes a folder. Unlike the file-system version, it does not require the folders to be empty.

Usage

```
rmdir(x, path)
```

Arguments

`x`	A `CrunchDataset` or `Folder` (`VariableFolder` or `ProjectFolder`)
`path`	A character "path" to the folder: either a vector of nested folder names or a single string with nested folders separated by a delimiter ("/" default, configurable via `options(crunch.delimiter)`). The path is interpreted as relative to the location of the folder `x` (when `x` is a dataset, that means the root, top-level folder). `path` may also be a `Folder` object.

Value

NULL

See Also

mv() to move entities to a folder; cd() to select a folder; file.remove() if you literally want to delete a directory from your local file system, which rmdir() does not do

Examples

```
## Not run:
ds <- loadDataset("Example survey")
rmdir(ds, "Demographics")
# Or with %>%
require(magrittr)
ds <- ds %>%
    rmdir("Demographics")

## End(Not run)
```

---

rowCount — *Create variables based on row-wise functions for crunch Multiple Response Variables*

Description

Quickly generate new variables that are based on row-wise summaries of Multiple Response Variables.

Usage

```
rowCount(x, name, ...)
```

Arguments

x	A crunch variable or expression
name	a character to use as the name of the case variable to create
...	description, alias, and other metadata passed to VarDef()

Value

A Variable Definition

See Also

expressions for the more flexible expressions that power these functions and rowDistinct() for other row-wise functions

rowDistinct *Create variables useful for determining whether a row's values are suspicious*

Description

`rowDistinct()` finds the number of unique values given per row of variables in an array `CrunchVariable`. `straightlineResponse()` returns a `selection` variable that indicates whether the responses are identical. When a row has all columns that are missing of the same type, it will return `Selected`, but will missing if any other number of values is missing (or there are multiple types of missing).

Usage

```
rowDistinct(x, name, ..., na.rm = TRUE)

straightlineResponse(x, name, ...)
```

Arguments

`x`	A `CrunchVariable`that is an an array, that unique values should be counted across.
`name`	a character to use as the name of the case variable to create
`...`	Optional attributes, like `description`, to set on the new variable (passed to `VarDef()`)
`na.rm`	Whether to count missing data as a separate category (all missing categories will be lumped together)

Value

A Variable Definition, which can be used to create a new `CrunchVariable`

See Also

`rowCount()` for other row-wise functions

runCrunchAutomation *Run a crunch automation script*

Description

Crunch automation is a custom scripting syntax that allows you to concisely describe the metadata of your data when importing. The syntax is described in the crunch API documentation

Usage

```
runCrunchAutomation(
  dataset,
  script,
  is_file = string_is_file_like(script),
  encoding = "UTF-8",
  ...
)

showScriptErrors()
```

Arguments

dataset	A crunch dataset
script	A path to a text file with crunch automation syntax or a string the syntax loaded in R.
is_file	The default guesses whether a file or string was used in the `script` argument, but you can override the heuristics by specifying `TRUE` for a file, and `FALSE` for a string.
encoding	Optional encoding to convert **from**, defaults to UTF-8. The API accepts only UTF-8, so all text will be converted to UTF-8 before being sent to the server.
...	Additional options, such as `dry_run = TRUE` passed on to the API

Value

For `runCrunchAutomation()`: an updated dataset (invisibly), For `showScriptErrors()`, when run after a failure, a list with two items: `script`: that contains the script string sent to the server and `errors` which is a `data.frame` with details about the errors sent from the server.

See Also

automation-undo & script-catalog

Examples

```
## Not run:
# Can use a path to a file:
script_file <- "crunch_automation.txt"
ds <- runCrunchAutomation(ds, script_file)

# Or a string directly:
ds <- runCrunchAutomation(ds, "RENAME v1 TO age;")

# A "dry run" that validates the script but does not run it:
runCrunchAutomation(ds, "RENAME V1 TO age;", dry_run = TRUE)

# After a failed run, some error information prints to console,
# But more details are available with function:
showScriptErrors()
```

```
# After a successful run, can look at scripts
scripts(ds)


## End(Not run)
```

---

saveVersion *Create a new saved version*

Description

Crunch datasets can be saved and restored using saveVersion and restoreVersion(). Some Crunch functions, such as appendDataset() create new savepoints automatically. To see the list of savepoints use versions().

Usage

```
saveVersion(
  dataset,
  description = paste("Version", length(versions(dataset)) + 1)
)
```

Arguments

dataset	a CrunchDataset
description	character name to give the saved version, as in a commit message. You are encouraged, though not strictly required, to give versions unique descriptions.

Value

invisibly, the URL of the newly created version

See Also

versions restoreVersion

scoreCatToFeat *Score similarity between a feature dataframe and categories*

Description

Implemented using the Jaccard index, where a number closer to 1 is more similar.

Usage

```
scoreCatToFeat(features, categories)
```

Arguments

features	a vector of features to match (usually from a subset of the output [availableGeodataFeatures]) with a single property for a single geodatum.
categories	a vector of categories to match

Value

the Jaccard index for the values of the property given in feat_df and the vector of categories

scripts *Crunch Automation scripts entities for a dataset*

Description

Crunch Automation scripts entities for a dataset

Usage

```
scripts(x)

## S4 method for signature 'CrunchDataset'
scripts(x)
```

Arguments

x	a CrunchDataset

Value

an object of class "ScriptCatalog" containing references to Script entities.

See Also

runCrunchAutomation() & automation-undo

searchDatasets *Search Crunch for datasets.*

Description

searchDatasets searches datasets' metadata for matches to the query argument. This search will include variable names, aliases, categories, but not the content of text variables. See the API Documentation for more information about searching Crunch.

Usage

```
searchDatasets(query, f = NULL, ...)
```

Arguments

query	the text to search for in datasets and their variables (note: only alpha characters will be used, numbers and other characters will be discarded.)
f	A list of filter parameters, see the filter parameters of the API Documentation for more details.
...	additional options provided to the search endpoint.

Value

If successful, an object of class SearchResults

self *Get the URL of this object*

Description

Get the URL of this object

Usage

```
self(x)

## S4 method for signature 'ShojiObject'
self(x)

## S4 method for signature 'CrunchVariable'
self(x)
```

Arguments

x	a Crunch object

Value

the URL for x

---

setName *Change the name of the current folder*

Description

If you just need to change the name of the folder you are currently in, you can use setName(). It doesn't move variables or change anything other than the name of the current folder.

Usage

```
setName(object, nm)
```

Arguments

object	A Folder
nm	A character that is the new name the folder should have

Value

object, with its name duly changed

See Also

cd() and mv()

Examples

```
## Not run:
ds <- ds %>%
    cd("Demographics") %>%
    setName("Key Demos.")

## End(Not run)
```

setNames *Change the name of the entities in a catalog*

Description

This is an alternative to assigning `names(catalog) <- something`, suitable for inclusion in a pipeline.

Usage

```
setNames(object, nm)

## S4 method for signature 'ShojiCatalog'
setNames(object, nm)
```

Arguments

`object`	A catalog object, such as `VariableFolder`
`nm`	A character vector of new names of the same length as the number of entities in the index

Value

`object`, with the names of its children duly changed

See Also

`cd()` and `mv()`

Examples

```
## Not run:
ds <- ds %>%
    cd("Demographics") %>%
    setNames(c("Gender (4 category)", "Birth year", "Race (5 category)"))

## End(Not run)
```

setOrder *Change the order of entities in folder*

Description

Change the order of entities in folder

Usage

```
setOrder(folder, ord)
```

Arguments

folder	A VariableFolder or other *Folder object
ord	A vector of integer indices or character references to objects contained in the folder

Value

folder with the order dictated by ord. The function also persists that order on the server.

settings *View and modify dataset-level settings*

Description

These methods allow access and control over dataset settings. Currently supported settings include:

- User Authorizations for view-only users ('viewers_can_export', 'viewers_can_share', and 'viewers_can_change_weight'); and
- 'weight', which determines the default weighting variable for the dataset Additional settings will be added in the future. See https://crunch.io/api/reference/#post-/datasets/ -> request body model -> settings key, for an up-to-date list of settings supported throughout the Crunch system. Clients may also provide and use custom settings if they choose.

Usage

```
settings(x)

settings(x) <- value
```

Arguments

x	CrunchDataset
value	A settings object (ShojiEntity), for the setter

Value

The getter returns a settings object (ShojiEntity). The setter returns the dataset (x), duly modified.

Examples

```
## Not run:
settings(ds)
settings(ds)$viewers_can_export <- TRUE
settings(ds)$weight <- ds$myWeightVariable

## End(Not run)
```

---

setupCrunchAuth *Helper for switching between API keys and urls*

Description

Credentials can be stored in the options or environment variables with the following structure (option = crunch.api.<ID> or environment variable R_CRUNCH_API_<ID>) where <ID> is a string. Then you can use this function to choose which credentials you want to use.

Usage

```
setupCrunchAuth(id)
```

Arguments

id	A string indicating the id of the credentials

Examples

```
## Not run:
# Using crunch options:
set_crunch_opts(
    crunch.api.account1 = "https://company1.crunch.io/api/",
    crunch.api.key.account1 = "MY KEY"
)

# Or with environment variables
Sys.setenv(
    "R_CRUNCH_API_ACCOUNT2" = "https://company2.crunch.io/api/",
    "R_CRUNCH_API_KEY_ACCOUNT2" = "ANOTHER KEY"
)

# Can now switch between accounts
setupCrunchAuth("account1")
crunch_sitrep()

setupCrunchAuth("account2")
```

```
crunch_sitrep()

## End(Not run)
```

---

share *Share a dataset*

Description

Share a dataset

Usage

```
share(dataset, users, edit = FALSE, notify = TRUE, message = NULL)
```

Arguments

dataset	a CrunchDataset
users	character: email address(es) or URLs of the users or teams with whom to share the dataset. If there is no Crunch user associated with an email, an invitation will be sent.
edit	logical: should the specified user(s) be given edit privileges on the dataset? Default is FALSE. edit can be a single value or, if inviting multiple users, a vector of logical values of equal length of the number of emails given.
notify	logical: should users who are getting new privileges on this dataset be sent an email informing them of this fact? Default is TRUE.
message	character: a message to send to the users who are receiving new privileges.

Value

Invisibly, the dataset.

See Also

unshare

ShojiObject-class *Mix-in class for multiple inheritance of variables and datasets.*

Description

Exists for common methods in interacting with Crunch API only. Has no Extract methods declared so as not to conflict with the vector/list/data.frame methods jointly inherited in CrunchVariable and CrunchDataset.

show *Show methods for Crunch objects*

Description

Show methods for Crunch objects

Usage

```
show(object)

## S4 method for signature 'ShojiObject'
show(object)

## S4 method for signature 'CrunchVariable'
show(object)

## S4 method for signature 'Category'
show(object)

## S4 method for signature 'Categories'
show(object)

## S4 method for signature 'Insertion'
show(object)

## S4 method for signature 'Insertions'
show(object)

## S4 method for signature 'CrunchExpr'
show(object)

## S4 method for signature 'CrunchLogicalExpr'
show(object)

## S4 method for signature 'AnalyticPalettes'
show(object)

## S4 method for signature 'AnalyticPalette'
show(object)

## S4 method for signature 'CrunchCube'
show(object)

## S4 method for signature 'OrderGroup'
show(object)
```

```
## S4 method for signature 'CrunchGeography'
show(object)

## S4 method for signature 'DeckCatalog'
show(object)

## S4 method for signature 'CrunchDeck'
show(object)

## S4 method for signature 'CrunchSlide'
show(object)

## S4 method for signature 'CrunchAnalysisSlide'
show(object)

## S4 method for signature 'CrunchMarkdownSlide'
show(object)

## S4 method for signature 'MultitableResult'
show(object)

## S4 method for signature 'ShojiFolder'
show(object)
```

Arguments

object	the object

Value

invisibly

See Also

methods::show

---

showMissing *Modify cube missing behavior*

Description

By default, CrunchCubes do not show entries for missing categories. You can include missing values in a cube with showMissing(cube) and hide them again with hideMissing(cube).

Usage

```
showMissing(cube)

hideMissing(cube)

showIfAny(cube)

## S4 method for signature 'CrunchCube'
showMissing(cube)

## S4 method for signature 'CrunchCube'
hideMissing(cube)

## S4 method for signature 'CrunchCube'
showIfAny(cube)
```

Arguments

cube	a CrunchCube

`showTransforms` *Show transformations on a Crunch object*

Description

`showTransforms([variable])` shows a summary of a categorical variable that has transforms with the transforms calculated and applied. This is useful to see what kind transforms exist before including the variable in a CrunchCube.

Usage

```
showTransforms(x)

## S4 method for signature 'CategoricalVariable'
showTransforms(x)

## S4 method for signature 'CrunchCube'
showTransforms(x)
```

Arguments

x	a Categorical variable or CrunchCube

Details

`showTransforms([CrunchCube])` shows the CrunchCube with all transforms calculated and applied. This is the default display method for cubes, so should not be frequently needed.

In both cases, an array is returned that includes the values of both the underlying data (either category counts or CrunchCube cell values) as well as the transformations applied.

Value

summary of the variable, or the full CrunchCube with transforms applied

Examples

```
## Not run:
showTransforms(ds$variable)

## End(Not run)
```

`slideCategories` *Create sliding subvariable definitions*

Description

Create a multiple response array variable by sliding through category levels and selecting potentially overlapping sets of categories.

Usage

```
slideCategories(variable, step, width, ..., complete = TRUE, useNA = FALSE)
```

Arguments

`variable`	A categorical crunch variable
`step`	number of categories between starting points of groups
`width`	number of categories wide the grouping should be
`...`	additional attributes to be included in the `VariableDefinition`, can be either functions that take the category names to be included in the sliding group and returns a single string, or a character vector the same length as the number of subvariables that will be created.
`complete`	whether to only include category groupings that are as wide as width (defaults to `TRUE`)
`useNA`	whether to use missing categories from the original variable (defaults to `FALSE`)

Value

A list of `VariableDefinitions` appropriate for use in `deriveArray()`

Examples

```
## Not run:
data <- data.frame(
    wave = factor(c("a", "b", "c", "d", "e"))
)

ds <- newDataset(data, "Sliding Categories")

# Make an MR variable where subvariable is 1 step apart, and with 3 categories wide
# and name subvariables with vector
ds$wave_step1_wide3 <- deriveArray(
   slideCategories(ds$wave, step = 1, width = 3, name = c("a - c", "b - d", "c - e")),
   "Sliding example 1"
)

# You can also make names (and other subvariable metadata like alias or description)
# with a function:
ds$wave_step2_wide2 <- deriveArray(
   slideCategories(
     ds$wave,
     step = 2,
     width = 2,
     name = function(x) paste(x[1], "-", x[length(x)])
   ),
   "Sliding example 2"
)

## End(Not run)
```

`slideMarkdown` *Add a new markdown slide to a deck*

Description

Markdown slides allow you to add rich text tiles to your Crunch Dashboards. `markdownSlideImage()` is a helper for embedding the data of an image from your computer into the slide.

Usage

```
slideMarkdown(x)

slideMarkdown(x) <- value

newMarkdownSlide(deck, ..., title = "", subtitle = "")

markdownSlideImage(file)

## S4 method for signature 'CrunchMarkdownSlide'
slideMarkdown(x)
```

```
## S4 replacement method for signature 'CrunchMarkdownSlide,character'
slideMarkdown(x) <- value
```

Arguments

x	A CrunchMarkdownSlide
value	A string to replace the markdown content with
deck	A Crunch Deck
...	Unnamed arguments are text that are combined to create the markdown body named arguments are passed to the API.
title	The slide's title
subtitle	The slide's subtitle
file	File path to an image

Value

A MarkdownCrunchSlide

See Also

newSlide() for creating an analysis slide

Examples

```
## Not run:
newMarkdownSlide(deck, "We contacted 1,000 people by telephone", title = "Methodology")

newMarkdownSlide(
    deck,
    "The 3 most **popular** vegetables are:\n",
    "- Fennel\n",
    "- Carrots\n",
    "- Avocado\n",
    title = "Key findings"
)

newMarkdownSlide(
    deck,
    "crunch.io: ",
    markdownSlideImage("logo.png")
)

## End(Not run)
```

slides *Access the slides of a CrunchDeck*

Description

Return a SlideCatalog from a CrunchDeck. All slide catalog methods should be available for CrunchDecks, but this function is used internally to model the API.

Usage

```
slides(x)

slides(x) <- value

## S4 method for signature 'CrunchDeck'
slides(x)

## S4 replacement method for signature 'CrunchDeck'
slides(x) <- value
```

Arguments

x	a CrunchDeck
value	a SlideCatalog or CrunchSlide to add

Value

a SlideCatalog

SO_schema *Schema for the 2017 Stack Overflow developer survey*

Description

Survey questions and variable names for the 2017 Stack Overflow Developers Survey #'

Usage

```
SO_schema
```

Format

A data frame with 23 rows and 2 variables.

Column The column name of the survey data frame

Question Question asked of respondents

SO_survey *R users who responded to the 2017 Stack Overflow developer survey*

Description

A slightly modified version of the 2017 Stack Overflow developer survey. The dataset is filtered to only include respondents who have used R before, and to include illustrative variable types.

Usage

```
SO_survey
```

Format

A data frame with 1634 rows and 25 variables.

Respondent Respondent ID number

Professional Which of the following best describes you?

Country In which country do you currently live?

CompanySize In terms of the number of employees, how large is the company or organization you work for?

CareerSatisfaction Career satisfaction rating

JobSatisfaction Job satisfaction rating

ImportantHiringAlgorithms Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Knowledge of algorithms and data structures

ImportantHiringTechExp Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Experience with specific tools (libraries, frameworks, etc.) used by the employer

ImportantHiringCommunication Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Communication skills

ImportantHiringOpenSource Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Contributions to open source projects

ImportantHiringPMExp Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Experience with specific project management tools & techniques

ImportantHiringCompanies Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Previous companies worked at

ImportantHiringTitles Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Previous job titles held

ImportantHiringEducation Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Educational credentials (e.g. schools attended, specific field of study, grades earned)

ImportantHiringRep Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Stack Overflow reputation

ImportantHiringGettingThingsDone Congratulations! You've just been put in charge of technical recruiting at Globex, a multinational high- tech firm. This job comes with a corner office, and you have an experienced staff of recruiters at your disposal. They want to know what they should prioritize when recruiting software developers. How important should each of the following be in Globex's hiring process? Track record of getting things done

Gender Which of the following do you currently identify as?

Race Which of the following do you identify as?

Salary What is your current annual base salary, before taxes, and excluding bonuses, grants, or other compensation?

ExpectedSalary You said before that you are currently learning how to program. When you have completed your studies, what annual salary do you expect to earn in your first job after graduation?

TabsSpaces Tabs or spaces?

WantWorkLanguage Which of the following languages have you done extensive development work in over the past year, and which do you want to work in over the next year?

HaveWorkedLanguage Which of the following languages have you done extensive development work in over the past year, and which do you want to work in over the next year?

Source

https://insights.stackoverflow.com/survey/

---

streaming *Set the streaming property of a dataset*

Description

Only datasets that have their streaming property set to "streaming" can have rows streamed to them. Before attempting to streaming rows (with streamRows for example), the dataset has to be set up to stream rows. Use `streaming(ds)` to get the streaming status, and `streaming(ds) <- "streaming"` to set the streaming status.

Usage

```
streaming(x)

streaming(x) <- value
```

Arguments

x	a CrunchDataset
value	for setting only (values can be: "no", "streaming", or "finished")

Value

the streaming status

Subtotal-class *Subtotals and headings*

Description

Subtotals and headings for Categorical Variables and Multiple Response Variables. These are especially useful for making aggregates across multiple categories (sometimes referred to as *nets*, *top box*, or *top 2 box*).

Subtotals and headings for Categorical Variables and Multiple Response Variables. These are especially useful for making aggregates across multiple categories (sometimes referred to as *nets*, *top box*, or *top 2 box*).

Usage

```
Subtotal(
  name,
  categories = NULL,
  position = c("relative", "top", "bottom"),
  after = NULL,
  before = NULL,
  negative = NULL,
  na.rm = TRUE,
  variable = NULL,
  id = NULL,
  alias = NULL
)

Heading(name, position = c("relative", "top", "bottom"), after = NULL)

subtotals(x)

subtotals(x) <- value

Subtotal(
  name,
  categories = NULL,
  position = c("relative", "top", "bottom"),
  after = NULL,
  before = NULL,
  negative = NULL,
  na.rm = TRUE,
  variable = NULL,
  id = NULL,
  alias = NULL
)

is.Subtotal(x)

is.Heading(x)

are.Subtotals(x)

are.Headings(x)

Heading(name, position = c("relative", "top", "bottom"), after = NULL)

## S4 method for signature 'CrunchVariable'
subtotals(x)

## S4 method for signature 'VariableTuple'
subtotals(x)
```

```
## S4 replacement method for signature 'CrunchVariable,ANY'
subtotals(x) <- value

## S4 replacement method for signature 'CrunchVariable,`NULL`'
subtotals(x) <- value

Subtotal(
  name,
  categories = NULL,
  position = c("relative", "top", "bottom"),
  after = NULL,
  before = NULL,
  negative = NULL,
  na.rm = TRUE,
  variable = NULL,
  id = NULL,
  alias = NULL
)

is.Subtotal(x)

is.Heading(x)

are.Subtotals(x)

are.Headings(x)

Heading(name, position = c("relative", "top", "bottom"), after = NULL)

## S4 method for signature 'CrunchVariable'
subtotals(x)

## S4 method for signature 'VariableTuple'
subtotals(x)

## S4 replacement method for signature 'CrunchVariable,ANY'
subtotals(x) <- value

## S4 replacement method for signature 'CrunchVariable,`NULL`'
subtotals(x) <- value
```

Arguments

name	character the name of the subtotal or heading
categories	character or numeric the category names or ids for subtotal only
position	character one of "relative", "top", or "bottom". Determines the position of the subtotal or heading, either at the top, bottom, or relative to another category in

	the cube (default).
after	character or numeric if position is "relative", then the category name or id to position the subtotal or heading after. If not supplied this defaults to the last of the categories supplied to Subtotal.
before	character or numeric if position is "relative" and also the subtotal is on a Multiple Response variable only.
negative	character or numeric of the category names or ids to be subtracted for subtotals only
na.rm	For Multiple Response subtotals, whether to remove missings before calculating the subtotal (so that if na.rm=TRUE and there is one missing and one selected, the response would selected instead of missing). This defaults to TRUE to match how the crunch web application behaves.
variable	For Multiple Response subtotals, the parent MR variable that contains the subvariables that are being subtotaled (defaults to the variable that is having the Subtotal added to it)
id	For Multiple Response subtotals, an optional number or string to use as the new insertion's id (defaults to a sequential number).
alias	For Multiple Response subtotals, an optional string to use as the new insertion's alias (defaults to letting the server choose the alias)
x	either a variable or CrunchCube object to add or get subtotal transforms for, for is.Subtotal() and is.Heading() an object to test if it is either a Subtotal or Heading
value	For [<-, the replacement Subtotal to insert

Details

To see the subtotals or headings set for a variable, use subtotals(variable)

To see the subtotals or headings set for a variable, use subtotals(variable)

Adding Subtotals and Headings

Subtotals and headings can be added either by passing a list of Subtotals or Headings, or they can be added one at a time by passing Subtotal or Heading to subtotals(variable) alone.

Adding subtotals or headings is additive; meaning that subtotals or headings that are already set on the variable are not removed when new subtotals or headings are added. To remove all subtotals and headings, set subtotals(variable) to NULL.

To get an array of just the subtotal rows from a CrunchCube, use the function subtotalArray(CrunchCube).

Subtotals and headings can be added either by passing a list of Subtotals or Headings, or they can be added one at a time by passing Subtotal or Heading to subtotals(variable) alone.

Adding subtotals or headings is additive; meaning that subtotals or headings that are already set on the variable are not removed when new subtotals or headings are added. To remove all subtotals and headings, set subtotals(variable) to NULL.

To get an array of just the subtotal rows from a CrunchCube, use the function subtotalArray(CrunchCube).

Working with Subtotals and headings

When interacting programmatically with Subtotals and Headings, it can be useful to be able to tell if something is a Subtotal or a Heading. The `is.*` family of methods are useful here: the singular versions (`is.Subtotal` and `is.Heading`) take a single object and returns `TRUE` if the object is either a Subtotal or a Heading and `FALSE` if not; the plural versions (`are.Subtotals` and `are.Headings`) take a list of objects (including an `Insertions` object) and returns a vector of `TRUE`/`FALSE`s.

When interacting programmatically with Subtotals and Headings, it can be useful to be able to tell if something is a Subtotal or a Heading. The `is.*` family of methods are useful here: the singular versions (`is.Subtotal` and `is.Heading`) take a single object and returns `TRUE` if the object is either a Subtotal or a Heading and `FALSE` if not; the plural versions (`are.Subtotals` and `are.Headings`) take a list of objects (including an `Insertions` object) and returns a vector of `TRUE`/`FALSE`s.

Removing transforms

`noTransforms()` is useful if you don't want to see or use any transformations like Subtotals and Headings. This action only applies to the CrunchCube object in R: it doesn't actually change the variables on Crunch servers or the query that generated the CrunchCube.

`noTransforms()` is useful if you don't want to see or use any transformations like Subtotals and Headings. This action only applies to the CrunchCube object in R: it doesn't actually change the variables on Crunch servers or the query that generated the CrunchCube.

Examples

```
## Not run:
# given a variable ds$opinion, with categories: Strongly Agree, Somewhat
# Agree, Neither Agree nor Disagree, Somewhat Disagree, and Strongly Disagree,
# to make two subtotals for Agree and Disagree:
subtotals(ds$opinion) <- list(
    Subtotal(
        name = "Agree",
        categories = c("Strongly Agree", "Somewhat Agree"),
        after = "Somewhat Agree"
    ),
    Subtotal(
        name = "Disagree",
        categories = c("Strongly Disagree", "Somewhat Disagree"),
        after = "Strongly Disagree"
    )
)

# headings can also be added:
subtotals(ds$opinion) <- Heading(name = "All opinions", position = "top")

# to see the subtotals and headings associated with a variable
subtotals(ds$opinion)
#        anchor         name     func    args
# 1      2        Agree subtotal 1 and 2
# 2      4     Disagree subtotal 4 and 5
# 3      0 All opinions     <NA>      NA
```

```
# when you use a variable with subtotals and headings in a cube, you see them
# by default
opinion_cube <- crtabs(~opinion, ds)
opinion_cube
#               All opinions
#             Strongly Agree 23
#             Somewhat Agree 24
#                      Agree 47
# Neither Agree nor Disagree 18
#          Somewhat Disagree 16
#          Strongly Disagree 19
#                   Disagree 35


# to get just the subtotals,
subtotalArray(opinion_cube)
#    Agree Disagree
#       47       35

# to remove all subtotals and headings
subtotals(ds$opinion) <- NULL
crtabs(~opinion, ds)
#             Strongly Agree 23
#             Somewhat Agree 24
# Neither Agree nor Disagree 18
#          Somewhat Disagree 16
#          Strongly Disagree 19

# if you want to temporarily remove subtotals and headings, you can with `noTransforms`
noTransforms(crtabs(~opinion, ds))
#             Strongly Agree             Somewhat Agree Neither Agree nor Disagree
#                          23                         24                         18
#          Somewhat Disagree          Strongly Disagree
#                          16                         19

## End(Not run)

## Not run:
# given a variable ds$opinion, with categories: Strongly Agree, Somewhat
# Agree, Neither Agree nor Disagree, Somewhat Disagree, and Strongly Disagree,
# to make two subtotals for Agree and Disagree:
subtotals(ds$opinion) <- list(
    Subtotal(
        name = "Agree",
        categories = c("Strongly Agree", "Somewhat Agree"),
        after = "Somewhat Agree"
    ),
    Subtotal(
        name = "Disagree",
        categories = c("Strongly Disagree", "Somewhat Disagree"),
        after = "Strongly Disagree"
    )
)
```

```
# headings can also be added:
subtotals(ds$opinion) <- Heading(name = "All opinions", position = "top")

# to see the subtotals and headings associated with a variable
subtotals(ds$opinion)
#        anchor         name     func    args
# 1      2        Agree subtotal 1 and 2
# 2      4     Disagree subtotal 4 and 5
# 3      0 All opinions     <NA>      NA

# when you use a variable with subtotals and headings in a cube, you see them
# by default
opinion_cube <- crtabs(~opinion, ds)
opinion_cube
#               All opinions
#             Strongly Agree 23
#             Somewhat Agree 24
#                      Agree 47
# Neither Agree nor Disagree 18
#          Somewhat Disagree 16
#          Strongly Disagree 19
#                   Disagree 35


# to get just the subtotals,
subtotalArray(opinion_cube)
#    Agree Disagree
#       47       35

# to remove all subtotals and headings
subtotals(ds$opinion) <- NULL
crtabs(~opinion, ds)
#             Strongly Agree 23
#             Somewhat Agree 24
# Neither Agree nor Disagree 18
#          Somewhat Disagree 16
#          Strongly Disagree 19

# if you want to temporarily remove subtotals and headings, you can with `noTransforms`
noTransforms(crtabs(~opinion, ds))
#             Strongly Agree             Somewhat Agree Neither Agree nor Disagree
#                         23                         24                         18
#          Somewhat Disagree          Strongly Disagree
#                         16                         19

## End(Not run)
```

---

subtotalArray *Calculate the transforms from a CrunchCube*

Description

applyTransforms calculates transforms (e.g. Subtotals) on a CrunchCube. Currently only the row transforms are supported. This is useful if you want to use the values from the subtotals of the CrunchCube in an analysis.

Usage

```
subtotalArray(x, ...)

## S4 method for signature 'CrunchCube'
subtotalArray(x, headings = FALSE)

applyTransforms(
  x,
  array = cubeToArray(x),
  transforms_list = transforms(x),
  dims_list = dimensions(x),
  useNA = x@useNA,
  ...
)
```

Arguments

x	a CrunchCube
...	arguments to pass to calcTransforms, for example include
headings	for subtotalArray: a logical indicating if the headings should be included with the subtotals (default: FALSE)
array	an array to use, if not using the default array from the cube itself. (Default: not used, pulls an array from the cube directly)
transforms_list	list of transforms to be applied (default: transforms(x))
dims_list	list of dimensions that correspond to array (default: dimensions(x))
useNA	useNA parameter from the CrunchCube to use (default: x@useNA)

Details

Including the include argument allows you to specify which parts of the CrunchCube to return. The options can be any of the following: "cube_cells" for the untransformed values from the cube itself, "subtotals" for the subtotal insertions, and "headings" for any additional headings. Any combination of these can be given, by default all will be given.

subtotalArray(cube) is a convenience function that is equivalent to applyTransforms(cube, include = c("subtotals"))

Value

an array with any transformations applied

Examples

```
## Not run:
# to get an array of just the subtotals
subtotalArray(crtabs(~opinion, ds))
#    Agree Disagree
#       47       35

# to get the full array with the subtotals but not headings
applyTransforms(crtabs(~opinion, ds), include = c("cube_cells", "subtotals"))
#             Strongly Agree             Somewhat Agree                      Agree
#                         23                         24                         47
# Neither Agree nor Disagree          Strongly Disagree                   Disagree
#                         18                         19                         35
#          Somewhat Disagree
#                         16
# to get the full array with the headings but not subtotals
applyTransforms(crtabs(~opinion, ds), include = c("cube_cells", "headings"))
#               All opinions             Strongly Agree             Somewhat Agree
#                         NA                         23                         24
# Neither Agree nor Disagree          Strongly Disagree          Somewhat Disagree
#                         18                         19                         16

## End(Not run)
```

Subvariables-class *Subvariables in Array Variables*

Description

Multiple-response and categorical-array variables are higher order variables which are made up of sets of subvariables. These methods allow you to retrieve and change the subvariables of a multiple-response or categorical-array variable.

Usage

```
subvariables(x)

subvariables(x) <- value

## S4 method for signature 'ArrayVariable'
subvariables(x)

## S4 method for signature 'CrunchVariable'
subvariables(x)

## S4 method for signature 'VariableTuple'
subvariables(x)
```

```
## S4 replacement method for signature 'ArrayVariable,ANY'
subvariables(x) <- value

## S4 replacement method for signature 'ArrayVariable,Subvariables'
subvariables(x) <- value
```

Arguments

x	A Variable or Subvariables object
value	For the setters, the appropriate values to set

Details

Subvariables can be accessed from array variables (including multiple response) with the `subvariables` method. They can be assigned back with the `subvariables<-` setter, but there are limitations to what is supported. Specifically, you can reorder subvariables, but you cannot add or remove subvariables by `subvariables<-` assignment. See `deleteSubvariable` to remove subvariables from an array.

Subvariables have a `names` attribute that can be accessed, showing the display names of the subvariables. These can be set with the `names<-` method.

Finally, subvariables can be accessed as regular (categorical) variables with the `$` and `[[` extract methods.

See the vignette on array variables for further details and examples.

See Also

`describe-catalog` `deleteSubvariable` `vignette("array-variables", package="crunch")`

SummaryStat-class *Summary insertions*

Description

Just like `subtotals()`s, summary statistics can be inserted into cubes. `SummaryStat()` makes an object of type `SummaryStat` which can be added on to a CrunchCube's `insertions` to add the specified summary statistic. Currently only `mean` and `median` are supported; both use weighted algorithms to go from counts and numeric values of categories to the expected statistic. Although `SummaryStat` objects can be made by hand, it is recommended instead to use the `addSummaryStat()` function which is much quicker and easier to simply add a summary statistic to an existing CrunchCube.

Usage

```
SummaryStat(
  name,
  stat,
  categories = NULL,
  position = c("relative", "top", "bottom"),
  after = NULL,
  before = NULL,
  includeNA = FALSE
)

SummaryStat(
  name,
  stat,
  categories = NULL,
  position = c("relative", "top", "bottom"),
  after = NULL,
  before = NULL,
  includeNA = FALSE
)

is.SummaryStat(x)

are.SummaryStats(x)
```

Arguments

name	character the name of the summary statistic
stat	a function to calculate the summary (e.g. mean or median)
categories	character or numeric the category names or ids to be included in the summary statistic, if empty all categories
position	character one of "relative", "top", or "bottom". Determines the position of the subtotal or heading, either at the top, bottom, or relative to another category in the cube (default)
after	character or numeric if position is "relative", then the category name or id to position the subtotal or heading after
before	character or numeric if position is relative (and the insertion type allows it - currently only MR subtotals).
includeNA	should missing categories be included in the summary?
x	for is.SummaryStat() only, an object to test if it is a SummaryStat object

Details

Summary statistics are intended only for CrunchCube objects, and are not able to be set on Crunch variables.

Removing transforms

noTransforms() is useful if you don't want to see or use any transformations like Subtotals and Headings. This action only applies to the CrunchCube object in R: it doesn't actually change the variables on Crunch servers or the query that generated the CrunchCube.

---

tabBook *Compute a Tab Book*

Description

This function allows you to generate a tab book from a multitable and data. As with other functions, you can select the rows and columns you want to work with by subsetting the dataset you pass into the function.

Usage

```
tabBook(
  multitable,
  dataset,
  weight = crunch::weight(dataset),
  output_format = c("json", "xlsx", "csv"),
  file,
  filter = NULL,
 use_legacy_endpoint = envOrOption("use.legacy.tabbook.endpoint", FALSE, expect_lgl =
    TRUE),
  ...
)
```

Arguments

multitable	a Multitable object
dataset	CrunchDataset, which may be subset with a filter expression on the rows, and a selection of variables to use on the columns.
weight	a CrunchVariable that has been designated as a potential weight variable for dataset, or NULL for unweighted results. Default is the currently applied weight.
output_format	character export format: currently supported values are "json" (default), "xlsx" and "csv".
file	character local filename to write to. A default filename will be generated from the multitable's name if one is not supplied and the "xlsx" format is requested. Not required for "json" format export.
filter	a Crunch filter object or a vector of names of filters defined in the dataset.
use_legacy_endpoint	Logical, indicating whether to use a 'legacy' endpoint for compatibility (this endpoint will be removed in the future). Defaults to FALSE, but can be set in the function, or with the environment variable R_USE_LEGACY_TABBOOK_ENDPOINT or R option use.legacy.tabbook.endpoint.

... Additional "options" passed to the tab book POST request. More details can be found in the crunch API documentation

Details

By specifying a "json" `format`, instead of generating an Excel workbook, you'll get a TabBookResult object, containing nested CrunchCube results. You can then further format these and construct custom tab reports.

Value

If "json" format is requested, the function returns an object of class `TabBookResult`, containing a list of `MultitableResult` objects, which themselves contain `CrunchCubes`. If "xlsx" or "csv", is requested, the function invisibly returns the filename (`file`, if specified, or the the autogenerated file name). If you request "json" and wish to access the JSON data underlying the `TabBookResult`, pass in a path for `file` and you will get a JSON file written there as well.

Examples

```
## Not run:
# Excel export
m <- newMultitable(~ gender + age4 + marstat, data = ds)
tabBook(m, ds, format = "xlsx", file = "wealthy-tab-book.xlsx", filter = "wealthy")

# csv export
tabBook(
    mt,
    ds[c("q5a", "q8", "q2a_1", "q2a_2")],
    output_format = "csv",
    file = "tabbook.csv",
    format = list(
        pval_colors = FALSE,
        decimal_places = list(percentages = 0L, other = 2L),
        show_empty = FALSE
    ),
    sig_threshold = 0.05,
    doc_layout = list(toc = FALSE, variable_sheets = "one_sheet"),
    fields = c(
        "col_percent", "row_percent", "count_unweighted", "mean",
        "valid_count_weighted", "valid_count_unweighted"
    ),
    page_layout = list(
        rows = list(
            top = c("base_weighted", "base_unweighted"),
            bottom = c("scale_mean", "scale_median")
        ),
        measure_layout = "long"
    )
)

# JSON export (loads into R)
book <- tabBook(m, ds)
```

```
tables <- prop.table(book, 2)


## End(Not run)
```

---

tabbook-dim *TabBookResult and MultitableResult dimension*

Description

TabBookResult and MultitableResult dimension

Usage

```
## S4 method for signature 'TabBookResult'
dim(x)
```

Arguments

x	a TabBookResult or MultitableResult

Value

Returns what you'd expect.

---

table *Table function for Crunch objects*

Description

Table function for Crunch objects

Usage

```
table(..., exclude, useNA = c("no", "ifany", "always"), dnn, deparse.level)
```

Arguments

...	CrunchVariables to tabulate
exclude	see base::table
useNA	see base::table
dnn	see base::table
deparse.level	see base::table

Value

a table object

See Also

base::table

---

team *Share Crunch assets with a team*

Description

You can share filters and multitables with a team that you are on. This will give all team members access to view and edit these filters. Use `getTeams()` to see what teams you are on.

Usage

```
team(x)

## S4 method for signature 'CrunchFilter'
team(x)

## S4 method for signature 'Multitable'
team(x)

## S4 method for signature 'CrunchDeck'
team(x)

team(x) <- value

## S4 replacement method for signature 'CrunchFilter'
team(x) <- value

## S4 replacement method for signature 'Multitable'
team(x) <- value

## S4 replacement method for signature 'CrunchDeck'
team(x) <- value
```

Arguments

x	a `CrunchFilter` or `Multitable`
value	a `CrunchTeam` or url for a Crunch team

Value

a `CrunchTeam` that the asset is shared with.

temp.options *Set some global options temporarily*

Description

Set some global options temporarily

Usage

```
temp.options(..., crunch = list())

temp.option(..., crunch = list())
```

Arguments

...	named options to set using `options()`
crunch	named list of options to set in crunch's higher priority options environment

Value

an S3 class "contextManager" object

See Also

`with-context-manager` `ContextManager`

titles *Manipulate deck titles*

Description

Crunch slides have titles and subtitles. You can change these features at either the deck level by assigning a character vector which is the same length as the deck to the CrunchDeck, or by assigning character strings to the the slide.

Usage

```
titles(x)

titles(x) <- value

title(x)

title(x) <- value
```

```
subtitles(x, value)

subtitles(x) <- value

subtitle(x, value)

subtitle(x) <- value

## S4 method for signature 'CrunchDeck'
titles(x)

## S4 replacement method for signature 'CrunchDeck'
titles(x) <- value

## S4 method for signature 'CrunchDeck'
subtitles(x)

## S4 replacement method for signature 'CrunchDeck'
subtitles(x) <- value

## S4 method for signature 'SlideCatalog'
titles(x)

## S4 replacement method for signature 'SlideCatalog'
titles(x) <- value

## S4 method for signature 'SlideCatalog'
subtitles(x)

## S4 replacement method for signature 'SlideCatalog'
subtitles(x) <- value

## S4 method for signature 'CrunchSlide'
title(x)

## S4 replacement method for signature 'CrunchSlide'
title(x) <- value

## S4 method for signature 'CrunchSlide'
subtitle(x)

## S4 replacement method for signature 'CrunchSlide'
subtitle(x) <- value
```

Arguments

x	a CrunchDeck or CrunchSlide
value	character, the new title or subtitle

Value

x, modified

Examples

```
## Not run:
titles(deck)
titles(deck) <- c(new_title1, new_title2)
slide <- deck[[1]]
title(slide) <- "new title"
subtitle(slide) <- "new subtitle"
subtitles(deck)

## End(Not run)
```

---

tojson-crunch *toJSON methods for Crunch objects*

Description

crunch uses the jsonlite package for JSON serialization and deserialization. Unfortunately, jsonlite::toJSON() does not allow for defining S4 methods for other object types. So, crunch::toJSON wraps jsonprep, which exists to translate objects to base R objects, which jsonlite::toJSON can handle. jsonprep is defined as an S4 generic, and it is exported, so you can define methods for it if you have other objects that you want to successfully serialize to JSON.

Usage

```
jsonprep(x, ...)

## S4 method for signature 'AbstractCategories'
jsonprep(x, ...)

## S4 method for signature 'ANY'
jsonprep(x, ...)

## S4 method for signature 'ShojiOrder'
jsonprep(x, ...)

## S4 method for signature 'OrderGroup'
jsonprep(x, ...)

toJSON(x, ..., for_query_string = FALSE)
```

Arguments

x	the object
...	additional arguments
for_query_string	If TRUE, and crunch.stabilize.query option is also set to TRUE, then dictionary items in the JSON are sorted alphabetically, which can be useful when capturing mocks using "httptest".

Value

jsonprep returns a base R object that jsonlite::toJSON can handle. toJSON returns the JSON-serialized character object.

See Also

jsonlite::toJSON()

---

toVariable *Generic method for converting objects to Crunch representations*

Description

R objects are converted to Crunch objects using the following rules:

Usage

```
toVariable(x, ...)

## S4 method for signature 'CrunchVarOrExpr'
toVariable(x, ...)

## S4 method for signature 'character'
toVariable(x, ...)

## S4 method for signature 'numeric'
toVariable(x, ...)

## S4 method for signature 'factor'
toVariable(x, ...)

## S4 method for signature 'Date'
toVariable(x, ...)

## S4 method for signature 'POSIXt'
toVariable(x, ...)
```

```
## S4 method for signature 'AsIs'
toVariable(x, ...)

## S4 method for signature 'VariableDefinition'
toVariable(x, ...)

## S4 method for signature 'logical'
toVariable(x, ...)

## S4 method for signature 'labelled'
toVariable(x, ...)

## S4 method for signature 'haven_labelled'
toVariable(x, ...)

## S4 method for signature 'labelled_spss'
toVariable(x, ...)

## S4 method for signature 'haven_labelled_spss'
toVariable(x, ...)
```

Arguments

x	An R vector you want to turn into a Crunch variable
...	Additional metadata fields for the variable, such as "name" and "description". See the API documentation for a complete list of valid attributes.

Details

- Character vectors are converted into Crunch text variables
- Numeric vectors are converted into Crunch numeric variables
- Factors are converted to categorical variables
- Date and POSIXt vectors are converted into Crunch datetime variables
- Logical vectors are converted to Crunch categorical variables
- `VariableDefinition()`s are not converted, but the function can still append additional metadata

If you have other object types you wish to convert to Crunch variables, you can declare methods for `toVariable`.

Value

A `VariableDefinition` object. To add this to a dataset, either assign it into the dataset (like `ds$newvar <- toVariable(...)`) or call `addVariables()`. If you're adding a column of data to a dataset, it must be as long as the number of rows in the dataset, or it may be a single value to be recycled for all rows.

See Also

VariableDefinition() addVariables()

Examples

```
var1 <- rnorm(10)
toVariable(var1)
toVariable(var1, name = "Random", description = "Generated in R")
## Not run:
ds$random <- toVariable(var1, name = "Random")
# Or, this way:
ds <- addVariables(ds, toVariable(var1, name = "Random"))

## End(Not run)
```

Transforms-class *Transformations of variable and cube views*

Description

Transformations allow you to change how a variable or cube is displayed without changing the underlying data.

Usage

```
Transforms(..., data = NULL)

TransformsList(..., data = NULL)

transforms(x)

transforms(x) <- value

## S4 method for signature 'CrunchVariable'
transforms(x)

## S4 method for signature 'VariableTuple'
transforms(x)

## S4 replacement method for signature 'CrunchVariable,Transforms'
transforms(x) <- value

## S4 replacement method for signature 'CrunchVariable,`NULL`'
transforms(x) <- value

## S4 method for signature 'CrunchCube'
transforms(x)
```

```
## S4 method for signature 'VariableCatalog'
transforms(x)

## S4 replacement method for signature 'CrunchCube,ANY'
transforms(x) <- value

## S4 replacement method for signature 'CrunchCube,TransformsList'
transforms(x) <- value

## S4 replacement method for signature 'CrunchCube,`NULL`'
transforms(x) <- value
```

Arguments

`...`	For the constructor function `Transforms` you can pass in attributes via `...`
`data`	For the constructor function `Transforms` you can either pass in attributes via `...` or you can create the objects with a fully defined `list` representation of the objects via the `data` argument. See the examples.
`x`	For the attribute getters and setters, an object of class Transforms
`value`	For the setter, the replacement Transforms to insert

Getting transformations

The `transforms(x)` methods can be used with Variables and CrunchCubes to get what transformations are currently set. For variables, they return a single `Transforms` object that includes all transformations for the variable. For CrunchCubes, it returns a named list with the same length as the number of dimensions of the cube with each dimension's transformations.

Currently, Insertions (e.g. Subtotal() and Heading()) are the only type of transformations that are supported.

Setting transformations on a variable

The `transforms(x) <- value` methods can be used to assign transformations for a specific variable. `value` must be a `Transforms` object. This allows you to set transformations on categorical variables. These transformations will automatically show up in any new CrunchCubes that contain this variable.

Setting transformations on a CrunchCube

The `transforms(x) <- value` methods can also be used to assign transformations to a CrunchCube that has already been calculated. `value` must be a named list of `Transforms` objects. The names of this list must correspond to dimensions in the cube (those dimensions correspondences are matched based on variable aliases). You don't have to provide an entry for each dimension, but any dimension you do provide will be overwritten fully.

Removing transformations

To remove transformations from a variable or CrunchCube, use `transforms(x) <- NULL`.

type *Change Crunch variable types*

Description

Numeric, text, and categorical variables can be cast to one another by assigning them a new "type". This modifies the storage of the data on the server and should only be done in narrow circumstances, as in when importing data from a different file format has resulted in incorrect types being specified.

Usage

```
type(x)

type(x) <- value

## S4 method for signature 'CrunchVariable'
type(x)

## S4 method for signature 'VariableEntity'
type(x)

## S4 replacement method for signature 'CrunchVariable'
type(x) <- value
```

Arguments

x	a Variable
value	For the setter, a character value in c("numeric", "text", "categorical")

Value

Getter returns character; setter returns x duly modified.

unbind *Split an array or multiple-response variable into its CategoricalVariables*

Description

Split an array or multiple-response variable into its CategoricalVariables

Usage

```
unbind(x)
```

Arguments

x	an ArrayVariable

Value

invisibly, the API response from DELETEing the array variable definition. If you refresh() the corresponding dataset after unbinding, you should see the array variable removed and its subvariables promoted to regular variables.

---

unshare *Revoke a user's access to a dataset*

Description

Revoke a user's access to a dataset

Usage

```
unshare(dataset, users)
```

Arguments

dataset	a CrunchDataset
users	character: email address(es) or URLs of the users or teams to unshare with.

Value

Invisibly, the dataset.

See Also

share

---

users *Get information about users who have access to a dataset*

Description

Get user metadata about all of the users that have access to a particular Crunch object like a dataset or project. Returns a UserCatalog object which can be translated into a data.frame with catalogToDataFrame() if information needs to be extracted, queried, transformed, etc.

Usage

```
users(x)

## S4 method for signature 'CrunchDataset'
users(x)

## S4 method for signature 'DatasetTuple'
users(x)

## S4 method for signature 'ProjectFolder'
users(x)
```

Arguments

x	a CrunchDataset, DatasetTuple, or ProjectFolder object to get users from

Value

a UserCatalog with information about users who have access to the dataset

---

var-categories *Get and set Categories on Variables*

Description

Get and set Categories on Variables

Usage

```
categories(x)

categories(x) <- value

## S4 method for signature 'VariableTuple'
categories(x)

## S4 method for signature 'CrunchVariable'
categories(x)

## S4 method for signature 'CategoricalVariable'
categories(x)

## S4 method for signature 'CategoricalArrayVariable'
categories(x)

## S4 method for signature 'VariableEntity'
categories(x)
```

```
## S4 replacement method for signature 'CategoricalVariable,Categories'
categories(x) <- value

## S4 replacement method for signature 'CategoricalArrayVariable,Categories'
categories(x) <- value

## S4 replacement method for signature 'CategoricalVariable,numeric'
categories(x) <- value

## S4 replacement method for signature 'CategoricalVariable,character'
categories(x) <- value

## S4 replacement method for signature 'CategoricalVariable,ANY'
categories(x) <- value

## S4 replacement method for signature 'CategoricalArrayVariable,numeric'
categories(x) <- value

## S4 replacement method for signature 'CategoricalArrayVariable,character'
categories(x) <- value

## S4 replacement method for signature 'CategoricalArrayVariable,ANY'
categories(x) <- value

## S4 replacement method for signature 'CrunchVariable,ANY'
categories(x) <- value
```

Arguments

x	a Variable
value	for the setters, an object of class Categories to set.

Value

Getters return Categories; setters return x duly modified.

`VariableCatalog-class` *Collection of Variables within a Dataset*

Description

A VariableCatalog contains references to all variables in a dataset, plus some descriptive metadata about each. Each VariableCatalog also contains a `VariableOrder` that governs how variables within it are organized.

`VariableDefinition` *Construct a variable definition*

Description

Crunch variables are created by posting a `VariableDefinition` to the Crunch server. The `VariableDefinition` contains the information the server requires to calculate the variable. This can information can either be in the form of the actual data which you would like to include in the variable, or a derivation which tells the server how to derive the new variable from existing ones. This function converts an R vector or set of attributes into a variable definition which can be posted to the server.

Usage

```
VariableDefinition(data, ...)

VarDef(data, ...)
```

Arguments

`data`	an R vector of data to convert to the Crunch payload format. See toVariable for how R data types are converted. This function can also be used to construct a `VariableDefinition` directly by passing attributes to `...`. This is only recommended for advanced users who are familiar with the Crunch API.
`...`	additional attributes to be included in the `VariableDefinition`

Value

a `VariableDefinition` object, ready to POST to Crunch.

See Also

`toVariable`

Examples

```
VariableDefinition(rnorm(5),
    name = "Some numbers",
    description = "Generated pseudorandomly from the normal distribution"
)
VarDef(
    name = "Integers", values = 1:5, type = "numeric",
    description = "When creating variable definitions with 'values', you must
    specify 'type', and categorical variables will require 'categories'."
)
```

variableMetadata *Get all variable metadata for a dataset*

Description

Crunch stores variable information in several catalogs containing information abut the variable class, its missingness and subvariables. This function allows you to access that information.

Usage

```
variableMetadata(dataset)
```

Arguments

dataset	CrunchDataset

Value

A VariableCatalog with all variable properties, including categories and subvariables.

VariableOrder-class *Organize Variables within a Dataset*

Description

Variables in the Crunch web application can be viewed in an ordered, hierarchical list. These objects and methods allow you to modify that order from R.

Details

A VariableOrder object is a subclass of `list` that contains VariableGroups. VariableGroup objects contain a group name and an set of "entities", which can be variable references or other nested VariableGroups.

Slots

`group` character, the name of the VariableGroup. In the constructor and more generally, this field can be referenced as "name" as well.

`entities` a character vector of variable URLs, or a list containing a combination of variable URLs and VariableGroup objects.

`duplicates` logical: should duplicate variable references be allowed in this object? Deprecated field: duplicates are never allowed.

`vars` either `NULL` or a `VariableCatalog()`. If not `NULL`, it will be used to look up variable names from the URLs.

variables *Access a catalog of variables*

Description

Datasets contain collections of variables. For some purposes, such as editing variables' metadata, it is helpful to access these variable catalogs more directly. Other objects, such as cubes and folders, also define `variables()` methods that expose variable metadata.

Usage

```
variables(x)

variables(x) <- value

allVariables(x)

allVariables(x) <- value

## S4 method for signature 'CubeDims'
variables(x)

## S4 method for signature 'CrunchCube'
variables(x)

## S4 method for signature 'CrunchDataset'
variables(x)

## S4 replacement method for signature 'CrunchDataset,VariableCatalog'
variables(x) <- value

## S4 method for signature 'CrunchDataset'
allVariables(x)

## S4 replacement method for signature 'CrunchDataset,VariableCatalog'
allVariables(x) <- value

## S4 method for signature 'SearchResults'
variables(x)

## S4 method for signature 'VariableFolder'
variables(x)
```

Arguments

x	a Dataset
value	For the setters, a VariableCatalog to assign.

Details

For datasets, `variables()` returns only the active variables in the dataset, while `allVariables()` returns all variables, including hidden variables. `allVariables()` is not defined for other objects.

Value

All methods return a `VariableCatalog` except the `VariableFolder` method, which returns a subset of `x` containing only variable references. Assignment functions return `x` with the changes made.

---

versions *Access the saved versions of a dataset*

Description

This function allows you to see a dataset's savepoints. These can then be passed to `restoreVersion()` to load the previously saved version of a dataset.

Usage

```
versions(x)
```

Arguments

`x`	a `CrunchDataset`

Value

an object of class `VersionCatalog`. Supported methods on the catalog include "names" and "timestamps".

See Also

saveVersion restoreVersion

---

webApp *View a Crunch Object in the Web Application*

Description

Convenience function that will use your system's "open" command to open a Crunch object in our web application in your default browser.

Usage

```
webApp(x)
```

Arguments

x	a Crunch Dataset or Variable

Value

Nothing; called for side effect of opening your web browser.

---

weightVariables *Get a dataset's weightVariables*

Description

Get a dataset's weightVariables

Usage

```
weightVariables(x)

## S4 method for signature 'CrunchDataset'
weightVariables(x)

## S4 method for signature 'VariableCatalog'
weightVariables(x)
```

Arguments

x	a CrunchDataset

Value

`weightVariables` returns a character vector of the aliases of the variables that are eligible to be used as weights.

See Also

`weight()` `makeWeight()` `modifyWeightVariables()`

`weightVariables<-` *Change which variables can be set as a dataset's weight.*

Description

`modifyWeightVariables` allows you to change the variables which are eligible to be used as a dataset's weight. You can also add variables to the weight variables catalog by assignment with `weightVariables(ds) <- "weight"` or `is.weightVariable(ds$weight) <- TRUE`.

Usage

```
weightVariables(x) <- value

is.weightVariable(x) <- value

modifyWeightVariables(x, vars, type = "append")

## S4 replacement method for signature 'CrunchDataset'
weightVariables(x) <- value

is.weightVariable(x)

## S4 replacement method for signature 'NumericVariable'
is.weightVariable(x) <- value
```

Arguments

`x`	a CrunchDataset
`value`	For the `weightVariables()` and `is.weightVariable` setters the variables to append to a dataset's weightVariables.
`vars`	Variables to add or remove this can be a numeric Crunch variable, list of numeric Crunch variables or a character vector with the aliases of numeric Crunch variables. Setting vars to `NULL` clears a datasets `weightVariables`
`type`	a character string determining how the weightVariables will be modified: • `"append"` : add `vars` to the current weight variables • `"remove"` : remove `vars` from the current list of weight variables • `"replace"`: replace the current weight variables with `vars`

Details

Editors can change which variables can be set as the weighting variable for a dataset. For instance if several weights have been calculated they can let the user choose which of those variables to use a weight, but prevent the user from choosing other variables as weight. This function allows you to change the `weightVariables` of a dataset.

Value

a CrunchDataset

Examples

```
## Not run:
modifyweightVariables(ds, "weight", "append")
weightVariables(ds) <- list(ds$weight, ds$weight2)
weightVariables(ds) <- NULL
weightVariables(ds) <- c("weight", "weight2")
is.weightVariables(ds$weight) <- TRUE

## End(Not run)
```

`which` *"which" method for CrunchLogicalExpr*

Description

"which" method for CrunchLogicalExpr

Usage

```
## S4 method for signature 'CrunchLogicalExpr'
which(x, arr.ind = FALSE, useNames = TRUE)
```

Arguments

`x`	CrunchLogicalExpr
`arr.ind`	Ignored
`useNames`	Ignored

Value

Integer row indices where `x` is true. Note that this does not return a Crunch expression. Use this when you need to translate to R values. For filtering a Crunch expression by `x`, don't use `which`.

with-context-manager *Context manager's "with" method*

Description

Context manager's "with" method

Usage

```
## S3 method for class 'contextManager'
with(data, expr, ...)
```

Arguments

data	contextManager
expr	code to evaluate within that context
...	additional arguments. One additional supported argument is "as", which lets you assign the return of your "enter" function to an object you can access.

Value

Nothing.

See Also

ContextManager

write.csv.gz *Write CSV to a compressed file*

Description

Write CSV to a compressed file

Usage

```
write.csv.gz(x, file, na = "", row.names = FALSE, ...)
```

Arguments

x	A data.frame or similar CSV-writable object
file	character destination to write the gzipped CSV to
na	See utils::write.csv(). This just changes the default to a Crunch-friendly empty string.
row.names	logical: write out row names? See utils::write.csv().
...	Additional arguments passed to write.csv.

Value

A csv file written to dist

Index