Package: teal.data (via r-universe)

October 10, 2024

Type Package

Title Data Model for 'teal' Applications

Version 0.6.0

Date 2024-04-29

Description Provides a 'teal_data' class as a unified data model for 'teal' applications focusing on reproducibility and relational data.

License Apache License 2.0

URL https://insightsengineering.github.io/teal.data/, https://github.com/insightsengineering/teal.data/

BugReports https://github.com/insightsengineering/teal.data/issues

Depends R (>= 4.0), teal.code (>= 0.5.0)

Imports checkmate (>= 2.1.0), lifecycle (>= 0.2.0), methods, rlang (>= 1.1.0), stats, utils

Suggests knitr (>= 1.42), rmarkdown (>= 2.19), testthat (>= 3.1.5), withr (>= 2.0.0)

VignetteBuilder knitr

RdMacros lifecycle

Config/Needs/verdepcheck insightsengineering/teal.code, mllg/checkmate, r-lib/lifecycle, r-lib/rlang, yihui/knitr, rstudio/rmarkdown, r-lib/testthat, r-lib/withr

Config/Needs/website insightsengineering/nesttemplate

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Collate 'cdisc_data.R' 'data.R' 'formatters_var_labels.R'
'deprecated.R' 'dummy_function.R' 'join_key.R' 'join_keys-c.R'
'join_keys-extract.R' 'join_keys-names.R' 'join_keys-parents.R'
'join_keys-print.R' 'join_keys-utils.R' 'join_keys.R'
'teal.data.R' 'teal_data-class.R' 'teal_data-datanames.R'
'teal_data-get_code.R' 'teal_data-show.R' 'teal_data.R'
'testhat-helpers.R' 'topological_sort.R'
'utils-get_code_dependency.R' 'verify.R' 'zzz.R'

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

RemoteUrl https://github.com/insightsengineering/teal.data

RemoteRef v0.6.0

RemoteSha 3f1ee88ea7b884501d3432fcd131d73f29cb284c

Contents

cdisc_data 2
col_labels 4
datanames 5
default_cdisc_join_keys 6
example_cdisc_data 6
get_code,teal_data-method 7
join_key 9
join_keys 10
names<-.join_keys 14
parents 14
show,teal_data-method 16
TealData 17
teal_data 19
verify 20

Index **22**

`cdisc_data` *Data input for* `teal` *app*

Description

[Stable]

Function is a wrapper around `teal_data()` and guesses `join_keys` for given datasets whose names match ADAM datasets names.

Usage

```
cdisc_data(
  ...,
  join_keys = teal.data::default_cdisc_join_keys[names(rlang::list2(...))],
  code = character(0),
  check
)
```

Arguments

...	any number of objects (presumably data objects) provided as name = value pairs.
join_keys	(join_keys or single join_key_set) optional object with datasets column names used for joining. If empty then it would be automatically derived basing on intersection of datasets primary keys. For ADAM datasets it would be automatically derived.
code	(character, language) optional code to reproduce the datasets provided in .... Note this code is not executed and the teal_data may not be reproducible
check	(logical) **[Deprecated]** Use verify() to verify code reproducibility .

Details

This function checks if there were keys added to all data sets.

Value

A teal_data object.

Examples

```
data <- cdisc_data(
  join_keys = join_keys(
    join_key("ADSL", "ADTTE", c("STUDYID" = "STUDYID", "USUBJID" = "USUBJID"))
  )
)

data <- within(data, {
  ADSL <- example_cdisc_data("ADSL")
  ADTTE <- example_cdisc_data("ADTTE")
})
```

`col_labels` *Variable labels*

Description

Get or set variable labels in a `data.frame`.

Usage

```
col_labels(x, fill = FALSE)

col_labels(x) <- value

col_relabel(x, ...)

get_labels(...)
```

Arguments

`x`	(`data.frame` or `DataFrame`) data object
`fill`	(`logical(1)`) specifying what to return if variable has no label
`value`	(`character`) vector of variable labels of length equal to number of columns in `x`; if named, names must match variable names in `x` and will be used as key to set labels; use `NA` to remove label from variable
`...`	name-value pairs, where name corresponds to a variable name in `x` and value is the new variable label; use `NA` to remove label from variable

Details

Variable labels can be stored as a `label` attribute set on individual variables. These functions get or set this attribute, either on all (`col_labels`) or some variables (`col_relabel`).

[Deprecated]

In previous versions of `teal.data` labels were managed with `get_labels()`. This function is deprecated as of `0.4.0`, use `col_labels` instead.

Value

For `col_labels`, named character vector of variable labels, the names being the corresponding variable names. If the `label` attribute is missing, the vector elements will be the variable names themselves if `fill = TRUE` and `NA` if `fill = FALSE`.

For `col_labels<-` and `col_relabel`, copy of `x` with variable labels modified.

Source

These functions were taken from formatters package, to reduce the complexity of the dependency tree and rewritten.

Examples

```
x <- iris
col_labels(x)
col_labels(x) <- paste("label for", names(iris))
col_labels(x)
y <- col_relabel(x, Sepal.Length = "Sepal Length of iris flower")
col_labels(y)
```

---

datanames *Names of data sets in* teal_data *object*

Description

Get or set the value of the datanames slot.

Usage

```
datanames(x)

datanames(x) <- value
```

Arguments

x	(teal_data) object to access or modify
value	(character) new value for @datanames; all elements must be names of variables existing in @env

Details

The @datanames slot in a teal_data object specifies which of the variables stored in its environment (the @env slot) are data sets to be taken into consideration. The contents of @datanames can be specified upon creation and default to all variables in @env. Variables created later, which may well be data sets, are not automatically considered such. Use this function to update the slot.

Value

The contents of @datanames or teal_data object with updated @datanames.

Examples

```
td <- teal_data(iris = iris)
td <- within(td, mtcars <- mtcars)
datanames(td)

datanames(td) <- c("iris", "mtcars")
datanames(td)
```

default_cdisc_join_keys *List containing default joining keys for* CDISC *datasets*

Description

This data object is created at loading time from `cdisc_datasets/cdisc_datasets.yaml`.

Usage

```
default_cdisc_join_keys
```

Format

An object of class `join_keys` (inherits from `list`) of length 19.

Source

internal

example_cdisc_data *Generate sample CDISC datasets*

Description

Retrieves example CDISC datasets for use in examples and testing.

Usage

```
example_cdisc_data(
  dataname = c("ADSL", "ADAE", "ADLB", "ADCM", "ADEX", "ADRS", "ADTR", "ADTTE", "ADVS")
)
```

Arguments

`dataname`	(`character(1)`) name of a CDISC dataset

Details

This function returns a dummy dataset and should only be used within `teal.data`. Note that the datasets are not created and maintained in `teal.data`, they are retrieved its dependencies.

Value

A CDISC dataset as a `data.frame`.

get_code,teal_data-method
Get code from teal_data *object*

Description

Retrieve code from teal_data object.

Usage

```
## S4 method for signature 'teal_data'
get_code(object, deparse = TRUE, datanames = NULL, ...)
```

Arguments

object	(teal_data)
deparse	(logical) flag specifying whether to return code as character (deparse = TRUE) or as expression (deparse = FALSE).
datanames	**[Experimental]** (character) vector of dataset names to return the code for. For more details see the "Extracting dataset-specific code" section.
...	Parameters passed to internal methods. Currently, the only supported parameter is check_names (logical(1)) flag, which is TRUE by default. Function warns about missing objects, if they do not exist in code but are passed in datanames. To remove the warning, set check_names = FALSE.

Details

Retrieve code stored in @code, which (in principle) can be used to recreate all objects found in @env. Use datanames to limit the code to one or more of the datasets enumerated in @datanames. If the code has not passed verification (with verify()), a warning will be prepended.

Value

Either a character string or an expression. If datanames is used to request a specific dataset, only code that *creates* that dataset (not code that uses it) is returned. Otherwise, all contents of @code.

Extracting dataset-specific code

When datanames is specified, the code returned will be limited to the lines needed to *create* the requested datasets. The code stored in the @code slot is analyzed statically to determine which lines the datasets of interest depend upon. The analysis works well when objects are created with standard infix assignment operators (see ?assignOps) but it can fail in some situations.

Consider the following examples:

Case 1: Usual assignments.

```
data <- teal_data() |>
  within({
    foo <- function(x) {
      x + 1
    }
    x <- 0
    y <- foo(x)
  })
get_code(data, datanames = "y")
```

x has no dependencies, so `get_code(data, datanames = "x")` will return only the second call.
y depends on x and foo, so `get_code(data, datanames = "y")` will contain all three calls.

Case 2: Some objects are created by a function's side effects.

```
data <- teal_data() |>
  within({
    foo <- function() {
      x <<- x + 1
    }
    x <- 0
    foo()
    y <- x
  })
get_code(data, datanames = "y")
```

Here, y depends on x but x is modified by foo as a side effect (not by reassignment) and so `get_code(data, datanames = "y")` will not return the `foo()` call.
To overcome this limitation, code dependencies can be specified manually. Lines where side effects occur can be flagged by adding "`# @linksto <object name>`" at the end.
Note that `within` evaluates code passed to `expr` as is and comments are ignored. In order to include comments in code one must use the `eval_code` function instead.

```
data <- teal_data() |>
  eval_code("
    foo <- function() {
      x <<- x + 1
    }
    x <- 0
    foo() # @linksto x
    y <- x
  ")
get_code(data, datanames = "y")
```

Now the `foo()` call will be properly included in the code required to recreate y.

Note that two functions that create objects as side effects, `assign` and `data`, are handled automatically.

Here are known cases where manual tagging is necessary:

- non-standard assignment operators, *e.g.* `%<>%`
- objects used as conditions in `if` statements: `if (<condition>)`
- objects used to iterate over in `for` loops: `for(i in <sequence>)`
- creating and evaluating language objects, *e.g.* `eval(<call>)`

Examples

```
tdata1 <- teal_data()
tdata1 <- within(tdata1, {
  a <- 1
  b <- a^5
  c <- list(x = 2)
})
get_code(tdata1)
get_code(tdata1, datanames = "a")
get_code(tdata1, datanames = "b")

tdata2 <- teal_data(x1 = iris, code = "x1 <- iris")
get_code(tdata2)
get_code(verify(tdata2))
```

`join_key` *Create a relationship between a pair of datasets*

Description

[Stable]

Create a relationship between two datasets, `dataset_1` and `dataset_2`. By default, this function establishes a directed relationship with `dataset_1` as the parent. If `dataset_2` is not specified, the function creates a primary key for `dataset_1`.

Usage

```
join_key(dataset_1, dataset_2 = dataset_1, keys, directed = TRUE)
```

Arguments

`dataset_1`, `dataset_2`
: (`character(1)`) Dataset names. When `dataset_2` is omitted, a primary key for `dataset_1` is created.

`keys`
: (optionally named `character`) Column mapping between the datasets, where `names(keys)` maps columns in `dataset_1` corresponding to columns of `dataset_2` given by the elements of `keys`.

- If unnamed, the same column names are used for both datasets.
- If any element of the `keys` vector is empty with a non-empty name, then the name is used for both datasets.

directed (logical(1)) Flag that indicates whether it should create a parent-child relationship between the datasets.

- TRUE (default) dataset_1 is the parent of dataset_2;
- FALSE when the relationship is undirected.

Value

object of class join_key_set to be passed into join_keys function.

See Also

join_keys(), parents()

Examples

```
join_key("d1", "d2", c("A"))
join_key("d1", "d2", c("A" = "B"))
join_key("d1", "d2", c("A" = "B", "C"))
```

---

join_keys *Manage relationships between datasets using* join_keys

---

Description

Facilitates the creation and retrieval of relationships between datasets. join_keys class extends list and contains keys connecting pairs of datasets. Each element of the list contains keys for specific dataset. Each dataset can have a relationship with itself (primary key) and with other datasets.

Note that join_keys list is symmetrical and assumes a default direction, that is: when keys are set between ds1 and ds2, it defines ds1 as the parent in a parent-child relationship and the mapping is automatically mirrored between ds2 and ds1.

Usage

```
## Constructor, getter and setter
join_keys(...)

## Default S3 method:
join_keys(...)

## S3 method for class 'join_keys'
join_keys(...)

## S3 method for class 'teal_data'
join_keys(...)
```

```
## S3 method for class 'join_keys'
x[i, j]

## S3 replacement method for class 'join_keys'
x[i, j, directed = TRUE] <- value

## S3 method for class 'join_keys'
c(...)

## S3 method for class 'join_key_set'
c(...)

join_keys(x) <- value

## S3 replacement method for class 'join_keys'
join_keys(x) <- value

## S3 replacement method for class 'teal_data'
join_keys(x) <- value

## S3 method for class 'join_keys'
format(x, ...)

## S3 method for class 'join_keys'
print(x, ...)
```

Arguments

`...`	optional, • either `teal_data` or `join_keys` object to extract `join_keys` • or any number of `join_key_set` objects to create `join_keys` • or nothing to create an empty `join_keys`
`x`	(`join_keys`) empty object to set the new relationship pairs. `x` is typically an object of `join_keys` class. When called with the `join_keys(x)` or `join_keys(x) <- value` then it can also take a supported class (`teal_data`, `join_keys`)
`i, j`	indices specifying elements to extract or replace. Index should be a a character vector, but it can also take numeric, logical, `NULL` or missing.
`directed`	(`logical(1)`) Flag that indicates whether it should create a parent-child relationship between the datasets. • `TRUE` (default) `dataset_1` is the parent of `dataset_2`; • `FALSE` when the relationship is undirected.
`value`	For `x[i, j, directed = TRUE)] <- value` (named/unnamed `character`) Column mapping between datasets. For `join_keys(x) <- value`: (`join_key_set` or list of `join_key_set`) relationship pairs to add to `join_keys` list. [i, j, directed = TRUE)]: R:i,%20j,%20directed%20=%20TRUE)

Value

`join_keys` object.

Methods (by class)

- `join_keys()`: Returns an empty `join_keys` object when called without arguments.
- `join_keys(join_keys)`: Returns itself.
- `join_keys(teal_data)`: Returns the `join_keys` object contained in `teal_data` object.
- `join_keys(...)`: Creates a new object with one or more `join_key_set` parameters.

Functions

- `x[datanames]`: Returns a subset of the `join_keys` object for given `datanames`, including parent `datanames` and symmetric mirror keys between `datanames` in the result.
- `x[i, j]`: Returns join keys between datasets `i` and `j`, including implicit keys inferred from their relationship with a parent.
- `x[i, j] <- value`: Assignment of a key to pair `(i, j)`.
- `x[i] <- value`: This (without `j` parameter) **is not** a supported operation for `join_keys`.
- `join_keys(x)[i, j] <- value`: Assignment to `join_keys` object stored in `x`, such as a `teal_data` object or `join_keys` object itself.
- `join_keys(x) <- value`: Assignment of the `join_keys` in object with `value`. `value` needs to be an object of class `join_keys` or `join_key_set`.

See Also

`join_key()` for creating `join_keys_set`, `parents()` for parent operations, `teal_data()` for `teal_data` constructor *and* default_cdisc_join_keys for default CDISC keys.

Examples

```
# Creating a new join keys ----

jk <- join_keys(
  join_key("ds1", "ds1", "pk1"),
  join_key("ds2", "ds2", "pk2"),
  join_key("ds3", "ds3", "pk3"),
  join_key("ds1", "ds2", c(pk1 = "pk2")),
  join_key("ds1", "ds3", c(pk1 = "pk3"))
)

jk

# Getter for join_keys ---

jk["ds1", "ds2"]

# Subsetting join_keys ----
```

```
jk["ds1"]
jk[1:2]
jk[c("ds1", "ds2")]

# Setting a new primary key ---

jk["ds4", "ds4"] <- "pk4"
jk["ds5", "ds5"] <- "pk5"

# Setting a single relationship pair ---

jk["ds1", "ds4"] <- c("pk1" = "pk4")

# Removing a key ---

jk["ds5", "ds5"] <- NULL
# Merging multiple `join_keys` objects ---

jk_merged <- c(
  jk,
  join_keys(
    join_key("ds4", keys = c("pk4", "pk4_2")),
    join_key("ds3", "ds4", c(pk3 = "pk4_2"))
  )
)
# note: merge can be performed with both join_keys and join_key_set

jk_merged <- c(
  jk_merged,
  join_key("ds5", keys = "pk5"),
  join_key("ds1", "ds5", c(pk1 = "pk5"))
)
# Assigning keys via join_keys(x)[i, j] <- value ----

obj <- join_keys()
# or
obj <- teal_data()

join_keys(obj)["ds1", "ds1"] <- "pk1"
join_keys(obj)["ds2", "ds2"] <- "pk2"
join_keys(obj)["ds3", "ds3"] <- "pk3"
join_keys(obj)["ds1", "ds2"] <- c(pk1 = "pk2")
join_keys(obj)["ds1", "ds3"] <- c(pk1 = "pk3")

identical(jk, join_keys(obj))
# Setter for join_keys within teal_data ----

td <- teal_data()
join_keys(td) <- jk

join_keys(td)["ds1", "ds2"] <- "new_key"
join_keys(td) <- c(join_keys(td), join_keys(join_key("ds3", "ds2", "key3")))
```

```
join_keys(td)
```

`names<-.join_keys` *The names of a* `join_keys` *object*

Description

The names of a `join_keys` object

Usage

```
## S3 replacement method for class 'join_keys'
names(x) <- value
```

Arguments

`x`	an R object.
`value`	a character vector of up to the same length as `x`, or `NULL`.

`parents` *Get and set parents in* `join_keys` *object*

Description

`parents()` facilitates the creation of dependencies between datasets by assigning a parent-child relationship.

Usage

```
parents(x)

## S3 method for class 'join_keys'
parents(x)

## S3 method for class 'teal_data'
parents(x)

parents(x) <- value

## S3 replacement method for class 'join_keys'
parents(x) <- value

## S3 replacement method for class 'teal_data'
parents(x) <- value

parent(x, dataset_name)
```

Arguments

x	(join_keys or teal_data) object that contains "parents" information to retrieve or manipulate.
value	(named list) of character vectors.
dataset_name	(character(1)) Name of dataset to query on their parent.

Details

Each element is defined by a list element, where list("child" = "parent").

Value

a list of character representing the parents.

For parent(x, dataset_name) returns NULL if parent does not exist.

Methods (by class)

- parents(join_keys): Retrieves parents of join_keys object.
- parents(teal_data): Retrieves parents of join_keys inside teal_data object.

Functions

- parents(x) <- value: Assignment of parents in join_keys object.
- parents(join_keys) <- value: Assignment of parents of join_keys object.
- parents(teal_data) <- value: Assignment of parents of join_keys inside teal_data object.
- parent(): Getter for individual parent.

See Also

join_keys()

Examples

```
# Get parents of join_keys ---

jk <- default_cdisc_join_keys["ADEX"]
parents(jk)
# Get parents of join_keys inside teal_data object ---

td <- teal_data(
  ADSL = rADSL,
  ADTTE = rADTTE,
  ADRS = rADRS,
  join_keys = default_cdisc_join_keys[c("ADSL", "ADTTE", "ADRS")]
)
parents(td)
# Assignment of parents ---
```

```
jk <- join_keys(
  join_key("ds1", "ds2", "id"),
  join_key("ds5", "ds6", "id"),
  join_key("ds7", "ds6", "id")
)

parents(jk) <- list(ds2 = "ds1")

# Setting individual parent-child relationship

parents(jk)["ds6"] <- "ds5"
parents(jk)["ds7"] <- "ds6"
# Assignment of parents of join_keys inside teal_data object ---

parents(td) <- list("ADTTE" = "ADSL") # replace existing
parents(td)["ADRS"] <- "ADSL" # add new parent
# Get individual parent ---

parent(jk, "ds2")
parent(td, "ADTTE")
```

---

show,teal_data-method *Show* teal_data *object*

Description

Prints teal_data object.

Usage

```
## S4 method for signature 'teal_data'
show(object)
```

Arguments

object	(teal_data)

Value

Input teal_data object.

Examples

```
teal_data()
teal_data(x = iris, code = "x = iris")
verify(teal_data(x = iris, code = "x = iris"))
```

`TealData` *Deprecated* `TealData` *class and related functions*

Description

[Deprecated]

The `TealData` class and associated functions have been deprecated. Use `teal_data()` instead. See the Migration guide for details.

Usage

```
as_cdisc(...)

callable_code(...)

callable_function(...)

code_dataset_connector(...)

code_cdisc_dataset_connector(...)

csv_dataset_connector(...)

csv_cdisc_dataset_connector(...)

python_code(...)

python_dataset_connector(...)

python_cdisc_dataset_connector(...)

cdisc_data_connector(...)

cdisc_dataset(...)

cdisc_dataset_connector(...)

cdisc_dataset_connector_file(...)

cdisc_dataset_file(...)

dataset(...)

dataset_connector(...)

dataset_connector_file(...)
```

```
dataset_file(...)

data_connection(...)

fun_dataset_connector(...)

fun_cdisc_dataset_connector(...)

relational_data_connector(...)

mae_dataset(...)

get_attrs(...)

get_dataset_label(...)

get_dataset(...)

get_datasets(...)

get_dataname(...)

get_key_duplicates(...)

get_keys(...)

get_raw_data(...)

is_pulled(...)

load_dataset(...)

load_datasets(...)

mutate_data(...)

mutate_dataset(...)

set_args(...)

rds_dataset_connector(...)

rds_cdisc_dataset_connector(...)

script_dataset_connector(...)

script_cdisc_dataset_connector(...)
```

```
set_keys(...)

read_script(...)

to_relational_data(...)

validate_metadata(...)

get_cdisc_keys(...)

cdisc_data_file(...)

teal_data_file(...)

get_join_keys(...)

get_join_keys(...) <- value
```

Arguments

`...`	any argument supported in `TealData` related functions.
`value`	value to assign

Value

nothing

See Also

`cdisc_data()` , `join_keys()`

`teal_data` *Comprehensive data integration function for* `teal` *applications*

Description

[Stable]

Universal function to pass data to teal application.

Usage

```
teal_data(..., join_keys = teal.data::join_keys(), code = character(0), check)
```

Arguments

...	any number of objects (presumably data objects) provided as name = value pairs.
join_keys	(join_keys or single join_key_set) optional object with datasets column names used for joining. If empty then no joins between pairs of objects.
code	(character, language) optional code to reproduce the datasets provided in ... . Note this code is not executed and the teal_data may not be reproducible
check	(logical) **[Deprecated]** Use verify() to verify code reproducibility .

Value

A teal_data object.

Examples

```
teal_data(x1 = iris, x2 = mtcars)
```

---

verify *Verify code reproducibility*

Description

Checks whether code in teal_data object reproduces the stored objects.

Usage

```
verify(x)
```

Arguments

x	teal_data object

Details

If objects created by code in the @code slot of x are all_equal to the contents of the @env slot, the function updates the @verified slot to TRUE in the returned teal_data object. Once verified, the slot will always be set to TRUE. If the @code fails to recreate objects in teal_data@env, an error is raised.

Value

Input teal_data object or error.

Examples

```
tdata1 <- teal_data()
tdata1 <- within(tdata1, {
  a <- 1
  b <- a^5
  c <- list(x = 2)
})
verify(tdata1)

tdata2 <- teal_data(x1 = iris, code = "x1 <- iris")
verify(tdata2)
verify(tdata2)@verified
tdata2@verified

tdata3 <- teal_data()
tdata3 <- within(tdata3, {
  stop("error")
})
try(verify(tdata3)) # fails


a <- 1
b <- a + 2
c <- list(x = 2)
d <- 5
tdata4 <- teal_data(
  a = a, b = b, c = c, d = d,
  code = "a <- 1
          b <- a
          c <- list(x = 2)
          e <- 1"
)
tdata4
## Not run:
verify(tdata4) # fails

## End(Not run)
```

Index

∗ **datasets**
default_cdisc_join_keys, 6
[.join_keys *(join_keys)*, 10
[<-.join_keys *(join_keys)*, 10

as_cdisc *(TealData)*, 17

c.join_key_set *(join_keys)*, 10
c.join_keys *(join_keys)*, 10
callable_code *(TealData)*, 17
callable_function *(TealData)*, 17
cdisc_data, 2
cdisc_data(), *19*
cdisc_data_connector *(TealData)*, 17
cdisc_data_file *(TealData)*, 17
cdisc_dataset *(TealData)*, 17
cdisc_dataset_connector *(TealData)*, 17
cdisc_dataset_connector_file *(TealData)*, 17
cdisc_dataset_file *(TealData)*, 17
code_cdisc_dataset_connector *(TealData)*, 17
code_dataset_connector *(TealData)*, 17
col_labels, 4
col_labels<- *(col_labels)*, 4
col_relabel *(col_labels)*, 4
csv_cdisc_dataset_connector *(TealData)*, 17
csv_dataset_connector *(TealData)*, 17

data_connection *(TealData)*, 17
datanames, 5
datanames,qenv.error-method *(datanames)*, 5
datanames,teal_data-method *(datanames)*, 5
datanames<- *(datanames)*, 5
datanames<-,qenv.error,character-method *(datanames)*, 5
datanames<-,teal_data,character-method *(datanames)*, 5
dataset *(TealData)*, 17
dataset_connector *(TealData)*, 17
dataset_connector_file *(TealData)*, 17
dataset_file *(TealData)*, 17
default_cdisc_join_keys, 6, *12*

example_cdisc_data, 6

format.join_keys *(join_keys)*, 10
fun_cdisc_dataset_connector *(TealData)*, 17
fun_dataset_connector *(TealData)*, 17

get_attrs *(TealData)*, 17
get_cdisc_keys *(TealData)*, 17
get_code,teal_data-method, 7
get_dataname *(TealData)*, 17
get_dataset *(TealData)*, 17
get_dataset_label *(TealData)*, 17
get_datasets *(TealData)*, 17
get_join_keys *(TealData)*, 17
get_join_keys<- *(TealData)*, 17
get_key_duplicates *(TealData)*, 17
get_keys *(TealData)*, 17
get_labels *(col_labels)*, 4
get_raw_data *(TealData)*, 17

is_pulled *(TealData)*, 17

join_key, 9
join_key(), *12*
join_keys, 10
join_keys(), *10*, *15*, *19*
join_keys<- *(join_keys)*, 10

load_dataset *(TealData)*, 17
load_datasets *(TealData)*, 17

mae_dataset *(TealData)*, 17

mutate_data *(*TealData*)*, 17
mutate_dataset *(*TealData*)*, 17

names<-.join_keys, 14

parent *(*parents*)*, 14
parents, 14
parents(), *10*, *12*
parents<- *(*parents*)*, 14
print.join_keys *(*join_keys*)*, 10
python_cdisc_dataset_connector
*(*TealData*)*, 17
python_code *(*TealData*)*, 17
python_dataset_connector *(*TealData*)*, 17

rds_cdisc_dataset_connector *(*TealData*)*,
17
rds_dataset_connector *(*TealData*)*, 17
read_script *(*TealData*)*, 17
relational_data_connector *(*TealData*)*, 17

script_cdisc_dataset_connector
*(*TealData*)*, 17
script_dataset_connector *(*TealData*)*, 17
set_args *(*TealData*)*, 17
set_keys *(*TealData*)*, 17
show,teal_data-method, 16

teal_data, 19
teal_data(), *2*, *12*, *17*
teal_data_file *(*TealData*)*, 17
TealData, 17
to_relational_data *(*TealData*)*, 17

validate_metadata *(*TealData*)*, 17
verify, 20
verify(), *3*, *7*, *20*
verify,qenv.error-method *(*verify*)*, 20
verify,teal_data-method *(*verify*)*, 20