Package 'integrator'

January 8, 2015

Type Package

Title Data Integrator

Version 0.1

Date 2012-05-29

Author Krzysztof Sakrejda

Maintainer Krzysztof Sakrejda <krzysztof.sakrejda@gmail.com>

Imports DBI

Depends RPostgreSQL, methods

Description Readable scripts for importing data depend on defining clear tasks. This package provides functions for common clear tasks.

License GPL

R topics documented:

`cmp_function_dependence` *A comparator ('<=' equivalent) for quicksort.*

Description

Compares functions based on dependencies!

Usage

```
cmp_function_dependence(f, g)
```

Arguments

`f`	A function.
`g`	Another function.

Value

Returns TRUE if g does not depend on f.

`db_connector-class` *A reference class which always yields a valid connection with x$conn.*

Description

A reference class which always yields a valid connection with x$conn.

Fields

`driver` A driver from RPostgreSQL.

`credentials` A filename for connection info. to the database (including host and port).

`conn` The always-valid connection.

Methods

`add_connection()` Adds a connection to the pool if necessary.

`check_crd(crd = NULL)` Checks whether credentials are available and complete.

`clean_connections()` Cleans out available connections.

`clear_connections()` Runs a simple command to clear pending results on available connections.

`connection_in_use(connection = NULL)` Tests whether a connection is in use.

`get_connection()` Yields an unused connection from the pool.

`initialize(credentials)` Checks for RPostgreSQL, uses credentials to generate a connection and test it.

dependency_resolver *Dependency resolver for function lists.*

Description

Returns the function with independent functions coming first.

Usage

```
dependency_resolver(f_list)
```

Arguments

f_list	A list of functions to be sorted.

See Also

cmp_function_dependence, quicksort.list

Examples

```
test_list <- list(
g = function(f, q) { f+q },
f = function(x) { x^2 },
h = function() { 10 }
)

sorted_test_list <- dependency_resolver(test_list)
```

eval_pipe *Runs expressions from a list in an independently scoped environment.*

Description

Runs expressions from a list in an independently scoped environment.

Usage

```
eval_pipe(data = NULL, pipeline = list())
```

Arguments

data	A list of environments (or data frames, or a data frame which is promoted to an environment. This creates a scoped context for evaluation expressions. This scope *skips* .GlobalEnv.
pipeline	A quoted list of expressions which are evaluted in scoped context provided by the data.

Value

An internally generated list of expression results.

Examples

```
data <- replicate(8,new.env())
data[[2]]$z <- pi
pipe=quote(list({a <- 3+3}, {b <- a*2}, {q <-a*b*z} ))
o <- eval_pipe(data,pipe)
```

`map` *Maps names in a list (or data frame) to new names based on a mapping from a list (map).*

Description

Maps names in a list (or data frame) to new names based on a mapping from a list (map).

Usage

```
map(x, map, one_to_one = TRUE)
```

Arguments

`x`	A list (or data frame) with names to be mapped.
`map`	A named list of character vectors. In re-mapping list names are the new target columns and source columns are named in the vectors.
`one_to_one`	A flag which (if TRUE) triggers a warning when multiple columns might be mapped (only the first is).

Value

The original list (or data frame) modified.

Examples

```
n <- 10
widgets <- data.frame(id=1:n, weight_dry=exp(rnorm(n)),
                      weight_wet=exp(rnorm(n))+10,
                      vol=exp(rnorm(n)))
widgets <- map(x=widgets, map=list(volume="vol"))
```

`map_unknowns` *Maps special "NA" values to NA's in a vector.*

Description

Maps special "NA" values to NA's in a vector.

Usage

```
map_unknowns(x, map)
```

Arguments

x	A vector with special values.
map	A vector of special values to be mapped to NA.

Value

x, with special values turned to NA.

---

map_values *Maps vectorvalues one-to-one.*

Description

Maps vectorvalues one-to-one.

Usage

```
map_values(x, input, output)
```

Arguments

x	A vector to operate on.
input	A vector of values to be modified.
output	A vector of values to replace input values.
x,	with modifications.

---

pipeline *Applies a list of functions to a data frame in order to transform columns.*

Description

Applies a list of functions to a data frame in order to transform columns.

Usage

```
pipeline(data = NULL, pipeline = NULL, envir = parent.frame(),
  final_names = NULL, multipath = FALSE)
```

Arguments

data	A data frame which provides data and where resulting columns are stored
pipeline	A list of functions. Columns with names taken from the list names are added to the data frame. These columns are generated by checking for pipeline function arguments first in the data frame and then in the environments. Once inputs are found, they are used to run the function. Inputs found in the data.frame are always the same lenght, but inputs found in 'envir' need not be.
envir	An environment which is searched for function arguments.

Details

The function is generally pretty flexible—when multipath is set, pipeline functions with missing arguments are simply ignored. When a pipeline element is a list of functions rather than a single function, the first applicable function is used and the rest are skipped, continuing with the first function of the top-level list. This is a good way to code preferential data sources.

`quicksort` *Sorting with an arbitrarily defined comparator ('<=' by default).*

Description

A quicksort implementation. It's generic so with the right comparator it will do dependency sorting on function lists...

Usage

```
quicksort(x, cmp = `<=`)
```

Arguments

`x`	The thing to be sorted.
`cmp`	The comparator, '<=' or similar.

Details

Naive, after reading a few web pages about how to do it... I just need to sort a short list with a given comparator... Based on:

http://algs4.cs.princeton.edu/23quicksort/ http://en.wikipedia.org/wiki/Quicksort http://rosettacode.org/wiki/Sorting_alg
Thanks internet, that Pascal intro course was a long time ago...

Value

x, but sorted according to cmp.

Examples

```
o <- quicksort(rbinom(n=30, size=15, prob=0.8))
```

`quicksort.list` *Sorting with an arbitrarily defined comparator ('<=' by default).*

Description

A quicksort implementation. It's generic so with the right comparator it will do dependency sorting on function lists...

Usage

```
quicksort.list(x, cmp = `<=`)
```

Arguments

`x`	The thing to be sorted.
`cmp`	The comparator, '<=' or similar.

Details

Naive, after reading a few web pages about how to do it... I just need to sort a short list with a given comparator... Based on:

http://algs4.cs.princeton.edu/23quicksort/ http://en.wikipedia.org/wiki/Quicksort http://rosettacode.org/wiki/Sorting_alg
Thanks internet, that Pascal intro course was a long time ago...

Value

x, but sorted according to cmp.

Examples

```
o <- quicksort.list(as.list(rbinom(n=30, size=15, prob=0.8)))
```

`safe_read_csv` *Read a .csv file with minimal modifications as allowed by R, then modify according to locally sourced files.*

Description

Read a .csv file with minimal modifications as allowed by R, then modify according to locally sourced files.

Usage

```
safe_read_csv(file, instructions = NULL, drop_uppercase = FALSE)
```

Arguments

file	A .csv type file.
instructions	A .R file (source-able) which defines column name mapping ("column_map"), NA mapping ("unknowns"), as well as value mapping ("value_map", with "input", and "output" vectors.
drop_uppercase	Turns column names to all lower-case.

Value

A data.frame with the .csv file loaded and modified as specified.

---

standard_string_transformations

Standard strings transformations.

Description

Standard strings transformations.

Usage

```
standard_string_transformations
```

Format

```
List of 4
 $ to_lower_case           :function (x)
 $ drop_leading_whitespace :function (x)
  ..- attr(*, "srcref")=Class 'srcref'  atomic [1:8] 5 28 5 94 35 101 5 5
 .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x2426380>
 $ drop_trailing_whitespace:function (x)
  ..- attr(*, "srcref")=Class 'srcref'  atomic [1:8] 6 29 6 95 36 102 6 6
 .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x2426380>
 $ whitespace_to_underscore:function (x)
  ..- attr(*, "srcref")=Class 'srcref'  atomic [1:8] 7 29 7 93 36 100 7 7
  .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x2426380>
```

---

string_masks

Takes strings, check if they match patterns, and construct expressions according to a string.

Description

Takes strings, check if they match patterns, and construct expressions according to a string.

Usage

```
string_masks(x, patterns, expr = NULL, all_strings = TRUE)
```

Arguments

x	A vector of strings which masks will be created for.
patterns	A list of patterns (for grepl) which the strings (might) match.
expr	A quoted list of logical expressions operating on pattern-derived masks which will further combine the masks

Value

A list (of length `length(patterns)+length(expr)`) of vectors (each of length `length(x)`).

Examples

```
n <- 10
widgets <- data.frame(id=1:n, weight_dry=exp(rnorm(n)),
                      weight_wet=exp(rnorm(n))+10, volume=exp(rnorm(n)))
patterns <- list(weights="^weight")
expr <- quote(list(measurement = weights | volume))
masks <- string_masks(x=names(widgets), patterns=patterns, expr=expr)
```

`string_pipeline` *Runs a series of string-transforming functions.*

Description

Simpler than the generic pipeline transformation.

Usage

```
string_pipeline(string = NULL, pipeline = NULL, string_args = rep("x",
  length(pipeline)), envir = parent.frame())
```

Arguments

string	A string to operate on.
pipeline	A list of functions to run on the strings.
string_args	A vector of names of function arguments which take the string for processing. Typically just 'x', but sometimes 'text', lets you use functions directly without rewriting thin wrappers.
envir	An environment where extra arguments can be found.

Value

A string, transformed. Better, faster, stronger!

Index