

# Package: `datastepr` (via `r-universe`)

September 14, 2024

**Type** Package

**Title** An Implementation of a SAS-Style Data Step

**Version** 0.0.2

**Author** Brandon Taylor

**Maintainer** Brandon Taylor <[brandon.taylor221@gmail.com](mailto:brandon.taylor221@gmail.com)>

**Description** Based on a SAS data step. This allows for row-wise dynamic building of data, iteratively importing slices of existing dataframes, conducting analyses, and exporting to a results frame. This is particularly useful for differential or time-series analyses, which are often not well suited to vector-based operations.

**Depends** R (>= 3.1.3)

**Imports** dplyr (>= 0.5.0), lazyeval (>= 0.1.10), R6 (>= 2.0.1), magrittr (>= 1.5), tibble (>= 1.1)

**License** CC0

**Suggests** knitr, covr, rmarkdown, testthat

**VignetteBuilder** knitr

**NeedsCompilation** no

**RoxygenNote** 5.0.1

**BugReports** <https://github.com/bramtayl/datastepr/issues>

**URL** <https://github.com/bramtayl/datastepr>

**Date/Publication** 2016-08-20 10:31:35

**Repository** <https://bramtayl.r-universe.dev>

**RemoteUrl** <https://github.com/cran/datastepr>

**RemoteRef** HEAD

**RemoteSha** aeaeb86a19aa3a1680ff1bf307a08933bc485ae5

## Contents

dataStepClass . . . . .	2
toDf . . . . .	3
toEnv . . . . .	4

<b>Index</b>	<b>5</b>
--------------	----------

---

dataStepClass	<i>An implementation of a SAS datastep in a class</i>
---------------	---

---

### Description

An implementation of a SAS datastep in a class

### Usage

dataStepClass

### Format

An [R6Class](#) generator object

### Fields

`i` `i` begins at 0 and is incremented for each iteration of the data step.

`results` The results frame is initialized as an empty data frame. It is populated row-wise with each iteration of the data step.

`continue` `continue` is a marker which signals that the step should continue repeating. When `continue` is 1, repetition will continue, and when `continue` is 0, repetition will cease. It is initialized to 0.

`eval` `eval` is initialized as NULL, but will store a pointer to the current evaluation environment. This pointer helps pass the evaluation environment from one iteration of the data step to the next.

### Methods

`begin(env)` `begin` does three things: imports the environment of the previous step to the current, stores the current environment (or the environment specified), and increments `i` by 1. It takes one argument, `envir`, which should typically be set to `environment()`.

`set(dataframe, group_id)` `set` takes two arguments: a data frame and an optional unquoted `group_id` variable. This `group_id` variable must contain a consecutive sequence of natural numbers from 1 to some maximum. In each data step, rows where `i` matches the `group_id` variable (or simply the `i`th row if no `group_id` variable is given) are selected, and the slice is split into vectors and imported into the evaluation environment. `continue` is set to 0 once `set` reaches the maximum value in the `group_id` column, ceasing repetition of the datastep, else `continue` is set to 1.

`set_(dataframe, group_id)` A standard evaluation version of `set_`, in which the `group_id` variable is included as a string, formula, or lazy object.

`output` `output` takes an optional list argument. Either the list, or, if none is given, all vectors in the evaluation environment are gathered into a `data.frame`, and this `data.frame` appended to `results`.

`end` `end` will, if `continue` is 1, evaluate the function given within the evaluation environment. Typically, the function given will be the current function: that is, steps are joined recursively.

### Examples

```
step = dataStepClass$new()

frame = data.frame(x = 1:10)

stairs = function() {
  step$begin(environment())
  step$set(frame)
  y = x + 1
  step$output()
  step$end(stairs)
}

stairs()

step$results
```

---

toDf

*Append an object to a dataframe.*


---

### Description

Convert an object to a list, select only vector entries, coerce to a `data.frame`, and append to the given data frame.

### Usage

```
toDf(object, dataframe)
```

### Arguments

<code>object</code>	An object which can be coerced to a list (e.g. an environment)
<code>dataframe</code>	A data frame

### Value

An appended dataframe

**Examples**

```
toDf(list(a = 1, b = 2, data.frame()), data.frame())  
toDf(environment(), data.frame())
```

---

toEnv

*Append an object to an environment*

---

**Description**

A function to coerce an object to a list and append the list to an environment

**Usage**

```
toEnv(object, environment)
```

**Arguments**

object	An object which can be coerced to a list (e.g. an environment)
environment	An environment

**Value**

An appended environment

**Examples**

```
toEnv(data.frame(a = 1, b = 2), environment())  
toEnv(list(a = 1, b = 2), environment())  
toEnv(environment(), new.env())
```

# Index

\* **datasets**

dataStepClass, [2](#)

dataStepClass, [2](#)

R6Class, [2](#)

toDf, [3](#)

toEnv, [4](#)