

Package ‘getDTeval’

May 8, 2026

Title Translating Coding Statements using get() and eval() for
Improved Run-Time Coding Efficiency

Version 0.0.2

Depends R (>= 3.6.0)

Description The getDTeval() function facilitates the translation of the original coding statement to an optimized form for improved runtime efficiency without compromising on the programmatic coding design.
The function can either provide a translation of the coding statement, directly evaluate the translation to return a coding result, or provide both of these outputs.

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, dplyr, testthat (>= 2.1.0), covr, devtools

Imports stats, data.table, formulaic, microbenchmark, utils

VignetteBuilder knitr

NeedsCompilation no

Author David Shilane [aut],
Mayur Bansal [ctb, cre],
Anderson Nelson [ctb],
Caffrey Lee [ctb],
Zichen Huang [ctb]

Maintainer Mayur Bansal <mb4511@columbia.edu>

Repository CRAN

Date/Publication 2021-06-21 06:10:04 UTC

Contents

benchmark.getDTeval	2
function.ending.point	3
getDTeval	3
translate.fn.calls	4

Index	6
--------------	----------

benchmark.getDTeval *benchmark.getDTeval*

Description

Performs a benchmarking experiment for data.table coding statements that use get() or eval() for programmatic designs. The a) original statement is compared to b) passing the original statement through getDTeval and also to c) an optimized coding statement. The results can demonstrate the overall improvement of using the coding translations offered by getDTeval()

Usage

```
benchmark.getDTeval(  
  the.statement,  
  times = 30,  
  seed = 47,  
  envir = .GlobalEnv,  
  ...  
)
```

Arguments

the.statement	refers to the original coding statement which needs to be translated to an optimized form. This value may be entered as either a character value or as an expression.
times	The number of iterations to run the benchmarking experiment
seed	an integer value specifying the seed of the pseudorandom number generator.
envir	The environment in which the calculation takes place, with the global environment .GlobalEnv set as the default.
...	provision for additional arguments

Examples

```
#Benchmarking runtime performances in calculating mean age  
dat<-formulaic::snack.dat  
age.name<-'Age'  
benchmark.getDTeval(the.statement = "dat[,.(mean_age=mean(get(age.name)))]", times = 5, seed = 10)
```

function.ending.point *function.ending.point*

Description

An Internal function to return the ending index

Usage

```
function.ending.point(all.chars, beginning.index, ...)
```

Arguments

all.chars	all the characters of the statement
beginning.index	specifies the index of the first character
...	provision for additional arguments

getDTeval *getDTeval*

Description

The getDTeval() function facilitates the translation of the original coding statement to an optimized form for improved runtime efficiency without compromising on the programmatic coding design. The function can either provide a translation of the coding statement, directly evaluate the translation to return a coding result, or provide both of these outputs

Usage

```
getDTeval(  
  the.statement,  
  return.as = "result",  
  coding.statements.as = "character",  
  eval.type = "optimized",  
  envir = .GlobalEnv,  
  ...  
)
```

Arguments

<code>the.statement</code>	refers to the original coding statement which needs to be translated to an optimized form. This value may be entered as either a character value or as an expression.
<code>return.as</code>	refers to the mode of output. It could return the results as a coding statement (<code>return.as = "code"</code>), an evaluated coding result (<code>return.as = "result"</code> , which is the default value), or a combination of both (<code>return.as = "all"</code>).
<code>coding.statements.as</code>	determines whether the coding statements provided as outputs are returned as expression objects (<code>return.as = "expression"</code>) or as character values (<code>return.as = "character"</code> , which is the default).
<code>eval.type</code>	a character value stating whether the coding statement should be evaluated in its current form (<code>eval.type = "as.is"</code>) or have its called to <code>get()</code> and <code>eval()</code> translated (<code>eval.type = "optimized"</code> , the default setting).
<code>envir</code>	Specify the environment for the required function. <code>.GlobalEnv</code> is set as default
<code>...</code>	provision for additional arguments

Examples

```
# Using getDTeval to calculate mean age
dat<-formulaic::snack.dat
age.name<-'Age'
getDTeval(the.statement = 'dat[,.(mean_age=mean(get(age.name)))]',return.as = 'result')
```

translate.fn.calls *translate.fn.calls*

Description

Internal Function that translates programmatic designs into optimized coding statements for faster calculations

Usage

```
translate.fn.calls(
  the.statement,
  function.name = "get(",
  envir = .GlobalEnv,
  ...
)
```

Arguments

<code>the.statement</code>	original coding statement to perform the required calculation. Must be provided as a character value.
<code>function.name</code>	Name of the function to be translated to an optimized form. Parameter values should be either 'get(' or 'eval('. 'get(' is set as default
<code>envir</code>	Specify the environment for the required function. .GlobalEnv is set as default
<code>...</code>	provision for additional arguments

Index

`benchmark.getDTeval`, [2](#)
`function.ending.point`, [3](#)
`getDTeval`, [3](#)
`translate.fn.calls`, [4](#)