Package 'globals'

July 22, 2025

Version 0.18.0

Depends R (>= 3.1.2)

Imports codetools

Title Identify Global Objects in R Expressions

Description Identifies global (``unknown'' or ``free'') objects in R expressions by code inspection using various strategies (ordered, liberal, conservative, or deep-first search). The objective of this package is to make it as simple as possible to identify global objects for the purpose of exporting them in parallel, distributed compute environments.

License LGPL (>= 2.1)

LazyLoad TRUE

ByteCompile TRUE

Language en-US

Encoding UTF-8

URL https://globals.futureverse.org, https://github.com/futureverse/globals

BugReports https://github.com/futureverse/globals/issues

RoxygenNote 7.3.2

NeedsCompilation no

Author Henrik Bengtsson [aut, cre, cph], Davis Vaughan [ctb]

Maintainer Henrik Bengtsson <henrikb@braju.com>

Repository CRAN

Date/Publication 2025-05-08 09:00:03 UTC

Contents

`cleanup.Globals` *Drop certain types of globals*

Description

Drop certain types of globals

Usage

```
## S3 method for class 'Globals'
cleanup(globals, drop = c("missing", "base-packages", "nativesymbolinfo"), ...)
```

Arguments

`globals`	A Globals object.
`drop`	A character vector specifying what type of globals to drop.
`...`	Not used

`findGlobals` *Get all global objects of an expression*

Description

Get all global objects of an expression

Usage

```
findGlobals(
  expr,
  envir = parent.frame(),
  ...,
  attributes = TRUE,
  tweak = NULL,
  dotdotdot = c("warning", "error", "return", "ignore"),
  method = c("ordered", "conservative", "liberal", "dfs"),
  substitute = FALSE,
  unlist = TRUE,
  trace = FALSE
)
```

```
globalsOf(
  expr,
  envir = parent.frame(),
  ...,
  method = c("ordered", "conservative", "liberal", "dfs"),
  tweak = NULL,
  locals = NA,
  substitute = FALSE,
  mustExist = TRUE,
  unlist = TRUE,
  recursive = TRUE,
  skip = NULL
)
```

Arguments

expr	An R expression.
envir	The environment from where to search for globals.
attributes	If TRUE (default), attributes of expr are also searched. If FALSE, they are not. If a character vector, then attributes with matching names are searched. Note, the attributes of the attributes elements are not searched, that is, attributes are not searched recursively. Also, attributes are searched with ‘dotdotdot = "ignore".
tweak	An optional function that takes an expression and returns a tweaked expression.
dotdotdot	TBD.
method	A character string specifying what type of search algorithm to use.
substitute	If TRUE, the expression is substitute():ed, otherwise not.
unlist	If TRUE, a list of unique objects is returned. If FALSE, a list of length(expr) sublists.
trace	TBD.
locals	Should globals part of any "local" environment of a function be included or not?
mustExist	If TRUE, an error is thrown if the object of the identified global cannot be located. Otherwise, the global is not returned.
recursive	If TRUE, globals that are closures (functions) and that exist outside of namespaces ("packages"), will be recursively scanned for globals.
skip	(internal) A list of globals not to be searched for additional globals. Ignored unless recursive is TRUE.
...	Not used.

Details

There currently three strategies for identifying global objects.

The method = "ordered" search method identifies globals such that a global variable preceding a local variable with the same name is not dropped (which the "conservative" method would).

The method = "conservative" search method tries to keep the number of false positive to a minimum, i.e. the identified objects are most likely true global objects. At the same time, there is a risk

that some true globals are not identified (see example). This search method returns the exact same result as the `findGlobals()` function of the **codetools** package.

The `method = "liberal"` search method tries to keep the true-positive ratio as high as possible, i.e. the true globals are most likely among the identified ones. At the same time, there is a risk that some false positives are also identified.

The `method = "dfs"` search method identifies globals in the abstract syntax tree (AST) using a depth-first search, which better emulates how the R engine identifies global variables.

With `recursive = TRUE`, globals part of locally defined functions will also be found, otherwise not.

Value

`findGlobals()` returns a character vector.

`globalsOf()` returns a Globals object.

See Also

Internally, the **codetools** package is utilized for code inspections.

Examples

```
b <- 2
expr <- substitute({ a <- b; b <- 1 })

## Will _not_ identify 'b' (because it's also a local)
globalsC <- globalsOf(expr, method = "conservative")
print(globalsC)

## Will identify 'b'
globalsL <- globalsOf(expr, method = "liberal")
print(globalsL)
```

Globals *A representation of a set of globals*

Description

A representation of a set of globals

Usage

```
Globals(object, ...)
```

Arguments

`object`	A named list.
`...`	Not used.

Value

An object of class `Globals`, which is a *named* list of the value of the globals, where the element names are the names of the globals. Attribute `where` is a named list of the same length and with the same names.

See Also

The `globalsOf()` function identifies globals from an R expression and returns a Globals object.

`globalsByName` *Locates and retrieves a set of global variables by their names*

Description

Locates and retrieves a set of global variables by their names

Usage

```
globalsByName(names, envir = parent.frame(), mustExist = TRUE, ...)
```

Arguments

`names`	A character vector of global variable names.
`envir`	The environment from where to search for globals.
`mustExist`	If TRUE, an error is thrown if the object of the identified global cannot be located. Otherwise, the global is not returned.
`...`	Not used.

Value

A Globals object of named elements and an attribute `where` with named elements. Both of sets have names according to `names`.

Special "argument" globals

If `names` specifies `"..."`, `"..1"`, `"..2"`, ..., then they are interpreted as arguments `...`, `..1`, `..2`, ..., respectively. If specified, then the corresponding elements in the results are lists of class `DotDotDotList` comprising the value of the latter. If the special argument does not exist, then the value is `NA`, and the corresponding `where` attributes is `NULL`.

Examples

```
f <- function(x = 42, ...) {
  globalsByName("x")
}

globals <- f()
str(globals)

globals <- f(3.14)
str(globals)


g <- function(x = 42, ...) {
  globalsByName("...")
}

globals <- g()
str(globals)

globals <- g(3.14)
str(globals)

globals <- g(3.14, 1L, b = 2L, c = 3L)
str(globals)


h <- function(x = 42, ...) {
  globalsByName("..2")
}

globals <- h(x = 3.14, a = 1, b = 2)
str(globals)

globals <- g(3.14)
str(globals)

globals <- g(3.14, 1L, b = 2L, c = 3L)
str(globals)
```

---

packagesOf.Globals *Identify the packages of the globals*

Description

Identify the packages of the globals

Usage

```
## S3 method for class 'Globals'
packagesOf(globals, ...)
```

Arguments

`globals`	A Globals object.
`...`	Not used.

Value

Returns a character vector of package names.

Index