Package 'refer'

October 14, 2022

Type Package

Title Create Object References

Version 0.1.0

Author Christopher Mann <cmann3@unl.edu>

Maintainer Christopher Mann <cmann3@unl.edu>

Description Allows users to easily create references to R objects then 'dereference' when needed or modify in place without using reference classes, environments, or active bindings as workarounds. Users can also create expression references that allow subsets of any object to be referenced or expressions containing references to multiple objects.

License MIT + file LICENSE

Encoding UTF-8

Imports utils, stats, eList, matchr

Suggests knitr, rmarkdown

RoxygenNote 7.1.1

Language en-US

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2021-11-08 12:10:04 UTC

R topics documented:

`decr` *Decrease Value In Place*

Description

Decrease the value of an object on the search path. Equivalent to `x--` or `x -= by` in other languages. See `incr` for details on implementation.

Usage

```
decr(x, by = 1)
```

Arguments

`x`	object to be decreased; can be a symbol, character, or extraction language object.
`by`	value to decrease `x` by; defaults to `1`.

Value

the value of `x` decreased by `by`, invisibly

Examples

```
z <- 1:10

incr(z)
identical(z, 2:11)       # TRUE

incr(z[1:3], by=2)
identical(z[1:3], 4:6)   # TRUE

l <- list(a = 1, b = 2)
decr(l$a)
l$a == 0   # TRUE

decr(l$b, by = 4)
l$b == -2  # TRUE
```

`deref` *Dereference Object*

Description

Return object from a `ref`. `` `!` `` can also be used to dereference an object. See `ref` for more details.

Usage

```
deref(x)

## S3 method for class 'ref'
!x
```

Arguments

`x`	reference object

Details

`deref` is used to obtain the object originally referenced from `ref`. `NULL` is returned if the object is no longer available. `ref` objects are automatically dereferenced when using generic functions such as arithmetic operators. Dereferencing a non-ref object just returns the object.

Value

R Obj or `NULL`

Examples

```
# Create a vectors of random numbers
x <- rnorm(10)
y <- runif(10)

# Create a reference to the random numbers
ref_to_x <- ref(x)
ref_to_y <- ref(y)

# Place references in a list
list_of_refs <- list(x = ref_to_x, y = ref_to_y)

# Check sum of refs 'x' and 'y'
# Note that both `+` and `sum` automatically deref
sum1 <- sum(list_of_refs$x + list_of_refs$y)

# Update 'x' and calculate new sum
x <- rnorm(10)
sum2 <- sum(list_of_refs$x + list_of_refs$y)

# check diff in sums to see if 'list_of_refs' updated
sum2 - sum1

# Obtain a reference to an expression
ref_to_part <- ref(x[2:5] + 3)
deref(ref_to_part)

# Another expression reference
refs_to_list <- ref(list(x, y))
deref(refs_to_list)

x <- "hello"
y <- "world"

deref(refs_to_list)

# Alternative, `!` can be used for dereferencing
!refs_to_list

identical(!refs_to_list, deref(refs_to_list))

# Referencing data.frame columns
dat <- data.frame(first = 1:4, second = 5:8)
ref_to_first <- ref(dat$first)
mean1 <- mean(!ref_to_first)

dat$first <- dat$first * 4
mean2 <- mean(!ref_to_first)

mean2 == 4*mean1

# Many operations automatically dereference
```

```
ref_to_first * 5
ref_to_x == ref_to_y
cos(ref_to_first)
max(ref_to_first)
```

`Extract` *Extract or Replace Parts of a Referenced Object*

Description

Operators acting on a `ref` object that extract part of the underlying object at the supplied indices, or replaces parts. These operators modify or extract from the object that is referenced, not the reference! Use `sref` is this behavior is undesirable.

Usage

```
## S3 method for class 'ref'
x$name

## S3 method for class 'sref'
x$..., value

## S3 replacement method for class 'ref'
x$name <- value

## S3 replacement method for class 'sref'
x$... <- value

## S3 method for class 'ref'
x[...]

## S3 method for class 'sref'
x[..., value]

## S3 replacement method for class 'ref'
x[...] <- value

## S3 replacement method for class 'sref'
x[...] <- value

## S3 method for class 'ref'
x[[...]]

## S3 method for class 'sref'
x[[..., value]]
```

```
## S3 replacement method for class 'ref'
x[[...]] <- value

## S3 replacement method for class 'sref'
x[[...]] <- value
```

Arguments

`x`	object of class "`ref`"
`name`	literal character string or a name
`...`	values passed to the function after dereferencing
`value`	object, usually of a similar class as the dereferenced value of `x`, used for assigning in place

Value

Object of class "`ref`"

Examples

```
x <- list(
  a = 1,
  b = "hello",
  "world"
)
ref_to_x <- ref(x)

# Extract parts of 'x' from the reference
ref_to_x$a
ref_to_x[2:3]
ref_to_x[["b"]]

# Replace parts of 'x' through the reference
ref_to_x[["a"]] <- 100
x$a == 100

ref_to_x$b <- "bye"
x$b == "bye"

ref_to_x[2:3] <- list(2, 3)
print(x)
```

getEnv — *Extract or Set Reference Environment*

Description

Functions to obtain or set the environment to which a `ref` or `sref` object points.

Usage

```
getEnv(x)

setEnv(x, e)
```

Arguments

`x`	object of class `"ref"` or `"sref"`
`e`	new environment to which the reference points

Value

environment for `getEnv` or reference object for `setEnv`

Examples

```
x <- 1:10
ref_to_x <- ref(x)
ref_env  <- getEnv(ref_to_x)
ref_sym  <- getSym(ref_to_x)

identical(ref_env, .GlobalEnv)
identical(ref_sym, "x")

e <- new.env()
e$x <- 100
ref_to_x <- setEnv(ref_to_x, e)
!ref_to_x
```

getIndex — *Extract or Set Slice Index*

Description

Functions to obtain or set the index to which a `slice` object points.

Usage

```
getIndex(x)

setIndex(x, ...)
```

Arguments

`x`	object of class "`slice`"
`...`	objects compatible with extracting or replacing a vector

Value

object of class "`slice`"

Examples

```
x <- matrix(1:9, nrow=3)
slice_x <- slice(x, 2:3, 1)
identical(getIndex(slice_x), list(2:3, 1)) # TRUE

setIndex(slice_x, list(1, substitute()))
identical(!slice_x, c(1, 4, 7))    # TRUE
```

`getSym` *Extract or Set Reference Symbol*

Description

Functions to obtain or set the object name to which a `ref` or `sref` object points.

Usage

```
getSym(x)

setSym(x, sym)
```

Arguments

`x`	object of class "`ref`"
`sym`	symbol or character naming the object to which the reference points

Value

character of length 1

Examples

```
x <- 1:10
ref_to_x <- ref(x)
ref_env  <- getEnv(ref_to_x)
ref_sym  <- getSym(ref_to_x)

identical(ref_env, .GlobalEnv)
identical(ref_sym, "x")

y <- 500
ref_to_x <- setSym(ref_to_x, y)
!ref_to_x
```

`incr` *Increment Value In Place*

Description

Increase the value of an object on the search path. Equivalent to `x++` or `x += by` in other languages.

Usage

```
incr(x, by = 1)
```

Arguments

`x`	object to be incremented; can be a symbol, character, or extraction language object.
`by`	value to increase `x` by; defaults to `1`.

Details

`incr` quotes object `x`, then attempts to determine the primary object to be modified. For example, `z` will be the 'primary object' in `incr(z[1:4])`. `incr` then searches for the primary object in the search path and records the environment. `x <- x + by` is then evaluated within the recorded environment.

The quoted object can be a symbol or character object. It can also be language object, though the primary call must be either `` `$` ``, `` `[` ``, or `` `[[` ``. These can be nested. For example, `x[1]` or `x[2, 1][3]` is acceptable, but `sqrt(x)` is not.

See `decr` to decrease the value.

Value

the value of `x` incremented by `by`, invisibly

Examples

```
z <- 1:10

incr(z)
identical(z, as.numeric(2:11))       # TRUE

incr(z[1:3], by=2)
identical(z[1:3], as.numeric(4:6))   # TRUE

l <- list(a = 1, b = 2)
decr(l$a)
l$a == 0   # TRUE

decr(l$b, by = 4)
l$b == -2  # TRUE
```

is.nullref   *Is Reference Null?*

Description

Check whether a ref points to a NULL object or an object that no longer exists.

Usage

```
is.nullref(x)
```

Arguments

x	object of class "ref"

Value

TRUE if x is not a reference or points to an object that does not exist; otherwise FALSE.

Examples

```
# Create a vectors of random numbers and a reference
x <- rnorm(10)
ref_to_x <- ref(x)

# Delete 'x' and check if NULL
is.nullref(ref_to_x) # FALSE
rm(x)
is.nullref(ref_to_x) # TRUE
```

is.ref *Is Object a Reference?*

Description

Check whether an R Object inherits a reference class.

Usage

```
is.ref(x)

is.sref(x)

is.rfexpr(x)

is.slice(x)

is.a.ref(x)
```

Arguments

x	object of any class

Value

TRUE if x is a reference object, otherwise FALSE

Functions

- is.sref: check whether object is an 'sref' object
- is.rfexpr: check whether object is a reference expression
- is.slice: check whether object references a slice of a vector
- is.a.ref: check whether object is any type of reference class

Examples

```
# Create a vectors of random numbers
x <- rnorm(10)

# Create a reference to the random numbers
ref_to_x <- ref(x)

is.ref(ref_to_x) # TRUE
```

iter.ref *Convert Reference to Iterable Object*

Description

ref methods for use with iter in the eList package. It allows ref objects to be used with the different vector comprehensions in the package and with functions such as lapply in base R.

Usage

```
## S3 method for class 'ref'
iter(x)

## S3 method for class 'slice'
iter(x)

## S3 method for class 'rfexpr'
iter(x)
```

Arguments

x	object to be looped across

Value

a vector

Examples

```
x <- sample(1:10, 5, replace=TRUE)
slice_x <- slice(x, 1:2)

lapply(eList::iter(slice_x), print)
```

match_cond.ref *Check and Evaluate Match Condition*

Description

ref methods for use with Match in the matchr package.

Usage

```
## S3 method for class 'ref'
match_cond(cond, x, do, ...)

## S3 method for class 'sref'
match_cond(cond, x, do, ...)

## S3 method for class 'slice'
match_cond(cond, x, do, ...)

## S3 method for class 'rfexpr'
match_cond(cond, x, do, ...)
```

Arguments

`cond`	match condition
`x`	object being matched
`do`	return expression associated with the condition. If cond is matched with `x`, then do should be evaluated and returned in a list with `TRUE`: `list(TRUE, eval(do))`.
`...`	arguments passed to evaluation

Details

See `Match` for details about the implementation of `match_cond`. When matching, `ref` conditions check whether `x` is a `ref` object. If so, then a match occurs if the condition and `x` point to the same object. Otherwise, the condition is dereferenced and the resulting value is checked using the appropriate match condition. Note that a `slice` is never matched with a `ref` and vice versa, though `ref` and `sref` objects may match if they point to the same object.

Value

`FALSE` if no match, or a list containing `TRUE` and the evaluated expression

Examples

```
x <- 1:10
ref_to_x <- ref(x)

matchr::Match(
  x,
  is.character -> "is a character",
  ref_to_x     -> "same as reference",   # <- MATCH
  .            -> "anything else"
)
```

Methods *S3 Methods for Automatic Dereferencing*

Description

These functions automatically call deref when applied to a ref or "rfexpr" object. Therefore, there is no need to explicitly call deref. sref objects will need to be explicitly dereferenced before applying these functions. All functions are from base R.

Usage

```
## S3 method for class 'ref'
Math(x, ...)

## S3 method for class 'ref'
Ops(e1, e2)

## S3 method for class 'ref'
Complex(z)

## S3 method for class 'ref'
Summary(..., na.rm = FALSE)

## S3 method for class 'rfexpr'
Math(x, ...)

## S3 method for class 'rfexpr'
Ops(e1, e2)

## S3 method for class 'rfexpr'
Complex(z)

## S3 method for class 'rfexpr'
Summary(..., na.rm = FALSE)

## S3 method for class 'ref'
all.equal(target, current, ...)

## S3 method for class 'ref'
anyDuplicated(x, incomparables = FALSE, ...)

## S3 method for class 'ref'
as.character(x, ...)

## S3 method for class 'ref'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

```
## S3 method for class 'ref'
as.Date(x, ...)

## S3 method for class 'ref'
as.double(x, ...)

## S3 method for class 'ref'
as.function(x, ...)

## S3 method for class 'ref'
as.list(x, ...)

## S3 method for class 'ref'
as.matrix(x, ...)

## S3 method for class 'ref'
as.POSIXct(x, tz = "", ...)

## S3 method for class 'ref'
as.POSIXlt(x, tz = "", ...)

## S3 method for class 'ref'
as.single(x, ...)

## S3 method for class 'ref'
as.table(x, ...)

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

## S3 method for class 'ref'
cut(x, ...)

## S3 method for class 'ref'
diff(x, ...)

## S3 method for class 'ref'
dim(x)

## S3 method for class 'ref'
droplevels(x, ...)

## S3 method for class 'ref'
duplicated(x, incomparables = FALSE, ...)

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

```
## S3 method for class 'ref'
isSymmetric(object, ...)

## S3 method for class 'ref'
kappa(z, ...)

## S3 method for class 'ref'
labels(object, ...)

## S3 method for class 'ref'
length(x)

## S3 method for class 'ref'
levels(x)

## S3 method for class 'ref'
mean(x, ...)

## S3 method for class 'ref'
merge(x, y, ...)

## S3 method for class 'ref'
qr(x, ...)

## S3 method for class 'ref'
rep(x, ...)

## S3 method for class 'ref'
rev(x)

## S3 method for class 'ref'
round(x, digits = 0)

## S3 method for class 'ref'
row.names(x)

## S3 method for class 'ref'
solve(a, b, ...)

## S3 method for class 'ref'
sort(x, decreasing = FALSE, ...)

## S3 method for class 'ref'
aggregate(x, ...)

## S3 method for class 'ref'
coef(object, ...)
```

```
## S3 method for class 'ref'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'ref'
fitted(object, ...)

## S3 method for class 'ref'
median(x, na.rm = FALSE, ...)

## S3 method for class 'ref'
model.frame(formula, ...)

## S3 method for class 'ref'
model.matrix(object, ...)

## S3 method for class 'ref'
na.omit(object, ...)

## S3 method for class 'ref'
plot(x, y, ...)

## S3 method for class 'ref'
predict(object, ...)

## S3 method for class 'ref'
residuals(object, ...)

## S3 method for class 'ref'
summary(object, ...)

## S3 method for class 'ref'
terms(x, ...)

## S3 method for class 'ref'
vcov(object, ...)

## S3 method for class 'ref'
window(x, ...)

## S3 method for class 'rfexpr'
all.equal(target, current, ...)

## S3 method for class 'rfexpr'
anyDuplicated(x, incomparables = FALSE, ...)

## S3 method for class 'rfexpr'
as.character(x, ...)
```

```
## S3 method for class 'rfexpr'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'rfexpr'
as.Date(x, ...)

## S3 method for class 'rfexpr'
as.double(x, ...)

## S3 method for class 'rfexpr'
as.function(x, ...)

## S3 method for class 'rfexpr'
as.list(x, ...)

## S3 method for class 'rfexpr'
as.matrix(x, ...)

## S3 method for class 'rfexpr'
as.POSIXct(x, tz = "", ...)

## S3 method for class 'rfexpr'
as.POSIXlt(x, tz = "", ...)

## S3 method for class 'rfexpr'
as.single(x, ...)

## S3 method for class 'rfexpr'
as.table(x, ...)

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

## S3 method for class 'rfexpr'
cut(x, ...)

## S3 method for class 'rfexpr'
diff(x, ...)

## S3 method for class 'rfexpr'
dim(x)

## S3 method for class 'rfexpr'
droplevels(x, ...)

## S3 method for class 'rfexpr'
duplicated(x, incomparables = FALSE, ...)
```

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

## S3 method for class 'rfexpr'
isSymmetric(object, ...)

## S3 method for class 'rfexpr'
kappa(z, ...)

## S3 method for class 'rfexpr'
labels(object, ...)

## S3 method for class 'rfexpr'
length(x)

## S3 method for class 'rfexpr'
levels(x)

## S3 method for class 'rfexpr'
mean(x, ...)

## S3 method for class 'rfexpr'
merge(x, y, ...)

## S3 method for class 'rfexpr'
qr(x, ...)

## S3 method for class 'rfexpr'
rep(x, ...)

## S3 method for class 'rfexpr'
rev(x)

## S3 method for class 'rfexpr'
round(x, digits = 0)

## S3 method for class 'rfexpr'
row.names(x)

## S3 method for class 'rfexpr'
solve(a, b, ...)

## S3 method for class 'rfexpr'
sort(x, decreasing = FALSE, ...)

## S3 method for class 'rfexpr'
aggregate(x, ...)
```

```
## S3 method for class 'rfexpr'
coef(object, ...)

## S3 method for class 'rfexpr'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'rfexpr'
fitted(object, ...)

## S3 method for class 'rfexpr'
median(x, na.rm = FALSE, ...)

## S3 method for class 'rfexpr'
model.frame(formula, ...)

## S3 method for class 'rfexpr'
model.matrix(object, ...)

## S3 method for class 'rfexpr'
na.omit(object, ...)

## S3 method for class 'rfexpr'
plot(x, y, ...)

## S3 method for class 'rfexpr'
predict(object, ...)

## S3 method for class 'rfexpr'
residuals(object, ...)

## S3 method for class 'rfexpr'
summary(object, ...)

## S3 method for class 'rfexpr'
terms(x, ...)

## S3 method for class 'rfexpr'
vcov(object, ...)

## S3 method for class 'rfexpr'
window(x, ...)
```

Arguments

`x, y, e1, e2, z, target, current, object, a, b, formula`	objects of class "`ref`"
`...`	other objects passed to the function
`incomparables, digits, tz, row.names, optional, decreasing, na.rm, parm, level`	function specific arguments. See the relevant functions for more details

Value

An R object depending on the function.

`modify_by` *Modify an Object In Place*

Description

Update the value pointed to by a `ref` object. If the new value is a function, the old values will be applied to the function and overwritten.

Usage

```
modify_by(x, value, ...)
```

Arguments

`x`	object of class `"ref"`
`value`	new value or function applied to the object at the referenced location
`...`	additional arguments passed to the function

Value

object of class `"ref"`

Examples

```
x <- 1:10
ref_to_x <- ref(x)

# Apply the square root function
modify_by(ref_to_x, sqrt)
print(x)

# Overwrite the original values
modify_by(ref_to_x, "hello world!")
print(x)
```

ref *Create Reference to an Object*

Description

Create a reference to an arbitrary R object. Use deref or `` `!` `` to obtain the values within the referenced object. Use sref to create a safer reference that limits modification in place.

Usage

```
ref(x)
```

Arguments

x	object to be referenced. x can be a symbol, character, or an expression containing a symbol.

Details

Since R does not have reference semantics outside of environments, ref records the environment location of an object rather than its memory address.ref(x) searches for object with name "x" within the search path. If found, a reference to the environment and the name "x" are recorded. Otherwise, an error is returned.

ref can also create a reference to objects within an expression. ref searches the uncalled names within the expression and replaces them with a reference to the object and a call to deref. For example, ref(x[[y]][2]) inserts a reference to variable x and variable y from the search path into the expression then wraps the expression into an object of class "ref_exp". These objects are dereferenced by evaluating the expression. An error is returned only if the corresponding variables cannot be found along the search path.

deref can be used to find the objects at the referenced location. This usually results in a copy of the objects. If the object is no longer available, NULL will be returned. Generic functions on a ref object, such as arithmetic or `` `sqrt` ``, will automatically dereference the object before applying the generic function. See Methods and Extract for a list of available functions where explicit dereferencing is not needed. If this behavior is not desired, then sref can be used to force the explicit use of deref.

See Extract and modify_by for functions that modify the underlying value in place.

An active binding could also be used instead of creating a reference. Active bindings, though, can be more difficult to pass around and may have additional overhead since they are functions.

ref can provide unsafe or inconsistent code that is susceptible to side-effects. Apply caution and restraint with its use and be sure to deref before exporting any ref objects.

Value

a list of class "ref" containing a reference to the environment of the object and the name of the object to be found within the environment, or an expression of class "rfexpr" containing references

Examples

```
# Create a vectors of random numbers
x <- rnorm(10)
y <- runif(10)

# Create a reference to the random numbers
ref_to_x <- ref(x)
ref_to_y <- ref(y)

# Place references in a list
list_of_refs <- list(x = ref_to_x, y = ref_to_y)

# Check sum of refs 'x' and 'y'
# Note that both `+` and `sum` automatically deref
sum1 <- sum(list_of_refs$x + list_of_refs$y)

# Update 'x' and calculate new sum
x <- rnorm(10)
sum2 <- sum(list_of_refs$x + list_of_refs$y)

# check diff in sums to see if 'list_of_refs' updated
sum2 - sum1

# Obtain a reference to an expression
ref_to_part <- ref(x[2:5] + 3)
deref(ref_to_part)

# Another expression reference
refs_to_list <- ref(list(x, y))
deref(refs_to_list)

x <- "hello"
y <- "world"

deref(refs_to_list)

# Alternative, `!` can be used for dereferencing
!refs_to_list

identical(!refs_to_list, deref(refs_to_list))

# Referencing data.frame columns
dat <- data.frame(first = 1:4, second = 5:8)
ref_to_first <- ref(dat$first)
mean1 <- mean(!ref_to_first)

dat$first <- dat$first * 4
mean2 <- mean(!ref_to_first)

mean2 == 4*mean1

# Many operations automatically dereference
```

```
ref_to_first * 5
ref_to_x == ref_to_y
cos(ref_to_first)
max(ref_to_first)
```

---

ref_list *Create A List of References*

Description

Create a list of references or referenced expressions. See `ref` for more details.

Usage

```
ref_list(...)
```

Arguments

`...`	objects to be referenced, possibly named.

Value

a list containing object references

Examples

```
x <- 1
y <- "hello"
z <- list(a = 1, b = 2, c = 3)

new_list <- ref_list(x, second = y, z)

!new_list[[1]]
(!new_list$second) == y  # TRUE

y <- 18
(!new_list$second) == 18 # TRUE
```

slice — *Create a Reference Slice to a Vector*

Description

Create a reference to a 'part' of an R object. Use deref or `` `!` `` to obtain the values within the referenced object.

Usage

```
slice(x, ...)
```

Arguments

`x`	object to be referenced; must be a symbol or character
`...`	objects passed to `x[...]` when dereferenced

Details

`slice` is similar to ref; it creates a reference to another R object. There are two main differences with `ref`. First, `slice` only accepts names or characters instead of expressions. Second, `slice` records a part of the underlying object. `slice(x, 1:2, 3)` is equivalent to the reference of `x[1:2, 3]`. This is similar to `ref(x[1:2, 3])`, though the implementation is different. `ref` would create an expression with a reference to `x`, while `slice(x, 1:2, 3)` creates a list with a reference to `x` and the extract inputs. `slice` is more efficient, but is limited in its capabilities.

Value

object of class `"slice"` and `"ref"`

Examples

```
## Vector Slice
x <- 10:1

slice_x <- slice(x, 2:4)
identical(!slice_x, 9:7)   # TRUE

x <- x - 2
identical(!slice_x, 7:5)   # TRUE

## Matrix Slice
y <- matrix(1:9, nrow=3)
slice_y <- slice(y, 2, 3)

identical(!slice_y, y[2, 3])   # TRUE
```

sref — *Create a Safer Reference to an Object*

Description

Create a reference to an arbitrary R object. See `ref` for more details. `sref` behaves similar to `ref`, but does not have support for direct operations on the referenced object.

Usage

```
sref(x)
```

Arguments

`x`	object to be referenced. `x` can be a symbol, character, or an expression containing a symbol.

Details

`sref` is similar to `ref`; it accepts either an R object or an expression, then records its location. `ref` objects prioritize convenience, while `sref` objects prioritize clarity and safety. For example, `` `[` `` and `` `$` `` can be used on a `ref` object to access the elements of the underlying object, while `` `[<-` `` and `` `$<-` `` can be used to overwrite elements within. These do not work for `sref` objects. Furthermore, base mathematical functions such as `` `+` `` and `sqrt` also will not automatically dereference before applying.

Examples

```
x <- 1:10
ref_x  <- ref(x)
sref_x <- sref(x)

## These operations will run:
ref_x + 5
ref_x[1:4]
ref_x[7] <- 5

## These operations will not run:
# sref_x + 5
# sref_x[1:4]
# sref_x[7] <- 5
```

sslice *Create a Safer Reference Slice to a Vector*

Description

Create a reference to a 'part' of an R object. sslice behaves similar to slice, but does not have support for direct operations on the referenced object. See sref for details about the behavior.

Usage

```
sslice(x, ...)
```

Arguments

x	object to be referenced; must be a symbol or character
...	objects passed to x[...] when dereferenced

Value

object of class "sslice" and "sref"

%.*=% *Matrix Multiplication In Place*

Description

Change the value of an object on the search path through matrix multiplication. Similar to '*=' in other languages, except with matrix multiplication. See incr for details on implementation.

Usage

```
x %.*=% value
```

Arguments

x	object to be modified; can be a symbol, character, or extraction language object.
value	value with which to change x by

Value

the new value of x, invisibly

Examples

```
x <- 1:5
x %*=% 6:10
identical(x, 130)  # TRUE
```

`%-=%` *Subtract In Place*

Description

Decrease the value of an object on the search path. Equivalent to `'-='` in other languages. See `incr` for details on implementation.

Usage

```
x %-=% value
```

Arguments

`x`	object to be modified; can be a symbol, character, or extraction language object.
`value`	value with which to change `x` by

Value

the new value of `x`, invisibly

Examples

```
x <- 11:20
x %-=% 10
identical(x, 1:10)  # TRUE
```

`%+=%` *Add In Place*

Description

Increase the value of an object on the search path. Equivalent to `'+='` in other languages. See `incr` for details on implementation.

Usage

```
x %+=% value
```

Arguments

`x`	object to be modified; can be a symbol, character, or extraction language object.
`value`	value with which to change `x` by

Value

the new value of `x`, invisibly

Examples

```
x <- 1:10
x %+=% 10
identical(x, 11:20)  # TRUE
```

`%^=%` *Power In Place*

Description

Change the value of an object on the search path through exponentiation Equivalent to `'^='` in other languages. See `incr` for details on implementation.

Usage

```
x %^=% value
```

Arguments

`x`	object to be modified; can be a symbol, character, or extraction language object.
`value`	value with which to change `x` by

Value

the new value of `x`, invisibly

Examples

```
x <- 10
x %^=% 2
identical(x, 100)  # TRUE
```

`%/=%` *Divide In Place*

Description

Change the value of an object on the search path through division. Equivalent to `'/='` in other languages. See `incr` for details on implementation.

Usage

```
x %/=% value
```

Arguments

`x`	object to be modified; can be a symbol, character, or extraction language object.
`value`	value with which to change `x` by

Value

the new value of `x`, invisibly

Examples

```
x <- 10
x %/=% 2
identical(x, 5)  # TRUE
```

`%*=%` *Multiply In Place*

Description

Change the value of an object on the search path through multiplication. Equivalent to `'*='` in other languages. See `incr` for details on implementation.

Usage

```
x %*=% value
```

Arguments

`x`	object to be modified; can be a symbol, character, or extraction language object.
`value`	value with which to change `x` by

Value

the new value of `x`, invisibly

Examples

```
x <- 5
x %*=% 2
identical(x, 10)  # TRUE
```

Index