Package: operator.tools (via r-universe)

October 17, 2024

Type Package

Title Utilities for Working with R's Operators

Version 1.6.3

Date 2017-02-28

Imports utils (>= 3.3.2)

Suggests operators (>= 0.1.8), magrittr (>= 1.5), testthat (>= 1.0.2)

Description Provides a collection of utilities that allow programming with R's operators. Routines allow classifying operators, translating to and from an operator and its underlying function, and inverting some operators (e.g. comparison operators), etc. All methods can be extended to custom infix operators.

License GPL-2 | file LICENSE

URL https://github.com/decisionpatterns/operator.tools

BugReports https://github.com/decisionpatterns/operator.tools/issues

RoxygenNote 6.0.1.9000

Repository https://decisionpatterns.r-universe.dev

RemoteUrl https://github.com/decisionpatterns/operator.tools

RemoteRef HEAD

RemoteSha cb5054bcbaea9130e5e409ac46f1f2e98fbc18b3

Contents

`.initOps` *Initialize operators*

Description

Initialize operators

Usage

```
.initOps()
```

`can.operator` *can.operator*

Description

tests whether an object can be coerced to a operator, optionally an operator of 'types'.

Usage

```
can.operator(x, ...)
```

Arguments

`x`	object; to test
`...`	additional arguments
	`can.operator` test whether an object can be coerced to an operator. Methods exist for name, function amd character classes

Value

logical

fun2name *Convert between a function and its name and vice versa.*

Description

fun2name compares a function (body) to all defined functions. If an identical match is found to a defined function, that function is returned. NB. This does not search through S4 methods.

Usage

```
fun2name(f)

name2fun(x)
```

Arguments

f	function
x	name; more specifically, an object to be converted into a name and eval'd fun2name compares the function against existing functions using identical. If a match is found, the name of the matching function (expressed as a character) is returned. fun2name will not work for S4 Methods.

Details

name2fun simply converts its argument to a name and than evals it to produce a function definition

Value

fun2name: character (name of function) name2fun: function

inverse *Invert an R operator*

Description

inverse is a S3 generic method for inverting an R operator in the mathematical sense. Presently, inverses are defined for relational operators, i.e. changing > to <= etc.

Usage

```
inverse(x, ...)
```

Arguments

x	object representing an R operator
...	additional arguments

Details

Arguments will be checked against the defined list of inverses, If an entry exists, the corresponding inverse is returned.

Value

`inverse` returns the inverse in the same form as the `x` argument. Thus, if a name is provided, a name is returned. If a function is provided, a function is returned.

Author(s)

Christopher Brown

References

http://en.wikipedia.org/wiki/Inverse_mathematics.

See Also

`operators` especially `operators(type="relational"))`

Examples

```
  ## Not run:
    inverse( as.name( '!=' ) )
    inverse( `==` )

## End(Not run)
```

`is.operator` *Utilities for operators*

Description

These S4 Methods are utilies for working with operators. In R, operators are functions with special syntax.

Usage

```
is.operator(x, ...)
```

Arguments

x	object to be tested or coerced. Can be function or name.
...	additional arguments passed to operators.

Details

is.operator tests whether the object is one of the defined operators.

can.operator tests whether the object can be coerced to an operator.

as.operator coerced the object to an operator.

Optionally, you can specify one of the that it tests for a specific type of operator. See details, below.

An operator is R function with special syntax.

(See ??operator for examples of each.)

is.operator tests whether the argumenst is an operator.

as.operator coerces x to a operator, otherwise fails.

can.operator test whether the object can be coerced to an operator.

All functions can accepts a types argument which is passed to link{operators}. By specifying one or more types, these functions test using those types only.

New operators can be "registered" using setOperator.

Value

is.operator and can.operator return logical.

as.operator returns the argument coerced to the concommitant R function.

Author(s)

Christopher Brown

See Also

operators, apropos, match.fun

Examples

```
## Not run:
  is.operator( `+` )
  is.operator( 'xyzzy' )
  is.operator( `+`, types="arithmetic" )
  is.operator( `+`, types="relational" )

  can.operator( `+` )
  can.operator( 'xyzzy' )
  can.operator( `+`, types="arithmetic" )
  can.operator( `+`, types="relational" )
```

```
   as.operator( `+` )
   as.operator( '+' )
   as.operator( as.name('+') )

## End(Not run)
```

`notin` *NOT IN*

Description

Commonly created NOT-IN operator

Usage

```
x %!in% table
```

Arguments

`x`	object on the lhs
`table`	object/list on the rhs

`operator.type` *Return the type for an operator.*

Description

Given an operator or its name/symbol, return the **type** of operator.

Usage

```
operator.type(op)
```

Arguments

`op`	An operator either as a name/symbol or function.

Details

The operator is first checked against all operators that have been registered with the `setOperator` command. If there is a match, its type is returned. If no matching operator is found, `op` is matched against unregistered operators that have been defined with the `%any%`-syntax. If a match is found, UNREGISTERED is returned.

The list of operators are maintained in `.Options\$operators` and be altered suing the `setOperator` command.

Value

A character value.

For registered operators, the registered type is returned. For Base R operators, the types come from Syntax.

For operators defined with the %any%-syntax but, not registered using setOperator, "UNREGISTERED" is returned.

NULL is returned otherwise.

Author(s)

Christopher Brown

See Also

operators, setOperator. Syntax

Examples

```
 ## Not run:
  operator.type( `+` )
  operator.type( `<=` )

  e <- quote( A +B )
  operator.type( e[[1]] )

  operator.type( as.name('+') )

## End(Not run)
```

---

operators *Return the _names_ of defined operators.*

Description

operators returns the names of defined operators. Argument types can be used to select operators of a specified type(s) or GROUPING(s). See Details for specifics.

Usage

```
operators(types = "REGISTERED")
```

Arguments

types	A character vector with the types of operators to return. The types may one or more of: 'namespace', 'component', 'indexing', 'sequence', 'arithmetic', 'relational', 'logical', 'tilde', 'assignment', 'help', 'user', or user-defined type specified in a call to `setOperator`. It may also be one of the special groups: 'REG(ISTERED)', 'UNREG(ISTERED)', 'SPECIAL', 'ALL'. See Details. `operators` provides the **names** of defined operators. These can be either registered operators (using `setOperators`), or unregistered operators definde by the `%any%` syntax. By default, only registered operators are returned. This is purely for performance reasons as an exhausting search for `%any%` functions is expensive. See Syntax.for the core R operators `types` may also be one a special operator groupings: • REG(ISTERED): (Default). Those registered by `setOperators` • UNREG(ISTERED): Unregisted operators, requires expensive search. • ALL: All operators, requires expensive search of environments. • SPECIAL: All operators defined using the `%any%` syntax.

Value

character vector of unique operator names.

Note

The right arrow assignment operators, `->` and `->>` is not an operator but a syntatic variant. Consequently, it does not behave properly as an operator. They are omitted from the operator list as they are not correctly identified as primitives or functions by the R language.

Author(s)

Christopher Brown

References

https://cran.r-project.org/doc/manuals/R-lang.html https://bugs.r-project.org/bugzilla3/show_bug.cgi?id=14310

See Also

Syntax, `setOperator`, `setOperators`, and the help files on the individial operators.

Examples

```
## Not run:
 operators()
 operators( types="arithmetic" )
 operators( types=c("arithmetic","logical" ) )
 operators( types='ALL' )
```

```
  operators( types='REG' )
  operators( types='UNREG' )
  operators( types='SPECIAL' )

## End(Not run)
```

`rel.type` *Get the relational type of a relational operator.*

Description

`rel.type` gets the relational type of a relational operator. The relational type is one of `'gt'`, `'lt'`, `'eq'`, `'ne'`.

Usage

```
rel.type(x)
```

Arguments

`x`	An operators expressed as a `function` or `name`

Details

A relational operator is an operate that relates the relationship between arguments. The core relational operators are: >, >=, <, <=, ==, !=,

The relational.type is a simple roll-up of these operators. > and >= are gt, etc. The value is retrieved from .Options$operators[[x]][['rel.type']] and can be defined for relational operators using `setOperator`.

A relational type provides an indication of nature of the relational operator.

Value

`character` value of the operator. One of: `'gt'`, `'lt'`, `'eq'`, `'ne'`.

Author(s)

Christopher Brown

See Also

`operators`, `setOperator`

Examples

```
 ## Not run:
  rel.type( `==` )
  rel.type( as.name('==') )

## End(Not run)
```

`removeOperator` *Unregister a an operator.*

Description

`removeOperator` unregisted an operator by removing it from the list of operators found at `.Options$operators`. All operator attributes are that have been set will be lost.

Usage

```
removeOperator(x)
```

Arguments

`x`	`character`. The name of the operator

Details

Warns if the operator has not been registered.

Value

None. Used for side-effects.

Author(s)

Christopher Brown

See Also

`setOperators` for registering Operators.

Examples

```
  # Unregister ? as an operator.
  ## Not run:
    removeOperator( '?' )

## End(Not run)
```

`setOperator` *Registers an operator for use with operator.tools package.*

Description

`setOperator` registers a user-defined operator as a given type. Subsequently, this operator can be treated as a member of a class of operators.

Usage

```
setOperator(name, type = "user", ...)

setOperators(...)
```

Arguments

`name`	A character vector containing the **names** of one or more functions which will be registered.
`type`	The type of operator. See Details.
`...`	Attributes for the operator(s).

Details

`setOperators` scans defined functions looking for any that have been defined by the user using the special `any` syntax. If found, these are registered with `setOperator` and given the default type='user'.

`setOperator` registers a single operator similar to the way that `setMethod` registers a method. The definition for these operators are defined by `.Options$operators`.

`setOperators` scans the environments for user-defined operators. If found and not already registered, these are registered by `setOperator`. Registered operators are much more efficient than unregisted ones, so it is often advantageous to register the operators. When `...` is supplied, these attributes are set for all unregistered operators.

Operators are allowed to have attributes. The one required attribute is `type`, which is just a character value that serves to classification the operator. On package load, All operators from base R are assigned a core type as specified in Syntax. These are: namespace, component, indexing, sequence, arithmetic, arithmetic, relational, logical, tilde, assignment, help.

Users may use one of these types or assign a type of their own choosing. The `type` is largely unrestricted, but cannot be one of the reserved operator groupings: ALL, REG(ISTERED), UNREG(ISTERED), SPECIAL or user. These have special meaning as described in `operators`. Users are encouraaged to make their own types in lower case.

Value

None. This function exists for assigning a operator to `options('operators')`.

Author(s)

Christopher Brown

See Also

operators, Syntax

Examples

```
  ## Not run:
    setOperator( '%!in%', 'relational' )
    operators( type='relational' )

## End(Not run)
```

Index