Package 'R.oo'

July 2, 2013

Version 1.13.9

Depends R (>= 2.4.0), R.methodsS3 (>= 1.4.4)

Imports methods, utils, R.methodsS3 (>= 1.4.4)

Suggests tools

Date 2013-07-01

Title R object-oriented programming with or without references

Author Henrik Bengtsson

Maintainer Henrik Bengtsson <henrikb@braju.com>

Description Methods and classes for object-oriented programming in R
with or without references. Large effort has been made on
making definition of methods as simple as possible with a
minimum of maintenance for package developers. The package has
been developed since 2001 and is now considered very stable.
This is a cross-platform package implemented in pure R that
defines standard S3 classes without any tricks.

License LGPL (>= 2.1)

LazyLoad TRUE

NeedsCompilation no

Repository CRAN

Date/Publication 2013-07-02 00:35:07

R topics documented:

R.oo-package   *Package R.oo*

Description

Methods and classes for object-oriented programming in R with or without references. Large effort
has been made on making definition of methods as simple as possible with a minimum of mainte-
nance for package developers. The package has been developed since 2001 and is now considered
very stable. This is a cross-platform package implemented in pure R that defines standard S3 classes
without any tricks.

Please note that the Rdoc syntax/grammar used to convert Rdoc comments in code into Rd files is
not strictly defined and is modified by the need of the author. Ideally, there will be a well defined
Rdoc language one day.

Installation and updates

To install this package do

```
install.packages("R.oo")
```

Dependancies and other requirements

This package requires a standard R installation and the **R.methodsS3** package.

To get started

To get started,It is very useful to understand that:

1. The setMethodS3() function, which is defined in the **R.methodsS3** package (used to be part of **R.oo**), is nothing but a conveniency wrapper for setting up S3 methods, and automatically create an S3 generic function, if missing. For more information, see the help of **R.methodsS3**.

2. The Object class is a top-level "root" class that provides support for *reference variables*. Any class inheriting from this class supports reference variables.

3. The Object class is basically a wrapper around an environment, which some additional accessors etc. It is the environment data type that provides the "emulation" of reference variables - the Object class structure makes it easier to extends this class and adds some level of coding protection. The Object class features is not intended for referencing individual elements of basic R data types, but rather for the whole variable of such. For instance, you can reassign a whole matrix X part of the object this way, but you cannot reassign X[1,1] without creating a completely new copy.

Further readings

For a detailed introduction to the package see [1] (part of the package distribution).

How to cite this package

Whenever using this package, please cite [1] as

```
@INPROCEEDINGS{BengtssonH_2003,
 author       = {Henrik Bengtsson},
 title        = {The {R.oo} package - Object-Oriented Programming
                  with References Using Standard {R} Code},
 booktitle    = {Proceedings of the 3rd International Workshop on
                  Distributed Statistical Computing (DSC 2003)},
 year         = {2003},
 editor       = {Kurt Hornik and Friedrich Leisch and Achim Zeileis},
 address      = {Vienna, Austria},
 month        = {March},
 issn         = {1609-395X},
 howpublished = {http://www.ci.tuwien.ac.at/Conferences/DSC-2003/},
}
```

License

The releases of this package is licensed under LGPL version 2.1 or newer.

Author(s)

Henrik Bengtsson

References

[1] H. Bengtsson, *The R.oo package - Object-Oriented Programming with References Using Standard R Code*, In Kurt Hornik, Friedrich Leisch and Achim Zeileis, editors, Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/

See Also

People interested in **R.oo** may also be interested in packages **proto** and **mutatr**.

---

charToInt	*Converts a vector of ASCII characters into a vector of integers*

Description

Converts a vector of ASCII characters to a equal length vector of ASCII integers.

Usage

```
## Default S3 method:
charToInt(ch, ...)
```

Arguments

ch   A character vector.
...   Not used.

Value

Returns an ASCII character vector.

Author(s)

Henrik Bengtsson

See Also

intToChar() utf8Conversion. rawConversion

Examples

```
i <- charToInt(unlist(strsplit("Hello world!", split=NULL)))
# Gives: 72 101 108 108 111  32 119 111 114 108 100  33
ch <- intToChar(c(72,101,108,108,111,32,119,111,114,108,100,33))
# Gives: "H" "e" "l" "l" "o" " " "w" "o" "r" "l" "d" "!"
```

| Class | *The Class class describes an Object class* |

Description

Package: R.oo
Class Class

Object
~~|
~~+--Class

Directly known subclasses:

public static class **Class**
extends Object

The Class class describes an Object class. First of all, this class is most commonly used *internally* and neither the end user nor the programmer need to no about the class Class.

Usage

```
Class(name=NULL, constructor=NULL)
```

Arguments

name	Name of the class.
constructor	Constructor (function) of any Object class.

Details

The class Class describes the Object class or one of its subclasses. All classes and constructors created by setConstructorS3() will be of class Class. Its methods provide ways of accessing static fields and static methods. Its *print()* method will print detailed information about the class and its fields and methods.

Fields and Methods

Methods:

$	-
$<-	-
[[	-
[[<-	-

argsToString	Gets the arguments of a function as a character string.
as.character	Returns a short string describing the class.
forName	Gets a Class object by a name of a class.
getDetails	Lists the fields and methods of a class.
getFields	Returns the field names of a class.
getKnownSubclasses	Gets all subclasses that are currently loaded.
getMethods	Returns the method names of class and its super classes.
getName	Gets the name of the class.
getPackage	Gets the package to which the class belongs.
getRdDeclaration	Gets the class declaraction in Rd format.
getRdHierarchy	Gets the class hierarchy in Rd format.
getRdMethods	Gets the methods of a class in Rd format.
getStaticInstance	Gets the static instance of this class.
getSuperclasses	Gets the super classes of this class.
isAbstract	Checks if a class is abstract or not.
isBeingCreated	Checks if a class is currently being initiated initiated.
isDeprecated	Checks if a class is deprecated or not.
isPrivate	Checks if a class is defined private or not.
isProtected	Checks if a class is defined protected or not.
isPublic	Checks if a class is defined public or not.
isStatic	Checks if a class is static or not.
newInstance	Creates a new instance of this class.
print	Prints detailed information about the class and its fields and methods.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

---

dimension   *Gets the dimension of the object*

Description

Gets the dimension of the object similar to what dim() does, but instead of NULL it will return the length of a vector. If a function is passed, NULL is returned.

Usage

```
## Default S3 method:
dimension(object, ...)
```

Arguments

object   The object for which the dimension should be obtained.

...   Not used.

Value

Returns an integer vector or NULL.

Author(s)

Henrik Bengtsson

See Also

ll.default(). [dim](dim)() and length().

Examples

```
dimension(matrix(1:100, ncol=10))    # 10 10
dimension(1:14)                      # 14
dimension(data.frame(a=1:10, b=10:1)) # 10  2
dimension(print)                     # NULL
```

---

equals   *Compares an object with another*

Description

Compares an object with another and returns TRUE if they are equal. The equal property must be

1) *reflexive*, i.e. equals(o1,o1) should be TRUE.

2) *symmetric*, i.e. equals(o1,o2) is TRUE if and only if equals(o2,o1) is TRUE.

3) *transitive*, i.e. equals(o1,o2) is TRUE and equals(o2,o3) is TRUE, then equals(o1,o3) should be TRUE.

5) *consistent*, i.e. equals(o1,o2) should return the same result on multiple invocations as long as nothing has changed.

6) equals(o1,NULL) should return FALSE.

By default identical() is used.

Usage

```
## Default S3 method:
equals(object, other, ...)
```

Arguments

object, other   Objects to be compared.

...   Not used.

Value

Returns [TRUE](#) if the objects are equal, otherwise [FALSE](#).

Author(s)

Henrik Bengtsson

See Also

[identical](#)().

Exception   *The Exception class to be thrown and caught*

Description

Package: R.oo
Class Exception

```
[Object]
~~|
~~+--try-error
~~~~~~~|
~~~~~~~+--condition
~~~~~~~~~~~~|
~~~~~~~~~~~~+--error
~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~+--simpleError
~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~+--Exception
```

Directly known subclasses:
[InternalErrorException](#), [RccViolationException](#), [RdocException](#)

public static class **Exception**
extends simpleError

Creates an Exception that can be thrown and caught. The Exception class is the root class of all
other Exception classes.

Usage

```
Exception(..., sep="", collapse=", ")
```

Arguments

`...`	One or several strings, which will be concatenated and contain informative message about the exception.
`sep`	The string to used for concatenating several strings.
`collapse`	The string to used collapse vectors together.

Fields and Methods

Methods:

as.character	Gets a character string representing of the Exception.
getCall	-
getCalls	Gets the active calls saved when the exception was created.
getLastException	Static method to get the last Exception thrown.
getMessage	Gets the message of the Exception.
getStackTrace	Gets the stack trace saved when the exception was created.
getStackTraceString	Gets the stack trace as a string.
getWhen	Gets the time when the Exception was created.
print	Prints the Exception.
printStackTrace	Prints the stack trace saved when the exception was created.
throw	Throws an Exception that can be caught.

Methods inherited from error:
as.character, throw

Methods inherited from condition:
abort, as.character, conditionCall, conditionMessage, print

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

See Also

See also tryCatch() (and try()).

Examples

```
##########################################################################
# 1. To catch a regular "error" exception thrown by e.g. stop().
##########################################################################
x <- NA
y <- NA
tryCatch({
  x <- log(123)
  y <- log("a")
}, error = function(ex) {
  print(ex)
})
print(x)
print(y)




##########################################################################
# 2. Always run a "final" expression regardless or error or not.
##########################################################################
filename <- tempfile("R.methodsS3.example")
con <- file(filename)
tryCatch({
  open(con, "r")
}, error = function(ex) {
  cat("Could not open ", filename, " for reading.\n", sep="")
}, finally = {
  close(con)
  cat("The id of the connection is ",
        ifelse(is.null(con), "NULL", con), ".\n", sep="")
})




##########################################################################
# 3. Creating your own Exception class
##########################################################################
setConstructorS3("NegativeLogValueException", function(
  msg="Trying to calculate the logarithm of a negative value", value=NULL) {
  extend(Exception(msg=msg), "NegativeLogValueException",
    .value = value
  )
})

setMethodS3("as.character", "NegativeLogValueException", function(this, ...) {
  paste(as.character.Exception(this), ": ", getValue(this), sep="")
})

setMethodS3("getValue", "NegativeLogValueException", function(this, ...) {
  this$.value
})


mylog <- function(x, base=exp(1)) {
  if (x < 0)
```

```
      throw(NegativeLogValueException(value=x))
    else
      log(x, base=base)
}


# Note that the order of the catch list is important:
l <- NA
x <- 123
tryCatch({
  l <- mylog(x)
}, NegativeLogValueException = function(ex) {
  cat(as.character(ex), "\n")
}, "try-error" = function(ex) {
  cat("try-error: Could not calculate the logarithm of ", x, ".\n", sep="")
}, error = function(ex) {
  cat("error: Could not calculate the logarithm of ", x, ".\n", sep="")
})
cat("The logarithm of ", x, " is ", l, ".\n\n", sep="")
```

---

extend   *Extends a object*

Description

via a mechanism known as "parasitic inheritance". Simply speaking this method "extends" the class
of an object. What is actually happening is that it creates an instance of class name `...className`,
by taking another object and add `...className` to the class list and also add all the named values
in `...` as attributes.

The method should be used by the constructor of a class and nowhere else.

Usage

```
## Default S3 method:
extend(this, ...className, ...)
```

Arguments

this	Object to be extended.
...className	The name of new class.
...	Attribute fields of the new class.

Value

Returns an object of class `...className`.

Author(s)

Henrik Bengtsson

Examples

```
setConstructorS3("MyDouble", function(value=0, ...) {
  extend(as.double(value), "MyDouble", ...)
})

setMethodS3("as.character", "MyDouble", function(object, ...) {
  fmtstr <- attr(object, "fmtstr")
  if (is.null(fmtstr))
    fmtstr <- "%.6f"
  sprintf(fmtstr, object)
})

setMethodS3("print", "MyDouble", function(object, ...) {
  print(as.character(object), ...)
})

x <- MyDouble(3.1415926)
print(x)

x <- MyDouble(3.1415926, fmtstr="%3.2f")
print(x)
attr(x, "fmtstr") <- "%e"
print(x)




setConstructorS3("MyList", function(value=0, ...) {
  extend(list(value=value, ...), "MyList")
})

setMethodS3("as.character", "MyList", function(object, ...) {
  fmtstr <- object$fmtstr
  if (is.null(fmtstr))
    fmtstr <- "%.6f"
  sprintf(fmtstr, object$value)
})

setMethodS3("print", "MyList", function(object, ...) {
  print(as.character(object), ...)
})

x <- MyList(3.1415926)
print(x)
x <- MyList(3.1415926, fmtstr="%3.2f")
print(x)
```

```
x$fmtstr <- "%e"
print(x)
```

---

getConstructorS3   *Get a constructor method*

Description

Get a constructor method.

Usage

```
## Default S3 method:
getConstructorS3(name, ...)
```

Arguments

name   The name of the constructor function.
...   Not used.

Author(s)

Henrik Bengtsson

See Also

[setConstructorS3](). [getMethodS3.](\) [isGenericS3.](\)

---

getName.environment   *Gets the name of an environment*

Description

Gets the name of an environment, e.g. "R_GlobalEnv" or "0x01ddd060".

Usage

```
## S3 method for class 'environment'
getName(env, ...)
```

Arguments

env   An [environment]().
...   Not used.

Value

Returns a [character](#) string.

Author(s)

Henrik Bengtsson

See Also

[environmentName](#)().

Examples

```
name <- getName(globalenv())
print(name)
stopifnot(identical(name, "R_GlobalEnv"))

getName(new.env())
```

---

hashCode   *Gets an integer hashcoded for R objects*

Description

Gets an integer hashcoded for R objects.

Usage

```
## Default S3 method:
hashCode(object, ...)
```

Arguments

object   A [vector](#) or [list](#) of R objects.

...   Not used.

Details

A [character](#) string is converted into a hashcode following Java conventions by s[1]*31^(n-1) + s[2]*31^(n-2) + ... using integer arithmetic, where s[i] is the i:th character of the string, n is the length of the string. The hash value of the empty string is zero.

For all other objects, by default as.integer() is called.

Value

Returns a [vector](#) of [integer](#)'s.

Author(s)

Henrik Bengtsson

InternalErrorException

InternalErrorException represents internal errors

Description

Package: R.oo
Class InternalErrorException

Object
~~|
~~+--try-error
~~~~~~~|
~~~~~~~+--condition
~~~~~~~~~~~~|
~~~~~~~~~~~~+--error
~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~+--simpleError
~~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~~~+--Exception
~~~~~~~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~~~~~~~+--InternalErrorException

Directly known subclasses:

public static class **InternalErrorException**
extends Exception

InternalErrorException represents internal errors that are likely to be due to implementation errors
done by the author of a specific package and not because the user made an error. Errors that are due
to unexpected input to functions etc falls under this error type.

Usage

```
InternalErrorException(..., package=NULL)
```

Arguments

...	Any arguments accepted by Exception.
package	The name (character string) of the package where the error exists. Can also be a Package object. If NULL, the source of the error is assumed to be unknown.

Fields and Methods

    Methods:

[getMessage](#)	Gets the message of the exception.
[getPackage](#)	Gets the suspicious package likely to contain an error.

    Methods inherited from Exception:
as.character, getCall, getCalls, getLastException, getMessage, getStackTrace, getWhen, print, printStack-Trace, throw

    Methods inherited from error:
as.character, throw

    Methods inherited from condition:
abort, as.character, conditionCall, conditionMessage, print

    Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

See Also

For detailed information about exceptions see [Exception](#).

---

intToChar	*Converts a vector of integers into a vector of ASCII characters*

Description

Converts a vector of ASCII integers to a equal length vector of ASCII characters. To make sure that all values in the input vector are in the range [0,255], the input vector is taken modulo 256.

Usage

```
## Default S3 method:
intToChar(i, ...)
```

Arguments

i	An [integer](#) [vector](#).
...	Not used.

Value

Returns a ASCII [integer](#) [vector](#).

Author(s)

Henrik Bengtsson

See Also

[utf8Conversion](#). [charToInt](#)()

Examples

```
i <- charToInt(unlist(strsplit("Hello world!", split=NULL)))
# Gives: 72 101 108 108 111  32 119 111 114 108 100  33
ch <- intToChar(c(72,101,108,108,111,32,119,111,114,108,100,33))
# Gives: "H" "e" "l" "l" "o" " " "w" "o" "r" "l" "d" "!"
```

---

ll	*Generates a list of informative properties of all members of an environment*

Description

Generates a list of informative properties of all members of an environment.

Usage

```
## Default S3 method:
ll(pattern=".*", ..., private=FALSE, properties=getOption("R.oo::ll/properties"),
  sortBy=NULL, envir=parent.frame())
```

Arguments

pattern	Regular expression pattern specifying which members to return. If ".*", all names are matched.
...	A named [vector](#) of format functionName=value, where functionName() will be called on each member found. If the result matches the value, the member is returned, otherwise not.
private	If [TRUE](#), also private members, i.e. members with a name starting with a . (period), will be listed, otherwise not.
properties	Names of properties to be returned. There must exist a [function](#) with the same name, because it will be called. This way one can extract any type of property by defining new methods.
sortBy	Name or index of column (property) to be sorted by. If [NULL](#), the objects are listed in the order they are found.
envir	An [environment](#), a search path index or a name of a package to be scanned.

Value

Returns a [data.frame](#) containing information about all the members.

Default properties returned

It is possible to set the default value of argument `properties` by setting option `"R.oo::ll/properties"`, e.g. `options("R.oo::ll/properties"=c("data.class", "dimension"))`. If this option is not set when the package is loaded, it is set to `c("data.class", "dimension", "objectSize")`.

Author(s)

Henrik Bengtsson

See Also

[ll.Object](#)().

Examples

```
 ## Not run:
  To list all objects in .GlobalEnv:
  > ll()
         member data.class dimension objectSize
  1               *tmp*     Person          1        428
  2   as.character.Person   function       NULL       1208
  3              country    character        1         44
  4         equals.Person   function       NULL       2324
  5              filename    character        1         84
  6               getAge    function       NULL        372
  7          getAge.Person   function       NULL        612
  8         getName.Person   function       NULL        628
  9        hashCode.Person   function       NULL       1196
  10         last.warning        list        1         192
  11                  obj      Person        1         428
  12               Person       Class       NULL       2292
  13               setAge    function       NULL        372
  14          setAge.Person   function       NULL       2088
  15              setName    function       NULL        372
  16         setName.Person   function       NULL        760
  17     staticCode.Person   function       NULL       2372

  To list all functions in the methods package:
  ll(mode="function", envir="methods")

  To list all numeric and character object in the base package:
  ll(mode=c("numeric", "character"), envir="base")

  To list all objects in the base package greater than 40kb:
  subset(ll(envir="base"), objectSize > 40000)

 ## End(Not run)
```

Object *The root class that every class must inherit from*

Description

R.oo
Class Object

public class **Object**

Object is the root class of all other classes. All classes *must* extends this class, directly or indirectly, which means that they all will inherit the methods in this class.

Usage

```
Object(core=NA)
```

Arguments

core   The core value of each *reference* referering to the Object. By default, this is just the smallest possible R object, but there are situations where it is useful to have another kind of core, which is the case with the Class class. *Note that this value belongs to the reference variable and not to the Object, which means it can not be referenced.*

Fields and Methods

Methods:

$	-
$<-	-
[[	-
[[<-	-
as.character	Gets a character string representing the object.
attach	Attaches an Object to the R search path.
attachLocally	Attaches an Object locally to an environment.
clearCache	Clear fields that are defined to have cached values.
clearLookupCache	Clear internal fields used for faster lookup.
clone	Clones an Object.
detach	Detach an Object from the R search path.
equals	Compares an object with another.
extend	Extends another class.
finalize	Finalizer methods called when object is clean out.
gc	Clear cached fields and calls the garbage collector.
getEnvironment	Gets the environment of this object.
getFieldModifier	-

getFieldModifiers	Gets all types of field modifiers.
getFields	Returns the field names of an Object.
getInstantiationTime	Gets the time when the object was instantiated.
getInternalAddress	Gets the memory location where the Object resides.
getStaticInstance	Gets the static instance of this objects class.
hasField	Checks if a field exists or not.
hashCode	Gets a hash code for the Object.
isReferable	Checks if the object is referable or not.
ll	Generates a list of informative properties of all members of an Object.
load	Static method to load an Object from a file or a connection.
names	-
newInstance	Creates a new instance of the same class as this object.
novirtual	Returns a reference to the same Object with virtual fields turned off.
objectSize	Gets the size of the Object in bytes.
print	Prints an Object.
registerFinalizer	Registers a finalizer hook for the object.
save	Saves an Object to a file or a connection.
staticCode	Method that will be call each time a new instance of a class is created.

Defining static fields

To define a static field of an Object class, use a private field <.field> and then create a virtual field <field> by defining methods get<Field>() and set<Field>(). These methods should retrieve and assign the value of the field <.field> of the *static* instance of the class. The second example below shows how to do this. The example modifies also the static field already in the constructor, which is something that otherwise may be tricky.

Author(s)

Henrik Bengtsson

References

[1] H. Bengtsson, *The R.oo package - Object-Oriented Programming with References Using Standard R Code*, In Kurt Hornik, Friedrich Leisch and Achim Zeileis, editors, Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/

Examples

```
##########################################################################
# Defines the class Person with private fields .name and .age, and
# with methods print(), getName(), setName(), getAge() and setAge().
##########################################################################
setConstructorS3("Person", function(name, age) {
  if (missing(name)) name <- NA;
  if (missing(age))  age <- NA;
```

```
  extend(Object(), "Person",
    .name=name,
    .age=age
  )
})


setMethodS3("as.character", "Person", function(this, ...) {
  paste(this$.name, "is", as.integer(this$.age), "years old.");
})

setMethodS3("equals", "Person", function(this, obj, ...) {
  ( identical(data.class(this), data.class(obj)) &&
    identical(this$getName(), obj$getName()) &&
    identical(this$getAge() , obj$getAge() )    );
})

setMethodS3("hashCode", "Person", function(this, ...) {
  # Get the hashCode() of the '.name' and the '.age' fields
  # using hashCode.default().
  hashCode(this$.name) * hashCode(this$.age);
})

setMethodS3("getName", "Person", function(this, ...) {
  this$.name;
})

setMethodS3("setName", "Person", function(this, newName, ...) {
  throw("It is not possible to change the name of a Person.");
})

setMethodS3("getAge", "Person", function(this, ...) {
  this$.age;
})

setMethodS3("setAge", "Person", function(this, newAge, ...) {
  if (!is.numeric(newAge))
    throw("Age must be numeric: ", newAge);
  if (newAge < 0)
    throw("Trying to set a negative age: ", newAge);
  this$.age <- newAge;
})




###########################################################################
# Code demonstrating different properties of the Object class using
# the example class Person.
###########################################################################

# Create an object (instance of) the class Person.
```

```
p1 <- Person("Dalai Lama", 67)

# 'p1' is an Object of class Person.
print(data.class(p1))  # "Person"

# Prints information about the Person object.
print(p1)              # "Dalai Lama is 67 years old."

# or equivalent (except that no generic method has to exist):

p1$print()             # "Dalai Lama is 67 years old."

# Shows that no generic method is required if the $ operator is used:
print(p1$getName())  # "Dalai Lama"

# The following will call p1$getName() since there exists a get-()
# method for the 'name' property.
print(p1$name)       # "Dalai Lama"

# and equivalent when using the [[ operator.
print(p1[["name"]])  # "Dalai Lama"

# The following shows that p1$setName(68) is called, simply because
# there exists a set-() method for the 'name' property.
p1$age <- 68         # Will call p1$setAge(68)

# Shows that the age of the Person has been updated:
print(p1)            # "Dalai Lama is 68 years old."

# If there would not exists such a set-() method or field a new
# field would be created:
p1$country <- "Tibet"

# Lists all (non-private) members of the Person object:
print(ll(p1))

# which gives
#      member class      mode     typeof length  dim bytes
#   1 country  NULL character character      1 NULL    44

# The following will call p1$setName("Lalai Dama") which will
# throw an exception saying one can not change the name of
# a Person.
tryCatch(p1$name <- "Lalai Dama", error=print)

# The following will call p1$setAge(-4) which will throw an
# exception saying that the age must be a non-negative number.
tryCatch(p1$age <- -100, error=print)

# Attaches Object 'p1' to the search path.
attach(p1)

# Accesses the newly created field 'country'.
```

```
    print(country)       # "Tibet"

    # Detaches Object 'p1' from the search path. Note that all
    # modifications to 'country' are lost.
    country <- "Sweden"
    detach(p1)
    print(p1$country)    # "Tibet"


    # Saves the Person object to a tempory file.
    filename <- tempfile("R.methodsS3.example")
    save(p1, filename)

    # Deletes the object
    rm(p1)

    # Loads an Object (of "unknown" class) from file using the
    # static method load() of class Object.
    obj <- Object$load(filename)

    # Prints information about the new Object.
    print(obj)

    # Lists all (non-private) members of the new Object.
    print(ll(obj))



    ########################################################################
    # Example illustrating how to "emulate" static fields using virtual
    # fields, i.e. get- and set-methods.  Here we use a private static
    # field '.count' of the static class instance 'MyClass', i.e.
    # MyClass$.count.  Then we define a virtual field 'count' via method
    # getCount() to access this static field.  This will make all queries
    # for 'count' of any object to use the static field instead.  In the
    # same way is assignment controlled via the setCount() method.  A
    # side effect of this way of coding is that all MyClass instances will
    # also have the private field '.count' (set to zero except for the
    # static field that is).
    ########################################################################
    setConstructorS3("MyClass", function(...) {
      # Create an instance (the static class instance included)
      this <- extend(Object(), "MyClass",
        .count = 0
      )

      # In order for a static field to be updated in the
      # constructor it has to be done after extend().
      this$count <- this$count + 1;

      # Return the object
      this;
    })
```

```
setMethodS3("as.character", "MyClass", function(this, ...) {
  paste(class(this)[1], ": Number of instances: ", this$count, sep="");
})


# Get virtual field 'count', e.g. obj$count.
setMethodS3("getCount", "MyClass", function(this, ...) {
  MyClass$.count;
})


# Set virtual field 'count', e.g. obj$count <- value.
setMethodS3("setCount", "MyClass", function(this, value, ...) {
  MyClass$.count <- value;
})


# Create four instances of class 'MyClass'
obj <- lapply(1:4, MyClass)
print(obj)
print(MyClass$count)
print(obj[[1]]$count)

stopifnot(obj[[1]]$count == length(obj))
stopifnot(MyClass$count == length(obj))
```

---

objectSize   *Gets the size of the object in bytes*

Description

Gets the size of the object in bytes. This method is just a wrapper for `object.size`.

Usage

```
## Default S3 method:
objectSize(...)
```

Arguments

```
...              Arguments passed to object.size.
```

Value

Returns an integer.

Author(s)

Henrik Bengtsson

See Also

Internally `object.size`.

---

objectSize.environment

Gets the size of an environment in bytes

Description

Gets the size of an environment in bytes.

Usage

```
## S3 method for class 'environment'
objectSize(envir, ...)
```

Arguments

envir	An `environment`().
...	Arguments passed to `ls`().

Value

Returns an `integer`.

Author(s)

Henrik Bengtsson

See Also

Internally `object.size` is used.

| Package | *The Package class provides methods for accessing package information* |

Description

Package: R.oo
Class Package

Object
~~|
~~+--Package

Directly known subclasses:

public class **Package**
extends Object

Creates a Package that can be thrown and caught. The Package class is the root class of all other
Package classes.

Usage

Package(name=NULL)

Arguments

name   Name of the package.

Fields and Methods

Methods:

as.character	Gets a string representation of this package.
getAuthor	Gets the Author of this package.
getBundle	Gets the Bundle that this package might belong to.
getBundlePackages	Gets the names of the other packages that is in the same bundle as this package.
getChangeLog	Gets the change log of this package.
getClasses	Gets all classes of a package.
getContents	Gets the contents of this package.
getContribUrl	Gets the URL(s) from where this package can be installed.
getDataPath	Gets the path to the data (data/) directory of this package.
getDate	Gets the date when package was build.
getDescription	Gets the description of the package.

getDescriptionFile	Gets the description file of this package.
getDevelUrl	Gets the URL(s) from where the developers version of this package can be installed.
getDocPath	Gets the path to the accompanying documentation (doc/) directory of this package.
getEnvironment	Gets the environment of a loaded package.
getExamplePath	Gets the path to the example (R-ex/) directory of this package.
getHistory	-
getHowToCite	Gets the howToCite of this package.
getLicense	Gets the License of this package.
getMaintainer	Gets the Maintainer of this package.
getName	Gets the name of this package.
getNews	-
getPath	Gets the library (system) path to this package.
getPosition	Gets the search path position of the package.
getTitle	Gets the Title of this package.
getUrl	Gets the URL of this package.
getVersion	Gets the version of this package.
isLoaded	Checks if the package is installed on the search path or not.
isOlderThan	Checks if the package is older than a given version.
ll	Generates a list of informative properties of all members of the package.
load	Loads a package.
showChangeLog	Show the change log of this package.
showContents	Show the CONTENTS file of this package.
showDescriptionFile	Show the DESCRIPTION file of this package.
showHistory	-
showHowToCite	Show the HOWTOCITE file of this package.
showNews	-
startupMessage	Generates a 'package successfully loaded' package startup message.
unload	Unloads a package.
update	Updates the package is a newer version is available.

Methods inherited from Object:

$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

Examples

```
## Not run: # By defining .onAttach() as follows in zzz.R for a package, an
# instance of class Package with the same name as the package will
# be made available on the search path. More over, the code below
# will also inform the user that the package has been loaded:
#
#  > library(R.oo)
#  R.oo v0.52 (2003/04/13) was successfully loaded.
```

```
#
.onAttach <- function(libname, pkgname) {
  pkg <- Package(pkgname);
  assign(pkgname, pkg, pos=getPosition(pkg));
  cat(getName(pkg), " v", getVersion(pkg), " (", getDate(pkg), ")",
    " was successfully loaded.\n", sep="");
}

# The Package class works for any packages, loaded or not.

# Some information about the base package
pkg <- Package("base")
print(pkg)
# [1] "Package: base v1.6.2 (NA) is loaded (pos=5). The official webpage
#       is NA and the maintainer is R Core Team <R-core@r-project.org>. The
#       package is installed in c:/PROGRA~1/R/rw1062/library/base/."
print(list.files(Package("base")$dataPath))

# Some information about the R.oo package
print(R.oo::R.oo)
# [1] "Package: R.oo v0.52 (2003/04/13) is loaded (pos=2). The official
#       webpage is http://www.braju.com/R/ and the maintainer is Henrik
#       Bengtsson <henrikb@braju.com>. The package is installed in
#       c:/PROGRA~1/R/rw1062/library/R.oo/."

## End(Not run)
```

RccViolationException   *An RccViolationException indicates a violation of the R Coding Con-
                        ventions (RCC)*

Description

Package: R.oo
Class RccViolationException

```
Object
~~|
~~+--try-error
~~~~~~~|
~~~~~~~+--condition
~~~~~~~~~~~~|
~~~~~~~~~~~~+--error
~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~+--simpleError
~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~+--Exception
~~~~~~~~~~~~~~~~~~~~~~~~~~|
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~+--RccViolationException
```

Directly known subclasses:

public static class **RccViolationException**
extends [Exception]

An RccViolationException indicates a violation of the R Coding Conventions (RCC). It is generated
by setConstructorS3() and setMethodS3(). It is *not* meant to be caught, but instead the source
code that violates the RCC should be fixed. For more information about RCC, see references below.

Usage

```
RccViolationException(...)
```

Arguments

...	Any arguments accepted by the constructor of Exception, i.e. one or several [character] strings, which will be concatenated and contain informative message about why the RCC was violated.

Details

Since it is not possible to assert that the RCC is followed during the parsing of the source code, but
first only when the source code is actually executed.

Fields and Methods

Methods:

[as.character]	Gets a string representing of the RCC violation.
[getRccUrl]	Static method to get a URL where the RCC can be found.

Methods inherited from Exception:
as.character, getCall, getCalls, getLastException, getMessage, getStackTrace, getWhen, print, printStack-
Trace, throw

Methods inherited from error:
as.character, throw

Methods inherited from condition:
abort, as.character, conditionCall, conditionMessage, print

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach,
equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstan-
tiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

See Also

See also [try](#)() and [tryCatch](#)(). For detailed information about exceptions see [Exception](#). The R Coding Conventions (RCC) can be found at http://aroma-project.org/developers/RCC/.

Examples

```
## Not run:
 setConstructorS3("myClass", function() { extends(Object(), .value=0) })
 setMethodS3("MyMethod", "myClass", function(this) { "Hullo!" })

## End(Not run)
```

Rdoc   *Class for converting Rdoc comments to Rd files*

Description

Package: R.oo
Class Rdoc

[Object](#)
~~|
~~+--Rdoc

Directly known subclasses:

public static class **Rdoc**
extends [Object](#)

Class for converting Rdoc comments to Rd files.

Usage

Rdoc()

Fields and Methods

    Methods:

argsToString	Gets the arguments signature of a function.
check	Checks the compiled Rd files.
compile	Compile source code files containing Rdoc comments into Rd files.
createManPath	Creates the directory where the Rd files should be saved.
createName	Creates a class-method name.
declaration	Gets the class declaration.
escapeRdFilename	Escape non-valid characters in a filename.
getClassS4Usage	Gets the usage of a S4 class.
getKeywords	Gets the keywords defined in R with descriptions.
getManPath	Gets the path to the directory where the Rd files will be saved.
getNameFormat	Gets the current name format.
getObject	-
getPackageNameOf	Gets the package of a method or an object.
getRdTitle	Extracts the title string of a Rd file.
getUsage	Gets the usage of a method.
hierarchy	Gets the class hierarchy.
isKeyword	Checks if a word is a Rd keyword.
isVisible	Checks if a member is visible given its modifiers.
methodsInheritedFrom	Gets all methods inherited from a class in Rd format.
setManPath	Sets the path to the directory where the Rd files should be saved.
setNameFormat	Sets the current name format.

    Methods inherited from Object:

$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

References

R developers, *Guidelines for Rd files*, <http://developer.r-project.org/Rds.html>, 2003

Examples

```
## Not run: # Set default author
author <- "Henrik Bengtsson, \url{http://www.braju.com/R/}"

# Show the file containing the Rdoc comments
rdocFile <- system.file("misc", "ASCII.R", package="R.oo")
file.show(rdocFile)
```

```
# Compile the Rdoc:s into Rd files (saved in the destPath directory)
destPath <- tempdir()
Rdoc$compile(rdocFile, destPath=destPath)

# List the generated Rd files
rdFiles <- list.files(destPath, full.names=TRUE)
print(rdFiles)

# Show one of the files
file.show(rdFiles[1])

# Clean up
file.remove(rdFiles)

## End(Not run)
```

RdocException   *RdocException are thrown by the Rdoc compiler*

Description

Package: R.oo
Class RdocException

[Object](#)
~~|
~~+--try-error
~~~~~~~|
~~~~~~~+--condition
~~~~~~~~~~~~|
~~~~~~~~~~~~+--error
~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~+--simpleError
~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~+--[Exception](#)
~~~~~~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~~~~~~+--RdocException

Directly known subclasses:

public static class **RdocException**
extends [Exception](#)

RdocException are thrown by the Rdoc compiler when it fails to generate a Rd file from an Rdoc
comment.

Usage

```
RdocException(..., source=NULL)
```

Arguments

...	Any arguments accepted by [Exception]().
source	Object specifying the source where the Rdoc error occured. This is commonly a filename [character]() string..

Fields and Methods

Methods:

[as.character]()	Gets a character string representing of the RdocException.
[getSource]()	Gets the source of the exception.

Methods inherited from Exception:
as.character, getCall, getCalls, getLastException, getMessage, getStackTrace, getWhen, print, printStack-Trace, throw

Methods inherited from error:
as.character, throw

Methods inherited from condition:
abort, as.character, conditionCall, conditionMessage, print

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstan-tiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, save

Author(s)

Henrik Bengtsson

See Also

For detailed information about exceptions see [Exception]().

---

setConstructorS3	*Defines a class in S3/UseMethod style*

Description

Defines a class in R.oo/S3 style. What this function currently does is simply creating a constructor function for the class.

Usage

```
## Default S3 method:
setConstructorS3(name, definition, private=FALSE, protected=FALSE, export=TRUE,
  static=FALSE, abstract=FALSE, trial=FALSE, deprecated=FALSE, envir=parent.frame(),
  enforceRCC=TRUE, ...)
```

Arguments

name	The name of the class.
definition	The constructor defintion. *Note: The constructor must be able to be called with no arguments, i.e. use default values for all arguments or make sure you use* missing() *or similar!*
static	If TRUE this class is defined to be static, otherwise not. Currently this has no effect expect as an indicator.
abstract	If TRUE this class is defined to be abstract, otherwise not. Currently this has no effect expect as an indicator.
private	If TRUE this class is defined to be private.
protected	If TRUE this class is defined to be protected.
export	A logical setting attribute "export".
trial	If TRUE this class is defined to be a trial class, otherwise not. A trial class is a class that is introduced to be tried out and it might be modified, replaced or even removed in a future release. Some people prefer to call trial versions, beta version. Currently this has no effect expect as an indicator.
deprecated	If TRUE this class is defined to be deprecated, otherwise not. Currently this has no effect expect as an indicator.
envir	The environment for where the class (constructor function) should be stored.
enforceRCC	If TRUE, only class names following the R Coding Convention is accepted. If the RCC is violated an RccViolationException is thrown.
...	Not used.
	Note: If a constructor is not declared to be private nor protected, it will be declared to be public.

A constructor must be callable without arguments

The requirement that a constructor function should be callable without arguments (e.g. MyConstructor()) is because that call is used to create the static instance of a class. The reason for this is that a static instance of the class is created automatically when the constructor is called *the first time* (only), that is, when the first of object of that class is created. All classes have to have a static instance.

To make a constructor callable without arguments, one can either make sure all arguments have default values or one can test for missing arguments using missing(). For instance the following defintion is *not* correct: setConstructorS3("Foo", function(x) extend(Object(), "Foo", x=x)) whereas this one is setConstructorS3("Foo", function(x=NA) extend(Object(), "Foo", x=x))

Code validation

If argument enforceRCC is [TRUE](), the class name is validated so it starts with a letter and it also gives a [warning]() if its first letter is *not* captial. The reason for this is to enforce a naming convention that names classes with upper-case initial letters and methods with lower-case initial letters (this is also the case in for instance Java).

Author(s)

Henrik Bengtsson

See Also

To define a method see [setMethodS3](). For information about the R Coding Conventions, see [RccViolationException](). For a thorough example of how to use this method see [Object]().

Examples

```
## Not run: For a complete example see help(Object).
```

---

throw	*Throws an Exception*

Description

Throws an exception similar to stop(), but with support for [Exception]() classes. The first argument (object) is by default pasted together with other arguments (...) and with seperator sep="". For instance, to throw an exception, write

throw("Value out of range: ", value, ".").

which is short for

throw(Exception("Value out of range: ", value, ".")).

Note that throw() can be defined for classes inheriting [Exception](), which can then be caught (or not) using [tryCatch]().

Usage

```
## Default S3 method:
throw(...)
```

Arguments

...	One or several strings that are concatenated and collapsed into on message string.

Value

Returns nothing.

Author(s)

Henrik Bengtsson

See Also

See the [Exception](#) class for more detailed information.

Examples

```
rbern <- function(n=1, prob=1/2) {
  if (prob < 0 || prob > 1)
    throw("Argument 'prob' is out of range: ", prob)
  rbinom(n=n, size=1, prob=prob)
}

rbern(10, 0.4)
# [1] 0 1 0 0 0 1 0 0 1 0
tryCatch(rbern(10, 10*0.4),
  error=function(ex) {}
)
```

---

throw.error   *Throws (rethrows) an object of class 'error'*

Description

Rethrows an 'error' object. The 'error' class was introduced in R v1.8.1 with the new error handling mechanisms.

Usage

```
## S3 method for class 'error'
throw(error, ...)
```

Arguments

error   An object or class 'error'.
...   Not used.

Value

Returns nothing.

Author(s)

Henrik Bengtsson

See Also

See the tryCatch() method etc. See the [Exception](#) class for more detailed information.

trim   *Converts to a string and removes leading and trailing whitespace*

Description

Converts to a string and removes leading and trailing whitespace.

Usage

```
## Default S3 method:
trim(object, ...)
```

Arguments

object   A [vector](#) of R objects to be trimmed.

...   Not used.

Value

Returns a [vector](#) of [character](#) strings.

Author(s)

Henrik Bengtsson

typeOfClass   *Gets the type of a class (S3 or S4)*

Description

Gets the type of a class (S3 or S4).

Usage

```
## Default S3 method:
typeOfClass(object, ...)
```

Arguments

object   The object to be checks.

...   Not used.

Value

Returns a [character](#) string ″S3″, ″S3-Object″ or ″S4″, or [NA](#) if neither.

Author(s)

Henrik Bengtsson

Index