Package: restorepoint (via r-universe)

August 24, 2024

Type Package

Title Debugging with Restore Points

Version 0.2

Date 2018-12-20

URL https://github.com/skranz/restorepoint

Author Sebastian Kranz [aut, cre], Roman Zenka [ctb]

Maintainer Roman Zenka <zenka.roman@mayo.edu>

Description Debugging with restore points instead of break points. A restore point stores all local variables when called inside a function. The stored values can later be retrieved and evaluated in a modified R console that replicates the function's environment. To debug step by step, one can simply copy & paste the function body from the R script. Particularly convenient in combination with ``RStudio". See the ``Github" page inst/vignettes for a tutorial.

License GPL (>= 2)

Collate 'restorepoint.R'

Suggests testthat, knitr

VignetteBuilder knitr

RoxygenNote 6.0.1

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

RemoteUrl https://github.com/skranz/restorepoint

RemoteRef master

RemoteSha 586ff0b271ca87d6b2e3c53ea93beac4fdf2d541

Contents

`add.restore.point.test` *Add one or several test functions*

Description

A test function is called after a restore point has stored data. It must have an argument env and name. It can check whether certain conditions are satisfied by the variables

Usage

```
add.restore.point.test(...)
```

Arguments

...	a slist of test functions that will be called with the stored arguments

assert *Checks whether cond holds true if not throws an error*

Description

Can be used for checking for errors in functions

Usage

```
assert(cond)
```

Arguments

cond	a condition that is checked

break.point *Sets a break point that can be debugged like a restore point*

Description

This function can be used as an alternative to browser(). When called inside a function, break.point stores all local objects and then does the following. i) If to global=FALSE (the default for break.point) starts the restore.point.browser for the local objects. ii) if to.global=TRUE copies the local objects to the global environment and stops execution.

Usage

```
break.point(name = "BREAK_POINT___",
  to.global = get.restore.point.options()$break.point.to.global,
  deep.copy = get.restore.point.options()$deep.copy, force = FALSE,
  dots = eval(substitute(list(...), env = parent.frame())))
```

Arguments

name	key under which the objects are stored. For restore points at the beginning of a function, I would suggest the name of that function.
to.global	if TRUE (default) objects are restored by simply copying them into the global environment. If FALSE a new environment will be created and the restore point browser will be invoked.
deep.copy	if TRUE try to make deep copies of objects that are by default copied by reference. Works so far for environments (recursivly). The function will search lists whether they contain reference objects, but for reasons of speed not yet in other containers. E.g. if an evironment is stored in a data.frame, only a shallow copy will be made. Setting deep.copy = FALSE (DEFAULT) may be useful if storing takes very long and variables that are copied by reference are not used or not modified.

force	store even if set.storing(FALSE) has been called
dots	by default a list of the ... argument of the function in whicht restore.point was called

Details

An alternative to break points are restore points. In the tutorial on GitHub, I provide some arguments how restore points can facilitate debugging compared to break points.

---

calls.to.trace — *Transforms a list returned by sys.calls into a vector of strings that looks like a result of traceback()*

Description

Transforms a list returned by sys.calls into a vector of strings that looks like a result of traceback()

Usage

```
calls.to.trace(calls = sys.calls(), max.lines = 4)
```

Arguments

calls	a list of calls, e.g. returned by sys.calls
max.lines	as in traceback()

Value

a character vector with one element for each call formated in a similar fashion as traceback() does

---

can.parse.multi.line — *Checks whether for the installed R version the function env.console is able to correctly parse R expressions that extend over more than a line*

Description

The current implementation of env.console is quite dirty in so far that it parses an error message of the parse() function to check whether a given R expression is assumed to be continued in the next line. That process may not work in R distributions that have error messages that are not in English. The function can.parse.multi.line() tries to check whether that process works or not @export

Usage

```
can.parse.multi.line()
```

clone.environment *Deep copy of an environment*

Description

Deep copy of an environment

Usage

```
clone.environment(env, use.copied.ref = FALSE, all.names = TRUE)
```

Arguments

env	the environment to be cloned
use.copied.ref	internal
all.names	passed to eapply

copy.into.env *Copies all members of a list or environment into an environment*

Description

Copies all members of a list or environment into an environment

Usage

```
copy.into.env(source = sys.frame(sys.parent(1)),
  dest = sys.frame(sys.parent(1)), names = NULL, exclude = NULL,
  from.restore.objects = FALSE, overwrite = TRUE, all.names = TRUE)
```

Arguments

source	a list or environment from which objects are copied
dest	the environment into which objects are copied
names	optionally a vector of names that shall be copied. If null all objects are copied
exclude	optionally a vector of names that shall not be copied
from.restore.objects	internal paramater keep FALSE
overwrite	should existing objects in dest with same name be overwritten?
all.names	if TRUE copy all objects if names=NULL, if FALSE omit variables starting with .

`default.error.string.fun` *A default error string function for eval with error trace*

Description

A default error string function for eval with error trace

Usage

```
default.error.string.fun(e, tb)
```

Arguments

`e`	the error object
`tb`	a character vector of the traceback

`disable.restore.points` *Globally disable or enable restore points*

Description

Globally disable or enable restore points

Usage

```
disable.restore.points(disable = TRUE)
```

Arguments

`disable`	if TRUE globaly disable restore points. This speeds up calls to restore.point quickly. Is faster than set.storing(FALSE), but has no informative messages when restore.point is called from the global env.

`env.console` *Emulates an R console that evaluates expressions in the specified environement env. You return to the standard R console by pressing ESC*

Description

Emulates an R console that evaluates expressions in the specified environement env. You return to the standard R console by pressing ESC

Usage

```
env.console(env = new.env(parent = parent.env), parent.env = parent.frame(),
  dots = NULL, prompt = ": ",
  startup.message = "Press ESC to return to standard R console",
  multi.line.parse.error = get.restore.point.options()$multi.line.parse.error,
  local.variables = NULL)
```

Arguments

`env`	The environment in which expressions shall be evaluated. If not specified then a new environment with the given parent.env is created.
`parent.env`	If env is not specified the parent environemnt in which the new environment shall be created
`dots`	a list that contains values for the ellipsies ... that will be used if you call other functions like fun(...) from within the console. You can access the values inside the console by typing list(...)
`prompt`	The prompt that shall be shown in the emulated console. Default = ": "
`startup.message`	The text that is shown when env.console is started
`multi.line.parse.error`	A substring used to identify an error by parse that is due to parsing the beginning of a multi-line expression. The substring can depend on the language of R error messages. The packages tries to find a correct substring automatically as default.
`local.variables`	additional variables that shall be locally available

Value

Returns nothing since the function must be stopped by pressing ESC.

`eval.with.error.trace` *Evals the expression such that if an error is encountered a traceback is added to the error message.*

Description

This function is mostly useful within a tryCatch clause Adapted from code in tools:::.try_quietly as suggested by Kurt Hornik in the following message https://stat.ethz.ch/pipermail/r-devel/2005-September/034546.html

Usage

```
eval.with.error.trace(expr, max.lines = 4, remove.early.calls = 0,
  error.string.fun = default.error.string.fun)
```

Arguments

`expr`	the expression to be evaluated
`max.lines`	as in traceback()
`remove.early.calls`	an integer specifying a number of calls that won't be shown in the trace.
`error.string.fun`	a function(e,tb) that takes as arguments an error e and a string vector tb of the stack trace resulting from a call to calls.to.trace() and returns a string with the extended error message

Value

If no error occurs the value of expr, otherwise an error is thrown with an error message that contains the stack trace of the error.

`get.restore.point.options` *Get global options for restore points*

Description

Get global options for restore points

Usage

```
get.restore.point.options()
```

`get.stored.dots` *Returns the ellipsis (...) that has been stored in restore.point name as a list*

Description

Returns the ellipsis (...) that has been stored in restore.point name as a list

Usage

```
get.stored.dots(name, deep.copy = FALSE)
```

Arguments

`name`	the name whith which restore.point or store.objects has been called.
`deep.copy`	shall a deep copy of stored objects be made

`get.stored.object.list` *Retrieves the list of all restore.points with the stored objects*

Description

Retrieves the list of all restore.points with the stored objects

Usage

```
get.stored.object.list()
```

`is.storing` *Check whether objects currently are stored or not*

Description

Check whether objects currently are stored or not

Usage

```
is.storing()
```

`restore.objects` *Restore stored objects by copying them into the specified environment. Is used by restore.point*

Description

Restore stored objects by copying them into the specified environment. Is used by restore.point

Usage

```
restore.objects(name, dest = globalenv(), was.forced = FALSE,
  deep.copy = get.restore.point.options()$deep.copy)
```

Arguments

`name`	name under which the variables have been stored
`dest`	environment into which the stored variables shall be copied. By default the global environment.
`was.forced`	flag whether storage of objects was forced. If FALSE (default) a warning is shown if restore.objects is called and is.storing()==FALSE, since probably no objects have been stored.
`deep.copy`	when storing or restoring tries to make a deep copy of R objects that are by default copied by reference, like environments. Setting deep.copy = FALSE can substantially speed up restore.point, however.

Value

returns nothing but automatically copies the stored variables into the global environment

`restore.point` *Sets a restore point*

Description

The function behaves different when called from a function or when called from the global environemnt. When called from a function, it makes a backup copy of all local objects and stores them internally under a key specified by name. When called from the global environment, it restores the previously stored objects by copying them into the global environment. See the package Vignette for an illustration of how this function can facilitate debugging.

Usage

```
restore.point(name, to.global = options$to.global,
  deep.copy = options$deep.copy, force = FALSE,
  display.restore.point = options$display.restore.point,
  indent.level = TRUE, trace.calls = options$trace.calls,
  max.trace.lines = 10, dots = eval(substitute(list(...), env =
  parent.frame())), options = get.restore.point.options())
```

Arguments

name	key under which the objects are stored. For restore points at the beginning of a function, I would suggest the name of that function.
to.global	if TRUE (default) objects are restored by simply copying them into the global environment. If FALSE a new environment will be created and the restore point browser will be invoked.
deep.copy	if TRUE try to make deep copies of objects that are by default copied by reference. Works so far for environments (recursivly). The function will search lists whether they contain reference objects, but for reasons of speed not yet in other containers. E.g. if an evironment is stored in a data.frame, only a shallow copy will be made. Setting deep.copy = FALSE (DEFAULT) may be useful if storing takes very long and variables that are copied by reference are not used or not modified.
force	store even if set.storing(FALSE) has been called
display.restore.point	shall a text be shown in the console if restore.point is called. Can be useful when informative tracebacks are not readily availbale, e.g. when debugging shiny apps.
indent.level	when display.restore.point=TRUE shall level of nestedness be illustrated by identation
trace.calls	when objects are restored, shall a traceback be shown
max.trace.lines	if trace.calls=TRUE how many lines shall be shown at most in the traceback.
dots	by default a list of the ... argument of the function in which restore.point was called
options	option list to fill the parameter defaults from

---

restore.point.browser *Examing a previously stored restore point by invoking the browser.*

Description

The function is mainly for internal use by restore.point.

Usage

```
restore.point.browser(name, was.forced = FALSE,
  message.text = paste("restore point", name, ", press ESC to return."),
  deep.copy = get.restore.point.options()$deep.copy)
```

Arguments

name	name under which the variables have been stored
was.forced	flag whether storage of objects was forced. If FALSE (default) a warning is shown if restore.objects is called and is.storing()==FALSE, since probably no objects have been stored.
message.text	initial shown message
deep.copy	when storing or restoring tries to make a deep copy of R objects that are by default copied by reference, like environments. Setting deep.copy = FALSE can substantially speed up restore.point, however.

Value

returns nothing

---

restore.point.options *Set global options for restore points*

Description

Set global options for restore points

Usage

```
restore.point.options(options = NULL, display.restore.point = FALSE, ...)
```

Arguments

options	a list of options that shall be set. Possible options are listed below
display.restore.point	Makes sure that the display.restore.point option is set to FALSE by default
...	options can also directly be passed. The following options can be set: - storing Default=TRUE enable or disable storing of options, setting storing = FALSE basicially turns off debugging via restore points - deep.copy Default = FALSE. If TRUE then when storing and restoring tries to make a deep copy of R objects that are by default copied by reference, like environments. deep.copy = FALSE substantially speeds up restore.point. - to.global Default=TRUE. If TRUE then when options are restored, they are simply copied into the global environment and the R console is directly used for debugging. If FALSE a browser mode will be started instead. It is still possible to parse all R commands into the browser and to use copy and paste. To quit the browser press ESC in the R

console. The advantage of the browser is that all objects are stored in a newly generated environment that mimics the environemnt of the original function, i.e. global varariables are not overwritten. Furthermore in the browser mode, one can pass the ... object to other functions, while this does not work in the global environment. The drawback is that the browser is still not as convenient as the normal R console, e.g. pressing arrow up does not restore the previous command. Also, one has to press Esc to leave the browser mode.

`set.storing` *Set whether objects shall be stored or not*

Description

Set whether objects shall be stored or not

Usage

```
set.storing(storing = TRUE)
```

Arguments

`storing`	if FALSE don't store objects if restore.point or store.objects is called. May save time. If TRUE (default) turn on storage again.

`store.objects` *Stores all local objects of the calling environment to be able to restore them later when debugging. Is used by restore.point*

Description

Stores all local objects of the calling environment to be able to restore them later when debugging. Is used by restore.point

Usage

```
store.objects(name = NULL, parent.num = -1,
  deep.copy = get.restore.point.options()$deep.copy, force = FALSE,
  store.if.called.from.global = FALSE, envir = sys.frame(parent.num),
  store.parent.env = "all.but.global", dots = eval(substitute(list(...), env
  = parent.frame())))
```

Arguments

name	key under which the objects are stored, typical the name of the calling function. If name is NULL by default the name of the calling function is chosen
parent.num	can be used to specify envir=sys.frame(parent.num)
deep.copy	if TRUE (default) variables that are copied by reference (in the moment environments) will be stored as deep copy. May take long for large variables but ensures that the value of the stored variable do not change
force	store even if do.store(FALSE) has been called
store.if.called.from.global	if the function is called from the global environment and store.if.called.from.global FALSE (default) does not store objects when called from the global environment but does nothing instead.
envir	the environment from which objects shall be stored. By default the local environemnt of the calling function.
store.parent.env	shall objects from enclosing environments of envir also be stored? So far this happens for all enclosing environments except for the global environment or baseenv.
dots	by default a list of the ... argument of the function in whicht restore.point was called

Value

returns nothing, just called for side effects

Index