

# Package ‘setRNG’

January 2, 2012

**Title** Set (Normal) Random Number Generator and Seed

**Description** SetRNG provides utilities to help set and record the setting of the seed and the uniform and normal generators used when a random experiment is run. The utilities can be used in other functions that do random experiments to simplify recording and/or setting all the necessary information for reproducibility. See the vignette and reference manual for examples.

**Depends** R (>= 2.5.0)

**Version** 2009.11-1

**Date** 2009-11-1

**LazyLoad** yes

**License** GPL-2 | file LICENSE

**Author** Paul Gilbert <pgilbert@bank-banque-canada.ca>

**Maintainer** Paul Gilbert <pgilbert@bank-banque-canada.ca>

**URL** <http://www.bank-banque-canada.ca/pgilbert>

**Repository** CRAN

**Repository/R-Forge/Project** distr

**Repository/R-Forge/Revision** 634

**Date/Publication** 2009-11-27 07:48:03

## R topics documented:

setRNG-package . . . . .	2
00.setRNG.Intro . . . . .	3
getRNG . . . . .	3
random.number.test . . . . .	4
setRNG . . . . .	5

<b>Index</b>	<b>6</b>
--------------	----------

---

 setRNG-package

 setRNG
 

---

## Description

Programs to set random number generator (and seed) in R and S.

## Usage

```
library("setRNG")
```

## Introduction

This library provides tools to simplify recording and resetting the random number generator, to help make monte carlo experiments easily reproducible. It uses the R/S tools for setting the seed, but also records and sets the mechanism for converting uniform numbers to normally distributed numbers. (It could be extended to other transformations, but I have not done that.)

The setRNG function would typically be called by simulation programs (see example) to set the RNG information if given, and record the RNG information in all cases. This information can be returned with the result of the simulation. That way the simulation can always be reproduced if necessary.

The library also implements an approach to random number generation which allows the same random experiments to be replicated in S and R. The functions in the S/ directory allow the R results using Wichmann-Hill and Box-Muller to be replicated in S. These were done with the aid of an example from B. D. Ripley. (The files in the S/ directory of the package are for use with S not R.) These functions are intended primarily as a way to confirm that simulations and estimations with simulated data work in the same way in both S and R, not as an improved RNG. (It has only been tested in Splus 3.3) Default and other RNGs can still be used and are preferred for both speed and theoretical reasons.

## Examples

```
setRNG(kind="Wichmann-Hill", seed=c(979,1479,1542), normal.kind="Box-Muller")
rnorm(10)
```

```
sim <-function(rng=NULL)
{if(!require("setRNG")) stop("This function requires the setRNG package.")
  if(is.null(rng)) rng <- setRNG() # returns setting so don't skip if NULL
  else          {old.rng <- setRNG(rng); on.exit(setRNG(old.rng)) }
  x <- list(numbers=rnorm(10))
  x$rng <- rng
  x
}
```

```
z <- sim()
sim()$numbers
sim(rng=getRNG(z))$numbers
z$numbers
```

---

00.setRNG.Intro	<i>setRNG</i>
-----------------	---------------

---

**Description**

Programs to set random number generator (and seed) in R and S.

**Details**

See [setRNG-package](#) ( in the help system use `package?setRNG` or `?setRNG-package`) for an overview.

---

getRNG	<i>get the RND and seed from an object</i>
--------	--

---

**Description**

Get the random number generator and seed used to generate an object.

**Usage**

```
getRNG(e=NULL)
## Default S3 method:
getRNG(e=NULL)
```

**Arguments**

`e` an object generated by simulation (which stored the RNG information).

**Details**

Extract the RNG information used to generate the object. If `e` is `NULL` then `getRNG` returns the RNG setting, as returned by `setRNG()`. Otherwise, the default method assumes the object is a list and the RNG information is in the element `rng` or `noise$rng`.

**Value**

The random seed and other random number generation information used to generate the object.

**See Also**

[setRNG](#), [.Random.seed](#)

**Examples**

```
## Not run:
if (require("dse")) {
  data("eg1.DSE.data.diff", package="dse")
  model <- estVARXls(eg1.DSE.data.diff)
  sim <- simulate(model)
  getRNG(sim)
}

## End(Not run)
```

---

random.number.test      *Test the random number generator*

---

**Description**

Test the random number generator.

**Usage**

```
random.number.test()
```

**Arguments**

None

**Details**

This function checks that the RNG is working properly and has not been changed. If the RNG does not return values as in previous versions of R then the function executes `stop()`. Since changes to the RNG will cause comparisons of simulation results to fail, this is a useful check before investigating more complicated problems that may be a result of other "improvements" in your simulation or estimation programs.

**Value**

logical

**Side Effects**

Executes `stop()` if the tests fail.

**See Also**

[set.seed](#) [RNGkind](#) [runif](#) [rnorm](#) [setRNG](#)

**Examples**

```
random.number.test()
```

---

`setRNG`*Set the Random Number Generator*

---

**Description**

Set the RNG or return information about the setting of the RNG.

**Usage**

```
setRNG(kind=NULL, seed=NULL, normal.kind=NULL)
```

**Arguments**

	None required
	a character string.
<code>seed</code>	a vector of numbers (depending on kind).
<code>normal.kind</code>	a character string.

**Details**

Sets the uniform and normal random number generators and the seed. The old setting is returned using `invisible()` in a format which can be used in another call to `setRNG`. (This would reset to the original value.) If no arguments are given the current setting is returned, not using `invisible()`. In R see `RNGkind` for more details.

Note that in a function using `setRNG` it is good practice to assign the old setting to a variable, then reset to the old value on exiting the function (using on `.exit`). This avoids the possibility that overall RNG behaviour in a session, other than within your function, may be disrupted by your function.

**Value**

The old setting.

**Side Effects**

Sets global variables controlling the uniform and normal random number generators and the global seed.

**See Also**

[RNGkind](#), [set.seed](#), [runif](#), [rnorm](#), [random.number.test](#)

**Examples**

```
setRNG(kind="Wichmann-Hill", seed=c(979,1479,1542), normal.kind="Box-Muller")
rnorm(10)
```

# Index

- \*Topic **distribution**
  - setRNG-package, [2](#)
- \*Topic **interface**
  - setRNG-package, [2](#)
- \*Topic **package**
  - 00.setRNG.Intro, [3](#)
- \*Topic **programming**
  - getRNG, [3](#)
  - random.number.test, [4](#)
  - setRNG, [5](#)
  - setRNG-package, [2](#)
- \*Topic **ts**
  - getRNG, [3](#)
- \*Topic **utilities**
  - getRNG, [3](#)
  - random.number.test, [4](#)
  - setRNG, [5](#)
  - setRNG-package, [2](#)
- .Random.seed, [3](#)
- 00.setRNG.Intro, [3](#)
  
- getRNG, [3](#)
  
- random.number.test, [4](#), [5](#)
- RNGkind, [4](#), [5](#)
- rnorm, [4](#), [5](#)
- runif, [4](#), [5](#)
  
- set.seed, [4](#), [5](#)
- setRNG, [3](#), [4](#), [5](#)
- setRNG-package, [3](#)
- setRNG-package, [2](#)
- setRNG.Intro (setRNG-package), [2](#)