Package ‘rTRNG’

February 24, 2026

Title Advanced and Parallel Random Number Generation via 'TRNG'

Version 4.23.1-5

Description Embeds sources and headers from Tina's Random Number Generator ('TRNG') C++ library. Exposes some functionality for easier access, testing and benchmarking into R. Provides examples of how to use parallel RNG with 'RcppParallel'. The methods and techniques behind 'TRNG' are illustrated in the package vignettes and examples. Full documentation is available in Bauke (2021) <https://github.com/rabauke/trng4/blob/v4.23.1/doc/trng.pdf>.

License GPL-3

URL https://github.com/miraisolutions/rTRNG#readme, https://mirai-solutions.ch

BugReports https://github.com/miraisolutions/rTRNG/issues

SystemRequirements GNU make, C++17

Imports methods, Rcpp (>= 0.11.6), RcppParallel

Suggests covr, knitr, R.rsp, rmarkdown, testthat (>= 2.0.0)

LinkingTo Rcpp, RcppParallel

VignetteBuilder knitr, R.rsp

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.3.2

Collate 'LdFlags.R' 'RcppExports.R' 'TRNG.Engine.R' 'TRNG.Random.R' 'TRNGkind.R' 'TRNGseed.R' 'TRNGjump.R' 'TRNGsplit.R' 'TRNG.Random.seed.R' 'TRNG.Version.R' 'currentEngine.R' 'defaultKind.R' 'inline.R' 'rTRNG-package.R' 'rbinom_trng.R' 'rlnorm_trng.R' 'rnorm_trng.R' 'rpois_trng.R' 'runif_trng.R' 'zzz.R'

Author Riccardo Porreca [aut, cre],
Roland Schmid [aut],
Mirai Solutions GmbH [cph],
Heiko Bauke [ctb, cph] (TRNG sources and headers)

Maintainer Riccardo Porreca <riccardo.porreca@mirai-solutions.com>

Repository CRAN

Date/Publication 2026-02-24 22:50:02 UTC

Contents

`rTRNG-package` *TRNG C++ library functionality exposed to R.*

Description

Tina's Random Number Generator Library (**TRNG**) is a state-of-the-art C++ pseudo-random number generator library for sequential and parallel Monte Carlo simulations (`https://www.numbercrunch.de/trng/`). It provides a variety of random number engines (pseudo-random number generators) and distributions. In particular, *parallel* random number engines provided by TRNG support techniques such as *block-splitting* and *leapfrogging* suitable for parallel algorithms. See 'References' for an introduction to the concepts and details around (parallel) random number generation.

Package **rTRNG** provides the R users with access to the functionality of the underlying TRNG C++ library in different ways and at different levels.

- Base-R Random-like usage via TRNG.Random functions, for selecting and manipulating the current engine. This is the simplest and more immediate way for R users to use **rTRNG**.
- Reference Objects wrapping the underlying C++ TRNG random number engines can be created and manipulated in OOP-style. This allows greater flexibility in using TRNG engines in R.
- TRNG C++ library and headers are made available to other R projects and packages using C++.
  - Standalone C++ code sourced via `sourceCpp` can rely on the `Rcpp::depends` attribute to correctly set up building against **rTRNG**:

    ```
    // [[Rcpp::depends(rTRNG)]]
    ```

  - Creating an R package with C++ code using the TRNG library is achieved by `LinkingTo: rTRNG` in the DESCRIPTION file, adding `importFrom(rTRNG, TRNG.Version)` in the NAMESPACE file, and setting relevant linker flags (via `LdFlags`) in Makevars[.win].
  - Note that C++ code using the TRNG library (sourced via `Rcpp::sourceCpp` or part of an R package) might fail on certain systems due to issues with building and linking against **rTRNG**. This is typically the case for macOS, and can generally be checked using `check_rTRNG_linking`.

See the package vignettes (`browseVignettes("rTRNG")`) for an overview and demos and refer to the examples in the documentation for further use cases.

Author(s)

Maintainer: Riccardo Porreca `<riccardo.porreca@mirai-solutions.com>`

Authors:

- Roland Schmid `<roland.schmid@mirai-solutions.com>`

Other contributors:

- Mirai Solutions GmbH `<info@mirai-solutions.com>` [copyright holder]
- Heiko Bauke (TRNG sources and headers) [contributor, copyright holder]

References

Heiko Bauke, *Tina's Random Number Generator Library*, Version 4.23.1, `https://github.com/rabauke/trng4/blob/v4.23.1/doc/trng.pdf`.

Stephan Mertens, *Random Number Generators: A Survival Guide for Large Scale Simulations*, 2009, `https://ui.adsabs.harvard.edu/abs/2009arXiv0905.4238M`

See Also

Useful links:

- `https://github.com/miraisolutions/rTRNG#readme`
- `https://mirai-solutions.ch`
- Report bugs at `https://github.com/miraisolutions/rTRNG/issues`

`check_rTRNG_linking` *Check rTRNG linking.*

Description

Check whether C++ code using the TRNG library can be built and linked against **rTRNG** on the current system.

Usage

```
check_rTRNG_linking(silent = FALSE)
```

Arguments

`silent`	logical: should the report of error messages be suppressed?

Value

A scalar logical with the result of the check. If `FALSE`, using the TRNG library from C++ code sourced via `sourceCpp` or part of an R package is not expected to work.

`defaultKind` *Default TRNG kind.*

Description

Return the name of the default TRNG random number engine.

Usage

```
defaultKind()
```

`LdFlags` *Linker flags for rTRNG.*

Description

Output the linker flags required to build against **rTRNG**.

Usage

```
LdFlags()
```

Details

`LdFlags` is typically called from Makevars as

```
PKG_LIBS += $(shell ${R_HOME}/bin/Rscript -e "rTRNG::LdFlags()")
```

and from Makevars.win as

```
PKG_LIBS += $(shell "${R_HOME}/bin${R_ARCH_BIN}/Rscript.exe" -e "rTRNG::LdFlags()")
```

Value

Returns NULL invisibly. The function is not called for its return value rather for the side effect of outputting the flags.

`rbinom_trng` *Binomial random numbers via TRNG.*

Description

Random number generation for the binomial distribution using the TRNG C++ library.

Usage

```
rbinom_trng(n, size, prob, engine = NULL, parallelGrain = 0L)
```

Arguments

`n`	Number of observations.
`size`, `prob`	Parameters of the distribution, with the same meaning as in `rbinom`. Note however that only scalar values are accepted.
`engine`	Optional TRNG engine object; if missing or `NULL`, the current engine controlled via TRNG.Random is used.
`parallelGrain`	Optional argument controlling the parallel simulation of random variates (see 'Parallel Simulation' below for details).

Value

Numeric vector of random variates generated with the given parameters. The length is determined by `n`.

Parallel Simulation

When a positive value of argument `parallelGrain` is supplied, random variates are simulated in parallel, provided a *parallel* random number `engine` is selected. This is done using **RcppParallel** via `parallelFor`, which uses the supplied `parallelGrain` to control the grain size (the number of threads being controlled by `setThreadOptions`). The grain size can greatly affect the overhead of performing the required *block splitting* `jump` operations and should be selected carefully. Note that TRNG guarantees the outcome of such parallel execution to be equivalent to a purely sequential simulation.

See Also

`rbinom`, `TRNG.Engine`, `TRNG.Random`.

Other TRNG distributions: `rlnorm_trng()`, `rnorm_trng()`, `rpois_trng()`, `runif_trng()`

Examples

```
## generate 10 random variates using the current TRNG engine
rbinom_trng(10, size = 1L, prob = 0.5)

## use a TRNG engine reference class object
r <- yarn2$new()
rbinom_trng(10, size = 1L, prob = 0.5, engine = r)

## generate 100k random variates in parallel, with 2 threads and 100 grain size
TRNGseed(117)
RcppParallel::setThreadOptions(numThreads = 2L)
x_parallel <- rbinom_trng(100e3, size = 1L, prob = 0.5, parallelGrain = 100L)
TRNGseed(117)
x_serial <- rbinom_trng(100e3, size = 1L, prob = 0.5)
identical(x_serial, x_parallel)
```

`rlnorm_trng` *Log-normal random numbers via TRNG.*

Description

Random number generation for the log-normal distribution using the TRNG C++ library.

Usage

```
rlnorm_trng(n, meanlog = 0, sdlog = 1, engine = NULL, parallelGrain = 0L)
```

Arguments

`n`	Number of observations.
`meanlog, sdlog`	Parameters of the distribution, with the same meaning as in `rlnorm`. Note however that only scalar values are accepted.
`engine`	Optional TRNG engine object; if missing or `NULL`, the current engine controlled via TRNG.Random is used.
`parallelGrain`	Optional argument controlling the parallel simulation of random variates (see 'Parallel Simulation' below for details).

Value

Numeric vector of random variates generated with the given parameters. The length is determined by `n`.

Parallel Simulation

When a positive value of argument `parallelGrain` is supplied, random variates are simulated in parallel, provided a *parallel* random number `engine` is selected. This is done using **RcppParallel** via `parallelFor`, which uses the supplied `parallelGrain` to control the grain size (the number of threads being controlled by `setThreadOptions`). The grain size can greatly affect the overhead of performing the required *block splitting* `jump` operations and should be selected carefully. Note that TRNG guarantees the outcome of such parallel execution to be equivalent to a purely sequential simulation.

See Also

`rlnorm`, `TRNG.Engine`, `TRNG.Random`.

Other TRNG distributions: `rbinom_trng()`, `rnorm_trng()`, `rpois_trng()`, `runif_trng()`

Examples

```
## generate 10 random variates using the current TRNG engine
rlnorm_trng(10, meanlog = 0, sdlog = 1)

## use a TRNG engine reference class object
r <- yarn2$new()
rlnorm_trng(10, meanlog = 0, sdlog = 1, engine = r)

## generate 100k random variates in parallel, with 2 threads and 100 grain size
TRNGseed(117)
RcppParallel::setThreadOptions(numThreads = 2L)
x_parallel <- rlnorm_trng(100e3, meanlog = 0, sdlog = 1, parallelGrain = 100L)
TRNGseed(117)
x_serial <- rlnorm_trng(100e3, meanlog = 0, sdlog = 1)
identical(x_serial, x_parallel)
```

`rnorm_trng` *Normal random numbers via TRNG.*

Description

Random number generation for the normal distribution using the TRNG C++ library.

Usage

```
rnorm_trng(n, mean = 0, sd = 1, engine = NULL, parallelGrain = 0L)
```

Arguments

`n`	Number of observations.
`mean, sd`	Parameters of the distribution, with the same meaning as in `rnorm`. Note however that only scalar values are accepted.

`engine`	Optional TRNG engine object; if missing or `NULL`, the current engine controlled via TRNG.Random is used.
`parallelGrain`	Optional argument controlling the parallel simulation of random variates (see 'Parallel Simulation' below for details).

Value

Numeric vector of random variates generated with the given parameters. The length is determined by `n`.

Parallel Simulation

When a positive value of argument `parallelGrain` is supplied, random variates are simulated in parallel, provided a *parallel* random number engine is selected. This is done using **RcppParallel** via `parallelFor`, which uses the supplied `parallelGrain` to control the grain size (the number of threads being controlled by `setThreadOptions`). The grain size can greatly affect the overhead of performing the required *block splitting* `jump` operations and should be selected carefully. Note that TRNG guarantees the outcome of such parallel execution to be equivalent to a purely sequential simulation.

See Also

`rnorm`, `TRNG.Engine`, `TRNG.Random`.

Other TRNG distributions: `rbinom_trng()`, `rlnorm_trng()`, `rpois_trng()`, `runif_trng()`

Examples

```
## generate 10 random variates using the current TRNG engine
rnorm_trng(10, mean = 0, sd = 1)

## use a TRNG engine reference class object
r <- yarn2$new()
rnorm_trng(10, mean = 0, sd = 1, engine = r)

## generate 100k random variates in parallel, with 2 threads and 100 grain size
TRNGseed(117)
RcppParallel::setThreadOptions(numThreads = 2L)
x_parallel <- rnorm_trng(100e3, mean = 0, sd = 1, parallelGrain = 100L)
TRNGseed(117)
x_serial <- rnorm_trng(100e3, mean = 0, sd = 1)
identical(x_serial, x_parallel)
```

`rpois_trng` *Poisson random numbers via TRNG.*

Description

Random number generation for the Poisson distribution using the TRNG C++ library.

Usage

```
rpois_trng(n, lambda, engine = NULL, parallelGrain = 0L)
```

Arguments

`n`	Number of observations.
`lambda`	Parameters of the distribution, with the same meaning as in `rpois`. Note however that only scalar values are accepted.
`engine`	Optional TRNG engine object; if missing or `NULL`, the current engine controlled via TRNG.Random is used.
`parallelGrain`	Optional argument controlling the parallel simulation of random variates (see 'Parallel Simulation' below for details).

Value

Numeric vector of random variates generated with the given parameters. The length is determined by `n`.

Parallel Simulation

When a positive value of argument `parallelGrain` is supplied, random variates are simulated in parallel, provided a *parallel* random number engine is selected. This is done using **RcppParallel** via `parallelFor`, which uses the supplied `parallelGrain` to control the grain size (the number of threads being controlled by `setThreadOptions`). The grain size can greatly affect the overhead of performing the required *block splitting* `jump` operations and should be selected carefully. Note that TRNG guarantees the outcome of such parallel execution to be equivalent to a purely sequential simulation.

See Also

`rpois`, `TRNG.Engine`, `TRNG.Random`.

Other TRNG distributions: `rbinom_trng()`, `rlnorm_trng()`, `rnorm_trng()`, `runif_trng()`

Examples

```
## generate 10 random variates using the current TRNG engine
rpois_trng(10, lambda = 4)

## use a TRNG engine reference class object
r <- yarn2$new()
rpois_trng(10, lambda = 4, engine = r)

## generate 100k random variates in parallel, with 2 threads and 100 grain size
TRNGseed(117)
RcppParallel::setThreadOptions(numThreads = 2L)
x_parallel <- rpois_trng(100e3, lambda = 4, parallelGrain = 100L)
TRNGseed(117)
x_serial <- rpois_trng(100e3, lambda = 4)
identical(x_serial, x_parallel)
```

runif_trng — *Uniform random numbers via TRNG.*

Description

Random number generation for the uniform distribution using the TRNG C++ library.

Usage

```
runif_trng(n, min = 0, max = 1, engine = NULL, parallelGrain = 0L)
```

Arguments

`n`	Number of observations.
`min, max`	Parameters of the distribution, with the same meaning as in `runif`. Note however that only scalar values are accepted.
`engine`	Optional TRNG engine object; if missing or `NULL`, the current engine controlled via TRNG.Random is used.
`parallelGrain`	Optional argument controlling the parallel simulation of random variates (see ‘Parallel Simulation’ below for details).

Value

Numeric vector of random variates generated with the given parameters. The length is determined by `n`.

Parallel Simulation

When a positive value of argument `parallelGrain` is supplied, random variates are simulated in parallel, provided a *parallel* random number `engine` is selected. This is done using **RcppParallel** via `parallelFor`, which uses the supplied `parallelGrain` to control the grain size (the number of threads being controlled by `setThreadOptions`). The grain size can greatly affect the overhead of performing the required *block splitting* `jump` operations and should be selected carefully. Note that TRNG guarantees the outcome of such parallel execution to be equivalent to a purely sequential simulation.

See Also

`runif`, `TRNG.Engine`, `TRNG.Random`.

Other TRNG distributions: `rbinom_trng()`, `rlnorm_trng()`, `rnorm_trng()`, `rpois_trng()`

Examples

```
## generate 10 random variates using the current TRNG engine
runif_trng(10, min = 0, max = 1)

## use a TRNG engine reference class object
r <- yarn2$new()
runif_trng(10, min = 0, max = 1, engine = r)

## generate 100k random variates in parallel, with 2 threads and 100 grain size
TRNGseed(117)
RcppParallel::setThreadOptions(numThreads = 2L)
x_parallel <- runif_trng(100e3, min = 0, max = 1, parallelGrain = 100L)
TRNGseed(117)
x_serial <- runif_trng(100e3, min = 0, max = 1)
identical(x_serial, x_parallel)
```

TRNG.Engine *TRNG random number engines.*

Description

Reference Classes exposing random number engines (pseudo-random number generators) in the TRNG C++ library. Engine objects of a class `engineClass` are created as `r <- engineClass$new(...)`, and a method `m` is invoked as `x$m(...)`. The engine object `r` can be then used for generating random variates via any of the `r<dist>_trng` functions (e.g., `runif_trng`), specifying the optional argument `engine = r`.

Classes

***Parallel* random number engines** `lcg64`, `lcg64_shift`, `mrg2`, `mrg3`, `mrg3s`, `mrg4`, `mrg5`, `mrg5s`, `yarn2`, `yarn3`, `yarn3s`, `yarn4`, `yarn5`, `yarn5s`.

***Conventional* random number engines** `lagfib2plus_19937_64`, `lagfib2xor_19937_64`, `lagfib4plus_19937_64`, `lagfib4xor_19937_64`, `mt19937_64`, `mt19937`.

Constructors

`$new()` Construct a random engine object using default seed and internal parameters.

`$new(seed)` Construct a random engine object with default internal parameters using the provided `seed`.

`$new(string)` Construct a random engine object restoring its internal state and parameters from a character `string`, falling back to `$new()` for empty strings. See method `$toString()`.

Methods

`$seed(seed)` Use the scalar integer `seed` to set the engine's internal state.

`$jump(steps)` Advance by `steps` the internal state of the engine. Applies to *parallel* engines only.

`$split(p, s)` Update the internal state and parameters of the engine for generating directly the `s`th of `p` subsequences, with `s` in [`1`, `p`], producing one element every `s` starting from the `p`th. Applies to *parallel* engines only.

`$name()`, `$kind()` Return the name of the random number engine (e.g., `"yarn2"`), also referred to as `kind` in **rTRNG** similarly to base R.

`$toString()` Return a character representation of the engine's internal state and parameters.

`$copy()` Specialization of the generic method for Reference Classes, ensuring the underlying C++ engine object is properly copied.

`$show()` Specialization of the generic show, displaying `$toString()` (truncated to 80 characters).

`$.Random.seed()` Return a two-element character vector with elements `$kind()` and `$toString()`, suitable for use in `TRNG.Random.seed` (and by a possible function returning an engine object given a `TRNG.Random.seed`).

Details

The TRNG C++ library provides a collection of random number engines (pseudo-random number generators). In particular, compared to *conventional* engines working in a purely sequential manner, *parallel* engines can be manipulated via `jump` and `split` operations. Jumping allows to advance the internal state by a number of steps without generating all intermediate states, whereas split operations allow to generate directly a subsequence obtained by decimating the original sequence. Please consult the TRNG C++ library documentation (see 'References') for an introduction to the concepts and details around (parallel) random number generation and engines, including details about the state size and period of the TRNG generators.

Random number engines from the C++ TRNG library are exposed to R using **Rcpp** `Module`s. As a consequence, the arguments to all Constructors and Methods above are not passed by name but by order. Moreover, arguments and return values are both defined in terms of C++ data types. Details can be displayed via the standard Reference Class documentation method `$help` (e.g., `yarn2$help(split)`).

Most of the Methods above are simple wrappers of analogous methods in the corresponding C++ class provided by the TRNG library. A few differences/details are worth being mentioned.

- Argument `s` of the `split` method is exposed to R according to R's `1`-based indexing, thus in the [`1`, `p`] interval, whereas the TRNG C++ implementation follows C++ `0`-based indexing, thus allowing values in [`0`, `p-1`].
- Constructor `new(string)` and method `toString()` rely on streaming operators >> and << available for all C++ TRNG classes.
- TRNG C++ random number engine objects are *copy-constructible* and *assignable*, whereas their R counterparts in **rTRNG** are purely reference-based. In particular, as for any R Reference Object, engines are not copied upon assignment but via the `$copy()` method.

Random number engines details

Parallel engines:

`lcg64` Linear congruential generator with modulus 2^{64}.

`lcg64_shift` Linear congruential generator with modulus 2^{64} and bit-shift transformation.

`mrg2`, `mrg3`, `mrg4`, `mrg5` Multiple recurrence generators based on a linear feedback shift register sequence with prime modulus $2^{31} - 1$.

`mrg3s`, `mrg5s` Multiple recurrence generators based on a linear feedback shift register with Sophie-Germain prime modulus.

`yarn2`, `yarn3`, `yarn4`, `yarn5` YARN generators based on the delinearization of a linear feedback shift register sequence with prime modulus $2^{31} - 1$.

`yarn3s`, `yarn5s` YARN generators based on the delinearization of a linear feedback shift register sequence with Sophie-Germain prime modulus.

Conventional engines:

`lagfib2plus_19937_64`, `lagfib4plus_19937_64` Lagged Fibonacci generator with 2 or 4 feedback taps and addition.

`lagfib2xor_19937_64`, `lagfib4xor_19937_64` Lagged Fibonacci generator with 2 or 4 feedback taps and exclusive-or operation.

`mt19937` Mersenne-Twister generating 32 random bit.

`mt19937_64` Mersenne-Twister generating 64 random bit.

References

Heiko Bauke, *Tina's Random Number Generator Library*, Version 4.23.1, https://github.com/rabauke/trng4/blob/v4.23.1/doc/trng.pdf.

See Also

ReferenceClasses, TRNG.Random.

TRNG distributions: rbinom_trng, rlnorm_trng, rnorm_trng, rpois_trng, runif_trng.

Examples

```
## Class yarn2 used in the examples below can be replaced by any other TRNG
## engine class (only of a parallel kind for jump and split examples).

## basic constructor with default internal state (and parameters)
r <- yarn2$new()
## show the internal parameters and state
r
## return internal parameters and state as character string
r$toString()

## seed the random number engine
r$seed(117)
r
## construct with given initial seed
s <- yarn2$new(117)
identical(s$toString(), r$toString())

## construct from string representation
s <- yarn2$new(r$toString()) # implicitly creates a copy
identical(s$toString(), r$toString())
s <- yarn2$new("") # same as yarn2$new()
```

```
identical(s$toString(), yarn2$new()$toString())
## Not run:
  ## error if the string is not a valid representation
  s <- yarn2$new("invalid")

## End(Not run)

## copy vs. reference
r_ref <- r # reference to the same engine object
r_cpy <- r$copy() # copy an engine
identical(r_cpy$toString(), r$toString())
rbind(c(runif_trng(4, engine = r), runif_trng(6, engine = r_ref)),
      runif_trng(10, engine = r_cpy))

## jump (and draw from reference)
runif_trng(10, engine = r_cpy)
r_ref$jump(7) # jump 7 steps ahead
runif_trng(3, engine = r) # jump has effect on the original r

## split
r_cpy <- r$copy()
runif_trng(10, engine = r)
r_cpy$split(5, 2) # every 5th element starting from the 2nd
runif_trng(2, engine = r_cpy)

## seed, jump and split can be used in c(...) as they return NULL
r <- yarn2$new()
r_cpy <- r$copy()
r$seed(117)
runif_trng(10, engine = r)
c(r_cpy$seed(117),
  r_cpy$jump(2), runif_trng(2, engine = r_cpy),
  r_cpy$split(3,2), runif_trng(2, engine = r_cpy))

## TRNG engine name/kind
r$kind()
r$name()

## use $.Random.seed() to set the current engine (as a copy)
r$.Random.seed()
TRNG.Random.seed(r$.Random.seed())
```

`TRNG.Random` *TRNG random number generation.*

Description

The functions below allow setting and manipulating the current TRNG random number engine (pseudo-random number generator), similar to base-R Random. The current engine is then used for generating random variates via any of the `r<dist>_trng` functions (e.g., `runif_trng`).

`TRNGkind` allows to query or set the kind of TRNG engine in use. See ‘Random number engines details’ for the available engines.

`TRNGseed` specifies the seed for the current engine.

If the current engine is of a *parallel* kind, `TRNGjump` advances its internal state without generating all intermediate steps.

If the current engine is of a *parallel* kind, `TRNGsplit` updates its internal state and parameters in order to generate directly a subsequence obtained by decimation, producing every `s`th element starting from the `p`th.

`TRNG.Random.seed` allows to get a full representation of the current state of the engine in use, and to restore the current engine from such representation.

Usage

```
TRNGkind(kind = NULL)

TRNGseed(seed)

TRNGjump(steps)

TRNGsplit(p, s)

TRNG.Random.seed(engspec)
```

Arguments

`kind`	Character string or `NULL`. If `kind` is not `NULL`, it defines the TRNG random number engine to be used. Use "default" for the **rTRNG** default kind (`"yarn2"`).
`seed`	Scalar integer seed, determining the internal state of the current engine.
`steps`	Number of steps to jump ahead.
`p`	Number of subsequences to split the engine by.
`s`	Index of the desired subsequence between `1` and `p`.
`engspec`	Optional two-element character vector `c(kind, state)`, where the second element is a character representation of the current engine’s internal state and parameters.

Value

`TRNGkind` returns the TRNG kind selected before the call, invisibly if argument `kind` is not NULL.

`TRNG.Random.seed()` called with no arguments returns a two-element character vector `c(kind, state)` fully representing the current state of the engine in use. When argument `engspec = c(kind, state)` is provided, it is used to set an engine of the given `kind` with internal state and parameters restored from `state`.

Details

The TRNG C++ library provides a collection of random number engines (pseudo-random number generators). In particular, compared to *conventional* engines working in a purely sequential manner, *parallel* engines can be manipulated via `jump` and `split` operations. Jumping allows to advance the internal state by a number of steps without generating all intermediate states, whereas split operations allow to generate directly a subsequence obtained by decimating the original sequence. Please consult the TRNG C++ library documentation (see 'References') for an introduction to the concepts and details around (parallel) random number generation and engines, including details about the state size and period of the TRNG generators.

The current engine is an instance of one TRNG engine class provided by **rTRNG**, and is stored as `"TRNGengine"` global option. If not explicitly set via `TRNGkind`, an engine of default kind is implicitly created upon the first call to any `TRNG*` or `r<dist>_trng` function. Note that the current engine is not persistent across R sessions. Function `TRNG.Random.seed` can be used to extract and restore the current engine and its internal state.

Random number engines details

Parallel engines:

`lcg64` Linear congruential generator with modulus 2^{64}.

`lcg64_shift` Linear congruential generator with modulus 2^{64} and bit-shift transformation.

`mrg2`, `mrg3`, `mrg4`, `mrg5` Multiple recurrence generators based on a linear feedback shift register sequence with prime modulus $2^{31} - 1$.

`mrg3s`, `mrg5s` Multiple recurrence generators based on a linear feedback shift register with Sophie-Germain prime modulus.

`yarn2`, `yarn3`, `yarn4`, `yarn5` YARN generators based on the delinearization of a linear feedback shift register sequence with prime modulus $2^{31} - 1$.

`yarn3s`, `yarn5s` YARN generators based on the delinearization of a linear feedback shift register sequence with Sophie-Germain prime modulus.

Conventional engines:

`lagfib2plus_19937_64`, `lagfib4plus_19937_64` Lagged Fibonacci generator with 2 or 4 feedback taps and addition.

`lagfib2xor_19937_64`, `lagfib4xor_19937_64` Lagged Fibonacci generator with 2 or 4 feedback taps and exclusive-or operation.

`mt19937` Mersenne-Twister generating 32 random bit.

`mt19937_64` Mersenne-Twister generating 64 random bit.

References

Heiko Bauke, *Tina's Random Number Generator Library*, Version 4.23.1, `https://github.com/rabauke/trng4/blob/v4.23.1/doc/trng.pdf`.

See Also

TRNG distributions: `rbinom_trng`, `rlnorm_trng`, `rnorm_trng`, `rpois_trng`, `runif_trng`.

Examples

```
## TRNG kind of the current engine
TRNGkind()
## set a specific TRNG kind
TRNGkind("yarn5s")
TRNGkind()
## Not run:
  ## error if kind is not valid
  TRNGkind("invalid")

## End(Not run)
## set the deafult TRNG kind
TRNGkind("default")
TRNGkind()

## seed the current random number engine
TRNGseed(117)

## full representation of the current state of the engine in use
s <- TRNG.Random.seed()
s

## draw 10 random variates using the current engine
runif_trng(10)

## restore the engine and its internal state
TRNG.Random.seed(s)

## jump and draw the last 3 variates out of the 10 above
TRNGjump(7) # jump 7 steps ahead
runif_trng(3)

## restore the internal state, split and draw every 5th element starting from
## the 2nd
TRNG.Random.seed(s)
TRNGsplit(5, 2)
runif_trng(2)

## TRNGseed, TRNGjump and TRNGsplit can be combined with r<dist>_trng in c(...)
## as they return NULL
c(TRNGseed(117),
  TRNGjump(2), runif_trng(2),
  TRNGsplit(3,2), runif_trng(2))
```

`TRNG.Version` *TRNG library version.*

Description

Return the version of the TRNG C++ library embedded in the `rTRNG` package.

Usage

```
TRNG.Version()
```

Index