Package: ironseed (via r-universe)

June 8, 2026

Title Improved Random Number Generator Seeding

Version 0.3.0.9000

Description A procedure for seeding R's built in random number generators using a variable-length sequence of values. Accumulates input entropy into a 256-bit hash digest or ``ironseed'' and is able to generate a variable-length sequence of output seeds from an ironseed.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

Roxygen list(markdown = TRUE)

Biarch TRUE

NeedsCompilation yes

URL https://github.com/reedacartwright/ironseed

BugReports https://github.com/reedacartwright/ironseed/issues

Suggests tinytest, gmp, Rmpfr

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-06-08 05:46:03 UTC

RemoteUrl https://github.com/reedacartwright/ironseed

RemoteRef HEAD

RemoteSha dcf291c62869207a330b8bff99280442751043e6

Contents

digest — *Create ironseed digests for arbitrary R objects*

Description

The `digest()` function creates a variable length ironseed digest for arbitrary R objects. Internally, it uses base::serialize to convert objects to a raw vector before calculating the digest.

Usage

```
digest(object, n = 1L, salt = 1L, serialize = TRUE, ascii = FALSE, xdr = FALSE)
```

Arguments

`object`	an arbitary R object. Will be serialized unless the `serialize` argument is `FALSE`.
`n`	a scalar integer. Specifies the length of the returned vector.
`salt`	a scalar integer. Used to vary the digests between applications.
`serialize`	a logical. Indicates whether to serialize object before calculating the digest.
`ascii`, `xdr`	Passed to base::serialize

Details

This algorithm uses different coefficients than create_ironseed and create_seedseq.

Value

an integer vector of length `n`

ironseed — *Ironseed: Improved Random Number Generator Seeding*

Description

An ironseed is a 256-bit hash digest constructed from a variable-length input sequence and can be used to generate a variable-length output sequence of seeds, including initializing R's built-in random number generator.

- `ironseed()` creates an ironseed from user supplied objects, from external arguments, or automatically from multiple sources of entropy on the local system. It also initializes R's built-in random number generator from an ironseed.
- `get_ironseed()` returns the ironseed most recently used to initialize `.Random.seed`.
- `set_ironseed()` is a wrapper around `ironseed()` for initializing `.Random.seed` with user-supplied data.

- `create_ironseed()` constructs an ironseed from a list of seed objects, following the rules described below. `auto_ironseed()` constructs an ironseed from multiple sources of entropy on the local system.
- `is_ironseed()` tests whether an object is an ironseed, and `is_ironseed_str()` tests if it is a string representing and ironseed.
- `as_ironseed()` casts an object to an ironseed, and `parse_ironseed_str()` parses a string to an ironseed.

Usage

```
ironseed(
  ...,
  set_seed = TRUE,
  quiet = FALSE,
  methods = c("dots", "args", "env", "auto", "null"),
  salt = 0L
)

set_ironseed(x, ..., quiet = FALSE, salt = 0L)

get_ironseed()

create_ironseed(x)

auto_ironseed()

is_ironseed(x)

is_ironseed_str(x)

as_ironseed(x)

parse_ironseed_str(x)
```

Arguments

`...`	objects
`set_seed`	a logical indicating whether to initialize `.Random.seed`.
`quiet`	a logical indicating whether to silence messages.
`methods`	a character vector.
`salt`	a scalar integer. Used to vary RNG seeding between applications.
`x`	a string, ironseed, list, or other object

Details

Ironseeds have a specific string representation, e.g. "rBQSjhjYv1d-z8dfMATEicf-sw1NSWAvVDi-bQaKSKKQmz1", where each element is a 64-bit number encoded in little-endian base58 format.

Ironseed generates an ironseed from an input sequence according to the methods included in `methods`. When generating an ironseed, `ironseed()` tries the listed methods starting from the first value and continuing until it can generate an ironseed. If no method works, an error will be raised.

- dots: Use the values passed as `...` to construct an ironseed. Most atomic types and lists of atomic types can be used. `ironseed(NULL)` constructs a default ironseed from no data (equivalent to the "null" method below). `ironseed()`, `ironseed(list())`, `ironseed(character())`, etc. are considered empty inputs and the next method will be tried.
- args: Use command line arguments to construct an ironseed. Any arguments that begins with `--seed=` or `-seed=` will be used as strings, after the argument names are trimmed. If no matching arguments are found, the next method will be tried.
- env: Use the value of the environmental variable "IRONSEED" as a scalar character to construct an ironseed. If this variable doesn't exist or is set to an empty string, the next method will be tried.
- auto: Use multiple sources of entropy from the system to generate an ironseed. This method always constructs an ironseed.
- null: Generate a "default" ironseed using no input. This method always constructs an ironseed.

If the input sequence has one value and it is an ironseed object, it is used as is. If the input sequence is a scalar character that matches an ironseed string, it is parsed to an ironseed. Otherwise, the input sequence is hashed to create an ironseed.

An ironseed is a finite-entropy (or fixed-entropy) hash digest that can be used to generate an unlimited sequence of seeds for initializing the state of a random number generator. It is inspired by the work of M.E. O'Neill and others.

An ironseed is a 256-bit hash digest constructed from a variable-length sequence of 32-bit inputs. Each ironseed consists of eight 32-bit sub-digests. The sub-digests are 32-bit multilinear hashes that accumulate entropy from the input sequence. Each input is included in every sub-digest. The coefficients for the multilinear hashes are generated by a Weyl sequence.

Multilinear hashes are also used to generate an output seed sequence from an ironseed. Each 32-bit output value is generated by uniquely hashing the sub-digests. The coefficients for the output are generated by a second Weyl sequence.

To improve the observed randomness of each hash output, bits are mixed using a finalizer adapted from SplitMix64. With the additional mixing from the finalizer, the output seed sequence passes PractRand tests.

Value

An ironseed. If `.Random.seed` was initialized, the ironseed used will be returned invisibly.

References

- O'Neill (2015) Developing a seed_seq Alternative. https://www.pcg-random.org/posts/developing-a-seed_seq-alternative.html
- O'Neill (2015) Simple Portable C++ Seed Entropy. https://www.pcg-random.org/posts/simple-portable-cpp-seed-entropy.html
- O'Neill (2015) Random-Number Utilities. https://gist.github.com/imneme/540829265469e673d045

- Lemire and Kaser (2018) Strongly universal string hashing is fast. https://arxiv.org/pdf/1202.4961
- Steele et al. (2014) Fast splittable pseudorandom number generators. doi:10.1145/2714064.2660195
- Weyl Sequence https://en.wikipedia.org/wiki/Weyl_sequence
- PractRand https://pracrand.sourceforge.net/

See Also

set.seed .Random.seed

Examples

```
# Generate an ironseed with user supplied data.
# This will initialize an uninitialized `.Random.seed`.
ironseed::ironseed("Experiment", 20251031, 1)

# Generate an ironseed automatically and force initialize
ironseed::ironseed(set_seed = TRUE)

# Generate a deterministic ironseed when user data is missing
ironseed::ironseed(NULL)

# Generate a random ironseed when user data is missing
ironseed::ironseed(character()) # list(), integer(), etc. also work.

# Return last used ironseed.
ironseed::get_ironseed()
```

---

ironseed_stream *Ironseed output seed sequences*

Description

Output sequences of 32-bit seeds are generated from an ironseed using multilinear hashes. The coefficients for these hashes are generated by a different Weyl sequence from the input hashes. A hash finalizer is also used to mix bits and improve observed randomness.

- `create_seedseq()` uses an ironseed to generate a sequence of 32-bit seeds.
- `ironseed_stream()` returns a function that can be used to generate a seed sequence iteratively.

Usage

```
ironseed_stream(
  ...,
  methods = c("dots", "args", "env", "auto", "null"),
  salt = 0L
)

create_seedseq(fe, n, salt = 0L)
```

Arguments

`...`	objects
`methods`	a character vector.
`salt`	a scalar integer that can be used to vary stream output between applications
`fe`	an ironseed
`n`	a scalar integer specifying the number of seeds to generate

Value

an integer vector containing 32-bit output seeds. If `n` is missing, `ironseed_stream()` returns the underlying ironseed.

See Also

ironseed

Examples

```
# Generate 20 seeds from an ironseed
fe <- ironseed("Experiment", 20251031, 1, set_seed = FALSE)
create_seedseq(fe, 20)

# Generate 20 seeds from the ironseed using a salt to change results
create_seedseq(fe, 20, salt = 142)

# Create a function that can be called multiple times to produce seeds
get_seeds <- ironseed_stream("Experiment", 20251031, 1)

# generate 10 seeds
get_seeds(10)

# generate 10 more seeds
get_seeds(10)

# output the ironseed used for the stream
get_seeds()
```

`with_ironseed` *Temporary ironseeds*

Description

`with_ironseed()` runs code with a specific ironseed and restores global state afterwards. `local_ironseed()` restores global state when the current evaluation state ends.

Usage

```
with_ironseed(seeds, code, quiet = FALSE, salt = 0L)

local_ironseed(
  seeds,
  ...,
  quiet = FALSE,
  salt = 0L,
  .local_envir = parent.frame()
)

with_ironseed_stream(func, code)

local_ironseed_stream(func, .local_envir = parent.frame())
```

Arguments

`seeds`	An object or list of objects suitable for constructing an ironseed.
`code`	Code to execute in the temporary environment.
`quiet`	a logical indicating whether to silence messages.
`salt`	a scalar integer. Used to vary RNG seeding between applications.
`...`	Additional objects.
`.local_envir`	The environment to use for scoping.
`func`	A stream function returned by `ironseed_stream()`

Value

`with_ironseed()` returns the results of the evaluation of the code argument. `local_ironseed()` returns the constructed ironseed.

See Also

ironseed ironseed_stream

Index