Package 'ids'

May 8, 2026

Title Generate Random Identifiers

Version 1.0.1

Description Generate random or human readable and pronounceable identifiers.

License MIT + file LICENSE

URL https://github.com/richfitz/ids

BugReports https://github.com/richfitz/ids/issues

Imports openssl, uuid

Suggests knitr, rcorpora, rmarkdown, testthat

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Rich FitzJohn [aut, cre]

Maintainer Rich FitzJohn <rich.fitzjohn@gmail.com>

Repository CRAN

Date/Publication 2017-05-31 08:49:59 UTC

Contents

adjective_animal 2
ids 3
int_to_proquint 4
proquint 5
random_id 6
sentence 7
uuid 8

Index **10**

`adjective_animal` *Ids based on a number of adjectives and an animal*

Description

Ids based on a number of adjectives and an animal

Usage

```
adjective_animal(n = 1, n_adjectives = 1, style = "snake",
  max_len = Inf)
```

Arguments

`n`	number of ids to return. If `NULL`, it instead returns the generating function
`n_adjectives`	Number of adjectives to prefix the anmial with
`style`	Style to join words with. Can be one of "Pascal", "camel", "snake", "kebab", "dot", "title", "sentence", "lower", "upper", and "constant".
`max_len`	The maximum length of a word part to include (this may be useful because some of the names are rather long. This stops you generating a `hexakosioihexekontahexaphobic_queenalex` A vector of length 2 can be passed in here in which case the first element will apply to the adjectives (all of them) and the second element will apply to the animals.

Details

The list of adjectives and animals comes from https://github.com/a-type/adjective-adjective-animal, and in turn from gfycat.com

Author(s)

Rich FitzJohn

Examples

```
# Generate a random identifier:
adjective_animal()

# Generate a bunch all at once:
adjective_animal(5)

# Control the style of punctuation with the style argument:
adjective_animal(style = "lower")
adjective_animal(style = "CONSTANT")
adjective_animal(style = "camel")
adjective_animal(style = "kebab")

# Control the number of adjectives used
```

```
adjective_animal(n_adjectives = 3)

# This can get out of hand quickly though:
adjective_animal(n_adjectives = 7)

# Limit the length of adjectives and animals used:
adjective_animal(10, max_len = 6)

# The lengths can be controlled for adjectives and animals
# separately, with Inf meaning no limit:
adjective_animal(10, max_len = c(6, Inf), n_adjectives = 2)

# Pass n = NULL to bind arguments to a function
id <- adjective_animal(NULL, n_adjectives = 2, style = "dot", max_len = 6)
id()
id(10)
```

`ids` *Generic id generating function*

Description

Generic id generating function

Usage

```
ids(n, ..., vals = list(...), style = "snake")
```

Arguments

`n`	number of ids to return. If `NULL`, it instead returns the generating function
`...`	A number of character vectors
`vals`	A list of character vectors, *instead* of `...`
`style`	Style to join words with. Can be one of "Pascal", "camel", "snake", "kebab", "dot", "title", "sentence", "lower", "upper", and "constant".

Value

Either a character vector of length `n`, or a function of one argument if `n` is `NULL`

Author(s)

Rich FitzJohn

Examples

```
# For an example, please see the vignette
```

`int_to_proquint` *Convert to and from proquints*

Description

Convert to and from proquints.

Usage

```
int_to_proquint(x, use_cache = TRUE)

proquint_to_int(p, as = "numeric", use_cache = TRUE)

proquint_word_to_int(w, use_cache = TRUE, validate = TRUE)

int_to_proquint_word(i, use_cache = TRUE, validate = TRUE)
```

Arguments

`x`	An integer (or integer-like) value to convert to a proquint
`use_cache`	Because there are relatively few combinations per word, and because constructing short strings is relatively expensive in R, it may be useful to cache all 65536 possible words. If `TRUE` then the first time that this function is used all words will be cached and the results used - the first time may take up to ~1/4 of a second and subsequent calls will be much faster. The identifiers selected will not change with this option (i.e., given a particular random seed, changing this option will not affect the identifiers randomly selected).
`p`	A character vector representing a proquint
`as`	The target data type for conversion from proquint. The options are `integer`, `numeric` and `bignum`. The first two will overflow given sufficiently large input - this will throw an error (overflow is at `.Machine$integer.max` and `2 / .Machine$double.eps - 1` for `integer` and `numeric` respectively). For `bignum` this will return a *list* of `bignum` elements *even if* `p` *is of length 1*.
`w`	A proquint *word* (five letter string)
`validate`	Validate the range of inputs? Because these functions are used internally, they can skip input validation. You can too if you promise to pass sanitised input in. If out-of-range values are passed in and validation is disabled the behaviour is undefined and subject to change.
`i`	An integer representing a single proquint word (in the range 0:65535)

Details

These functions try to be type safe and predictable about what they will and will not return.

For `proquint_to_int`, because numeric overflow is a possibility, it is important to consider whether a proquint can be meaninfully translated into an integer or a numeric and the functions will throw an error rather than failing in a more insidious way (promoting the type or returning NA).

`proquint_word_to_int` always retuns an integer vector of the same length as the input.

Missing values are allowed; a missing integer representation of a proquint will translate as `NA_character_` and a missing proquint will translate as `NA_integer_` (if `as = "integer"`), `NA_real_`, if `as = "numeric"` or as `NULL` (if `as = "bignum"`).

Names are always discarded. Future versions may gain an argument `named` with a default of `FALSE`, but that setting to `TRUE` would preseve names. Let me know if this would be useful.

`proquint` *Generate random proquint identifiers*

Description

Generate random "proquint" identifiers. "proquint" stands for PRO-nouncable QUINT-uplets and were described by Daniel Wilkerson in https://arxiv.org/html/0901.4016. Each "word" takes one of 2^16 possibilities. A four word proquint has a keyspace of 10^19 possibilities but takes only 23 characters. Proquint identifiers can be interchanged with integers (though this is totally optional); see `proquint_to_int` and the other functions documented on that page.

Usage

```
proquint(n = 1, n_words = 2L, use_cache = TRUE, use_openssl = FALSE)
```

Arguments

`n`	number of ids to return. If `NULL`, it instead returns the generating function
`n_words`	The number of words for each identifier; each word has `2^16` (65536) possible combinations, a two-word proquint has `2^32` possible combinations and an k-word proquint has `2^(k * 16)` possible combinations.
`use_cache`	Because there are relatively few combinations per word, and because constructing short strings is relatively expensive in R, it may be useful to cache all 65536 possible words. If `TRUE` then the first time that this function is used all words will be cached and the results used - the first time may take up to ~1/4 of a second and subsequent calls will be much faster. The identifiers selected will not change with this option (i.e., given a particular random seed, changing this option will not affect the identifiers randomly selected).
`use_openssl`	Use openssl for random number generation, with the primary effect that the identifiers will not be affected by R's random seed (at a small speed cost).

Details

In the abstract of their paper, Wilkerson introduces proquints:

"Identifiers (IDs) are pervasive throughout our modern life. We suggest that these IDs would be easier to manage and remember if they were easily readable, spellable, and pronounceable. As a solution to this problem we propose using PRO-nouncable QUINT-uplets of alternating unambiguous consonants and vowels: proquints."

Examples

```
# A single, two word, proquint
proquint()

# Longer identifier:
proquint(n_words = 5)

# More identifiers
proquint(10)
```

`random_id` *Cryptographically generated random identifiers*

Description

Random identifiers. By default this uses the `openssl` package to produce a random set of bytes, and expresses that as a hex character string. This does not affect R's random number stream.

Usage

```
random_id(n = 1, bytes = 16, use_openssl = TRUE)
```

Arguments

`n`	number of ids to return. If `NULL`, it instead returns the generating function
`bytes`	The number of bytes to include for each identifier. The length of the returned identifiers will be twice this long with each pair of characters representing a single byte.
`use_openssl`	A logical, indicating if we should use the openssl for generating the random identifiers. The openssl random bytestream is not affected by the state of the R random number generator (e.g., via `set.seed`) so may not be suitable for use where reproducibility is important. The speed should be very similar for both approaches.

Author(s)

Rich FitzJohn

Examples

```
# Generate a random id:
random_id()

# Generate 10 of them!
random_id(10)

# Different length ids
random_id(bytes = 8)
```

```
# (note that the number of characters is twice the number of bytes)

# The ids are not affected by R's RNG state:
set.seed(1)
(id1 <- random_id())
set.seed(1)
(id2 <- random_id())
# The generated identifiers are different, despite the seed being the same:
id1 == id2

# If you need these identifiers to be reproducible, pass use_openssl = FALSE
set.seed(1)
(id1 <- random_id(use_openssl = FALSE))
set.seed(1)
(id2 <- random_id(use_openssl = FALSE))
# This time they are the same:
id1 == id2

# Pass \code{n = NULL} to generate a function that binds your arguments:
id8 <- random_id(NULL, bytes = 8)
id8(10)
```

`sentence` *Sentence style identifiers*

Description

Create a sentence style identifier. This uses the approach described by Asana on their blog https://blog.asana.com/2011/09/6-sad-squid-snuggle-softly/. This approach encodes 32 bits of information (so 2^32 ~= 4 billion possibilities) and in theory can be remapped to an integer if you really wanted to.

Usage

```
sentence(n = 1, style = "snake", past = FALSE)
```

Arguments

`n`	number of ids to return. If `NULL`, it instead returns the generating function
`style`	Style to join words with. Can be one of "Pascal", "camel", "snake", "kebab", "dot", "title", "sentence", "lower", "upper", and "constant".
`past`	Use the past tense for verbs (e.g., slurped or jogged rather than slurping or jogging)

Author(s)

Rich FitzJohn

Examples

```
# Generate an identifier
sentence()

# Generate a bunch
sentence(10)

# As with adjective_animal, use "style" to control punctuation
sentence(style = "Camel")
sentence(style = "dot")
sentence(style = "Title")

# Change the tense of the verb:
set.seed(1)
sentence()
set.seed(1)
sentence(past = TRUE)

# Pass n = NULL to bind arguments to a function
id <- sentence(NULL, past = TRUE, style = "dot")
id()
id(10)
```

---

uuid *Generate UUIDs*

Description

Generate UUIDs using the uuid package. This is simply a thin wrapper around `uuid::UUIDgenerate` that matches the interface in the rest of the ids package.

Usage

```
uuid(n = 1, drop_hyphens = FALSE, use_time = NA)
```

Arguments

`n`	number of ids to return. If `NULL`, it instead returns the generating function
`drop_hyphens`	Drop the hyphens from the UUID?
`use_time`	Passed through to `UUIDgenerate` as `use.time`.

Author(s)

Rich FitzJohn

Examples

```
# Generate one id
uuid()

# Or a bunch
uuid(10)

# More in the style of random_id()
uuid(drop_hyphens = TRUE)
```

Index

adjective_animal, 2

ids, 3
int_to_proquint, 4
int_to_proquint_word *(*int_to_proquint*)*, 4

proquint, 5
proquint_to_int, *5*
proquint_to_int *(*int_to_proquint*)*, 4
proquint_word_to_int *(*int_to_proquint*)*, 4

random_id, 6

sentence, 7
set.seed, *6*

uuid, 8