Package 'UNF'

May 16, 2021

Version 2.0.7

Title Tools for Creating Universal Numeric Fingerprints for Data

Date 2021-05-16

Description Computes a universal numeric fingerprint (UNF) for an R data object. UNF is a cryptographic hash or signature that can be used to uniquely identify (a version of) a rectangular dataset, or a subset thereof. UNF can be used, in tandem with a DOI, to form a persistent citation to a versioned dataset.

Imports utils, stats, tools, base64enc, digest

Suggests knitr, rmarkdown, testthat

License GPL-2

URL https://github.com/leeper/UNF

BugReports https://github.com/leeper/UNF/issues

VignetteBuilder knitr

RoxygenNote 7.1.0

NeedsCompilation no

Author Thomas J. Leeper [aut, cre] (<https://orcid.org/0000-0003-4097-6326>), Micah Altman [aut]

Maintainer Thomas J. Leeper <thosjleeper@gmail.com>

Repository CRAN

Date/Publication 2021-05-16 14:10:13 UTC

R topics documented:

UNF-package *Tools for creating universal numeric fingerprints for data*

Description

Computes a univeral numeric fingerprint of data objects.

Details

This package calculates a Universal Numeric Fingerprint (UNF) on an R data object. UNF is a crypographic hash or signature that can be used to uniquely identify a (version of a) dataset, or a subset thereof. UNF is used by the Dataverse archives and this package can be used to verify a dataset against one listed available in a Dataverse study (e.g., as returned by the dataverse and dvn packages).

A UNF is created by rounding data values (or truncating strings) to a known number of digits (or characters), representing those values in a standard form (as 8-bit [for versions 4.1 and 5] or 32-bit [for versions 3 and 4] unicode-formatted strings), and applying a fingerprinting method (a cryptographic hashing function) to this representation (md5 for versions 3 and 4 or sha256 for versions 4.1, 5, and 6). UNFs are computed from data values (independent of variable naming and column arrangement), so they directly reflect the internal representation of the data.

A UNF differs from an ordinary file checksum in several important ways:

1. *UNFs are format independent.* The UNF for a dataset will be the same regardless of whether the data is saved as a R binary format, SAS formatted file, Stata formatted file, etc., but file checksums will differ. The UNF is also independent of variable arrangement and naming, which can be unintentionally changed during file reading.
2. *UNFs are robust to insignificant rounding error.* This important when dealing with floating-point numeric values. A UNF will also be the same if the data differs in non-significant digits, a file checksum not.
3. *UNFs detect misinterpretation of the data by the statistical software.* If the statistical software misreads the file, the resulting UNF will not match the original, but the file checksums may match. For example, numeric values read as character will produce a different UNF than those values read in as numerics.
4. *UNFs are strongly tamper resistant.* Any accidental or intentional changes to data values will change the resulting UNF. Most file checksums and descriptive statistics detect only certain types of changes.

Author(s)

Thomas J. Leeper and Micah Altman.

See Also

unf %unf%

as.unfvector *UNF Vector Representation*

Description

Standardize a vector according to UNF specifications

Usage

```
as.unfvector(x, ...)
```

Arguments

x	A vector to be coerced to a character string representation according to the UNF specification.
...	Additional arguments passed to methods.

Details

The UNF specifications describes how to coerce all R data types to a standardized character representation. This S3 method exposes that coercion functionality.

Value

A character string with class "unfvector" manipulated to follow the UNF specification. These are used internally by unf6.

Author(s)

Thomas J. Leeper (<thosjleeper@gmail.com>)

See Also

unf, unf6, %unf%

signifz *Round values to specified number of significant digits*

Description

Rounds the value to a specified number of significant digits, using IEEE 754 rounding towards zero rounding mode.

Usage

```
signifz(x, digits = 6)
```

Arguments

x	A numeric vector.
digits	An integer indicating the precision to be used.

Details

This function rounds the values in its first argument to the specified number of significant digits, using IEC 60559/IEEE 754 "round towards zero" mode. This is an alternative to the round function, which rounds toward nearest, ties to even. This is designed to be used internally by unf3 and unf4 (though the original implementations do not seem to actually use the function). unf5 uses round instead. Rounding toward zero assures that signifz(signifz(x,digits=m),digits=n) = signifz(x,digits=n) for $m>n$, an important property for creating approximate fingerprints. It can, however, produce more rounding error than rounding toward nearest. The maximum log relative error (LRE) for the former is (digits-1) while the maximum LRE for the latter is 'digits'. Hence, you may wish to use one more significant digit with signifz than with signif.

Author(s)

Micah Altman

See Also

signif, unf

Examples

```
# note the difference
signif(pi,digits=5)
signifz(pi,digits=5)
```

---

unf *Universal Numeric Fingerprint*

Description

UNF is a cryptographic hash or signature that can be used to uniquely identify (a version of) a dataset, or a subset thereof.

Usage

```
unf(x, version = 6, ...)

unf3(
  x,
  digits = 7L,
  characters = 128L,
```

```
  factor_as_character = TRUE,
  nonfinites_as_missing = FALSE,
  empty_character_as_missing = FALSE,
  dvn_zero = FALSE,
  ...
)

unf4(
  x,
  digits = 7L,
  characters = 128L,
  truncation = 128L,
  version = 4,
  factor_as_character = TRUE,
  nonfinites_as_missing = FALSE,
  empty_character_as_missing = FALSE,
  dvn_zero = FALSE,
  ...
)

unf5(
  x,
  digits = 7L,
  characters = 128L,
  truncation = 128L,
  raw_as_character = TRUE,
  factor_as_character = TRUE,
  nonfinites_as_missing = FALSE,
  empty_character_as_missing = FALSE,
  dvn_zero = FALSE,
  timezone = "",
  date_format = "%Y-%m-%d",
  decimal_seconds = 5,
  ...
)

unf6(
  x,
  digits = 7L,
  characters = 128L,
  truncation = 128L,
  raw_as_character = TRUE,
  factor_as_character = TRUE,
  complex_as_character = TRUE,
  nonfinites_as_missing = FALSE,
  timezone = "",
  date_format = "%Y-%m-%d",
  decimal_seconds = 5,
```

```
  ...
)
```

Arguments

x	For unf, a vector, matrix, dataframe, or list; for unf3, unf4, unf5, a vector. If x is a dataframe or list with one variable or one vector element, respectively, unf returns the UNF for the single vector (which is consistent with the Dataverse implementation but ambiguous in the UNF standard). For algorithm versions < 5, all non-numeric vectors are treated as character.
version	Version of the UNF algorithm. Allowed values are 3, 4, 4.1, 5, and 6. Always use the same version of the algorithm to check a UNF. Default for unf is 6 and default for unf4 is 4 (but can also be set to 4.1, which is identical except for using SHA256 instead of MD5).
digits	The number of significant digits for rounding for numeric values. Default is 7L. Must be between 1 and 15.
characters	The number of characters for truncation. Default is 128L. Must be greater than 1.
factor_as_character	A logical indicating whether to treat an factors as character. If FALSE, factor variables are treated as integer (and thus handled as any numeric value).
nonfinites_as_missing	A logical indicating whether to treat nonfinite values (NaN, Inf, -Inf) as NA. This is supplied to create compatibility with a Dataverse UNFv5 implementation.
empty_character_as_missing	A logical indicating whether to treat an empty character string as a missing value. This is supplied to create compatibility with a Dataverse UNFv5 implementation.
dvn_zero	A logical indicating whether to format a zero (0) numeric value as +0.e-6 instead of the default +0.e+. This is supplied to create compatibility with a Dataverse UNFv5 implementation, backwards compatibility with v1.0 of the UNF package (for UNFv3, UNFv4, UNFv4.1).
truncation	The number of bits to truncate the UNF signature to. Default is 128L. Must be one of: 128,192,196,256.
raw_as_character	A logical indicating whether to format raw vectors as character.
timezone	A character string containing a valid timezone. This is used for formatting "Date" and "POSIXt" class variables. Because of different implementations of datetime classes across computer applications, UNF signatures may vary due to the timezone in which they are calculated. This parameter allows for the comparison of UNFs calculated in different timezones.
date_format	A character string containing a formatting pattern for "Date" class variables. One of '%Y-%m-%d' (the default), '%Y-%m', '%Y', '%F'.
decimal_seconds	A number indicating the number of decimal places to round fractional seconds to. The UNF specification (and default) is 5.

`complex_as_character`
: A logical indicating whether to format raw vectors as character. If `TRUE`, UNF should match Dataverse UNFv5 implementation. If `FALSE`, complex numbers are formatted as `A,iB`.

`...`
: Additional arguments passed to specific algorithm functions. Ignored.

Details

The Dataverse Network implements a potentially incorrect version of the UNF algorithm with regard to the handling of zero values and logical `FALSE` values in data (though the specification is unclear). Setting the `dvn` argument to `TRUE` (the default), uses the Dataverse implementation (for comparison to files stored in that archive).

Value

The `unf` function returns a list of class `UNF`, containing:

- `unf`: A character string containing the universal numeric fingerprint.
- `hash`: A raw vector expressing the unencoded universal numeric fingerprint. This can be converted to a UNF using `base64Encode`.
- `unflong`: For `unf5`, a character string containing the un-truncated universal numeric fingerprint.
- `formatted`: A character string containing the formatted UNF, including version number and header attributes.

The object additionally contains several attributes:

- `version`: A one-element numeric vector specifying which version of the UNF algorithm was used to generate the object.
- `digits`: A one-element numeric vector specifying how many significant digits were used in rounding numeric values.
- `characters`: A one-element numeric vector specifying how many characters were preserved during truncation of character values.
- `truncation`: A one-element numeric vector specifying how many bits the UNF hash was truncated to.

The default print method displays the UNF along with these attributes. For example: `UNF:3:4,128:ZNQRI14053UZq389x0Bf`
This representation identifies the signature as UNF, using version 3 of the algorithm, computed to 4 significant digits for numbers and 128 for characters. The segment following the final colon is the actual fingerprint in base64-encoded format.

References

https://guides.dataverse.org/en/latest/developers/unf/index.html

Altman, M., J. Gill and M. P. McDonald. 2003. *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley \& Sons. [Describes version 3 of the algorithm]

Altman, M., \& G. King. 2007. A Proposed Standard for the Scholarly Citation of Quantitative Data. *D-Lib* 13(3/4). http://dlib.org/dlib/march07/altman/03altman.html [Describes a citation standard using UNFs]

Altman, M. 2008. A Fingerprint Method for Scientific Data Verification. In T. Sobh, editor, Advances in Computer and Information Sciences and Engineering, chapter 57, pages 311–316. Springer Netherlands, Netherlands, 2008. https://link.springer.com/chapter/10.1007/978-1-4020-8741-7_57 [Describes version 5 of the algorithm]

Data Citation Synthesis Group. 2013. Declaration of Data Citation Principles [DRAFT]. https://www.force11.org/datacitationprinciples. [Describes general principles of data citation, of which UNF is likely to be a part]

See Also

%unf%

Examples

```
# Version 6 #

### FORTHCOMING ###

# Version 5 #
## vectors

### just numerics
unf5(1:20) # UNF:5:/FIOZM/29oC3TK/IE52m2A==
unf5(-3:3, dvn_zero = TRUE) # UNF:5:pwzm1tdPaqypPWRWDeW6Jw==

### characters and factors
unf5(c('test','1','2','3')) # UNF:5:fH4NJMYkaAJ16OWMEE+zpQ==
unf5(as.factor(c('test','1','2','3'))) # UNF:5:fH4NJMYkaAJ16OWMEE+zpQ==

### logicals
unf5(c(TRUE,TRUE,FALSE), dvn_zero=TRUE)# UNF:5:DedhGlU7W6o2CBelrIZ3iw==

### missing values
unf5(c(1:5,NA)) # UNF:5:Msnz4m7QVvqBUWxxrE7kNQ==

## variable order and object structure is irrelevant
unf(data.frame(1:3,4:6,7:9)) # UNF:5:ukDZSJXck7fn4SlPJMPFTQ==
unf(data.frame(7:9,1:3,4:6))
unf(list(1:3,4:6,7:9))

# Version 4 #
# version 4
data(longley)
unf(longley, ver=4, digits=3) # PjAV6/R6Kdg0urKrDVDzfMPWJrsBn5FfOdZVr9W8Ybg=

# version 4.1
unf(longley, ver=4.1, digits=3) # 8nzEDWbNacXlv5Zypp+3YCQgMao/eNusOv/u5GmBj9I=

# Version 3 #
x1 <- 1:20
x2 <- x1 + .00001
```

```
unf3(x1) # HRSmPi9QZzlIA+KwmDNP8w==
unf3(x2) # OhFpUw1lrpTE+csF30Ut4Q==

# UNFs are identical at specified level of rounding
identical(unf3(x1), unf3(x2))
identical(unf3(x1, digits=5),unf3(x2, digits=5))

# dataframes, matrices, and lists are all treated identically:
unf(cbind.data.frame(x1,x2),ver=3) # E8+DS5SG4CSoM7j8KAkC9A==
unf(list(x1,x2), ver=3)
unf(cbind(x1,x2), ver=3)
```

`%unf%` *Compare two objects*

Description

Function to compare the size, structure, arrangement, and UNFs of two objects.

Usage

```
x %unf% y

unf_equal(x, y, ...)
```

Arguments

`x`	A vector, matrix, dataframe, list, or object of class "UNF", or a one-element character vector containing a UNF signature.
`y`	A vector, matrix, dataframe, list, or object of class "UNF", or a one-element character vector containing a UNF signature.
`...`	Additional arguments passed to `unf`.

Details

Compares two objects using `all.equal` and additional details based on the UNF of the two objects (and, for lists, dataframes, and matrices) the constituent vectors thereof. The print method for class `UNFtest` prints the UNFs for both objects and summarizes any differences between the objects. This is helpful for identifying mismatching variables.

Value

An object of class `UNFtest` containing the results of `unf` for both objects and both `identical` and `all.equal` for the comparison of the two.

Author(s)

Thomas J. Leeper

See Also

unf

Examples

```
a <- data.frame(x1=1:10, x2=11:20)
b <- data.frame(x1=1:10, x2=11:20+.0005)
a %unf% a
try(a %unf% b)
unf_equal(a, b, digits = 3)

unf(a) %unf% "UNF6:aKW4lAFNBH8vfrnrDbQZjg=="
```

Index