Package 'haven'

May 30, 2025

Title Import and Export 'SPSS', 'Stata' and 'SAS' Files

Version 2.5.5

Description Import foreign statistical formats into R via the embedded
'ReadStat' C library, <https://github.com/WizardMac/ReadStat>.

License MIT + file LICENSE

URL https://haven.tidyverse.org, https://github.com/tidyverse/haven,
https://github.com/WizardMac/ReadStat

BugReports https://github.com/tidyverse/haven/issues

Depends R (>= 3.6)

Imports cli (>= 3.0.0), forcats (>= 0.2.0), hms, lifecycle, methods,
readr (>= 0.1.0), rlang (>= 0.4.0), tibble, tidyselect, vctrs
(>= 0.3.0)

Suggests covr, crayon, fs, knitr, pillar (>= 1.4.0), rmarkdown,
testthat (>= 3.0.0), utf8

LinkingTo cpp11

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements GNU make, zlib: zlib1g-dev (deb), zlib-devel (rpm)

NeedsCompilation yes

Author Hadley Wickham [aut, cre],
Evan Miller [aut, cph] (Author of included ReadStat code),
Danny Smith [aut],
Posit Software, PBC [cph, fnd]

Maintainer Hadley Wickham <hadley@posit.co>

Repository CRAN

Date/Publication 2025-05-30 14:10:02 UTC

Contents

---

as_factor *Convert labelled vectors to factors*

Description

The base function as.factor() is not a generic, but forcats::as_factor() is. haven provides as_factor() methods for labelled() and labelled_spss() vectors, and data frames. By default, when applied to a data frame, it only affects labelled columns.

Usage

```
## S3 method for class 'data.frame'
as_factor(x, ..., only_labelled = TRUE)

## S3 method for class 'haven_labelled'
as_factor(
  x,
  levels = c("default", "labels", "values", "both"),
  ordered = FALSE,
  ...
)

## S3 method for class 'labelled'
as_factor(
  x,
  levels = c("default", "labels", "values", "both"),
  ordered = FALSE,
  ...
)
```

Arguments

`x`	Object to coerce to a factor.
`...`	Other arguments passed down to method.
`only_labelled`	Only apply to labelled columns?
`levels`	How to create the levels of the generated factor: • "default": uses labels where available, otherwise the values. Labels are sorted by value. • "both": like "default", but pastes together the level and value • "label": use only the labels; unlabelled values become `NA` • "values": use only the values
`ordered`	If `TRUE` create an ordered (ordinal) factor, if `FALSE` (the default) create a regular (nominal) factor.

Details

Includes methods for both class `haven_labelled` and `labelled` for backward compatibility.

Examples

```
x <- labelled(sample(5, 10, replace = TRUE), c(Bad = 1, Good = 5))

# Default method uses values where available
as_factor(x)
# You can also extract just the labels
as_factor(x, levels = "labels")
# Or just the values
as_factor(x, levels = "values")
# Or combine value and label
as_factor(x, levels = "both")

# as_factor() will preserve SPSS missing values from values and ranges
y <- labelled_spss(1:10, na_values = c(2, 4), na_range = c(8, 10))
as_factor(y)
# use zap_missing() first to convert to NAs
zap_missing(y)
as_factor(zap_missing(y))
```

`labelled` *Create a labelled vector.*

Description

A labelled vector is a common data structure in other statistical environments, allowing you to assign text labels to specific values. This class makes it possible to import such labelled vectors in to R without loss of fidelity. This class provides few methods, as I expect you'll coerce to a standard R class (e.g. a `factor()`) soon after importing.

Usage

```
labelled(x = double(), labels = NULL, label = NULL)

is.labelled(x)
```

Arguments

`x`	A vector to label. Must be either numeric (integer or double) or character.
`labels`	A named vector or `NULL`. The vector should be the same type as `x`. Unlike factors, labels don't need to be exhaustive: only a fraction of the values might be labelled.
`label`	A short, human-readable description of the vector.

Examples

```
s1 <- labelled(c("M", "M", "F"), c(Male = "M", Female = "F"))
s2 <- labelled(c(1, 1, 2), c(Male = 1, Female = 2))
s3 <- labelled(
  c(1, 1, 2),
  c(Male = 1, Female = 2),
  label = "Assigned sex at birth"
)

# Unfortunately it's not possible to make as.factor work for labelled objects
# so instead use as_factor. This works for all types of labelled vectors.
as_factor(s1)
as_factor(s1, levels = "values")
as_factor(s2)

# Other statistical software supports multiple types of missing values
s3 <- labelled(
  c("M", "M", "F", "X", "N/A"),
  c(Male = "M", Female = "F", Refused = "X", "Not applicable" = "N/A")
)
s3
as_factor(s3)

# Often when you have a partially labelled numeric vector, labelled values
# are special types of missing. Use zap_labels to replace labels with missing
# values
x <- labelled(c(1, 2, 1, 2, 10, 9), c(Unknown = 9, Refused = 10))
zap_labels(x)
```

`labelled_spss` *Labelled vectors for SPSS*

Description

This class is only used when `user_na = TRUE` in `read_sav()`. It is similar to the `labelled()` class but it also models SPSS's user-defined missings, which can be up to three distinct values, or for numeric vectors a range.

Usage

```
labelled_spss(
  x = double(),
  labels = NULL,
  na_values = NULL,
  na_range = NULL,
  label = NULL
)
```

Arguments

`x`	A vector to label. Must be either numeric (integer or double) or character.
`labels`	A named vector or `NULL`. The vector should be the same type as `x`. Unlike factors, labels don't need to be exhaustive: only a fraction of the values might be labelled.
`na_values`	A vector of values that should also be considered as missing.
`na_range`	A numeric vector of length two giving the (inclusive) extents of the range. Use `-Inf` and `Inf` if you want the range to be open ended.
`label`	A short, human-readable description of the vector.

Examples

```
x1 <- labelled_spss(1:10, c(Good = 1, Bad = 8), na_values = c(9, 10))
is.na(x1)

x2 <- labelled_spss(
  1:10,
  c(Good = 1, Bad = 8),
  na_range = c(9, Inf),
  label = "Quality rating"
)
is.na(x2)

# Print data and metadata
x2
```

print_labels — *Print the labels of a labelled vector*

Description

This is a convenience function, useful to explore the variables of a newly imported dataset.

Usage

```
print_labels(x, name = NULL)
```

Arguments

`x`	A labelled vector
`name`	The name of the vector (optional)

Examples

```
s1 <- labelled(c("M", "M", "F"), c(Male = "M", Female = "F"))
s2 <- labelled(c(1, 1, 2), c(Male = 1, Female = 2))
labelled_df <- tibble::tibble(s1, s2)

for (var in names(labelled_df)) {
  print_labels(labelled_df[[var]], var)
}
```

read_dta — *Read and write Stata DTA files*

Description

Currently haven can read and write logical, integer, numeric, character and factors. See `labelled()` for how labelled variables in Stata are handled in R.

Character vectors will be stored as `strL` if any components are `strl_threshold` bytes or longer (and `version` >= 13); otherwise they will be stored as the appropriate `str#`.

Usage

```
read_dta(
  file,
  encoding = NULL,
  col_select = NULL,
  skip = 0,
  n_max = Inf,
  .name_repair = "unique"
)

read_stata(
  file,
  encoding = NULL,
  col_select = NULL,
  skip = 0,
  n_max = Inf,
  .name_repair = "unique"
)

write_dta(
  data,
  path,
  version = 14,
  label = attr(data, "label"),
  strl_threshold = 2045,
  adjust_tz = TRUE
)
```

Arguments

`file`	Either a path to a file, a connection, or literal data (either a single string or a raw vector). Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically uncompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote gz files can also be automatically downloaded and decompressed. Literal data is most useful for examples and tests. To be recognised as literal data, the input must be either wrapped with `I()`, be a string containing at least one new line, or be a vector containing at least one string with a new line. Using a value of `clipboard()` will read from the system clipboard.
`encoding`	The character encoding used for the file. Generally, only needed for Stata 13 files and earlier. See Encoding section for details.
`col_select`	One or more selection expressions, like in `dplyr::select()`. Use `c()` or `list()` to use more than one expression. See `?dplyr::select` for details on available selection options. Only the specified columns will be read from `data_file`.
`skip`	Number of lines to skip before reading data.
`n_max`	Maximum number of lines to read.
`.name_repair`	Treatment of problematic column names: • `"minimal"`: No name repair or checks, beyond basic existence, • `"unique"`: Make sure names are unique and not empty, • `"check_unique"`: (default value), no name repair, but check they are `unique`, • `"universal"`: Make the names `unique` and syntactic

- a function: apply custom name repair (e.g., `.name_repair = make.names` for names in the style of base R).
- A purrr-style anonymous function, see `rlang::as_function()`

This argument is passed on as `repair` to `vctrs::vec_as_names()`. See there for more details on these terms and the strategies used to enforce them.

`data`	Data frame to write.
`path`	Path to a file where the data will be written.
`version`	File version to use. Supports versions 8-15.
`label`	Dataset label to use, or `NULL`. Defaults to the value stored in the "label" attribute of `data`. Must be <= 80 characters.
`strl_threshold`	Any character vectors with a maximum length greater than `strl_threshold` bytes will be stored as a long string (strL) instead of a standard string (str#) variable if `version` >= 13. This defaults to 2045, the maximum length of str# variables. See the Stata long string documentation for more details.
`adjust_tz`	Stata, SPSS and SAS do not have a concept of time zone, and all date-time variables are treated as UTC. `adjust_tz` controls how the timezone of date-time values is treated when writing. • If `TRUE` (the default) the timezone of date-time values is ignored, and they will display the same in R and Stata/SPSS/SAS, e.g. `"2010-01-01 09:00:00 NZDT"` will be written as `"2010-01-01 09:00:00"`. Note that this changes the underlying numeric data, so use caution if preserving between-time-point differences is critical. • If `FALSE`, date-time values are written as the corresponding UTC value, e.g. `"2010-01-01 09:00:00 NZDT"` will be written as `"2009-12-31 20:00:00"`.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

If a dataset label is defined in Stata, it will stored in the "label" attribute of the tibble.

`write_dta()` returns the input `data` invisibly.

Character encoding

Prior to Stata 14, files did not declare a text encoding, and the default encoding differed across platforms. If `encoding = NULL`, haven assumes the encoding is windows-1252, the text encoding used by Stata on Windows. Unfortunately Stata on Mac and Linux use a different default encoding, "latin1". If you encounter an error such as "Unable to convert string to the requested encoding", try `encoding = "latin1"`

For Stata 14 and later, you should not need to manually specify `encoding` value unless the value was incorrectly recorded in the source file.

Examples

```
path <- system.file("examples", "iris.dta", package = "haven")
read_dta(path)

tmp <- tempfile(fileext = ".dta")
write_dta(mtcars, tmp)
read_dta(tmp)
read_stata(tmp)
```

`read_sas` *Read SAS files*

Description

`read_sas()` supports both sas7bdat files and the accompanying sas7bcat files that SAS uses to record value labels.

Usage

```
read_sas(
  data_file,
  catalog_file = NULL,
  encoding = NULL,
  catalog_encoding = encoding,
  col_select = NULL,
  skip = 0L,
  n_max = Inf,
  cols_only = deprecated(),
  .name_repair = "unique"
)
```

Arguments

`data_file`, `catalog_file`	Path to data and catalog files. The files are processed with `readr::datasource()`.
`encoding`, `catalog_encoding`	The character encoding used for the `data_file` and `catalog_encoding` respectively. A value of `NULL` uses the encoding specified in the file; use this argument to override it if it is incorrect.
`col_select`	One or more selection expressions, like in `dplyr::select()`. Use `c()` or `list()` to use more than one expression. See `?dplyr::select` for details on available selection options. Only the specified columns will be read from `data_file`.
`skip`	Number of lines to skip before reading data.
`n_max`	Maximum number of lines to read.
`cols_only`	**[Deprecated]** `cols_only` is no longer supported; use `col_select` instead.

`.name_repair` Treatment of problematic column names:

- `"minimal"`: No name repair or checks, beyond basic existence,
- `"unique"`: Make sure names are unique and not empty,
- `"check_unique"`: (default value), no name repair, but check they are `unique`,
- `"universal"`: Make the names `unique` and syntactic
- a function: apply custom name repair (e.g., `.name_repair = make.names` for names in the style of base R).
- A purrr-style anonymous function, see `rlang::as_function()`

This argument is passed on as `repair` to `vctrs::vec_as_names()`. See there for more details on these terms and the strategies used to enforce them.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

`write_sas()` returns the input `data` invisibly.

Examples

```
path <- system.file("examples", "iris.sas7bdat", package = "haven")
read_sas(path)
```

`read_spss` *Read and write SPSS files*

Description

`read_sav()` reads both `.sav` and `.zsav` files; `write_sav()` creates `.zsav` files when `compress = TRUE`. `read_por()` reads `.por` files. `read_spss()` uses either `read_por()` or `read_sav()` based on the file extension.

Usage

```
read_sav(
  file,
  encoding = NULL,
  user_na = FALSE,
  col_select = NULL,
  skip = 0,
  n_max = Inf,
  .name_repair = "unique"
)

read_por(
    file,
    user_na = FALSE,
    col_select = NULL,
    skip = 0,
    n_max = Inf,
    .name_repair = "unique"
  )

  write_sav(data, path, compress = c("byte", "none", "zsav"), adjust_tz = TRUE)

  read_spss(
    file,
    user_na = FALSE,
    col_select = NULL,
    skip = 0,
    n_max = Inf,
    .name_repair = "unique"
  )
```

Arguments

`file`	Either a path to a file, a connection, or literal data (either a single string or a raw vector). Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically uncompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote gz files can also be automatically downloaded and decompressed. Literal data is most useful for examples and tests. To be recognised as literal data, the input must be either wrapped with `I()`, be a string containing at least one new line, or be a vector containing at least one string with a new line. Using a value of `clipboard()` will read from the system clipboard.
`encoding`	The character encoding used for the file. The default, `NULL`, use the encoding specified in the file, but sometimes this value is incorrect and it is useful to be able to override it.
`user_na`	If `TRUE` variables with user defined missing will be read into `labelled_spss()` objects. If `FALSE`, the default, user-defined missings will be converted to `NA`.
`col_select`	One or more selection expressions, like in `dplyr::select()`. Use `c()` or `list()` to use more than one expression. See `?dplyr::select` for details on available selection options. Only the specified columns will be read from `data_file`.
`skip`	Number of lines to skip before reading data.
`n_max`	Maximum number of lines to read.
`.name_repair`	Treatment of problematic column names: • `"minimal"`: No name repair or checks, beyond basic existence, • `"unique"`: Make sure names are unique and not empty, • `"check_unique"`: (default value), no name repair, but check they are `unique`,

- "universal": Make the names unique and syntactic
- a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R).
- A purrr-style anonymous function, see rlang::as_function()

This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.

data	Data frame to write.
path	Path to a file where the data will be written.
compress	Compression type to use: • "byte": the default, uses byte compression. • "none": no compression. This is useful for software that has issues with byte compressed .sav files (e.g. SAS). • "zsav": uses zlib compression and produces a .zsav file. zlib compression is supported by SPSS version 21.0 and above. TRUE and FALSE can be used for backwards compatibility, and correspond to the "zsav" and "none" options respectively.
adjust_tz	Stata, SPSS and SAS do not have a concept of time zone, and all date-time variables are treated as UTC. adjust_tz controls how the timezone of date-time values is treated when writing. • If TRUE (the default) the timezone of date-time values is ignored, and they will display the same in R and Stata/SPSS/SAS, e.g. "2010-01-01 09:00:00 NZDT" will be written as "2010-01-01 09:00:00". Note that this changes the underlying numeric data, so use caution if preserving between-time-point differences is critical. • If FALSE, date-time values are written as the corresponding UTC value, e.g. "2010-01-01 09:00:00 NZDT" will be written as "2009-12-31 20:00:00".

Details

Currently haven can read and write logical, integer, numeric, character and factors. See labelled_spss() for how labelled variables in SPSS are handled in R.

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

write_sav() returns the input data invisibly.

Examples

```
path <- system.file("examples", "iris.sav", package = "haven")
read_sav(path)

tmp <- tempfile(fileext = ".sav")
write_sav(mtcars, tmp)
read_sav(tmp)
```

read_xpt *Read and write SAS transport files*

Description

The SAS transport format is a open format, as is required for submission of the data to the FDA.

Usage

```
read_xpt(
  file,
  col_select = NULL,
  skip = 0,
  n_max = Inf,
  .name_repair = "unique"
)

write_xpt(
  data,
  path,
  version = 8,
  name = NULL,
  label = attr(data, "label"),
  adjust_tz = TRUE
)
```

Arguments

`file`	Either a path to a file, a connection, or literal data (either a single string or a raw vector). Files ending in `.gz`, `.bz2`, `.xz`, or `.zip` will be automatically uncompressed. Files starting with `http://`, `https://`, `ftp://`, or `ftps://` will be automatically downloaded. Remote gz files can also be automatically downloaded and decompressed. Literal data is most useful for examples and tests. To be recognised as literal data, the input must be either wrapped with `I()`, be a string containing at least one new line, or be a vector containing at least one string with a new line. Using a value of `clipboard()` will read from the system clipboard.
`col_select`	One or more selection expressions, like in `dplyr::select()`. Use `c()` or `list()` to use more than one expression. See `?dplyr::select` for details on available selection options. Only the specified columns will be read from `data_file`.
`skip`	Number of lines to skip before reading data.
`n_max`	Maximum number of lines to read.
`.name_repair`	Treatment of problematic column names:

	• "minimal": No name repair or checks, beyond basic existence, • "unique": Make sure names are unique and not empty, • "check_unique": (default value), no name repair, but check they are unique, • "universal": Make the names unique and syntactic • a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R). • A purrr-style anonymous function, see rlang::as_function() This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.
data	Data frame to write.
path	Path to a file where the data will be written.
version	Version of transport file specification to use: either 5 or 8.
name	Member name to record in file. Defaults to file name sans extension. Must be <= 8 characters for version 5, and <= 32 characters for version 8.
label	Dataset label to use, or NULL. Defaults to the value stored in the "label" attribute of data. Note that although SAS itself supports dataset labels up to 256 characters long, dataset labels in SAS transport files must be <= 40 characters.
adjust_tz	Stata, SPSS and SAS do not have a concept of time zone, and all date-time variables are treated as UTC. adjust_tz controls how the timezone of date-time values is treated when writing. • If TRUE (the default) the timezone of date-time values is ignored, and they will display the same in R and Stata/SPSS/SAS, e.g. "2010-01-01 09:00:00 NZDT" will be written as "2010-01-01 09:00:00". Note that this changes the underlying numeric data, so use caution if preserving between-time-point differences is critical. • If FALSE, date-time values are written as the corresponding UTC value, e.g. "2010-01-01 09:00:00 NZDT" will be written as "2009-12-31 20:00:00".

Value

A tibble, data frame variant with nice defaults.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

If a dataset label is defined, it will be stored in the "label" attribute of the tibble.

write_xpt() returns the input data invisibly.

Examples

```
tmp <- tempfile(fileext = ".xpt")
write_xpt(mtcars, tmp)
read_xpt(tmp)
```

`tagged_na` *"Tagged" missing values*

Description

"Tagged" missing values work exactly like regular R missing values except that they store one additional byte of information a tag, which is usually a letter ("a" to "z"). When by loading a SAS and Stata file, the tagged missing values always use lower case values.

Usage

```
tagged_na(...)

na_tag(x)

is_tagged_na(x, tag = NULL)

format_tagged_na(x, digits = getOption("digits"))

print_tagged_na(x, digits = getOption("digits"))
```

Arguments

`...`	Vectors containing single character. The letter will be used to "tag" the missing value.
`x`	A numeric vector
`tag`	If `NULL`, will only return true if the tag has this value.
`digits`	Number of digits to use in string representation

Details

`format_tagged_na()` and `print_tagged_na()` format tagged NA's as NA(a), NA(b), etc.

Examples

```
x <- c(1:5, tagged_na("a"), tagged_na("z"), NA)

# Tagged NA's work identically to regular NAs
x
is.na(x)

# To see that they're special, you need to use na_tag(),
# is_tagged_na(), or print_tagged_na():
is_tagged_na(x)
na_tag(x)
print_tagged_na(x)
```

```
# You can test for specific tagged NAs with the second argument
is_tagged_na(x, "a")

# Because the support for tagged's NAs is somewhat tagged on to R,
# the left-most NA will tend to be preserved in arithmetic operations.
na_tag(tagged_na("a") + tagged_na("z"))
```

`zap_empty` *Convert empty strings into missing values*

Description

Convert empty strings into missing values

Usage

```
zap_empty(x)
```

Arguments

`x`	A character vector

Value

A character vector with empty strings replaced by missing values.

See Also

Other zappers: `zap_formats()`, `zap_label()`, `zap_labels()`, `zap_widths()`

Examples

```
x <- c("a", "", "c")
zap_empty(x)
```

`zap_formats` *Remove format attributes*

Description

To provide some mild support for round-tripping variables between Stata/SPSS and R, haven stores variable formats in an attribute: `format.stata`, `format.spss`, or `format.sas`. If this causes problems for your code, you can get rid of them with `zap_formats`.

Usage

```
zap_formats(x)
```

Arguments

x	A vector or data frame.

See Also

Other zappers: zap_empty(), zap_label(), zap_labels(), zap_widths()

---

zap_label *Zap variable labels*

Description

Removes variable label, leaving unlabelled vectors as is.

Usage

```
zap_label(x)
```

Arguments

x	A vector or data frame

See Also

zap_labels() to remove value labels.

Other zappers: zap_empty(), zap_formats(), zap_labels(), zap_widths()

Examples

```
x1 <- labelled(1:5, c(good = 1, bad = 5), label = "rating")
x1
zap_label(x1)

x2 <- labelled_spss(c(1:4, 9), label = "score", na_values = 9)
x2
zap_label(x2)

# zap_label also works with data frames
df <- tibble::tibble(x1, x2)
str(df)
str(zap_label(df))
```

zap_labels *Zap value labels*

Description

Removes value labels, leaving unlabelled vectors as is. Use this if you want to simply drop all `labels` from a data frame.

Zapping labels from `labelled_spss()` also removes user-defined missing values by default, replacing with standard `NA`s. Use the `user_na` argument to override this behaviour.

Usage

```
zap_labels(x, ...)

## S3 method for class 'haven_labelled_spss'
zap_labels(x, ..., user_na = FALSE)
```

Arguments

`x`	A vector or data frame
`...`	Other arguments passed down to method.
`user_na`	If `FALSE`, the default, `zap_labels()` will convert `labelled_spss()` user-defined missing values to `NA`. If `TRUE` they will be treated like normal values.

See Also

`zap_label()` to remove variable labels.

Other zappers: `zap_empty()`, `zap_formats()`, `zap_label()`, `zap_widths()`

Examples

```
x1 <- labelled(1:5, c(good = 1, bad = 5))
x1
zap_labels(x1)

x2 <- labelled_spss(c(1:4, 9), c(good = 1, bad = 5), na_values = 9)
x2
zap_labels(x2)

# Keep the user defined missing values
zap_labels(x2, user_na = TRUE)

# zap_labels also works with data frames
df <- tibble::tibble(x1, x2)
df
zap_labels(df)
```

zap_missing — *Zap special missings to regular R missings*

Description

This is useful if you want to convert tagged missing values from SAS or Stata, or user-defined missings from SPSS, to regular R `NA`.

Usage

```
zap_missing(x)
```

Arguments

x	A vector or data frame

Examples

```
x1 <- labelled(
  c(1, 5, tagged_na("a", "b")),
  c(Unknown = tagged_na("a"), Refused = tagged_na("b"))
)
x1
zap_missing(x1)

x2 <- labelled_spss(
  c(1, 2, 1, 99),
  c(missing = 99),
  na_value = 99
)
x2
zap_missing(x2)

# You can also apply to data frames
df <- tibble::tibble(x1, x2, y = 4:1)
df
zap_missing(df)
```

zap_widths — *Remove display width attributes*

Description

To provide some mild support for round-tripping variables between SPSS and R, haven stores display widths in an attribute: `display_width`. If this causes problems for your code, you can get rid of them with `zap_widths`.

Usage

```
zap_widths(x)
```

Arguments

`x`	A vector or data frame.

See Also

Other zappers: `zap_empty()`, `zap_formats()`, `zap_label()`, `zap_labels()`

Index