Package: ksformat (via r-universe)

June 4, 2026

Type Package

Title 'SAS'-Style 'PROC FORMAT' for R

Version 0.8.1

Author Vladimir Larchenko [aut, cre], Igor Aleschenkov [aut]

Maintainer Vladimir Larchenko

<vladimir.larchenko@keystatsolutions.com>

Description Provides 'SAS' 'PROC FORMAT'-like functionality for creating and applying value formats in R. Supports discrete and range-based mapping of values to labels, reverse formatting (invalue), date/time/datetime formatting with built-in 'SAS' format names, multi-label formats, expression labels evaluated at apply-time, case-insensitive matching, import/export of format definitions, and proper handling of missing values (NA, NULL, NaN).

URL https://github.com/crow16384/ksformat

BugReports https://github.com/crow16384/ksformat/issues

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cli

Suggests dplyr, knitr, rmarkdown, shiny, testthat (>= 3.0.0), tidyr

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Repository https://crow16384.r-universe.dev

Date/Publication 2026-06-04 17:01:47 UTC

RemoteUrl https://github.com/crow16384/ksformat

RemoteRef HEAD

RemoteSha 59fbf3490cc6dc3f3b09a94f014fc2e8391e1dde

Contents

`ksformat-package` *ksformat: 'SAS'-Style 'PROC FORMAT' for R*

Description

Provides 'SAS' 'PROC FORMAT'-like functionality for creating and applying value formats in R. The package supports mapping values to labels, range-based formatting, reverse formatting (invalue), date/time/datetime formatting, and proper handling of missing values (NA, NULL, NaN).

Details

Format creation:

- `fnew` — create value-to-label mappings (formats)
- `finput` — create reverse mappings (label-to-value invalues)
- `fnew_bid` — create both format and invalue simultaneously
- `fnew_date` — create date/time/datetime formats ('SAS'-style or custom `strftime` patterns)
- `fparse` — parse 'SAS'-like format definitions from text or file
- `fimport` — import formats from a 'SAS' CNTLOUT CSV file
- `e` — mark a label for expression evaluation at apply-time

Format application:

- `fput` — apply a format to a vector (value to label)
- `fputn` — apply a numeric format by name (like 'SAS' PUTN)
- `fputc` — apply a character format by name (like 'SAS' PUTC)
- `fput_all` — apply a multilabel format returning all matching labels
- `fput_df` — apply formats to data frame columns

Reverse formatting:

- `finputn` — apply a numeric invalue by name (like 'SAS' INPUTN)
- `finputc` — apply a character invalue by name (like 'SAS' INPUTC)

Format library:

- `format_get` — retrieve a format from the global library
- `fprint` — list or display registered formats
- `fclear` — remove one or all formats from the library
- `format_library_app` — open interactive library browser (Shiny)
- `fexport` — export formats to 'SAS'-like text

Utilities:

- `is_missing` — check for NA, NaN, and empty strings
- `range_spec` — create a range specification object

Key features:

- *Discrete and range-based* numeric formatting with configurable inclusive/exclusive bounds
- *Multilabel* formats — a value can match multiple labels (`multilabel = TRUE` in `fnew`, retrieved with `fput_all`)
- *Case-insensitive matching* (`ignore_case = TRUE` in `fnew`)
- *Expression labels* — labels containing `.x1`, `.x2`, etc. are evaluated at apply-time; see also `e`
- *Date/time/datetime* formatting with built-in 'SAS' format names (auto-resolved) or custom `strftime` patterns

- *Global format library* with auto-registration and case-insensitive name lookup
- *CNTLOUT import* — read format catalogues exported from 'SAS'

Cheat sheet: run `ksformat_cheatsheet()` to open the HTML version in your browser, or see the files in `system.file("doc", package = "ksformat")`.

Author(s)

Maintainer: Vladimir Larchenko <vladimir.larchenko@keystatsolutions.com>

Authors:

- Vladimir Larchenko <vladimir.larchenko@keystatsolutions.com>
- Igor Aleschenkov <igor.aleschenkov@keystatsolutions.com>

See Also

Source repository and issue tracker: https://github.com/crow16384/ksformat

Examples

```
# Discrete format
fnew("M" = "Male", "F" = "Female", .missing = "Unknown", name = "sex")
fput(c("M", "F", NA), "sex")

# Numeric range format (parsed from text)
fparse(text = '
VALUE age (numeric)
  [0, 18)  = "Child"
  [18, 65) = "Adult"
;
')
fputn(c(5, 25), "age")

# Bidirectional format + invalue
fnew_bid("A" = "Active", "I" = "Inactive", name = "status")
fputc("A", "status")
finputc("Active", "status_inv")

# Multilabel format
ml <- fnew(
  "0,17,TRUE,TRUE" = "Pediatric",
  "18,Inf,TRUE,TRUE" = "Adult",
  "0,Inf,TRUE,TRUE" = "Any Age",
  name = "agegrp", type = "numeric", multilabel = TRUE
)
fput_all(c(10, 30), ml)

# Date format (SAS-style, auto-resolved)
fputn(Sys.Date(), "DATE9.")

# Export and library management
cat(fexport(sex = format_get("sex")))
```

```
flist()          # character vector of registered names
fprint()
fclear()
```

---

e *Mark a Label for Expression Evaluation*

Description

Marks a format label string so it will be evaluated as an R expression at apply-time (fput), even when it does not contain .x1, .x2, etc. placeholders.

Usage

```
e(expr)
```

Arguments

expr	Character string. The R expression to evaluate.

Details

This is useful when a label should call a function that does not need positional .xN arguments. The expression is evaluated in the caller's environment of fput, so user-defined functions are accessible.

Labels containing .x1, .x2, etc. are still evaluated automatically without needing e().

Value

The same character string with an "eval" attribute set to TRUE.

Examples

```
# Mark an expression for evaluation at apply-time
fmt <- fnew(
  "timestamp" = e("format(Sys.time(), '%Y-%m-%d')"),
  "static"    = "Hello",
  name = "demo_eval"
)
fput(c("timestamp", "static"), fmt)
fclear()
```

fclear *Remove Format(s) from Library*

Description

Removes one or all formats from the global format library. When called without arguments, clears all formats. When called with a name, removes only that format.

Usage

```
fclear(name = NULL)
```

Arguments

name	Character. Optional name of a specific format to remove. If NULL (default), removes all formats.

Value

Invisible NULL

Examples

```
fnew("M" = "Male", "F" = "Female", name = "sex")
fclear("sex")   # remove one format
fclear()        # remove all formats
```

fexport *Export Formats to 'SAS'-like Text*

Description

Converts ks_format and/or ks_invalue objects to human-readable 'SAS'-like text representation.

Usage

```
fexport(..., formats = NULL, file = NULL)
```

Arguments

...	Named ks_format or ks_invalue objects to export.
formats	A named list of format objects. Alternative to ....
file	Optional file path to write the output to. If NULL, returns the text as a character string.

Value

If `file` is `NULL`, returns a character string with the 'SAS'-like text. If `file` is specified, writes to the file and returns the path invisibly.

Examples

```
# Export a character format
sex_fmt <- fnew("M" = "Male", "F" = "Female",
                .missing = "Unknown", name = "sex")
cat(fexport(sex = sex_fmt))

# Export a numeric range format
fparse(text = '
VALUE bmi (numeric)
  [0, 18.5)  = "Underweight"
  [18.5, 25) = "Normal"
  [25, 30)   = "Overweight"
  [30, HIGH] = "Obese"
  .missing   = "No data"
;
')
bmi_fmt <- format_get("bmi")
cat(fexport(bmi = bmi_fmt))

# Export a multilabel format
risk_fmt <- fnew(
  "0,3,TRUE,TRUE" = "Low Risk",
  "0,7,TRUE,TRUE" = "Monitored",
  "3,7,FALSE,TRUE" = "Medium Risk",
  "7,10,FALSE,TRUE" = "High Risk",
  name = "risk", type = "numeric", multilabel = TRUE
)
cat(fexport(risk = risk_fmt))

# Export a date format
enrl_fmt <- fnew_date("DATE9.", name = "enrldt", .missing = "Not Enrolled")
cat(fexport(enrldt = enrl_fmt))
fclear()
```

`fimport` *Import Formats from 'SAS' PROC FORMAT CNTLOUT CSV*

Description

Reads a CSV file produced by 'SAS' `PROC FORMAT` with `CNTLOUT=` option (typically exported via `PROC EXPORT`) and converts compatible format definitions into `ks_format` and `ks_invalue` objects.

Usage

```
fimport(file, register = TRUE, overwrite = TRUE)
```

Arguments

file	Path to the CSV file exported from a SAS format catalogue.
register	Logical; if TRUE (default), each imported format is registered in the global format library.
overwrite	Logical; if TRUE (default), existing library entries with the same name are overwritten.

Details

The 'SAS' format catalogue CSV is expected to contain the standard CNTLOUT columns: FMTNAME, START, END, LABEL, TYPE, HLO, SEXCL, EEXCL.

Supported SAS format types:

N Numeric VALUE format → ks_format with type = "numeric"

C Character VALUE format → ks_format with type = "character"

I Numeric INVALUE (informat) → ks_invalue with target_type = "numeric"

J Character INVALUE (informat) → ks_invalue with target_type = "character"

Incompatible types (logged with a warning):

P PICTURE formats − no equivalent in ksformat

Rows with SAS special missing values (.A−.Z, ._) in the HLO field are logged as incompatible entries and skipped because R has no equivalent concept.

Value

A named list of ks_format and ks_invalue objects that were successfully imported. Returned invisibly.

Examples

```
# In SAS:
# proc format library=work cntlout=fmts; run;
# proc export data=fmts outfile="formats.csv" dbms=csv replace; run;

csv_file <- system.file("extdata", "test_cntlout.csv", package = "ksformat")
imported <- fimport(csv_file)
flist()
fprint()
fclear()
```

finput *Create Invalue Format (Reverse Formatting like 'SAS' INVALUE)*

Description

Creates an invalue format that converts formatted labels back to values. This is similar to 'SAS' PROC FORMAT with INVALUE statement. The invalue is automatically stored in the global format library if `name` is provided.

Usage

```
finput(
  ...,
  name = NULL,
  target_type = "numeric",
  missing_value = NA,
  ignore_case = FALSE
)
```

Arguments

`...`	Named arguments defining label-value mappings (reverse of `fnew`), or one or more named vectors/lists using `c(Label = value)`. Example: `"Male" = 1, "Female" = 2` or `c(Male = 1, Female = 2)`.
`name`	Character. Optional name for the invalue format. If provided, the invalue is automatically registered in the global format library.
`target_type`	Character. Type to convert to: `"numeric"` (default), `"integer"`, `"character"`, or `"logical"`. INVALUE formats produce numeric output by default; character-to-character conversion should use a regular VALUE format (`fnew`) instead.
`missing_value`	Value to use for missing inputs (default: `NA`)
`ignore_case`	Logical. If `TRUE`, label matching is case-insensitive (default: `FALSE`).

Value

An object of class `"ks_invalue"` containing the invalue definition. The object is also stored in the format library if `name` is given.

Examples

```
# Convert text labels to numeric codes
finput(
  "Male" = 1,
  "Female" = 2,
  name = "sex_inv"
)

# Apply using finputn (numeric invalue by name)
```

```
finputn(c("Male", "Female", "Unknown"), "sex_inv")
# [1]  1  2 NA
fclear()

# From a named vector
finput(c(Male = 1, Female = 2), name = "sex_inv2")
finputn(c("Male", "Female"), "sex_inv2")
# [1] 1 2
fclear()
```

`finputc` *Apply Character Invalue by Name (like 'SAS' INPUTC)*

Description

Looks up an INVALUE format by name from the global format library and applies it to convert labels to character values.

Usage

```
finputc(x, invalue_name)
```

Arguments

`x`	Character vector of labels to convert
`invalue_name`	Character. Name of a registered INVALUE format.

Value

Character vector

Examples

```
# Bidirectional: use finputc for reverse direction
fnew_bid(
  "A" = "Active",
  "I" = "Inactive",
  "P" = "Pending",
  name = "status"
)

# Forward: code -> label
fputc(c("A", "I", "P"), "status")
# [1] "Active" "Inactive" "Pending"

# Reverse: label -> code
finputc(c("Active", "Pending", "Inactive"), "status_inv")
# [1] "A" "P" "I"
fclear()
```

finputk *Apply Invalue Using a Composite Label*

Description

Convenience wrapper around an INVALUE lookup that pastes multiple vectors together into a composite label before reverse lookup. Mirrors [fputk()] on the invalue side, for INVALUE formats built with composite labels such as ‘fmap(paste(col1, col2, sep = "|"), codes)‘.

Usage

```
finputk(..., invalue_name, sep = "|", na_as_string = FALSE)
```

Arguments

`...`	Vectors to paste together into a composite label. All vectors are recycled to a common length by [paste()].
`invalue_name`	Character. Name of a registered INVALUE format.
`sep`	Separator inserted between the pasted components (default ‘"|"‘).
`na_as_string`	If ‘FALSE‘ (default), an ‘NA‘ in any component propagates to the composite label (restored to ‘NA_character_‘ after the [paste()] step) so the invalue’s ‘missing_value‘ applies. If ‘TRUE‘, the literal string ‘"NA"‘ produced by [paste()] is kept, which is useful when the invalue was built with composite labels via ‘fmap(paste(..., sep = "|"), values)‘.

Details

The output type is determined by the stored invalue’s ‘target_type‘ (numeric / integer → numeric, character → character, logical → logical).

Value

A vector whose type depends on the invalue’s ‘target_type‘.

See Also

[finput()], [finputn()], [finputc()], [fputk()]

Examples

```
# Build an INVALUE keyed on two columns via paste()
finput(
  fmap(paste(c("A", "A", "B"), c(1, 2, 1), sep = "|"), c(10, 20, 30)),
  name = "ab_inv"
)

finputk(c("A", "A", "B"), c(1, 2, 1), invalue_name = "ab_inv")
```

```
# -> 10 20 30

fclear()
```

---

finputn *Apply Numeric Invalue by Name (like 'SAS' INPUTN)*

Description

Looks up a numeric INVALUE format by name from the global format library and applies it to convert labels to numeric values.

Usage

```
finputn(x, invalue_name)
```

Arguments

x	Character vector of labels to convert
invalue_name	Character. Name of a registered INVALUE format.

Value

Numeric vector

Examples

```
# Create numeric invalue and apply
finput(
  "Male" = 1,
  "Female" = 2,
  name = "sex_inv"
)
finputn(c("Male", "Female", "Male", "Unknown", "Female"), "sex_inv")
# [1]  1  2  1 NA  2
fclear()

# Parse invalue from text and apply
fparse(text = '
INVALUE race_inv
  "White" = 1
  "Black" = 2
  "Asian" = 3
;
')
finputn(c("White", "Black"), "race_inv")
# [1] 1 2
fclear()
```

flist *List Format Names from Library*

Description

Returns a character vector of all format and invalue names currently registered in the global format library.

Usage

```
flist()
```

Value

A character vector of registered format names, sorted alphabetically. Returns `character(0)` if the library is empty.

Examples

```
fnew("M" = "Male", "F" = "Female", name = "sex")
flist()
fclear()
```

fmap *Create a Key-Value Mapping for Format Creation*

Description

Convenience helper for building data-driven formats with `fnew`. Returns a named vector (or list) with class `"ks_fmap"` that signals `fnew()` to use the natural direction: names are **input keys**, values are **output labels/objects** — regardless of the format type.

Usage

```
fmap(keys, values)
```

Arguments

`keys`	Character vector of input keys (lookup values).
`values`	Vector of output labels or objects (character, numeric, Date, POSIXct, logical, etc.).

Details

Without `fmap()`, `fnew()` reverses named vectors for character and numeric types (the `factor()` convention `c(Label = "Code")`). Wrapping your data in `fmap()` suppresses this reversal, so `fmap(keys, values)` works identically for character, numeric, Date, POSIXct, and logical formats.

Value

A named vector (or list, for non-atomic values) with class `c("ks_fmap", <original class>)`. Names are `keys`, values are `values`.

See Also

`fnew` for format creation.

Examples

```
# Character lookup: keys -> labels
fmap(c("M", "F"), c("Male", "Female")) |> fnew(name = "sex")
fput(c("M", "F"), "sex")
fclear()

# Date lookup from a data frame
ids   <- c("SUBJ-001", "SUBJ-002")
dates <- as.Date(c("2023-03-09", "2024-08-13"))
fmap(ids, dates) |> fnew(type = "Date", name = "icdtn")
fput("SUBJ-001", "icdtn")
fclear()
```

---

fmap_ranges *Build a Vector of Range Mappings*

Description

Construct a `ks_fmap`-classed named character vector whose names encode numeric / Date / POSIXct range bounds and whose values are the corresponding labels. The result is intended to be passed to `fnew` as a single positional argument (it suppresses the default name reversal).

Usage

```
fmap_ranges(
  low,
  high,
  label,
  inc_low = TRUE,
  inc_high = FALSE,
  date_format = NULL
)
```

Arguments

`low, high`	Numeric, `Date`, or `POSIXct` vectors of equal length giving the lower / upper bounds of each range.
`label`	Character vector of labels (same length as `low`).

`inc_low`, `inc_high`
: Logical, length 1 or `length(low)`. Whether each bound is inclusive. Defaults match `range_spec`: `[low, high)`.

`date_format`
: Optional strptime format string used when formatting `Date`/`POSIXct` bounds into key strings.

Details

Bounds are formatted as ISO 8601: `"%Y-%m-%d"` for Date, `"%Y-%m-%d %H:%M:%S"` (UTC) for POSIXct. Override with `date_format` if needed.

Value

A `ks_fmap` object (named character vector) suitable for passing to `fnew()`.

See Also

`fmap`, `fmap_strata`, `fnew`

Examples

```
rng <- fmap_ranges(
  low   = c(0, 18, 65),
  high  = c(18, 65, Inf),
  label = c("Child", "Adult", "Senior"),
  inc_high = c(FALSE, FALSE, TRUE)
)
fnew(rng, type = "numeric", name = "age_groups")
fput(c(5, 25, 90), "age_groups")
fclear()
```

---

fmap_strata *Build a Vector of Stratified Range Mappings*

Description

Companion to `fmap_ranges` for the `stratified_range` format type. Each row pairs a stratum (e.g. study arm, subject id, or a composite key produced by `fputk()`) with a numeric / Date / POSIXct range and a label. The returned `ks_fmap` vector carries the chosen `sep` as an attribute so that `fnew(type = "stratified_range")` picks it up automatically.

Usage

```
fmap_strata(
  stratum,
  low,
  high,
  label,
```

```
  inc_low = TRUE,
  inc_high = FALSE,
  sep = "|",
  date_format = NULL
)
```

Arguments

stratum	Character vector of stratum identifiers.
low, high	Range bounds. See fmap_ranges.
label	Character vector of labels.
inc_low, inc_high	Logical, length 1 or length(low). See fmap_ranges.
sep	Separator inserted between stratum and range key. Must match the sep subsequently passed to fputk.
date_format	Optional strptime format string.

Value

A ks_fmap object with an attached "strata_sep" attribute.

See Also

fmap_ranges, fputk, fnew

Examples

```
visits <- fmap_strata(
  stratum  = c("ARM_A", "ARM_A", "ARM_B"),
  low      = c(0, 7, 0),
  high     = c(7, 14, 10),
  label    = c("Baseline", "Week 1", "Baseline")
)
fnew(visits, type = "stratified_range", range_subtype = "numeric",
     name = "visit_window")
fputk(c("ARM_A", "ARM_B"), c(3, 5), format = "visit_window")
fclear()
```

---

fmap_to_ranges *Reverse-Lookup Range Bounds from Labels*

Description

Given a vector of values that match the *labels* of a range-based format, returns the corresponding low / high bounds (and inclusivity flags) for each input. Useful for reconstructing the underlying range from a coded value.

Usage

```
fmap_to_ranges(x, fmt)
```

Arguments

x	A vector of values to look up against the format's labels. Coerced to character before matching.
fmt	A ks_format object, or a character name of a format registered in the global library.

Details

For multilabel formats where the same label maps to several ranges, only the *first* matching range is returned. For full multi-match behaviour, call franges() directly and join on label.

Value

A data.frame with one row per element of x and columns low, high, inc_low, inc_high. Rows where the input does not match any range label contain NA.

See Also

franges

Examples

```
fparse(text = '
VALUE visit_ther (numeric)
  [LOW,  1] =  0
  [ 8, 22] =  2
  [22, 36] =  4
  [37, 50] =  6
;
')
fmap_to_ranges(c(0, 2, 4, 6), "visit_ther")
fclear()
```

---

fnew *Create a Format Definition (like 'SAS' PROC FORMAT)*

Description

Creates a format object that maps values to labels, similar to 'SAS' PROC FORMAT. Supports discrete value mapping, ranges, and special handling of missing values. The format is automatically stored in the global format library if name is provided.

Usage

```
fnew(
  ...,
  name = NULL,
  type = "auto",
  default = NULL,
  multilabel = FALSE,
  ignore_case = FALSE,
  date_format = NULL,
  range_subtype = c("numeric", "date", "datetime"),
  strata_sep = "|",
  verbose = FALSE
)
```

Arguments

`...`	Named arguments defining value-label mappings, or one or more named vectors/lists using the R convention `c(Label = "Code")`. Can be: • Discrete values: `"M" = "Male"`, `"F" = "Female"` • Named vector: `c(Male = "M", Female = "F")` • Named list: `list(Male = "M", Female = "F")` • `fmap` vector: `fmap(keys, values)` for data-driven formats (no reversal) • Special values: `.missing = "Missing"`, `.other = "Other"` Named vectors use the R idiom where names are labels and values are codes, which is the reverse of the `...` argument convention. **Named-vector reversal:** For character and numeric formats, named vectors are automatically reversed so that `c(Male = "M")` becomes the mapping `"M" -> "Male"` (following the `factor()` convention). For value types (`"Date"`, `"POSIXct"`, `"logical"`), no reversal occurs — the named vector is used as-is with names as input keys and values as output objects, because non-character objects cannot serve as vector names. **Data-driven formats:** For formats built programmatically from data, wrap your data in `fmap(keys, values)` to suppress automatic reversal for all types. See `vignette("usage_examples")` Example 21 for a detailed walkthrough.
`name`	Character. Optional name for the format. If provided, the format is automatically registered in the global format library.
`type`	Character. Type of format: `"character"`, `"numeric"`, `"Date"`, `"POSIXct"`, `"logical"`, `"date_range"`, `"datetime_range"`, or `"auto"` (default) for auto-detection. Value types (`"Date"`, `"POSIXct"`, `"logical"`) store native R objects instead of character labels. For value types, `.missing` and `.other` are always typed NA. Range-bucketing types (`"date_range"`, `"datetime_range"`) bucket `Date`/`POSIXct` input into character labels using ISO date/datetime strings as range bounds.
`default`	Character. Default label for unmatched values (overrides .other)

multilabel	Logical. If TRUE, the format supports overlapping ranges where a single value can match multiple labels. Used with fput_all to retrieve all matching labels. Default FALSE.
ignore_case	Logical. If TRUE, key matching for character formats is case-insensitive. Default FALSE.
date_format	Character. Optional strptime-style format string used when parsing date/datetime range keys (e.g. "%d/%m/%Y"). When NULL, ISO 8601 parsing is used. Applies to "date_range", "datetime_range", and the value types "Date" / "POSIXct" when keys are ranges.
range_subtype	Character. For type = "stratified_range" only. One of "numeric" (default), "date", or "datetime". Determines how the range part of each stratified key is parsed and compared.
strata_sep	Character. For type = "stratified_range" only. Single-character (or multi-character) separator between the stratum identifier and the range key in each mapping key. Default "|".
verbose	Logical. If TRUE, returns the format object visibly; otherwise returns it invisibly. Default FALSE.

Details

Special directives:

- .missing: Label for NA, NULL, NaN values
- .other: Label for values not matching any rule

Named-vector direction (reverse convention):

When a named vector or list is passed as an unnamed argument (e.g., fnew(c(Male = "M"))), the direction of the name-to-value mapping depends on the output type:

- For **character / numeric** types, names are *labels* and values are *codes*. The pairs are reversed internally so that the format maps code -> label. This follows the standard R idiom used by factor(), where c(Label = "Code").
- For **value types** (Date, POSIXct, logical), names are *input keys* and values are the native R objects returned by the format. No reversal is applied, because non-character objects cannot be used as vector names.

This means the *same data* may need to be arranged differently depending on the target type. To avoid this inconsistency for data-driven formats, use fmap(keys, values) which works identically for all types:

```
fnew(fmap(ids, dates), type = "Date")
fnew(fmap(ids, date_strings), type = "character")
```

When in doubt, use explicit key = "label" arguments — these are never reversed regardless of type.

Expression labels: If a label contains .x1, .x2, etc., it is treated as an R expression that is evaluated at apply-time. Extra arguments are passed positionally via ... in fput:

```
stat_fmt <- fnew("n" = "sprintf('%s', .x1)",
                 "pct" = "sprintf('%.1f%%', .x1 * 100)")
fput(c("n", "pct"), stat_fmt, c(42, 0.15))
# Returns: "42" "15.0%"
```

Value

An object of class "ks_format" containing the format definition. The object is also stored in the format library if name is given.

Examples

```
# Discrete value format (auto-stored as "sex")
fnew(
  "M" = "Male",
  "F" = "Female",
  .missing = "Unknown",
  .other = "Other Gender",
  name = "sex"
)

# Apply immediately
fput(c("M", "F", NA, "X"), "sex")
# [1] "Male" "Female" "Unknown" "Other Gender"
fclear()

# Multilabel format: a value can match multiple labels
fnew(
  "0,5,TRUE,TRUE"   = "Infant",
  "6,11,TRUE,TRUE"  = "Child",
  "12,17,TRUE,TRUE" = "Adolescent",
  "0,17,TRUE,TRUE"  = "Pediatric",
  "18,64,TRUE,TRUE" = "Adult",
  "65,Inf,TRUE,TRUE" = "Elderly",
  "18,Inf,TRUE,TRUE" = "Non-Pediatric",
  name = "age_categories",
  type = "numeric",
  multilabel = TRUE
)

# fput returns first match; fput_all returns all matches
fput(c(3, 14, 25, 70), "age_categories")
fput_all(c(3, 14, 25, 70), "age_categories")
fclear()

# From a named vector (Label = Code convention)
sex_vec <- fnew(c(Male = "M", Female = "F"), .missing = "Unknown",
                name = "sex_vec")
fput(c("M", "F", NA), sex_vec)
# [1] "Male" "Female" "Unknown"
fclear()
```

fnew_bid *Create Bidirectional Format*

Description

Creates both a format and its corresponding invalue for bidirectional conversion. Both are automatically stored in the global format library if `name` is provided.

Usage

```
fnew_bid(..., name = NULL, type = "auto")
```

Arguments

`...`	Named arguments for format mappings
`name`	Character. Base name for both formats. The invalue will be named `paste0(name, "_inv")`.
`type`	Character. Format type

Value

List with `format` (ks_format) and `invalue` (ks_invalue) components.

Examples

```
# Bidirectional status format
status_bi <- fnew_bid(
  "A" = "Active",
  "I" = "Inactive",
  "P" = "Pending",
  name = "status"
)

# Forward: code -> label
fputc(c("A", "I", "P", "A"), "status")
# [1] "Active" "Inactive" "Pending" "Active"

# Reverse: label -> code
finputc(c("Active", "Pending", "Inactive"), "status_inv")
# [1] "A" "P" "I"
fclear()

# From a named vector (Label = Code convention, same as fnew)
fnew_bid(c(Male = "M", Female = "F"), name = "sex_bid")
fputc(c("M", "F"), "sex_bid")
finputc(c("Male", "Female"), "sex_bid_inv")
fclear()
```

`fnew_date` *Create Date/Time Format*

Description

Creates a format object for date, time, or datetime values using SAS format names or custom R `strftime` patterns. The format is automatically registered in the global format library.

Usage

```
fnew_date(pattern, name = NULL, type = "auto", .missing = NULL)
```

Arguments

`pattern`	Character. Either a SAS format name (e.g., `"DATE9."`, `"MMDDYY10."`, `"TIME8."`, `"DATETIME20."`) or a custom R `strftime` pattern (e.g., `"%Y-%m-%d"`).
`name`	Character. Name to register the format under. Defaults to the SAS format name (with period) or the pattern itself.
`type`	Character. Type of format: `"date"`, `"time"`, `"datetime"`, or `"auto"` (auto-detect from SAS name). Must be specified for custom strftime patterns.
`.missing`	Character. Label for missing values (NA). Default `NULL`.

Details

SAS format names are resolved automatically:

- **Date:** DATE9., DDMMYY10., MMDDYY10., YYMMDD10., MONYY7., YEAR4., WEEKDATE., WORDDATE., etc.
- **Time:** TIME8., TIME5., HHMM., HOUR., MMSS.
- **Datetime:** DATETIME20., DATETIME13., etc.

Numeric input is converted using R epoch (`"1970-01-01"`):

- Dates: numeric values are interpreted as days since 1970-01-01
- Datetimes: numeric values are interpreted as seconds since 1970-01-01
- Times: always treated as seconds since midnight

Value

A `ks_format` object with date/time type, registered in the library.

Examples

```
# Use a SAS format name
fnew_date("DATE9.", name = "mydate")
fput(as.Date("2020-01-01"), "mydate")
# [1] "01JAN2020"

# Use directly without pre-creating
fputn(as.Date("2020-06-15"), "MMDDYY10.")
# [1] "06/15/2020"

# Custom strftime pattern (e.g., Russian style: DD.MM.YYYY)
fnew_date("%d.%m.%Y", name = "ru_date", type = "date")
fput(as.Date(c("1990-03-25", "1985-11-03", "2000-07-14")), "ru_date")

# Custom format with missing value label
fnew_date("MMDDYY10.", name = "us_date", .missing = "NO DATE")
fput(c(as.Date("2025-01-01"), NA, as.Date("2025-12-31")), "us_date")
# [1] "01/01/2025" "NO DATE" "12/31/2025"

# Numeric dates (days since 1970-01-01, R epoch)
r_days <- as.numeric(as.Date("2025-01-01"))
fputn(r_days, "DATE9.")

# Multiple SAS date formats applied directly
today <- Sys.Date()
fputn(today, "DATE9.")
fputn(today, "MMDDYY10.")
fputn(today, "YYMMDD10.")
fputn(today, "MONYY7.")
fputn(today, "WORDDATE.")
fputn(today, "QTR.")

# Time formatting (seconds since midnight)
fputn(c(0, 3600, 45000, 86399), "TIME8.")
fputn(c(0, 3600, 45000), "HHMM.")

# Datetime formatting
now <- Sys.time()
fputn(now, "DATETIME20.")
fputn(now, "DTDATE.")
fputn(now, "DTYYMMDD.")
fclear()
```

`format_get` *Retrieve a Format from the Library*

Description

Returns a format or invalue object by name. Used when you need the object (e.g. for `fput_df` or `fexport`) rather than applying by name with `fput`, `fputn`, or `fputc`.

Usage

```
format_get(name)
```

Arguments

name	Character. Name of a registered format or invalue.

Value

A ks_format or ks_invalue object.

Examples

```
fnew("M" = "Male", "F" = "Female", name = "sex")
sex_fmt <- format_get("sex")
fput_df(data.frame(sex = c("M", "F")), sex = sex_fmt)
fclear()
```

---

format_library_app *Launch Shiny Browser for Format Library*

Description

Opens an interactive Shiny app for browsing and managing objects currently registered in the global ksformat format library.

Usage

```
format_library_app(port = getOption("shiny.port"), launch.browser = TRUE)
```

Arguments

port	Integer or NULL. Port passed to shiny::runApp(). Default: getOption("shiny.port").
launch.browser	Logical. Passed to shiny::runApp(). Default: TRUE.

Details

The app displays both ks_format (VALUE) and ks_invalue (INVALUE) objects, supports filtering and name search, shows object details with a formatted mapping table, and provides management actions to remove one object, clear the full library, or quit the app.

Value

Invisibly returns NULL.

Examples

```
## Not run:
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  fnew("M" = "Male", "F" = "Female", name = "sex")
  finput("Male" = 1, "Female" = 2, name = "sex_inv")
  format_library_app()
}

## End(Not run)
```

`fparse` *Parse Format Definitions from 'SAS'-like Text*

Description

Reads format definitions written in a human-friendly 'SAS'-like syntax and returns a list of `ks_format` and/or `ks_invalue` objects. All parsed formats are automatically stored in the global format library.

Usage

```
fparse(text = NULL, file = NULL, verbose = FALSE)
```

Arguments

`text`	Character string or character vector containing format definitions. If a character vector, lines are concatenated with newlines.
`file`	Path to a text file containing format definitions. Exactly one of `text` or `file` must be provided.
`verbose`	Logical. If `TRUE`, the parsed formats are printed to the console. Default is `FALSE` to suppress output (the result is returned invisibly).

Details

The syntax supports two block types:

VALUE blocks define formats (value -> label):

```
VALUE name (type)
  "value1" = "Label 1"
  "value2" = "Label 2"
  [low, high) = "Range Label (half-open)"
  (low, high] = "Range Label (open-low, closed-high)"
  .missing = "Missing Label"
  .other = "Other Label"
;
```

INVALUE blocks define reverse formats (label -> numeric value):

```
INVALUE name
  "Label 1" = 1
  "Label 2" = 2
;
```

Syntax rules:

- Blocks start with `VALUE` or `INVALUE` keyword and end with `;`
- The type in parentheses is optional; defaults to `"auto"` for VALUE, `"numeric"` for INVALUE
- Values can be quoted or unquoted
- Ranges use interval notation with explicit bounds
- Legacy range syntax `low - high` is also supported
- Special range keywords: `LOW` (-Inf) and `HIGH` (Inf)
- `.missing` and `.other` are special directives
- Lines starting with `/*`, `*`, `//`, or `#` are comments

Block options:

Comma-separated options can be placed inside the parentheses after the type:

- `nocase` — enables case-insensitive key matching (equivalent to `ignore_case = TRUE` in `fnew`).
- `multilabel` — allows overlapping ranges where a single value matches multiple labels (used with `fput_all`).

Options can be combined: `VALUE name (character, nocase, multilabel)`.

Value

A named list of `ks_format` and/or `ks_invalue` objects. Names correspond to the format names defined in the text. All formats are automatically registered in the global format library.

Examples

```
# Parse multiple format definitions from text
fparse(text = '
VALUE sex (character)
  "M" = "Male"
  "F" = "Female"
  .missing = "Unknown"
;

VALUE age (numeric)
  [0, 18)    = "Child"
  [18, 65)   = "Adult"
  [65, HIGH]  = "Senior"
  .missing   = "Age Unknown"
;

// Invalue block
INVALUE race_inv
```

```
  "White" = 1
  "Black" = 2
  "Asian" = 3
;
')

fput(c("M", "F", NA), "sex")
fputn(c(5, 25, 70, NA), "age")
finputn(c("White", "Black"), "race_inv")
flist()
fprint()
fclear()

# Parse date/time/datetime format definitions
fparse(text = '
VALUE enrldt (date)
  pattern = "DATE9."
  .missing = "Not Enrolled"
;

VALUE visit_time (time)
  pattern = "TIME8."
;

VALUE stamp (datetime)
  pattern = "DATETIME20."
;
')

fput(as.Date("2025-03-01"), "enrldt")
fput(36000, "visit_time")
fput(as.POSIXct("2025-03-01 10:00:00", tz = "UTC"), "stamp")
fclear()

# Case-insensitive format (nocase option)
fparse(text = '
VALUE yesno (character, nocase)
  "Y" = "Yes"
  "N" = "No"
  .other = "Unknown"
;
')
fput(c("y", "N", "YES"), "yesno")
# [1] "Yes" "No" "Unknown"
fclear()

# Parse multilabel format
fparse(text = '
VALUE risk (numeric, multilabel)
  [0, 3]  = "Low Risk"
  [0, 7]  = "Monitored"
  (3, 7]  = "Medium Risk"
  (7, 10] = "High Risk"
```

```
;
')
fput_all(c(2, 5, 9), "risk")
fclear()
```

`fprint` *Print Format(s) from Library*

Description

Displays format information from the global format library. When called without arguments, lists all registered format names. When called with a name, displays the full definition of that format.

Usage

```
fprint(name = NULL)
```

Arguments

`name`	Character. Optional name of a specific format to display. If `NULL` (default), lists all registered formats.

Value

Invisible `NULL`. This function is for display only.

See Also

`flist` for a programmatic alternative that returns a character vector of registered names.

Examples

```
fnew("M" = "Male", "F" = "Female", name = "sex")
flist()          # character vector of names
fprint()         # list all formats
fprint("sex")    # show specific format
fclear()
```

fput *Apply Format to Data (like 'SAS' PUT function)*

Description

Applies a format definition to a vector of values, returning formatted labels. Properly handles NA, NULL, NaN, and other missing values.

Usage

```
fput(x, format, ..., keep_na = FALSE)
```

Arguments

x	Vector of values to format
format	A ks_format object or a character string naming a format in the global format library.
...	Additional arguments for expression labels. Positional arguments are mapped to .x1, .x2, etc. inside expression labels. Can be vectors of the same length as x or scalars (recycled).
keep_na	Logical. If TRUE, preserve NA in output instead of applying missing label.

Details

The function handles missing values in the following order:

1. NA, NULL, NaN -> Uses format's missing_label if defined
2. Exact matches -> Uses defined value-label mapping
3. Range matches (for numeric) -> Uses range label
4. No match -> Uses format's other_label or returns original value

Expression labels: If a label string contains .x1, .x2, etc., it is evaluated as an R expression at apply-time. Extra data is passed as positional arguments:

```
stat_fmt <- fnew("n" = "sprintf('%s', .x1)",
                 "pct" = "sprintf('%.1f%%', .x1 * 100)")
fput(c("n", "pct"), stat_fmt, c(42, 0.15))
# Returns: "42" "15.0%"
```

Case-insensitive matching: When a format has ignore_case = TRUE, key matching is case-insensitive for character formats.

Value

Character vector with formatted labels

Examples

```
# Basic discrete formatting
fnew("M" = "Male", "F" = "Female", .missing = "Unknown", name = "sex")
fput(c("M", "F", NA, "X"), "sex")
# [1] "Male" "Female" "Unknown" "X"

# Preserve NA instead of applying missing label
sex_f <- fnew("M" = "Male", "F" = "Female", .missing = "Unknown")
fput(c("M", "F", NA), sex_f, keep_na = TRUE)
# [1] "Male" "Female" NA

# Numeric range formatting
fparse(text = '
VALUE score (numeric)
  (0, 50]  = "Low"
  (50, 100] = "High"
  .other   = "Out of range"
;
')
fput(c(0, 1, 50, 51, 100, 101), "score")
# [1] "Out of range" "Low" "Low" "High" "High" "Out of range"
fclear()
```

`fput_all` *Apply Format and Return All Matches (Multilabel)*

Description

For multilabel formats, returns all matching labels for each input value. Regular `fput` returns only the first match; this function returns all matches as a list of character vectors.

Usage

```
fput_all(x, format, ..., keep_na = FALSE)
```

Arguments

`x`	Vector of values to format
`format`	A `ks_format` object or a character string naming a format in the global format library.
`...`	Additional arguments for expression labels (mapped to `.x1`, `.x2`, etc.).
`keep_na`	Logical. If TRUE, preserve NA in output.

Value

A list of character vectors. Each element contains all matching labels for the corresponding input value.

Examples

```
# Basic multilabel: a value can match multiple labels
age_ml <- fnew(
  "0,5,TRUE,TRUE" = "Infant",
  "6,11,TRUE,TRUE" = "Child",
  "12,17,TRUE,TRUE" = "Teen",
  "0,17,TRUE,TRUE" = "Minor",
  "18,64,TRUE,TRUE" = "Adult",
  "65,Inf,TRUE,TRUE" = "Senior",
  name = "age_ml", type = "numeric", multilabel = TRUE
)

fput_all(c(3, 15, 25), age_ml)
# [[1]] "Infant" "Minor"
# [[2]] "Teen" "Minor"
# [[3]] "Adult"

# Multilabel with .missing and .other
fnew(
  "0,100,TRUE,TRUE"  = "Valid Score",
  "0,49,TRUE,TRUE"   = "Below Average",
  "50,100,TRUE,TRUE" = "Above Average",
  "90,100,TRUE,TRUE" = "Excellent",
  .missing = "No Score",
  .other = "Out of Range",
  name = "score_ml", type = "numeric", multilabel = TRUE
)
fput_all(c(95, 45, NA, 150), "score_ml")
# [[1]] "Valid Score" "Above Average" "Excellent"
# [[2]] "Valid Score" "Below Average"
# [[3]] "No Score"
# [[4]] "Out of Range"

# Parse multilabel from text
fparse(text = '
VALUE risk (numeric, multilabel)
  [0, 3]  = "Low Risk"
  [0, 7]  = "Monitored"
  (3, 7]  = "Medium Risk"
  (7, 10] = "High Risk"
;
')
fput_all(c(2, 5, 9), "risk")
# [[1]] "Low Risk" "Monitored"
# [[2]] "Monitored" "Medium Risk"
# [[3]] "High Risk"
fclear()
```

---

fput_df *Apply Format to Data Frame Columns*

Description

Applies formats to one or more columns in a data frame.

Usage

```
fput_df(data, ..., suffix = "_fmt", replace = FALSE)
```

Arguments

data	Data frame
...	Named format specifications: column_name = format_object_or_name
suffix	Character. Suffix to add to formatted column names (default: "_fmt")
replace	Logical. If TRUE, replace original columns; if FALSE, create new columns

Value

Data frame with formatted columns

Examples

```
# Apply formats to multiple columns
df <- data.frame(
  id = 1:6,
  sex = c("M", "F", "M", "F", NA, "X"),
  age = c(15, 25, 45, 70, 35, NA),
  stringsAsFactors = FALSE
)

sex_f <- fnew("M" = "Male", "F" = "Female", .missing = "Unknown")
fparse(text = '
VALUE age (numeric)
  [0, 18)   = "Child"
  [18, 65)  = "Adult"
  [65, HIGH] = "Senior"
  .missing  = "Age Unknown"
;
')
age_f <- format_get("age")

fput_df(df, sex = sex_f, age = age_f, suffix = "_label")

# Date formatting in data frames
patients <- data.frame(
  id = 1:4,
  visit_date = as.Date(c("2025-01-10", "2025-02-15", "2025-03-20", NA)),
  stringsAsFactors = FALSE
)
visit_fmt <- fnew_date("DATE9.", name = "visit_fmt", .missing = "NOT RECORDED")
fput_df(patients, visit_date = visit_fmt)
fclear()
```

fputc *Apply Character Format by Name (like 'SAS' PUTC)*

Description

Looks up a character VALUE format by name from the global format library and applies it to the input vector.

Usage

```
fputc(x, format_name, ...)
```

Arguments

x	Character vector of values to format
format_name	Character. Name of a registered character format, or a character vector of format names (same length as x) to apply a different format per element (like 'SAS' PUTC with a variable format).
...	Additional arguments passed to fput for expression labels (mapped to .x1, .x2, etc.).

Value

Character vector with formatted labels

Examples

```
# Apply character format by name
fnew("M" = "Male", "F" = "Female", name = "sex")
fputc(c("M", "F"), "sex")
# [1] "Male" "Female"

# Bidirectional: forward direction
fnew_bid(
  "A" = "Active",
  "I" = "Inactive",
  "P" = "Pending",
  name = "status"
)
fputc(c("A", "I", "P", "A"), "status")
# [1] "Active" "Inactive" "Pending" "Active"
fclear()
```

fputk *Apply Format Using a Composite Key*

Description

Convenience wrapper around [fput()] that pastes multiple vectors together into a composite key before lookup. Useful when a format is keyed on the combination of several columns (e.g., 'USUBJID|VISITNUM').

Usage

```
fputk(..., format, sep = "|", keep_na = FALSE, na_as_string = FALSE)
```

Arguments

...	Vectors to paste together into a composite key. All vectors are recycled to a common length by [paste()].
format	A [ks_format] object or a registered format name (character string).
sep	Separator inserted between the pasted components (default '"|"').
keep_na	If 'TRUE', 'NA' inputs remain 'NA' in the output instead of being mapped via '.missing'. Passed through to [fput()].
na_as_string	If 'FALSE' (default), an 'NA' in any component propagates to the composite key (restored to 'NA_character_' after the [paste()] step) so that [fput()] can apply '.missing' handling. If 'TRUE', the literal string '"NA"' produced by [paste()] is kept, which is useful when the format was built with composite keys via 'fmap(paste(..., sep = "|"), values)' — because [paste()] converts 'NA' to '"NA"' on both sides, the round-trip lookup then matches.

Value

A character vector of formatted labels, the same length as the (recycled) input vectors.

See Also

[fput()], [fputn()], [fputc()], [finputk()]

Examples

```
# Build a lookup keyed on two columns
fnew(
  "A|1" = "2025-01-15",
  "A|2" = "2025-02-20",
  "B|1" = "2025-03-10",
  .other = "NOT FOUND",
  name = "visit_date",
  type = "character"
)
```

```
subj  <- c("A", "A", "B", "B")
visit <- c(1, 2, 1, 3)

fputk(subj, visit, format = "visit_date")
# -> "2025-01-15" "2025-02-20" "2025-03-10" "NOT FOUND"

fclear()

# Composite key with NA components matching a paste()-built format
fnew(
  fmap(
    paste(c("CHEM", "COAG"), c("ALB", "INR"), c("g/L", NA), sep = "|"),
    c("ALB", "INR")
  ),
  name = "lb_param", type = "character"
)

fputk(c("CHEM", "COAG"), c("ALB", "INR"), c("g/L", NA),
      format = "lb_param", na_as_string = TRUE)
# -> "ALB" "INR"

fclear()
```

`fputn` *Apply Numeric Format by Name (like 'SAS' PUTN)*

Description

Looks up a numeric VALUE format by name from the global format library and applies it to the input vector.

Usage

```
fputn(x, format_name, ...)
```

Arguments

`x`	Numeric vector of values to format
`format_name`	Character. Name of a registered numeric format, or a character vector of format names (same length as `x`) to apply a different format per element (like 'SAS' PUTN with a variable format).
`...`	Additional arguments passed to `fput` for expression labels (mapped to `.x1`, `.x2`, etc.).

Value

Character vector with formatted labels

Examples

```
# Numeric range formatting
fparse(text = '
VALUE age (numeric)
  [0, 18)    = "Child"
  [18, 65)   = "Adult"
  [65, HIGH]  = "Senior"
  .missing   = "Age Unknown"
;
')
fputn(c(5, 25, 70, NA), "age")
# [1] "Child" "Adult" "Senior" "Age Unknown"

# SAS date format (auto-resolved, no pre-creation needed)
fputn(as.Date("2025-01-15"), "DATE9.")
# [1] "15JAN2025"

# Time format (seconds since midnight)
fputn(c(0, 3600, 45000), "TIME8.")
# [1] "00:00:00" "01:00:00" "12:30:00"
fclear()
```

`franges` *Extract Range Entries from a Format*

Description

Returns the range-based mappings of a `ks_format` object as a tidy data frame. Discrete entries (plain values, `.missing`, `.other`) are excluded.

Usage

```
franges(fmt)
```

Arguments

`fmt`	A `ks_format` object, or a character name of a format registered in the global library.

Details

Range keys are stored internally as strings such as `"0,18,TRUE,FALSE"`. `franges()` parses these keys back into their numeric bounds and inclusivity flags, making it easy to inspect, filter, or programmatically reuse range definitions.

Bounds parsed as `HIGH` or `LOW` appear as `Inf` and `-Inf` respectively.

Value

A `data.frame` with columns `low`, `high`, `inc_low`, `inc_high`, and `label`. Rows are returned in the order ranges appear in the format. If the format has no range entries, an empty data frame with the same columns is returned.

Examples

```
fparse(text = '
VALUE age (numeric)
  [0, 18)    = "Child"
  [18, 65)   = "Adult"
  [65, HIGH] = "Senior"
  .missing   = "Unknown"
;
')
franges("age")
fclear()
```

`is_missing` *Check if Value is Missing*

Description

Element-wise check for missing values including NA and NaN. Optionally treats empty strings as missing.

Usage

```
is_missing(x)
```

Arguments

`x`	Value to check

Value

Logical vector. NULL input returns `logical(0)`.

Examples

```
is_missing(NA)          # TRUE
is_missing(NaN)         # TRUE
is_missing("")          # TRUE
is_missing("text")      # FALSE
is_missing(c(1, NA, NaN)) # FALSE TRUE TRUE
```

ksformat_cheatsheet *Open the ksformat cheat sheet*

Description

Opens the package cheat sheet in the default browser (HTML) or viewer (PDF). The files are installed under `system.file("doc", ..., package = "ksformat")`.

Usage

```
ksformat_cheatsheet(format = c("html", "pdf"))
```

Arguments

format	Character: `"html"` (default) or `"pdf"`. Which version to open.

Value

Invisibly, the path to the opened file. If the file is not found, an error is thrown.

Examples

```
## Not run:
ksformat_cheatsheet()           # open HTML in browser
ksformat_cheatsheet("pdf")      # open PDF

## End(Not run)
```

print.ks_format *Print Format Object*

Description

Print Format Object

Usage

```
## S3 method for class 'ks_format'
print(x, ...)
```

Arguments

x	A ks_format object
...	Additional arguments (unused)

Value

The input `x`, returned invisibly.

print.ks_invalue *Print Invalue Object*

Description

Print Invalue Object

Usage

```
## S3 method for class 'ks_invalue'
print(x, ...)
```

Arguments

x	A ks_invalue object
...	Additional arguments (unused)

Value

The input x, returned invisibly.

range_spec *Create Range Specification*

Description

Helper function to create range specifications for numeric formats.

Usage

```
range_spec(low, high, label, inc_low = TRUE, inc_high = FALSE)
```

Arguments

low	Numeric. Lower bound of the range.
high	Numeric. Upper bound of the range.
label	Character. Label for values in this range.
inc_low	Logical. If TRUE (default), the lower bound is inclusive (>=). If FALSE, exclusive (>).
inc_high	Logical. If TRUE, the upper bound is inclusive (<=). If FALSE (default), exclusive (<).

Details

By default, ranges are half-open: `[low, high)` — the lower bound is included and the upper bound is excluded. This matches 'SAS' PROC FORMAT range semantics and prevents overlap between adjacent ranges.

Value

A `range_spec` object (list with low, high, label, inc_low, inc_high).

Examples

```
range_spec(0, 18, "Child")          # [0, 18)
range_spec(18, 65, "Adult")         # [18, 65)
range_spec(65, Inf, "Senior", inc_high = TRUE)  # [65, Inf]
```

Index