

# ksformat :: Cheat Sheet

SAS-style **PROC FORMAT** for R — value–label mappings, numeric ranges, reverse formatting (invalue), date/time/datetime, expression labels, multilabel & more

## INSTALL & LOAD

```
# From CRAN
install.packages("ksformat")
# From GitHub (dev)
remotes::install_github("crown16384/ksformat")
library(ksformat)
```

## FORMAT CREATION

### FNEW() — DISCRETE VALUE–LABEL VALUE

**fnew(..., name=NULL, type="auto", default=NULL, multilabel=FALSE, ignore\_case=FALSE)**  
... = named pairs key="label" | .missing/.other = special labels | name = register in library

```
fnew(
  "M" = "Male",
  "F" = "Female",
  .missing = "Unknown",
  .other = "Other Gender",
  name = "sex"
)
```

**sex CHAR**

M → Male  
F → Female  
.missing → Unknown  
.other → Other Gender

### FINPUT() — REVERSE (LABEL–VALUE) INVALUE

**finput(..., name=NULL, target\_type="numeric", missing\_value=NA)**  
... = named pairs label=value | target\_type = "numeric"|"character" output type

```
finput(
  "Male" = 1,
  "Female" = 2,
  name = "sex_inv"
)
```

**sex\_inv INVALUE –NUM**

Male → 1  
Female → 2

### FNEW\_BID() — BIDIRECTIONAL VALUE + INVALUE

**fnew\_bid(..., name=NULL, type="auto")**  
Creates both a VALUE format and matching \_inv INVALUE simultaneously

```
fnew_bid(
  "A" = "Active",
  "I" = "Inactive",
  name = "status"
)
```

Creates both:  
"status" VALUE format  
"status\_inv" INVALUE

### FNEW\_DATE() — DATE/TIME/DATETIME FORMAT

**fnew\_date(pattern, name=NULL, type="auto", .missing=NULL)**  
pattern = SAS format name (DATE9.) or R strftime pattern (%d.%m.%Y)

**fnew\_date("DATE9.", name = "bday")**

```
fnew_date("%d.%m.%Y",
  name = "ru", type = "date")
```

**bday DATE**

Pattern: %d.%m.%Y (DATE9.)

```
fnew_date("MMDDYY10.",
  name = "us",
  .missing = "NO DATE")
```

SAS names auto-resolved; R patterns also supported

### OPTIONS FOR FNEW()

PARAM	DEFAULT	WHAT IT DOES
name	NULL	Register in format library
type	"auto"	"numeric"/"character" key type
multilabel	FALSE	Allow multiple labels per value
ignore_case	FALSE	Case-insensitive matching
.missing	–	Label for NA/NaN/"
.other	–	Fallback for unmatched values

## FORMAT APPLICATION

### FPUT() — APPLY FORMAT (GENERIC)

**fput(x, format, ..., keep\_na=FALSE)**  
x = values | format = name or object | ... = extra args for expression labels

```
fput(
  c("M", "F", "M", NA, "X"),
  "sex"
)
```

"Male" "Female" "Male" "Unknown"  
"Other Gender"

### FPUTN() FPUTC() — TYPED APPLY

**fputn(x, format\_name, ...) · fputc(x, format\_name, ...)**  
Numeric/character shorthand → format\_name is always a string name

```
fputn(c(5, 30, 70), "age")
```

"Child" "Adult" "Senior"

```
fputc(c("M", "F"), "sex")
```

"Male" "Female"

### FPUT\_DF() — FORMAT DATA FRAME COLUMNS

**fput\_df(data, ..., suffix="\_fmt", replace=FALSE)**  
... = col=format pairs | suffix = new column suffix | replace = overwrite original

```
fput_df(df,
  sex = format_get("sex"),
  age = format_get("age"),
  suffix = "_lbl"
)
```

	SEX	AGE	SEX_LBL	AGE_LBL
M	15	Male	Child	
F	25	Female	Adult	
NA	35	Unknown	Adult	

### CASE-INSENSITIVE MATCHING

```
fnew(
  "M" = "Male",
  "F" = "Female",
  name = "sex_nc",
  ignore_case = TRUE
)
```

**fput(c("m", "F", "M"), "sex\_nc")**

"Male" "Female" "Male"

"m" matches "M" with ignore\_case

## MISSING VALUE HANDLING

#	CONDITION	RESULT
1	NA / NaN / ""	.missing   label
2	Exact match	Mapped label
3	Range match	Range label
4	No match	.other   / original

### IS\_MISSING() — CHECK FOR MISSING VALUES

**is\_missing(x)**  
Returns TRUE for NA, NaN, and empty string ""

```
is_missing(NA) # TRUE
is_missing(NaN) # TRUE
is_missing("") # TRUE
is_missing("x") # FALSE
```

TRUE  
TRUE  
TRUE  
FALSE

### KEEP\_NA OPTION

```
fput(c("M", NA), "sex")
```

"Male" "Unknown"

```
fput(c("M", NA), "sex",
  keep_na = TRUE)
```

"Male" "NA"

## TEXT PARSING & NUMERIC RANGES

### FPARSE() — PARSE VALUE/INVALUE TEXT

**fparse(text=NULL, file=NULL)**  
text = format definition string | file = path to .txt file with definitions

```
fparse(text = '
VALUE age (numeric)
[0, 18] = "Child"
[18, 65] = "Adult"
[65, HIGH] = "Senior"
.missing = "Age Unknown"
;')
fputn(cc(5,18,65,NA), "age")
```

"Child" "Adult" "Senior" "Age Unknown"

### RANGE SYNTAX REFERENCE

SYNTAX	MEANING	MATH
[a, b]	both inclusive	a ≤ x ≤ b
[a, b)	right exclusive	a ≤ x < b
(a, b]	left exclusive	a < x ≤ b
(a, b)	both exclusive	a < x < b
HIGH	↔ upper bound	
LOW	↔ lower bound	

### BMI WITH DECIMALS + .MISSING

VALUE bmi (numeric)	BMI	RESULT
[0, 18.5)	16.2	Underweight
[18.5, 25)	22.7	Normal
[25, 30)	25.0	Overweight
[30, HIGH)	35.1	Obese
.missing	NA	No data

### EXCLUSIVE / INCLUSIVE BOUNDS + .OTHER

VALUE score (numeric)	SCORE	LABEL
[0, 50]	0	Out of range
(50, 100]	50	Low
(100, 150]	51	High
(150, 200]	101	Out of range

### PARSE MULTIPLE FORMATS AT ONCE

```
fparse(text = '
VALUE race (character)
"M"="White" "B"="Black"
"A"="Asian"
.missing = "Unknown"
;')
INVALUE race_inv
"White"=1 "Black"=2
"Asian"=3
;')
```

Registers both in library:  
race VALUE (char)  
race\_inv INVALUE (num)

### RANGE\_SPEC() — BUILD RANGE STRINGS

**range\_spec(low, high, label, inc\_low=TRUE, inc\_high=FALSE)**  
Programmatic range key builder for fnew() – returns "low,high,inc\_low,inc\_high"

```
range_spec(0, 18)
```

"0,18,TRUE,FALSE"

```
range_spec(18, Inf,
  inc_high = FALSE)
```

"18,Inf,TRUE,FALSE"

## REVERSE FORMATTING (INVALUE)

### FINPUTN() — LABELS → NUMERIC

**finputn(x, invalue\_name)**  
Applies named invalue, returns numeric vector

```
finputn(
  c("Male", "Female", "Unknown"),
  "sex_inv"
)
```

1 2 NA

### FINPUTC() — LABELS → CHARACTER

**finputc(x, invalue\_name)**  
Applies named invalue, returns character vector

```
finputc(
  c("Active", "Pending"),
  "status_inv"
)
```

"A" "P"

### ROUND-TRIP: VALUE ↔ INVALUE

"A" → fputc("status") → "Active" → finputc("status\_inv") → "A"

## DATE, TIME & DATETIME

### SAS DATE FORMATS (AUTO-RESOLVED)

	FORMAT	→ RESULT
fputn(Sys.Date(), "DATE9.")	DATE9.	18MAR2026
fputn(Sys.Date(), "MMDDYY10.")	MMDDYY10.	03/18/2026
fputn(Sys.Date(), "YYMMDD10.")	YYMMDD10.	2026-03-18
fputn(Sys.Date(), "WORDDATE.")	WORDDATE.	18/03/2026
fputn(Sys.Date(), "QTR.")	QTR.	March 18, 2026
	YEAR4.	2026
	QTR.	1

### R NUMERIC DATES (DAYS SINCE 1970-01-01)

```
days ← as.numeric(
  as.Date("2025-01-01")
)
```

days = 20089

"01JAN2025"  
"01/01/2025"

### TIME FORMATS (SECONDS SINCE MIDNIGHT)

SECS	TIME8.	TIMES.	HHMM.
0	0:00:00	0:00	00:00
3600	1:00:00	1:00	01:00
45000	12:30:00	12:30	12:30
86399	23:59:59	23:59	23:59

### DATETIME FORMATS

	FORMAT	→ RESULT
fputn(Sys.time(), "DATETIME20.")	DATETIME20.	18MAR2026:13:48:58
	DATETIME13.	18MAR26:13:48
	DTDATE.	18MAR2026
	DTYYMMDD.	2026-03-18

### CUSTOM + DATA FRAMES

```
fnew_date("%d.%m.%Y",
  name = "ru_date")
```

"25.03.1998" "03.11.1985" "14.07.2080"

```
fput(birthdays, "ru_date")
```

v ← fnew\_date("DATE9.",  
name = "vfmt",  
.missing = "NOT RECORDED")

```
fput_df(patients,
  visit_date = v)
```

### PUTN WORKFLOW (SAS-STYLE DYNAMIC DISPATCH)

```
fnew(
  "1" = "date9.",
  "2" = "mmddyy10.",
  name = "wfmt",
  type = "numeric")
```

ID	VISIT_DATE	VISIT_FMT
1	2025-01-10	10JAN2025
2	2025-02-15	15FEB2025
3	<NA>	NOT RECORDED

```
datefmt ← fputn(key, "wfmt")
date ← fputn(number, datefmt)
```

### PARSE DATE FORMATS FROM TEXT

```
fparse(text = '
VALUE enlde (date)
pattern = "DATE9."
.missing = "Not Enrolled"
;')
VALUE stamp (datetime)
pattern = "DATETIME20."
;')
```

## FORMAT LIBRARY

### FPRINT() — INSPECT REGISTERED FORMATS

**fprint(name=NULL)**  
No args = list all | name = show detail for one format

```
fprint() # list all
fprint("sex") # detail
```

age VALUE (num) 3  
sex VALUE (char) 2  
sex\_inv INVALUE (num) 2

### FORMAT\_GET() FCLEAR()

**format\_get(name)** – retrieve format object **fclear(name=NULL)** – remove one or all

```
fnt ← format_get("sex")
```

Retrieve format object  
Remove single format  
Clear entire library

### FEXPORT() — EXPORT TO PARSEABLE TEXT

**fexport(..., formats=NULL, file=NULL)**  
... or formats = format objects | file = write to file (else returns string)

```
cat(fexport(bmi = bmi_fmt))
```

VALUE bmi (numeric)  
[0, 18.5) = "Underweight"  
[18.5, 25) = "Normal"  
[25, 30) = "Overweight"  
[30, HIGH] = "Obese"  
;

### FIMPORT() — SAS CNTLOUT CSV

**fimport(file, register=TRUE, overwrite=TRUE)**  
file = CNTLOUT CSV path | register = auto-add to library | overwrite = replace existing

```
x ← fimport("cntlout.csv")
```

```
m ← fimport(csv,
  register = FALSE)
```

AGEGRP VALUE (num)  
BMICAT VALUE (num)  
GENDER VALUE (char)

```
fput(v, m[["GENDER"]])
```

## MULTILABEL FORMATS

### FPUT\_ALL() — ONE VALUE – MULTIPLE LABELS MULTILABEL=TRUE

**fput\_all(x, format, ..., keep\_na=FALSE)**  
Returns list of character vectors – all matching labels per value

```
fnew(
  "0,5,T,T" = "Infant",
  "6,11,T,T" = "Child",
  "12,17,T,T" = "Adolescent",
  "18,17,T,T" = "Pediatric",
  "18,64,T,T" = "Adult",
  "65,Inf,T,T" = "Elderly",
  "18,Inf,T,T" = "Non-Pediatric",
  name = "age_cat",
  type = "numeric",
  multilabel = TRUE
)
```

3 → Infant Pediatric  
14 → Adolescent Pediatric  
25 → Adult Non-Pediatric  
70 → Elderly Non-Pediatric

fput() → first match only · fput\_all() → all matching labels (list)

```
fput_all(
  c(3, 14, 25, 70),
  "age_cat"
)
```

### AE SEVERITY GRADING (CLINICAL)

```
fnew(
  "1,1,T,T" = "Mild",
  "2,2,T,T" = "Moderate",
  "3,3,T,T" = "Severe",
  "4,4,T,T" = "Life-threat.",
  "5,5,T,T" = "Fatal",
  "3,5,T,T" = "Serious",
  "1,2,T,T" = "Non-serious",
  name = "ae",
  type = "numeric",
  multilabel = TRUE
)
```

1 → Mild Non-serious  
2 → Moderate Non-serious  
3 → Severe Serious  
4 → Life-threat. Serious  
5 → Fatal Serious

## EXPRESSION LABELS

### SPRINTF WITH .X1 (EVALUATED AT APPLY-TIME)

```
stat ← fnew(
  "n" = "sprintf('%s', .x1)",
  "pct" = "sprintf('%1f%%',
    .x1 * 100)",
  name = "stat"
)
```

"42" "5.3%" "100" "25.5%"

```
fput(
  c("n", "pct", "n", "pct"),
  stat,
  c(42, -.053, 100, .255)
)
```

### TWO ARGS (.X1, .X2) + SCALAR RECYCLING

```
r ← fnew(
  "ratio" = "sprintf('%s/%s',
    .x1, .x2)"
)
```

fput("ratio", r, 3, 10)

"3/10"

"42(N=100)" "55(N=100)"  
x2=100 recycled for all elements

```
lbl ← fnew(
  "val" = "sprintf('%s(N=%s)',
    .x1, .x2)"
)
```

fput(cc("val", "val"), lbl,  
c(42, 55), 100)

### IFELSE + MIXED STATIC/EXPRESSION

```
sign ← fnew(
  "val" = "ifelse(.x1 > 0,
    paste0('+', .x1),
    as.character(.x1))"
)
```

fput(rep("val", 3), sign,  
c(5, 0, -3))

"+5" "0" "-3"

"OK" "Err(timeout)"  
"ok" → static · "E01" → other expression

# Mixed: static + expression  
fnew(
 "ok" = "OK",
 .other = "sprintf('Err(ks)',
 .x1)"
)

## VECTORIZED FORMAT NAMES

### EACH ELEMENT USES A DIFFERENT FORMAT (SAS PUTC/PUTN)

```
fnew(
  "1" = "groupx",
  "2" = "groupy",
  "3" = "groupz",
  name = "typefmt",
  type = "numeric"
)
```

	TYPE	RESP	FMT	WORD
1	1	positive	groupx	agree
1	1	negative	groupx	disagree
2	2	positive	groupy	accept
2	2	negative	groupy	reject
3	3	positive	groupz	pass
3	3	negative	groupz	fail

# define groupx, y, z ...  
respfmt ← fput(type,  
"typefmt")  
word ← fputc(response,  
respfmt)

## FUNCTION REFERENCE

CATEGORY	FUNCTION	PURPOSE
Create	fnew(..., name) finput(..., name) fnew_bid(..., name) fnew_date(pattern, name)	Discrete value–label Reverse label–value Both format + invalue Date/time/datetime
Apply	fput(x, format, ...) fputn(x, name, ...) fputc(x, name, ...) fput_all(x, format) fput_df(data, ...)	Generic apply Numeric apply Character apply All labels (multilabel) Data frame columns
Reverse	finputn(x, name) finputc(x, name) format_get(name) fprint(name) fclear(name) fimport(file)	Labels → numeric Labels → character Get from library List/inspect Remove format(s) SAS CNTLOUT CSV
Utils	fparse(text, file) fexport(..., file) range_spec(lo, hi) is_missing(x)	Parse definitions Export to text Build range key NA/NaN/" → TRUE