Package 'fmtr'

March 15, 2026

Type Package

Title Easily Apply Formats to Data

Version 1.7.3

Maintainer David Bosak <dbosak01@gmail.com>

Description Contains a set of functions that can be used to apply formats to data frames or vectors. The package aims to provide functionality similar to that of SAS® formats. Formats are assigned to the format attribute on data frame columns. Then when the fdata() function is called, a new data frame is created with the column data formatted as specified. The package also contains a value() function to create a user-defined format, similar to a SAS® user-defined format.

License CC0

Encoding UTF-8

URL https://fmtr.r-sassy.org, https://github.com/dbosak01/fmtr

BugReports https://github.com/dbosak01/fmtr/issues

Depends R (>= 4.0.0), common

Suggests testthat, knitr, rmarkdown, covr, logr, utils, dplyr, libr, hms

Imports tibble, stats, crayon, Rcpp

RoxygenNote 7.3.3

VignetteBuilder knitr

LinkingTo Rcpp

NeedsCompilation yes

Author David Bosak [aut, cre], Chen Ling [aut]

Repository CRAN

Date/Publication 2026-03-15 11:00:02 UTC

Contents

`as.data.frame.fcat` *Convert a format catalog to a data frame*

Description

This function takes the information stored in a format catalog, and converts it to a data frame. This data frame is useful for storage, editing, saving to a spreadsheet, etc. The data frame shows the name of the formats, their type, and the format expression. For user-defined formats, the data frame populates additional columns for the label and order.

Usage

```
## S3 method for class 'fcat'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	The format catalog to convert.
`row.names`	Row names of the return data frame. Default is NULL.
`optional`	TRUE or FALSE value indicating whether converting to syntactic variable names is desired. In the case of formats, the resulting data frame will always be returned with syntactic names, and this parameter is ignored.
`...`	Any follow-on parameters.

Value

A data frame that contains the values stored in the format catalog.

See Also

Other fcat: `as.fcat()`, `as.fcat.data.frame()`, `as.fcat.fmt_lst()`, `as.fcat.list()`, `fcat()`, `is.fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

Examples

```
# Create a format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d%b%Y")

# Convert catalog to data frame to view the structure
df <- as.data.frame(c1)
print(df)
```

```
#       Name Type Expression   Label Order
# 1   num_fmt    S       %.1f            NA
# 2 label_fmt    U   x == "A" Label A    NA
# 3 label_fmt    U   x == "B" Label B    NA
# 4 label_fmt    U       TRUE   Other    NA
# 5  date_fmt    S     %d%b%Y            NA

# Convert data frame back to a format catalog
c2 <- as.fcat(df)
c2
# # A format catalog: 3 formats
# - $date_fmt: type S, "%d%b%Y"
# - $label_fmt: type U, 3 conditions
# - $num_fmt: type S, "%.1f"
```

`as.data.frame.fmt` *Casts a format to a data frame*

Description

Cast a format object to a data frame. This function is a class-specific implementation of the the generic `as.data.frame` method.

Usage

```
## S3 method for class 'fmt'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  name = deparse(substitute(x, env = environment()))
)
```

Arguments

`x`	An object of class "fmt".
`row.names`	Row names of the return data frame. Default is NULL.
`optional`	TRUE or FALSE value indicating whether converting to syntactic variable names is options. In the case of formats, the resulting data frame will always be returned with syntactic names, and this parameter is ignored.
`...`	Any follow-on parameters.
`name`	An optional name for the format. By default, the name of the variable holding the format will be used.

See Also

Other fmt: `as.fmt()`, `as.fmt.data.frame()`, `condition()`, `is.format()`, `labels.fmt()`, `print.fmt()`, `value()`

`as.data.frame.fmt_lst` *Convert a formatting list to a data frame*

Description

This function takes the information stored in a formatting list, and converts it to a data frame. The data frame format is useful for storage, editing, saving to a spreadsheet, etc. The data frame shows the name of the formats, their type, and the format expression. For user-defined formats, the data frame populates additional columns for the label and order.

Usage

```
## S3 method for class 'fmt_lst'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	The formatting list to convert.
`row.names`	Row names for the returned data frame. Default is NULL.
`optional`	TRUE or FALSE value indicating whether converting to syntactic variable names is desired. In the case of formats, the resulting data frame will always be returned with syntactic names, and this parameter is ignored.
`...`	Any follow-on parameters.

Value

A data frame that contains the values stored in the formatting list.

See Also

Other flist: `as.flist()`, `as.flist.data.frame()`, `as.flist.fcat()`, `as.flist.list()`, `as.flist.tbl_df()`, `flist()`, `is.flist()`, `print.fmt_lst()`, `read.flist()`, `write.flist()`

Examples

```
# Create a formatting list
c1 <- flist(num_fmt  = "%.1f",
            label_fmt = value(condition(x == "A", "Label A"),
                              condition(x == "B", "Label B"),
                              condition(TRUE, "Other")),
            date_fmt = "%d%b%Y")

# Convert catalog to data frame to view the structure
df <- as.data.frame(c1)
print(df)
#       Name Type Expression   Label Order
# 1   num_fmt    S       %.1f            NA
# 2 label_fmt    U   x == "A" Label A    NA
```

```
# 3 label_fmt    U   x == "B" Label B    NA
# 4 label_fmt    U       TRUE   Other    NA
# 5  date_fmt    S     %d%b%Y            NA

# Convert data frame back to a formatting list
c2 <- as.flist(df)
c2
# # A formatting list: 3 formats
# - type: column
# - simplify: TRUE
#        Name Type Expression   Label Order
# 1  date_fmt    S     %d%b%Y          <NA>
# 2 label_fmt    U   x == "A" Label A  <NA>
# 3 label_fmt    U   x == "B" Label B  <NA>
# 4 label_fmt    U       TRUE   Other  <NA>
# 5   num_fmt    S       %.1f          <NA>
```

`as.fcat` *Generic casting method for format catalogs*

Description

A generic method for casting objects to a format catalog. Individual objects will inherit from this function.

Usage

```
as.fcat(x)
```

Arguments

x	The object to cast.

Value

A format catalog, created using the information in the input object.

See Also

For class-specific methods, see `as.fcat.data.frame`, `as.fcat.list`, and `as.fcat.fmt_lst`.

Other fcat: `as.data.frame.fcat()`, `as.fcat.data.frame()`, `as.fcat.fmt_lst()`, `as.fcat.list()`, `fcat()`, `is.fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

`as.fcat.data.frame` *Convert a data frame to a format catalog*

Description

This function takes a data frame as input and converts it to a format catalog based on the information contained in the data frame. The data frame should have 5 columns: "Name", "Type", "Expression", "Label" and "Order".

Usage

```
## S3 method for class 'data.frame'
as.fcat(x)
```

Arguments

`x`	The data frame to convert.

Details

The `as.fcat.data.frame` converts a data frame to a format catalog. A corresponding conversion for class "tbl_df" converts a tibble.

To understand the structure of the input data frame, create a format and use the `as.data.frame` method to convert the format to a data frame. Then observe the columns and organization of the data.

Value

A format catalog based on the information contained in the input data frame.

Input Data Frame Specifications

The input data frame should contain the following columns:

- **Name**: The name of the format
- **Type**: The type of format. See the type codes below.
- **Expression**: The formatting expression. The expression will hold different types of values depending on the format type.
- **Label**: The label for user-defined, "U" type formats.
- **Order**: The order for user-defined, "U" type formats.

Any additional columns will be ignored. Column names are case-insensitive.

Valid values for the "Type" column are as follows:

- **U**: User Defined List created with the `value` function.
- **S**: A formatting string of formatting codes. See FormattingStrings.

- **F**: A vectorized function.
- **V**: A named vector lookup.

The "Label" and "Order" columns are used only for a type "U", user-defined format created with the `value` function.

See Also

Other fcat: `as.data.frame.fcat()`, `as.fcat()`, `as.fcat.fmt_lst()`, `as.fcat.list()`, `fcat()`, `is.fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

Examples

```
# Create a format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d-%b-%Y")

# Convert catalog to data frame to view the structure
df <- as.data.frame(c1)
print(df)
#       Name Type Expression   Label Order
# 1   num_fmt    S       %.1f            NA
# 2 label_fmt    U   x == "A" Label A    NA
# 3 label_fmt    U   x == "B" Label B    NA
# 4 label_fmt    U       TRUE   Other    NA
# 5  date_fmt    S   %d-%b-%Y            NA

# Convert data frame back to a format catalog
c2 <- as.fcat(df)
c2
# # A format catalog: 3 formats
# - $date_fmt: type S, "%d-%b-%Y"
# - $label_fmt: type U, 3 conditions
# - $num_fmt: type S, "%.1f"

# Use re-converted catalog
fapply(123.456, c2$num_fmt)
# [1] "123.5"

fapply(c("A", "B", "C", "B"), c2$label_fmt)
# [1] "Label A" "Label B" "Other"   "Label B"

fapply(Sys.Date(), c2$date_fmt)
# [1] "07-Jan-2024"
```

as.fcat.fmt_lst *Convert a formatting list to a format catalog*

Description

The `as.fcat.list` function converts a formatting list to a format catalog. For additional information on formatting lists, see `flist`.

Usage

```
## S3 method for class 'fmt_lst'
as.fcat(x)
```

Arguments

x	The formatting list to convert.

Value

A format catalog based on the formats contained in the input formatting list.

See Also

Other fcat: `as.data.frame.fcat()`, `as.fcat()`, `as.fcat.data.frame()`, `as.fcat.list()`, `fcat()`, `is.fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

as.fcat.list *Convert a list to a format catalog*

Description

The `as.fcat.list` function converts a list of formats to a format catalog. Items in the list must be named.

Usage

```
## S3 method for class 'list'
as.fcat(x)
```

Arguments

x	The list to convert. List must contained named formats.

Value

A format catalog based on the formats contained in the input list.

See Also

Other fcat: `as.data.frame.fcat()`, `as.fcat()`, `as.fcat.data.frame()`, `as.fcat.fmt_lst()`, `fcat()`, `is.fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

`as.fcat.tbl_df` *Convert a tibble to a format catalog*

Description

This function takes a data frame as input and converts it to a format catalog based on the information contained in the data frame. The data frame should have 5 columns: "Name", "Type", "Expression", "Label" and "Order".

Usage

```
## S3 method for class 'tbl_df'
as.fcat(x)
```

Arguments

x	The data frame to convert.

Details

The `as.fcat.data.frame` converts a data frame to a format catalog. A corresponding conversion for class "tbl_df" converts a tibble.

To understand the structure of the input data frame, create a format and use the `as.data.frame` method to convert the format to a data frame. Then observe the columns and organization of the data.

Value

A format catalog based on the information contained in the input data frame.

Input Data Frame Specifications

The input data frame should contain the following columns:

- **Name**: The name of the format
- **Type**: The type of format. See the type codes below.
- **Expression**: The formatting expression. The expression will hold different types of values depending on the format type.
- **Label**: The label for user-defined, "U" type formats.
- **Order**: The order for user-defined, "U" type formats.

Any additional columns will be ignored. Column names are case-insensitive.

Valid values for the "Type" column are as follows:

- **U**: User Defined List created with the value function.
- **S**: A formatting string of formatting codes. See FormattingStrings.
- **F**: A vectorized function.
- **V**: A named vector lookup.

The "Label" and "Order" columns are used only for a type "U", user-defined format created with the value function.

See Also

Other fcat: as.data.frame.fcat(), as.fcat(), as.fcat.fmt_lst(), as.fcat.list(), fcat(), is.fcat(), print.fcat(), read.fcat(), write.fcat()

Examples

```
# Create a format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d-%b-%Y")

# Convert catalog to data frame to view the structure
df <- as.data.frame(c1)
print(df)
#       Name Type Expression   Label Order
# 1   num_fmt    S       %.1f            NA
# 2 label_fmt    U   x == "A" Label A    NA
# 3 label_fmt    U   x == "B" Label B    NA
# 4 label_fmt    U       TRUE   Other    NA
# 5  date_fmt    S   %d-%b-%Y            NA

# Convert data frame back to a format catalog
c2 <- as.fcat(df)
c2
# # A format catalog: 3 formats
# - $date_fmt: type S, "%d-%b-%Y"
# - $label_fmt: type U, 3 conditions
# - $num_fmt: type S, "%.1f"

# Use re-converted catalog
fapply(123.456, c2$num_fmt)
# [1] "123.5"

fapply(c("A", "B", "C", "B"), c2$label_fmt)
# [1] "Label A" "Label B" "Other"   "Label B"

fapply(Sys.Date(), c2$date_fmt)
# [1] "07-Jan-2024"
```

`as.flist` *Convert to a formatting list*

Description

Converts an object to a formatting list. All other parameters are the same as the `flist` function.

Usage

```
as.flist(x, type = "column", lookup = NULL, simplify = TRUE)
```

Arguments

`x`	Object to convert.
`type`	The type of formatting list. Valid values are 'row' or 'column'. The default value is 'column'.
`lookup`	A lookup vector. Used for looking up the format from the formatting list. This parameter is only used for 'row' type formatting lists.
`simplify`	Whether to simplify the results to a vector. Valid values are TRUE or FALSE. Default is TRUE. If the value is set to FALSE, the return type will be a list.

Details

To apply more than one formatting object to a vector, use a formatting list. There are two types of formatting list: column and row. The column type formatting lists applies all formats to all values in the vector. The row type formatting list can apply a different format to each value in the vector.

Further, there are two styles of row type list: ordered and lookup. The ordered style applies each format in the list to the vector values in the order specified. The ordered style will recycle the formats as needed. The lookup style formatting list uses a lookup to determine which format from the list to apply to a particular value of the vector. The lookup column values should correspond to names on the formatting list.

Examples of column type and row type formatting lists are given below.

Value

A formatting list object.

See Also

Other flist: `as.data.frame.fmt_lst()`, `as.flist.data.frame()`, `as.flist.fcat()`, `as.flist.list()`, `as.flist.tbl_df()`, `flist()`, `is.flist()`, `print.fmt_lst()`, `read.flist()`, `write.flist()`

Examples

```
## Example 1: Formatting List - Column Type ##
# Set up data
v1 <- c(Sys.Date(), Sys.Date() + 30, Sys.Date() + 60)

# Create formatting list
fl1 <- flist("%B", "The month is: %s")

# Apply formatting list to vector
fapply(v1, fl1)
# [1] "The month is: October"  "The month is: November" "The month is: December"

## Example 2: Formatting List - Row Type ordered ##
# Set up data
# Notice each row has a different data type
l1 <- list("A", 1.263, as.Date("2020-07-21"),
          "B", 5.8732, as.Date("2020-10-17"))

# These formats will be recycled in the order specified
fl2 <- flist(type = "row",
             c(A = "Label A", B = "Label B"),
             "%.1f",
             "%d%b%Y")

fapply(l1, fl2)
# [1] "Label A"   "1.3"       "21Jul2020" "Label B"   "5.9"       "17Oct2020"


## Example 3: Formatting List - Row Type with lookup ##

#' # Create formatting list
fl3 <- flist(type = "row",
             DEC1 = "%.1f",
             DEC2 = "%.2f",
             PCT1 = "%.1f%%")

# Set up data
df <- data.frame(CODE = c("DEC1", "DEC2", "PCT1", "DEC2", "PCT1"),
                 VAL = c(41.258, 62.948, 12.125, 65.294, 15.825))

# Assign lookup
fl3$lookup <- df$CODE

# Apply Formatting List
fapply(df$VAL, fl3)
# [1] "41.3"  "62.95" "12.1%" "65.29" "15.8%"

## Example 4: Formatting List - Values with Units ##

#' # Create formatting list
fl4 <- flist(type = "row",
             BASO = "%.2f x10(9)/L",
```

```
              EOS  = "%.2f x10(9)/L",
              HCT = "%.1f%%",
              HGB = "%.1f g/dL")

# Set up data
df <- data.frame(CODE = c("BASO", "EOS", "HCT", "HGB"),
                 VAL = c(0.02384, 0.14683, 40.68374, 15.6345))

# Assign lookup
fl4$lookup <- df$CODE

# Apply Formatting List
df$VALC <- fapply(df$VAL, fl4)

# View results
df
#   CODE      VAL          VALC
# 1 BASO  0.02384 0.02 x10(9)/L
# 2  EOS  0.14683 0.15 x10(9)/L
# 3  HCT 40.68374         40.7%
# 4  HGB 15.63450     15.6 g/dL
```

`as.flist.data.frame` *Convert a data frame to a formatting list*

Description

Converts a data frame to a formatting list. All other parameters are the same as the `flist` function.

Usage

```
## S3 method for class 'data.frame'
as.flist(x, type = "column", lookup = NULL, simplify = TRUE)
```

Arguments

`x`	Data frame to convert.
`type`	The type of formatting list. Valid values are 'row' or 'column'. The default value is 'column'.
`lookup`	A lookup vector. Used for looking up the format from the formatting list. This parameter is only used for 'row' type formatting lists.
`simplify`	Whether to simplify the results to a vector. Valid values are TRUE or FALSE. Default is TRUE. If the value is set to FALSE, the return type will be a list.

Details

To apply more than one formatting object to a vector, use a formatting list. There are two types of formatting list: column and row. The column type formatting lists applies all formats to all values in the vector. The row type formatting list can apply a different format to each value in the vector.

Further, there are two styles of row type list: ordered and lookup. The ordered style applies each format in the list to the vector values in the order specified. The ordered style will recycle the formats as needed. The lookup style formatting list uses a lookup to determine which format from the list to apply to a particular value of the vector. The lookup column values should correspond to names on the formatting list.

Examples of column type and row type formatting lists are given below.

Value

A formatting list object.

See Also

Other flist: `as.data.frame.fmt_lst()`, `as.flist()`, `as.flist.fcat()`, `as.flist.list()`, `as.flist.tbl_df()`, `flist()`, `is.flist()`, `print.fmt_lst()`, `read.flist()`, `write.flist()`

Examples

```
## Example 1: Formatting List - Column Type ##
# Set up data
v1 <- c(Sys.Date(), Sys.Date() + 30, Sys.Date() + 60)

# Create formatting list
fl1 <- flist("%B", "The month is: %s")

# Apply formatting list to vector
fapply(v1, fl1)
# [1] "The month is: October"  "The month is: November" "The month is: December"

## Example 2: Formatting List - Row Type ordered ##
# Set up data
# Notice each row has a different data type
l1 <- list("A", 1.263, as.Date("2020-07-21"),
          "B", 5.8732, as.Date("2020-10-17"))

# These formats will be recycled in the order specified
fl2 <- flist(type = "row",
             c(A = "Label A", B = "Label B"),
             "%.1f",
             "%d%b%Y")

fapply(l1, fl2)
# [1] "Label A"   "1.3"       "21Jul2020" "Label B"   "5.9"       "17Oct2020"


## Example 3: Formatting List - Row Type with lookup ##
```

```
#' # Create formatting list
fl3 <- flist(type = "row",
             DEC1 = "%.1f",
             DEC2 = "%.2f",
             PCT1 = "%.1f%%")

# Set up data
df <- data.frame(CODE = c("DEC1", "DEC2", "PCT1", "DEC2", "PCT1"),
                 VAL = c(41.258, 62.948, 12.125, 65.294, 15.825))

# Assign lookup
fl3$lookup <- df$CODE

# Apply Formatting List
fapply(df$VAL, fl3)
# [1] "41.3"  "62.95" "12.1%" "65.29" "15.8%"

## Example 4: Formatting List - Values with Units ##

#' # Create formatting list
fl4 <- flist(type = "row",
             BASO = "%.2f x10(9)/L",
             EOS  = "%.2f x10(9)/L",
             HCT = "%.1f%%",
             HGB = "%.1f g/dL")

# Set up data
df <- data.frame(CODE = c("BASO", "EOS", "HCT", "HGB"),
                 VAL = c(0.02384, 0.14683, 40.68374, 15.6345))

# Assign lookup
fl4$lookup <- df$CODE

# Apply Formatting List
df$VALC <- fapply(df$VAL, fl4)

# View results
df
#   CODE      VAL          VALC
# 1 BASO  0.02384 0.02 x10(9)/L
# 2  EOS  0.14683 0.15 x10(9)/L
# 3  HCT 40.68374         40.7%
# 4  HGB 15.63450     15.6 g/dL
```

---

as.flist.fcat *Convert a format catalog to a formatting list*

Description

Converts a format catalog to a formatting list. All other parameters are the same as the `flist` function.

Usage

```
## S3 method for class 'fcat'
as.flist(x, type = "column", lookup = NULL, simplify = TRUE)
```

Arguments

`x`	Format catalog to convert.
`type`	The type of formatting list. Valid values are 'row' or 'column'. The default value is 'column'.
`lookup`	A lookup vector. Used for looking up the format from the formatting list. This parameter is only used for 'row' type formatting lists.
`simplify`	Whether to simplify the results to a vector. Valid values are TRUE or FALSE. Default is TRUE. If the value is set to FALSE, the return type will be a list.

Details

To apply more than one formatting object to a vector, use a formatting list. There are two types of formatting list: column and row. The column type formatting lists applies all formats to all values in the vector. The row type formatting list can apply a different format to each value in the vector.

Further, there are two styles of row type list: ordered and lookup. The ordered style applies each format in the list to the vector values in the order specified. The ordered style will recycle the formats as needed. The lookup style formatting list uses a lookup to determine which format from the list to apply to a particular value of the vector. The lookup column values should correspond to names on the formatting list.

Examples of column type and row type formatting lists are given below.

Value

A formatting list object.

See Also

Other flist: `as.data.frame.fmt_lst()`, `as.flist()`, `as.flist.data.frame()`, `as.flist.list()`, `as.flist.tbl_df()`, `flist()`, `is.flist()`, `print.fmt_lst()`, `read.flist()`, `write.flist()`

Examples

```
## Example 1: Formatting List - Column Type ##
# Set up data
v1 <- c(Sys.Date(), Sys.Date() + 30, Sys.Date() + 60)

# Create formatting list
fl1 <- flist("%B", "The month is: %s")
```

```
# Apply formatting list to vector
fapply(v1, fl1)
# [1] "The month is: October"  "The month is: November" "The month is: December"

## Example 2: Formatting List - Row Type ordered ##
# Set up data
# Notice each row has a different data type
l1 <- list("A", 1.263, as.Date("2020-07-21"),
          "B", 5.8732, as.Date("2020-10-17"))

# These formats will be recycled in the order specified
fl2 <- flist(type = "row",
             c(A = "Label A", B = "Label B"),
             "%.1f",
             "%d%b%Y")

fapply(l1, fl2)
# [1] "Label A"   "1.3"       "21Jul2020" "Label B"   "5.9"       "17Oct2020"


## Example 3: Formatting List - Row Type with lookup ##

#' # Create formatting list
fl3 <- flist(type = "row",
             DEC1 = "%.1f",
             DEC2 = "%.2f",
             PCT1 = "%.1f%%")

# Set up data
df <- data.frame(CODE = c("DEC1", "DEC2", "PCT1", "DEC2", "PCT1"),
                 VAL = c(41.258, 62.948, 12.125, 65.294, 15.825))

# Assign lookup
fl3$lookup <- df$CODE

# Apply Formatting List
fapply(df$VAL, fl3)
# [1] "41.3"  "62.95" "12.1%" "65.29" "15.8%"

## Example 4: Formatting List - Values with Units ##

#' # Create formatting list
fl4 <- flist(type = "row",
             BASO = "%.2f x10(9)/L",
             EOS  = "%.2f x10(9)/L",
             HCT = "%.1f%%",
             HGB = "%.1f g/dL")

# Set up data
df <- data.frame(CODE = c("BASO", "EOS", "HCT", "HGB"),
                 VAL = c(0.02384, 0.14683, 40.68374, 15.6345))
```

```
# Assign lookup
fl4$lookup <- df$CODE

# Apply Formatting List
df$VALC <- fapply(df$VAL, fl4)

# View results
df
#   CODE      VAL           VALC
# 1 BASO  0.02384 0.02 x10(9)/L
# 2  EOS  0.14683 0.15 x10(9)/L
# 3  HCT 40.68374         40.7%
# 4  HGB 15.63450     15.6 g/dL
```

`as.flist.list` *Convert a list to a formatting list*

Description

Converts a normal list to a formatting list. All other parameters are the same as the `flist` function.

Usage

```
## S3 method for class 'list'
as.flist(x, type = "column", lookup = NULL, simplify = TRUE)
```

Arguments

`x`	List to convert.
`type`	The type of formatting list. Valid values are 'row' or 'column'. The default value is 'column'.
`lookup`	A lookup vector. Used for looking up the format from the formatting list. This parameter is only used for 'row' type formatting lists.
`simplify`	Whether to simplify the results to a vector. Valid values are TRUE or FALSE. Default is TRUE. If the value is set to FALSE, the return type will be a list.

Details

To apply more than one formatting object to a vector, use a formatting list. There are two types of formatting list: column and row. The column type formatting lists applies all formats to all values in the vector. The row type formatting list can apply a different format to each value in the vector.

Further, there are two styles of row type list: ordered and lookup. The ordered style applies each format in the list to the vector values in the order specified. The ordered style will recycle the formats as needed. The lookup style formatting list uses a lookup to determine which format from the list to apply to a particular value of the vector. The lookup column values should correspond to names on the formatting list.

Examples of column type and row type formatting lists are given below.

Value

A formatting list object.

See Also

flist function documentation for additional details.

Other flist: as.data.frame.fmt_lst(), as.flist(), as.flist.data.frame(), as.flist.fcat(), as.flist.tbl_df(), flist(), is.flist(), print.fmt_lst(), read.flist(), write.flist()

Examples

```
# Example 1: Create flist from list - column type
lst1 <- list("%d%b%Y", "%.1f")
fl1  <- as.flist(lst1, type = "column")

# Example 2: Create flist from list - row type
lst2 <- list(lkup = c(A = "Label A", B = "Label B"),
             dec1 = "%.1f",
             dt1  = "%d%b%Y")
fl2 <- as.flist(lst2, type = "row")
```

---

as.flist.tbl_df *Convert a tibble to a formatting list*

Description

Converts a tibble to a formatting list. All other parameters are the same as the flist function.

Usage

```
## S3 method for class 'tbl_df'
as.flist(x, type = "column", lookup = NULL, simplify = TRUE)
```

Arguments

x	Tibble to convert.
type	The type of formatting list. Valid values are 'row' or 'column'. The default value is 'column'.
lookup	A lookup vector. Used for looking up the format from the formatting list. This parameter is only used for 'row' type formatting lists.
simplify	Whether to simplify the results to a vector. Valid values are TRUE or FALSE. Default is TRUE. If the value is set to FALSE, the return type will be a list.

Details

To apply more than one formatting object to a vector, use a formatting list. There are two types of formatting list: column and row. The column type formatting lists applies all formats to all values in the vector. The row type formatting list can apply a different format to each value in the vector.

Further, there are two styles of row type list: ordered and lookup. The ordered style applies each format in the list to the vector values in the order specified. The ordered style will recycle the formats as needed. The lookup style formatting list uses a lookup to determine which format from the list to apply to a particular value of the vector. The lookup column values should correspond to names on the formatting list.

Examples of column type and row type formatting lists are given below.

Value

A formatting list object.

See Also

Other flist: `as.data.frame.fmt_lst()`, `as.flist()`, `as.flist.data.frame()`, `as.flist.fcat()`, `as.flist.list()`, `flist()`, `is.flist()`, `print.fmt_lst()`, `read.flist()`, `write.flist()`

Examples

```
## Example 1: Formatting List - Column Type ##
# Set up data
v1 <- c(Sys.Date(), Sys.Date() + 30, Sys.Date() + 60)

# Create formatting list
fl1 <- flist("%B", "The month is: %s")

# Apply formatting list to vector
fapply(v1, fl1)
# [1] "The month is: October"  "The month is: November" "The month is: December"

## Example 2: Formatting List - Row Type ordered ##
# Set up data
# Notice each row has a different data type
l1 <- list("A", 1.263, as.Date("2020-07-21"),
          "B", 5.8732, as.Date("2020-10-17"))

# These formats will be recycled in the order specified
fl2 <- flist(type = "row",
             c(A = "Label A", B = "Label B"),
             "%.1f",
             "%d%b%Y")

fapply(l1, fl2)
# [1] "Label A"   "1.3"       "21Jul2020" "Label B"   "5.9"       "17Oct2020"


## Example 3: Formatting List - Row Type with lookup ##
```

```
#' # Create formatting list
fl3 <- flist(type = "row",
             DEC1 = "%.1f",
             DEC2 = "%.2f",
             PCT1 = "%.1f%%")

# Set up data
df <- data.frame(CODE = c("DEC1", "DEC2", "PCT1", "DEC2", "PCT1"),
                 VAL = c(41.258, 62.948, 12.125, 65.294, 15.825))

# Assign lookup
fl3$lookup <- df$CODE

# Apply Formatting List
fapply(df$VAL, fl3)
# [1] "41.3"  "62.95" "12.1%" "65.29" "15.8%"

## Example 4: Formatting List - Values with Units ##

#' # Create formatting list
fl4 <- flist(type = "row",
             BASO = "%.2f x10(9)/L",
             EOS  = "%.2f x10(9)/L",
             HCT = "%.1f%%",
             HGB = "%.1f g/dL")

# Set up data
df <- data.frame(CODE = c("BASO", "EOS", "HCT", "HGB"),
                 VAL = c(0.02384, 0.14683, 40.68374, 15.6345))

# Assign lookup
fl4$lookup <- df$CODE

# Apply Formatting List
df$VALC <- fapply(df$VAL, fl4)

# View results
df
#   CODE      VAL          VALC
# 1 BASO  0.02384 0.02 x10(9)/L
# 2  EOS  0.14683 0.15 x10(9)/L
# 3  HCT 40.68374         40.7%
# 4  HGB 15.63450     15.6 g/dL
```

`as.fmt` *Generic casting method for formats*

Description

A generic method for casting objects to a format. Individual objects will inherit from this function.

Usage

```
as.fmt(x)
```

Arguments

x	The object to cast.

Value

A formatting object, created using the information in the input object.

See Also

Other fmt: `as.data.frame.fmt()`, `as.fmt.data.frame()`, `condition()`, `is.format()`, `labels.fmt()`, `print.fmt()`, `value()`

---

as.fmt.data.frame *Convert a data frame to a user-defined format*

Description

This function takes a data frame as input and converts it to a user-defined format based on the information contained in the data frame. The data frame should have 5 columns: "Name", "Type", "Expression", "Label" and "Order".

Usage

```
## S3 method for class 'data.frame'
as.fmt(x)
```

Arguments

x	The data frame to convert.

Details

The `as.fmt.data.frame` function converts a data frame to a user-defined format.

To understand the structure of the input data frame, create a user-defined format and use the `as.data.frame` method to convert the format to a data frame. Then observe the columns and organization of the data.

Value

A format catalog based on the information contained in the input data frame.

Input Data Frame Specifications

The input data frame should contain the following columns:

- **Name**: The name of the format
- **Type**: The type of format. See the type codes below.
- **Expression**: The formatting expression. The expression will hold different types of values depending on the format type. Within the data frame, this expression is stored as a character string.
- **Label**: The label for user-defined, "U" type formats.
- **Order**: The order for user-defined, "U" type formats.
- **Factor**: An optional column for "U" type formats that sets the "as.factor" parameter. Valid values are TRUE, FALSE, or NA.

Any additional columns will be ignored. Column names are case-insensitive.

Valid values for the "Type" column are as follows:

- **U**: User Defined List created with the `value` function.
- **S**: A formatting string of formatting codes. See FormattingStrings.
- **F**: A vectorized function.
- **V**: A named vector lookup.

The "Label", "Order", and "Factor" columns are used only for a type "U", user-defined format created with the `value` function.

See Also

Other fmt: `as.data.frame.fmt()`, `as.fmt()`, `condition()`, `is.format()`, `labels.fmt()`, `print.fmt()`, `value()`

Examples

```
# Create a user-defined format
f1 <- value(condition(x == "A", "Label A"),
            condition(x == "B", "Label B"),
            condition(TRUE, "Other"))

# Convert user-defined format to data frame to view the structure
df <- as.data.frame(f1)
print(df)

# Name Type Expression   Label Order Factor
# 1 f1    U   x == "A" Label A    NA  FALSE
# 2 f1    U   x == "B" Label B    NA  FALSE
# 3 f1    U       TRUE   Other    NA  FALSE

# Convert data frame back to a user-defined format
f2 <- as.fmt(df)
```

```
# Use re-converted format
fapply(c("A", "B", "C", "B"), f2)
# [1] "Label A" "Label B" "Other"   "Label B"
```

`condition` *Define a condition for a user-defined format*

Description

The `condition` function creates a condition for a user-defined format. It is typically used in conjunction with the `value` function.

Usage

```
condition(expr, label, order = NULL)
```

Arguments

`expr`	A valid R expression. The value in the expression is identified by the variable 'x', i.e. x == 'A' or x > 3 & x < 6. The expression should not be quoted. The expression parameter will accept equality, relational, and logical operators. It will also accept numeric or string literals. String literals should be quoted. It will not accept functions or any expression that includes a comma. For these more complex operations, it is best to use a vectorized function. See `fapply` for an example of a vectorized function.
`label`	A label to be assigned if the expression is TRUE. The label can any valid literal value. Typically, the label will be a character string. However, the label parameter does not restrict the data type. Meaning, the label could also be a number, date, or other R object type. The label may also be a string format, which allows you to perform conditional formatting.
`order`	An optional integer order number. When used, this parameter will effect the order of the labels returned from the `labels.fmt` function. The purpose of the parameter is to control ordering of the format labels independently of the order they are assigned in the conditions. The order parameter is useful when you are using the format labels to assign ordered levels in a factor.

Details

The `condition` function creates a condition as part of a format definition. The format is defined using the `value` function. The condition is defined as an expression/label pair. The expression parameter can be any valid R expression. The label parameter can be any valid literal. Conditions are evaluated in the order they are assigned. A default condition is created by assigning the expression parameter to TRUE. If your data can contain missing values, it is recommended that you test for those values first. Any data values that do not meet one of the conditions will fall through the format as-is.

The condition object is an S3 class of type "fmt_cond". The condition labels can be extracted from the format using the `labels` function.

The format object may be applied to a vector using the `fapply` function. See `fapply` for further details.

Value

The new condition object.

See Also

`fdata` to apply formatting to a data frame, `value` to define a format, `levels` or `labels.fmt` to access the labels, and `fapply` to apply the format to a vector.

Other fmt: `as.data.frame.fmt()`, `as.fmt()`, `as.fmt.data.frame()`, `is.format()`, `labels.fmt()`, `print.fmt()`, `value()`

Examples

```
# Set up vector
v1 <- c("A", "B", "C", "B")

# Define format
fmt1 <- value(condition(x == "A", "Label A"),
              condition(x == "B", "Label B"),
              condition(TRUE, "Other"))

# Apply format to vector
v2 <- fapply(v1, fmt1)
v2
# [1] "Label A" "Label B" "Other"   "Label B"
```

---

descriptions *Get or set descriptions for data frame columns*

Description

The `descriptions` function extracts all assigned description attributes from a data frame, and returns them in a named list. The function also assigns description attributes from a named list.

Usage

```
descriptions(x)

descriptions(x, verbose = FALSE) <- value
```

Arguments

`x`	A data frame or tibble
`verbose`	If TRUE, the function will emit messages regarding which descriptions have been added or revised, and which are still missing. This option can help you if you are filling out a lot of descriptions on a large dataset.
`value`	A named list of description values.

Details

If descriptions are assigned to the "description" attributes of the data frame columns, the `descriptions` function will extract those values. The function will return the description values in a named list, where the names correspond to the name of the column that the description was assigned to. If a column does not have a description attribute assigned, that column will not be included in the list.

When used on the receiving side of an assignment, the function will assign descriptions to a data frame. The description values should be in a named list, where each name corresponds to the name of the data frame column to assign values to.

Finally, if you wish to clear out the description attributes, assign a NULL value to the `descriptions` function.

Value

A named list of description values.

See Also

`fdata` to display formatted data, `value` to create user-defined formats, and `fapply` to apply formatting to a vector.

Examples

```
# Take subset of data
df1 <- mtcars[1:5, c("mpg", "cyl") ]

# Print current state
print(df1)
#                   mpg cyl
# Mazda RX4         21.0   6
# Mazda RX4 Wag     21.0   6
# Datsun 710        22.8   4
# Hornet 4 Drive    21.4   6
# Hornet Sportabout 18.7   8

# Assign descriptions
descriptions(df1) <- list(mpg = "Miles per Gallon", cyl = "Cylinders")

# Display descriptions
descriptions(df1)
# $mpg
# [1] "Miles per Gallon"
#
# $cyl
# [1] "Cylinders"

# Clear descriptions
descriptions(df1) <- NULL

# Confirm descriptions are cleared
descriptions(df1)
# list()
```

`fapply` *Apply formatting to a vector*

Description

The `fapply` function applies formatting to a vector.

Usage

```
fapply(x, format = NULL, width = NULL, justify = NULL)
```

Arguments

`x`	A vector, factor, or list to apply the format to.
`format`	A format to be applied.
`width`	The desired character width of the formatted vector. Default value is NULL, meaning the vector will be variable width.
`justify`	Whether to justify the return vector. Valid values are 'left', 'right', 'center', 'centre', or 'none'.

Details

The `fapply` function accepts several types of formats: formatting strings, named vectors, vectorized functions, or user-defined formats. It also accepts a formatting list, composed of any of the previous types. The function will first determine the type of format, and then apply the format in the appropriate way. Results are returned as a vector.

The function also has parameters for width and justification.

Parameters may also be passed as attributes on the vector. See the `fattr` function for additional information on setting formatting attributes.

Value

A vector of formatted values.

Types of Formats

The `fapply` function will process any of the following types of formats:

- **Formatting string:** A single string will be interpreted as a formatting string. Formatting strings include R-style formatting codes, and some SAS-style format names like "best", "date", and "datetime". See the FormattingStrings documentation and the sections below for further details.
- **Named vector:** A named vector can serve as a lookup list or decode for a vector. You can use a named vector to perform simple lookups on character vectors.
- **Format object:** A format object may be created using the `value` function. The format object is included in the **fmtr** package, and is specially designed for data categorization.

- **Vectorized formatting function:** A vectorized function provides the most flexibility and power over your formatting. You can use an existing formatting function from any package, or create your own vectorized formatting function using `Vectorize`.

`fapply` will also accept a formatting list, which can contain any number of formats from the above list. To create a formatting list, see the `flist` function.

"best" Format

The SAS "best" format is used to fit numeric values within a certain width. The word "best" is followed by the desired width, i.e. "best6" or "best12". The format will then use the most optimal display for the available width.

The format will use the entire value if the number of digits fits in the desired width. If not, the format may round the value. The format may also use scientific notation if the value is very large or very small. If the format cannot fit the value in the desired width at all, it will emit stars ("*") in the desired width.

For input values that are less than the desired width, the result will be left-padded with spaces. The output value will then always contain the exact number of characters requested.

Such a format has no direct equivalent in R, and is indeed difficult to replicate. For this reason, the **fmtr** package added this format option for those situations when you want to replicate SAS "best" formatting as closely as possible.

The "best" format accepts widths between 1 and 32. The default width is 12. The R "best" format syntax does not accept a number of decimals, as in "bestW.d".

Note that "best" widths between 8 and 16 will match SAS most reliably. Small widths have many special cases, and the logic is difficult to replicate. For large values, there are some differences between SAS and R in how they represent these numbers, and sometimes they will not match.

"date" Format

The "date" format is used to display date values in a readable character form, such as "01JAN70" or "01-JAN-1970", depending on the specified width. The word "date" is followed by the desired width, e.g. "date7" or "date9". This format replicates similar capabilities in SAS.

The format converts numeric or Date values into character strings using a pattern that depends on the width. Smaller widths display shorter forms, while larger widths display more detail. For example:

- **date5** – Displays as `mmmyy` (e.g., "JAN70")
- **date7** – Displays as `ddmmmyy` (e.g., "01JAN70")
- **date9** – Displays as `ddmmmyyyy` (e.g., "01JAN1970")
- **date11** – Displays as `dd-mmm-yyyy` (e.g., "01-JAN-1970")

The "date" format accepts widths between 5 and 11. Widths outside this range are not valid and will result in an error. The default width is 7. Both `"dateW"` and `"dateW."` are accepted, the trailing dot (".") is optional and does not affect behavior.

For input values that are numeric, the function will interpret them as the number of days since 1970-01-01, consistent with R's internal date representation. If the input is already an R `Date` or `POSIXt` object, it will be used directly. Missing values will be returned as missing.

The output value is left-padded with spaces if it is shorter than the requested width, ensuring the formatted result always occupies exactly the specified number of characters. For example, for the date 1970-01-01, the result of `date7` is `"01JAN70"`, while the result of `date8` is `" 01JAN70"`, with one additional leading space.

This format has no direct equivalent in base R. The **fmtr** package adds this capability for users who wish to replicate SAS-style "date" formatting behavior as closely as possible.

"time" Format

The **"time"** format is used to display time-of-day values in a readable character form, such as `"9:00"` or `"23:59:59.995"`, depending on the specified width and number of decimal places. The word "time" is followed by the desired width *w* and optional decimal precision *d*, e.g. `"time8"`, `"time12.3"`, or `"time20.9"`. This format closely replicates the behavior of SAS `TIMEw.d`.

The format converts numeric or time-like objects into character strings representing elapsed time in hours, minutes, and seconds. Hours are not limited to the 0–23 range and may exceed 24, allowing the format to represent durations such as `"119:26:40"` or `"1388:53:20"`.

Width and precision: The total width *w* controls the minimum number of characters in the output, while the optional decimal precision *d* controls the number of digits displayed after the decimal point for seconds.

- **time5** – Displays hours and minutes or only hours (e.g. " 9:00", "24:00", " 120")
- **time7** – Displays h:mm:ss or hh:mm (e.g., "1:03:10", " 23:59")
- **time8** – Displays as hh:mm:ss (e.g. " 1:03:10", "23:59:59")
- **time9.1** – Displays seconds with one decimal place (e.g., "1:00:00.0")
- **time12.3** – Displays seconds with three decimal places (e.g., " 9:00:01.005")
- **time20.9** – Displays seconds with up to nine decimal places (e.g., " 9:00:00.987654321")

Valid widths range from 2 to 20. The decimal precision *d*, when specified, must be between 0 and `w - 1`. Widths or precisions outside these ranges are not valid and will result in an error.

If the width *w* is omitted (e.g., `"time"` or `"time."`), it defaults to `8`. If the decimal precision *d* is omitted, it defaults to `0`. Both `"TIMEw"` and `"TIMEw."` are accepted, and the trailing dot is optional.

Input handling: The `TIMEw.d` format accepts the following input types:

- Numeric values, interpreted as the number of seconds since midnight
- `POSIXt` objects, using the time-of-day component
- `hms` objects from the **hms** package
- `difftime` objects

For numeric, `hms` and `difftime` inputs, negative values and values larger than 24 hours are allowed and formatted accordingly, consistent with SAS behavior. For example, values such as `-3600` or `430000` are valid. For `POSIXt` inputs, only the clock time is used. As a result, negative times and times exceeding 24 hours are not applicable to `POSIXt` objects.

The `TIMEw.d` format resolves known differences in fractional-second rounding between base R and SAS, it applies SAS-compatible rounding to ensure that formatted results match SAS output exactly, particularly near rounding boundaries.

In addition, whereas base R effectively limits fractional seconds to 6 digits, `TIMEw.d` supports up to 12 digits of decimal precision, padding with trailing zeros when necessary, consistent with SAS behavior.

"datetime" Format

The **"datetime"** format is used to display date-time values in a readable character form, such as `"01JAN70:00:00:00"` or `"31DEC1999:23:59:59.995"`, depending on the specified width and number of decimal places. The word `"datetime"` is followed by the desired width *w* and optional decimal precision *d*, e.g. `"datetime16"`, `"datetime22.3"`, or `"datetime40.12"`. This format closely replicates the behavior of SAS `DATETIMEw.d`.

The format accepts numeric values and `POSIXt` objects. Numeric inputs follow R conventions and are interpreted as seconds since `1970-01-01 00:00:00`. Negative numeric values are allowed and represent datetimes before 1970. For `POSIXt` inputs, the full date and time components are used.

Width and precision: The total width *w* controls the minimum number of characters in the output, while the optional decimal precision *d* controls the number of digits displayed after the decimal point for seconds. Valid widths range from 7 to 40. The decimal precision *d*, when specified, must be between 0 and `w - 1`. Widths or precisions outside these ranges are not valid and will result in an error.

The exact appearance depends on the requested width. Narrower widths may use abbreviated forms, while wider widths allow full date-time values, four-digit years, and fractional seconds. Common examples include:

- **datetime16** – Displays date and time with a two-digit year (e.g., "01JAN70:00:00:00")
- **datetime18** – Displays date and time with a four-digit year (e.g., "01JAN1970:00:00:00")
- **datetime22.3** – Displays seconds with three decimal places (e.g., "01JAN1970:00:00:00.000")
- **datetime25.6** – Displays seconds with six decimal places (e.g., "31DEC1999:23:59:59.995000")
- **datetime40.12** – Displays seconds with up to twelve decimal places and pads to the requested width

If the width *w* is omitted (e.g., `"datetime"` or `"datetime."`), it defaults to `16`. If the decimal precision *d* is omitted, it defaults to `0`. Both `"DATETIMEw"` and `"DATETIMEw."` are accepted, and the trailing dot is optional.

Input handling: The `DATETIMEw.d` format accepts the following input types:

- `POSIXt` objects, which have both a date and time component.
- Numeric values, interpreted as the number of seconds since `1970-01-01 00:00:00` in the local time zone.

For numeric input, negative values and values far beyond a single day are allowed and formatted accordingly, consistent with SAS behavior. For `POSIXt` input, the full calendar date and time components are used. If the time zone is not included in the `POSIXt` value, the function will use the local time zone. Likewise for numeric datetime values, because numeric values do not inherently carry time zone information, `DATETIMEw.d` interprets them in the local time zone.

To view the time zone for your session, use `Sys.timezone()`. To change the time zone, use `Sys.setenv(TZ = "<timezone code>")`.

In addition, whereas base R effectively limits fractional seconds to 6 digits, `DATETIMEw.d` supports display of up to 23 digits after the decimal point. Precision is maintained through 15 digits; beyond that, additional decimal places are padded with zeros.

See Also

fcat to create a format catalog, value to define a format, fattr to easily set the formatting attributes of a vector, and flist to define a formatting list. Also see fdata to apply formats to an entire data frame, and FormattingStrings for how to define a formatting string.

Examples

```
## Example 1: Formatting string ##
v1 <- c(1.235, 8.363, 5.954, 2.465)

# Apply string format.
fapply(v1, "%.1f")
# [1] "1.2" "8.4" "6.0" "2.5"

# Apply width and two decimals
fapply(v1, "%5.2f")
# [1] " 1.24" " 8.36" " 5.95" " 2.46"

# Apply "best" format
fapply(v1, "best3")
# [1] "1.2" "8.4" "  6" "2.5"

## Example 2: Named vector ##
# Set up vector
v2 <- c("A", "B", "C", "B")

# Set up named vector for formatting
fmt2 <- c(A = "Label A", B = "Label B", C = "Label C")

# Apply format to vector
fapply(v2, fmt2)
# [1] "Label A" "Label B" "Label C" "Label B"

## Example 3: User-defined format ##
# Define format
fmt3 <- value(condition(x == "A", "Label A"),
              condition(x == "B", "Label B"),
              condition(TRUE, "Other"))

# Apply format to vector
fapply(v2, fmt3)
# [1] "Label A" "Label B" "Other"   "Label B"

## Example 4: Formatting function ##
# Set up vectorized function
fmt4 <- Vectorize(function(x) {

  if (x %in% c("A", "B"))
    ret <- paste("Label", x)
  else
    ret <- "Other"
```

```
  return(ret)
})

# Apply format to vector
fapply(v2, fmt4)
# [1] "Label A" "Label B" "Other"   "Label B"

## Example 5: Formatting List - Row Type ##
# Set up data
# Notice each row has a different data type
v3 <- list(2841.258, "H", Sys.Date(),
           "L", Sys.Date() + 60, 1382.8865)
v4 <- c("int", "char", "date", "char", "date", "int")

# Create formatting list
lst <- flist(type = "row", lookup = v4,
       int = function(x) format(x, digits = 2, nsmall = 1,
                                big.mark=","),
       char = value(condition(x == "H", "High"),
                    condition(x == "L", "Low"),
                    condition(TRUE, "NA")),
       date = "%d%b%Y")

# Apply formatting list to vector
fapply(v3, lst)
# [1] "2,841.3"   "High"      "06Jan2024" "Low"       "06Mar2024" "1,382.9"

## Example 6: Formatting List - Column Type ##
# Set up data
v5 <- c(Sys.Date(), Sys.Date() + 30, Sys.Date() + 60)
v5
# [1] "2024-01-06" "2024-02-05" "2024-03-06"

# Create formatting list
lst <- flist("%B", "This month is: %s", type = "column")

# Apply formatting list to vector
fapply(v5, lst)
# [1] "This month is: January"  "This month is: February" "This month is: March"

# Example 7: Conditional Formatting
# Data vector
v6 <- c(8.38371, 1.46938, 3.28783, NA, 0.98632)

# User-defined format
fmt5 <- value(condition(is.na(x), "Missing"),
              condition(x < 1, "Low"),
              condition(x > 5, "High"),
              condition(TRUE, "%.2f"))

# Apply format to data vector
fapply(v6, fmt5)
# [1] "High"    "1.47"    "3.29"    "Missing" "Low"
```

```
# Example 8: "best" Format
#' # Data vector
v7 <- c(12.3456, 1234567.89, NA, 0.123456, 0.000012345)

fapply(v7, "best6")
# [1] "12.346" "1.23E6" NA       "0.1235" "123E-7"

# Example 9: "date" Format
# Data Vector
v8 <- as.Date(c("1924-02-29",NA,"1980-12-31","2019-12-31","2020-02-29","2030-08-20"))

fapply(v8, "date7")
# [1] "29FEB24" NA   "31DEC80" "31DEC19" "29FEB20" "20AUG30"

fapply(v8, "date11")
# [1] "29-FEB-1924" NA   "31-DEC-1980" "31-DEC-2019" "29-FEB-2020" "20-AUG-2030"

# Example 10: "time" format
# Data vector
v9 <- c(-3600, NA, 0, 59.9, 3600.12345, 86399.995, 90000)
v10 <- strptime(c("01:00:00.123", "09:00:01.456", NA), format="%H:%M:%OS")
fapply(v9, "time8")
# [1]  "-1:00:00" NA         " 0:00:00" " 0:01:00" " 1:00:00" "24:00:00" "25:00:00"

fapply(v10, "time10.2")
# [1] "1:00:00.12" "9:00:01.46" NA

# Example 11: "datetime" format
# Data vectors
v11 <- c(-1, NA, 0, 315619199, 1267446896, 1752566950)
v12 <- as.POSIXct(c("1960-01-01 00:00:00.123",
                    "1999-12-31 23:59:59.995",
                    NA),
                  format = "%Y-%m-%d %H:%M:%OS", tz = "UTC")

fapply(v11, "datetime18")

# [1] "  31DEC69:23:59:59" NA                   "  01JAN70:00:00:00" "  01JAN80:23:59:59"
# [5] "  01MAR10:12:34:56" "  15JUL25:08:09:10"

fapply(v12, "datetime22.3")
# [1] "01JAN1960:00:00:00.123" "31DEC1999:23:59:59.995" NA
```

`fapply2` *Apply formatting to two vectors*

Description

The `fapply2` function applies formatting to two different vectors, and combines them into a single vector. This function is useful in cases where your data is in two different variables, and you would

like them displayed as a single column for reporting purposes. For example, if you wish to create one column to display mean and standard deviation.

Usage

```
fapply2(
  x1,
  x2,
  format1 = NULL,
  format2 = NULL,
  sep = " ",
  width = NULL,
  justify = NULL
)
```

Arguments

x1	A vector, factor, or list to apply the format1 to.
x2	A second vector, factor, or list to which format2 will be applied.
format1	A format to be applied to the first input.
format2	A format to be applied to the second input.
sep	A separator to use between the two formatted values. Default is a single blank space (" ").
width	The desired character width of the formatted vector. Default value is NULL, meaning the vector will be variable width.
justify	Whether to justify the return vector. Valid values are 'left', 'right', 'center', 'centre', or 'none'.

Details

The fapply2 function works nearly the same as fapply. The difference is it has parameters for two vectors and formats instead of one. The output of the function is a single vector. The function essentially calls fapply on each vector and pastes them together afterwards.

There is an additional sep parameter so you can define a separator between the two formatted values. The width and justify parameters will apply to the single vector result. The function will also pick up format attributes on the supplied vectors.

The fapply2 function accepts any of the format types that fapply accepts. See fapply for additional information on the types of formats that can be applied.

Parameters may also be passed as attributes on the vector. See the fattr function for additional information on setting formatting attributes.

Value

A vector of formatted values.

See Also

fapply to format a single input, fcat to create a format catalog, value to define a format, fattr to easily set the formatting attributes of a vector, and flist to define a formatting list. Also see fdata to apply formats to an entire data frame, and FormattingStrings for how to define a formatting string.

Examples

```
# Create sample data
dt <- c(2.1, 5, 6, 9, 2, 7, 3)

# Calculate mean and standard deviation
v1 <- mean(dt)
v2 <- sd(dt)

# Apply formats and combine
fapply2(v1, v2, "%.1f", "(%.2f)")
# [1] "4.9 (2.66)"
```

---

fattr *Set formatting attributes*

Description

Assign formatting attributes to a vector.

Usage

```
fattr(
  x,
  format = NULL,
  width = NULL,
  justify = NULL,
  label = NULL,
  description = NULL,
  keep = TRUE
)
```

Arguments

x	The vector or data frame column to assign attributes to.
format	The format to assign to the format attribute. The format can be a formatting string, a named vector decode, a vectorized formatting function, or a formatting list.
width	The desired width of the formatted output.
justify	Justification of the output vector. Valid values are 'none', 'left', 'right', 'center', or 'centre'.

label	A label string to assign to the vector. This parameter was added for convenience, as the label is frequently assigned at the same time the formatting attributes are assigned.
description	A description string to assign to the vector. This parameter was added for convenience, as the description is frequently assigned at the same time the formatting attributes are assigned.
keep	Whether to keep any existing formatting attributes and transfer to the new vector. Default value is TRUE.

Details

The fattr function is a convenience function for assigning formatting attributes to a vector. The function accepts parameters for format, width, and justify. Any formatting attributes assigned can be applied using fapply or fdata.

Value

The vector with formatting attributes assigned.

See Also

fdata to apply formats to a data frame, fapply to apply formats to a vector. See FormattingStrings for documentation on formatting strings.

Examples

```
# Create vector
a <- c(1.3243, 5.9783, 2.3848)

# Assign format attributes
a <- fattr(a, format = "%.1f", width = 10, justify = "center")

# Apply format attributes
fapply(a)
# [1] "   1.3    " "   6.0    " "   2.4    "
```

---

fattr<- *Set formatting attributes*

Description

Assign formatting attributes to a vector

Usage

```
fattr(x) <- value
```

Arguments

x	The vector or data frame column to assign attributes to.
value	A named vector of attribute values.

Details

The fattr function is a convenience function for assigning formatting attributes to a vector. The function accepts a named list of formatting attributes. Valid names are 'format', 'width', 'justify', 'label' and 'description'. See fattr for additional details.

See Also

fdata to apply formats to a data frame, fapply to apply formats to a vector.

Examples

```
# Create vector
a <- c(1.3243, 5.9783, 2.3848)

# Assign format attributes
fattr(a) <- list(format = "%.1f")

# Apply format attributes
fapply(a)
# [1] "1.3" "6.0" "2.4"
```

---

fcat *Create a format catalog*

Description

A format catalog is a collection of formats. A format collection allows you to manage and store formats as a unit. The fcat function defines the format catalog.

Usage

```
fcat(..., log = TRUE)
```

Arguments

...	A set of formats. Pass the formats as a name/value pair. Multiple name/value pairs are separated by a comma.
log	Whether to log the creation of the format catalog. Default is TRUE. This parameter is used internally.

Details

A format catalog is an S3 object of class "fcat". The purpose of the catalog is to combine related formats, and allow you to manipulate all of them as a single object. The format catalog can be saved to/from a file using the `write.fcat` and `read.fcat` functions. A format catalog can also be converted to/from a data frame using the `as.fcat.data.frame` and `as.data.frame.fcat` functions. Formats are accessed in the catalog using list syntax.

A format catalog can be used to assign formats to a data frame or tibble using the `formats` function. Formats may be applied using the `fdata` and `fapply` functions.

A format catalog may contain any type of format except a formatting list. Allowed formats include a formatting string, a named vector lookup, a user-defined format, and a vectorized formatting function. A formatting list can be converted to a format catalog and saved independently. See the `flist` function for more information on formatting lists.

Value

The format catalog object.

See Also

`formats` function for assigning formats to a data frame, and the `fdata` and `fapply` functions for applying formats.

Other fcat: `as.data.frame.fcat()`, `as.fcat()`, `as.fcat.data.frame()`, `as.fcat.fmt_lst()`, `as.fcat.list()`, `is.fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

Examples

```
# Create format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d%b%Y")

# Use formats in the catalog
fapply(2, c1$num_fmt)
# [1] "2.0"

fapply(c("A", "B", "C", "B"), c1$label_fmt)
# [1] "Label A" "Label B" "Other"   "Label B"

fapply(Sys.Date(), c1$date_fmt)
# [1] "06Jan2024"
```

fdata *Format a data frame or tibble*

Description

The fdata function applies formatting attributes to the entire data frame.

Usage

```
fdata(x, ...)
```

Arguments

x	A data frame or tibble to be formatted.
...	Any follow-on parameters to the format function.

Details

If formats are assigned to the "format" attributes of the data frame columns, the fdata function will apply those formats to the specified columns, and return a new, formatted data frame. Formats can be specified as formatting strings, named vectors, user-defined formats, or vectorized formatting functions. The fdata function will apply the format to the associated column data using the fapply function. A format can also be specified as a formatting list of the previous four types. See the fapply function for additional information.

After formatting each column, the fdata function will call the base R format function on the data frame. Any follow on parameters will be sent to the format function.

The fdata function will also apply any width or justify attributes assigned to the data frame columns. These attributes can be controlled at the column level. Using attributes to assign formatting and fdata to apply those attributes gives you a great deal of control over how your data is presented.

Value

A new, formatted data frame or tibble with the formats applied.

See Also

fcat to create a format catalog, fapply to apply a format to a vector, value to define a format object, fattr to assign formatting specifications to a single column/vector, and the formats, widths, and justification functions to get or set formatting for an entire data frame. Also see FormattingStrings for documentation on formatting strings.

Examples

```
# Construct data frame from state vectors
df <- data.frame(state = state.abb, area = state.area)[1:10, ]

# Calculate percentages
df$pct <- df$area / sum(state.area) * 100

# Before formatting
df
#    state   area         pct
# 1     AL  51609  1.42629378
# 2     AK 589757 16.29883824
# 3     AZ 113909  3.14804973
# 4     AR  53104  1.46761040
# 5     CA 158693  4.38572418
# 6     CO 104247  2.88102556
# 7     CT   5009  0.13843139
# 8     DE   2057  0.05684835
# 9     FL  58560  1.61839532
# 10    GA  58876  1.62712846

# Create state name lookup list
name_lookup <- state.name
names(name_lookup) <- state.abb

# Assign formats
formats(df) <- list(state = name_lookup,
                    area  = function(x) format(x, big.mark = ","),
                    pct   = "%.1f%%")

# Apply formats
fdata(df)
#          state    area   pct
# 1      Alabama  51,609  1.4%
# 2       Alaska 589,757 16.3%
# 3      Arizona 113,909  3.1%
# 4     Arkansas  53,104  1.5%
# 5   California 158,693  4.4%
# 6     Colorado 104,247  2.9%
# 7  Connecticut   5,009  0.1%
# 8     Delaware   2,057  0.1%
# 9      Florida  58,560  1.6%
# 10     Georgia  58,876  1.6%
```

`flist` *Create a formatting list*

Description

A formatting list contains more than one formatting object.

Usage

```
flist(..., type = "column", lookup = NULL, simplify = TRUE)
```

Arguments

...	A set of formatting objects.
type	The type of formatting list. Valid values are 'row' or 'column'. The default value is 'column'.
lookup	A lookup vector. Used for looking up the format from the formatting list. This parameter is only used for 'row' type formatting lists.
simplify	Whether to simplify the results to a vector. Valid values are TRUE or FALSE. Default is TRUE. If the value is set to FALSE, the return type will be a list.

Details

To apply more than one formatting object to a vector, use a formatting list. There are two types of formatting list: column and row. The column type formatting lists applies all formats to all values in the vector. The row type formatting list can apply a different format to each value in the vector.

Further, there are two styles of row type list: ordered and lookup. The ordered style applies each format in the list to the vector values in the order specified. The ordered style will recycle the formats as needed. The lookup style formatting list uses a lookup to determine which format from the list to apply to a particular value of the vector. The lookup column values should correspond to names on the formatting list.

Examples of column type and row type formatting lists are given below.

Value

A vector or list of formatted values. The type of return value can be controlled with the `simplify` parameter. The default return type is a vector.

See Also

`fapply` for information on how formats are applied to a vector, `value` for how to create a user-defined format, and `as.flist` to convert an existing list of formats to a formatting list. Also see FormattingStrings for details on how to use formatting strings.

Other flist: `as.data.frame.fmt_lst()`, `as.flist()`, `as.flist.data.frame()`, `as.flist.fcat()`, `as.flist.list()`, `as.flist.tbl_df()`, `is.flist()`, `print.fmt_lst()`, `read.flist()`, `write.flist()`

Examples

```
## Example 1: Formatting List - Column Type ##
# Set up data
v1 <- c(Sys.Date(), Sys.Date() + 30, Sys.Date() + 60)

# Create formatting list
fl1 <- flist("%B", "The month is: %s")

# Apply formatting list to vector
```

```
fapply(v1, fl1)
# [1] "The month is: October"  "The month is: November" "The month is: December"

## Example 2: Formatting List - Row Type ordered ##
# Set up data
# Notice each row has a different data type
l1 <- list("A", 1.263, as.Date("2020-07-21"),
          "B", 5.8732, as.Date("2020-10-17"))

# These formats will be recycled in the order specified
fl2 <- flist(type = "row",
             c(A = "Label A", B = "Label B"),
             "%.1f",
             "%d%b%Y")

fapply(l1, fl2)
# [1] "Label A"   "1.3"       "21Jul2020" "Label B"   "5.9"       "17Oct2020"


## Example 3: Formatting List - Row Type with lookup ##

#' # Create formatting list
fl3 <- flist(type = "row",
             DEC1 = "%.1f",
             DEC2 = "%.2f",
             PCT1 = "%.1f%%")

# Set up data
df <- data.frame(CODE = c("DEC1", "DEC2", "PCT1", "DEC2", "PCT1"),
                 VAL = c(41.258, 62.948, 12.125, 65.294, 15.825))

# Assign lookup
fl3$lookup <- df$CODE

# Apply Formatting List
fapply(df$VAL, fl3)
# [1] "41.3"  "62.95" "12.1%" "65.29" "15.8%"

## Example 4: Formatting List - Values with Units ##

#' # Create formatting list
fl4 <- flist(type = "row",
             BASO = "%.2f x10(9)/L",
             EOS  = "%.2f x10(9)/L",
             HCT = "%.1f%%",
             HGB = "%.1f g/dL")

# Set up data
df <- data.frame(CODE = c("BASO", "EOS", "HCT", "HGB"),
                 VAL = c(0.02384, 0.14683, 40.68374, 15.6345))

# Assign lookup
fl4$lookup <- df$CODE
```

```
# Apply Formatting List
df$VALC <- fapply(df$VAL, fl4)

# View results
df
#   CODE      VAL          VALC
# 1 BASO  0.02384 0.02 x10(9)/L
# 2  EOS  0.14683 0.15 x10(9)/L
# 3  HCT 40.68374         40.7%
# 4  HGB 15.63450     15.6 g/dL
```

`fmt_cnt_pct` *Formatted count and percent*

Description

A function to calculate and format a count and percent.

Usage

```
fmt_cnt_pct(x, denom = NULL, format = "%5.1f", na = NULL, zero = NULL)
```

Arguments

`x`	The input data vector or data frame column.
`denom`	The denominator to use for the percentage. By default, the parameter is NULL, meaning the function will use the number of non-missing values of the data vector as the denominator. Otherwise, supply the denominator as a numeric value.
`format`	A formatting string suitable for input into the `sprintf` function. By default, this format is defined as "%5.1f", which displays the value with one decimal place.
`na`	The value to return for any NA value encountered in the input vector. Usually this parameter is passed as a string, such as "-", yet any value can be supplied.
`zero`	The value to return for any zero values encountered in the input vector. Usually this value is supplied as string such as "0 (-)".

Details

This function calculates a percent and appends to the provided count. The input vector is assumed to contain the counts. This function will not perform counting. It will calculate percentages and append to the given counts.

The result is then formatted using `sprintf`. By default, the number of non-missing values in the input data vector is used as the denominator. Alternatively, you may supply the denominator using the **denom** parameter. You may also control the percent format using the **format** parameter. The function will return any NA values in the input data unaltered.

If the calculated percentage is between 0% and 1%, the function will display "(< 1.0%)" as the percentage value. Zero values will be displayed as "(0.0%)"

Value

A character vector of formatted counts and percents.

See Also

Other helpers: fmt_mean_sd(), fmt_mean_stderr(), fmt_median(), fmt_n(), fmt_quantile_range(), fmt_range()

Examples

```
v1 <- c(4, 3, 8, 6, 9, 5, NA, 0, 7, 4, 3, 7)

# Format count and percent
fmt_cnt_pct(v1)

# Output
# [1] "4 ( 36.4%)" "3 ( 27.3%)" "8 ( 72.7%)" "6 ( 54.5%)"
# [5] "9 ( 81.8%)" "5 ( 45.5%)" NA           "0 (  0.0%)"
# [9] "7 ( 63.6%)" "4 ( 36.4%)" "3 ( 27.3%)" "7 ( 63.6%)"

# Custom values for NA and zero
fmt_cnt_pct(v1, na = "N/A", zero = "0 (-)")

# Custom NA and zero output
# [1] "4 ( 36.4%)" "3 ( 27.3%)" "8 ( 72.7%)" "6 ( 54.5%)"
# [5] "9 ( 81.8%)" "5 ( 45.5%)" "N/A"        "0 (-)"
# [9] "7 ( 63.6%)" "4 ( 36.4%)" "3 ( 27.3%)" "7 ( 63.6%)"
```

---

fmt_mean_sd *Formatted mean and standard deviation*

Description

A function to calculate and format a mean and standard deviation.

Usage

```
fmt_mean_sd(x, format = "%.1f", sd_format = NULL)
```

Arguments

x	The input data vector or data frame column.
format	A formatting string suitable for input into the sprintf function. By default, this format is defined as "%.1f", which displays the mean and standard deviation with one decimal place.
sd_format	An optional format for the standard deviation. If this parameter is not supplied, the standard deviation will be formatted the same as the mean, according to the 'format' parameter.

Details

This function calculates a mean and standard deviation, and formats using sprintf. You may control the format using the **format** parameter. Function will ignore NA values in the input data. Results are returned as a character vector.

Value

The formatted mean and standard deviation.

See Also

Other helpers: fmt_cnt_pct(), fmt_mean_stderr(), fmt_median(), fmt_n(), fmt_quantile_range(), fmt_range()

Examples

```
v1 <- c(4.3, 3.7, 8.7, 6.1, 9.2, 5.6, NA, 0.7, 7.8, 4.9)

# Format mean and standard deviation
fmt_mean_sd(v1)
# "5.7 (2.7)"
```

---

fmt_mean_stderr *Formatted mean and standard error*

Description

A function to calculate and format a mean and standard error.

Usage

```
fmt_mean_stderr(x, format = "%.1f", stderr_format = NULL)
```

Arguments

x	The input data vector or data frame column.
format	A formatting string suitable for input into the sprintf function. By default, this format is defined as "%.1f", which displays the mean and standard error with one decimal place.
stderr_format	An optional format for the standard error. If this parameter is not supplied, the standard error will be formatted the same as the mean, according to the 'format' parameter.

Details

This function calculates a mean and standard error, and formats using sprintf. You may control the format using the **format** parameter. Function will ignore NA values in the input data. Results are returned as a character vector.

Value

The formatted mean and standard error.

See Also

Other helpers: fmt_cnt_pct(), fmt_mean_sd(), fmt_median(), fmt_n(), fmt_quantile_range(), fmt_range()

Examples

```
v1 <- c(4.3, 3.7, 8.7, 6.1, 9.2, 5.6, NA, 0.7, 7.8, 4.9)

# Format mean and standard error
fmt_mean_stderr(v1)
# "5.7 (0.9)"
```

---

fmt_median *Formatted Median*

Description

A function to calculate and format a median.

Usage

```
fmt_median(x, format = "%.1f")
```

Arguments

x	The input data vector or data frame column.
format	A formatting string suitable for input into the sprintf function. By default, this format is defined as "%.1f", which displays the value with one decimal place.

Details

This function calculates a median using the stats package median function, and then formats the output using sprintf. You may control the format using the **format** parameter. Function will ignore any NA values in the input data. Results are returned as a character vector.

Value

The formatted median value.

See Also

Other helpers: fmt_cnt_pct(), fmt_mean_sd(), fmt_mean_stderr(), fmt_n(), fmt_quantile_range(), fmt_range()

Examples

```
v1 <- c(4.3, 3.7, 8.7, 6.1, 9.2, 5.6, NA, 0.7, 7.8, 4.9)

# Format median
fmt_median(v1)
# "5.6"
```

`fmt_n` *Formatted Count*

Description

A function to calculate and format a numeric count.

Usage

```
fmt_n(x)
```

Arguments

x	The input data vector or data frame column.

Details

This function calculates a count using the Base R `sum` function. NA values are not counted. Results are returned as a character vector.

Value

The formatted count value.

See Also

Other helpers: `fmt_cnt_pct()`, `fmt_mean_sd()`, `fmt_mean_stderr()`, `fmt_median()`, `fmt_quantile_range()`, `fmt_range()`

Examples

```
# Create example vector
v1 <- c(4.3, 3.7, 8.7, 6.1, 9.2, 5.6, NA, 0.7, 7.8, 4.9)

# Format n
fmt_n(v1)
# "9"
```

`fmt_quantile_range` *Formatted Quantile Range*

Description

A function to calculate and format a quantile range.

Usage

```
fmt_quantile_range(
  x,
  format = "%.1f",
  sep = "-",
  lower = 0.25,
  upper = 0.75,
  type = 7
)
```

Arguments

`x`	The input data vector or data frame column.
`format`	A formatting string suitable for input into the `sprintf` function. By default, this format is defined as "%.1f", which displays the value with one decimal place.
`sep`	The character to use as a separator between the two quantiles.
`lower`	The lower quantile range. Default is .25.
`upper`	The upper quantile range. Default is .75.
`type`	An integer between 1 and 9 selecting one of the nine quantile algorithms. The default is 7, which is the standard R default. If you are trying to match SAS results, use type 2. See the `quantile` function documentation for further details.

Details

This function calculates a quantile range using the stats package `quantile` function, and then formats the output using `sprintf`. You may control the format using the **format** parameter. Function will ignore any NA values in the input data. Results are returned as a character vector.

By default, the function calculates the 1st and 3rd quantiles at .25 and .75. The upper and lower quantile ranges may be changed with the `upper` and `lower` parameters.

Value

The formatted quantile range.

See Also

Other helpers: `fmt_cnt_pct()`, `fmt_mean_sd()`, `fmt_mean_stderr()`, `fmt_median()`, `fmt_n()`, `fmt_range()`

Examples

```
# Create example vector
v1 <- c(4.3, 3.7, 8.7, 6.1, 9.2, 5.6, NA, 0.7, 7.8, 4.9)

# Format Quantiles
fmt_quantile_range(v1)
# "4.3 - 7.8"
```

`fmt_range` *Formatted Range*

Description

A function to calculate and format a numeric range.

Usage

```
fmt_range(x, format = "%s", sep = "-")
```

Arguments

`x`	The input data vector or data frame column.
`format`	A formatting string suitable for input into the `sprintf` function. By default, this format is defined as "%s", which simply converts the value to a string with no specific formatting.
`sep`	The token used to separate the minimum and maximum range values. Default value is a hyphen ("-").

Details

This function calculates a range using the Base R `range` function, and then formats the output using `sprintf`. You may control the format using the **format** parameter. Any NA values in the input data are ignored. Results are returned as a character vector.

Value

The formatted range values.

See Also

Other helpers: `fmt_cnt_pct()`, `fmt_mean_sd()`, `fmt_mean_stderr()`, `fmt_median()`, `fmt_n()`, `fmt_quantile_range()`

Examples

```
# Create example vector
v1 <- c(4.3, 3.7, 8.7, 6.1, 9.2, 5.6, NA, 0.7, 7.8, 4.9)

# Format range
fmt_range(v1)
# "0.7 - 9.2"
```

---

formats *Get or set formats for a data frame*

Description

The `formats` function extracts all assigned formats from a data frame, and returns them in a named list. The function also assigns formats from a named list.

Usage

```
formats(x)

formats(x) <- value
```

Arguments

`x`	A data frame or tibble
`value`	A named list of formats

Details

If formats are assigned to the "format" attributes of the data frame columns, the `formats` function will extract those formats. The function will return the formats in a named list, where the names correspond to the name of the column that the format was assigned to. If a column does not have a format attribute assigned, that column will not be included in the list.

When used on the receiving side of an assignment, the function will assign formats to a data frame. The formats should be in a named list, where each name corresponds to the data frame column to assign the format to.

The `formats` function can also accept a format catalog as input. This feature allows you to save formats in metadata, load them into a format catalog, and quickly assign them to a data frame or tibble. See the `fcat` function for additional information.

Finally, if you wish to clear out format attributes, assign a NULL value to the `formats` function.

Value

A named list of formats.

See Also

fdata to display formatted data, value to create user-defined formats, fapply to apply formats to a vector, and fcat to create a format catalog. Also see FormattingStrings for documentation on formatting strings.

Examples

```
# Take subset of data
df1 <- mtcars[1:5, c("mpg", "cyl") ]

# Print current state
print(df1)
#                    mpg cyl
# Mazda RX4         21.0   6
# Mazda RX4 Wag     21.0   6
# Datsun 710        22.8   4
# Hornet 4 Drive    21.4   6
# Hornet Sportabout 18.7   8

# Assign formats
attr(df1$mpg, "format") <- value(condition(x >= 20, "High"),
                                 condition(x < 20, "Low"))
attr(df1$cyl, "format") <- function(x) format(x, nsmall = 1)

# Display formatted data
fdata(df1)
#                    mpg cyl
# Mazda RX4         High 6.0
# Mazda RX4 Wag     High 6.0
# Datsun 710        High 4.0
# Hornet 4 Drive    High 6.0
# Hornet Sportabout  Low 8.0

# Extract format list
lst <- formats(df1)

# Alter format list and reassign
lst$mpg <- value(condition(x >= 22, "High"),
                 condition(x < 22, "Low"))
lst$cyl <- function(x) format(x, nsmall = 2)
formats(df1) <-  lst

# Display formatted data
fdata(df1)
#                    mpg  cyl
# Mazda RX4          Low 6.00
# Mazda RX4 Wag      Low 6.00
# Datsun 710        High 4.00
# Hornet 4 Drive     Low 6.00
# Hornet Sportabout  Low 8.00

# Clear formats
```

```
formats(df1) <- NULL

# Confirm formats are cleared
formats(df1)
# list()
```

FormattingStrings *Formatting Strings*

Description

Formatting codes for formatting strings follow the conventions of the base R `strptime` and `sprintf` functions. See below for further details.

Details

The **fmtr** packages accepts single strings as formatting specifications. These formatting strings are interpreted differently depending on the data type of the vector being formatted. For date and datetime vectors, the string will be interpreted as an input to the base R `strptime` function. For all other types of vectors, the formatting string will be interpreted as an input to the `sprintf` function.

The formatting codes for these functions are simple to use. For example, the code `fapply(as.Date("1970-01-01"), "%B %d, %Y")` will produce the output `"January 01, 1970"`. The code `fapply(1.2345, "%.1f")` will produce the output `"1.2"`.

Date Formatting

Below are some commonly used formatting codes for dates:

- %d = day as a number
- %a = abbreviated weekday
- %A = unabbreviated weekday
- %m = month
- %b = abbreviated month
- %B = unabbreviated month
- %y = 2-digit year
- %Y = 4-digit year
- %H = hour (24 hour clock)
- %I = hour (12 hour clock)
- %M = minute
- %S = second
- %p = AM/PM indicator
- %q = Quarter as integer
- %Q = Quarter as "Q?"

See the `strptime` function for additional codes and examples of formatting dates and times.

"date" Format

The "dateW." format is a date-display format that replicates the behavior of the SAS "DATEw." family of formats. The "dateW." format converts either numeric date values, R `Date` objects, or `POSIXt` date-time objects into standard fixed-width character representations.

Smaller widths show abbreviated forms (e.g., "JAN70" for "date5"), while larger widths show full day/month/year values (e.g., "01JAN1970" for "date9" or "01-JAN-1970" for "date11"). For `POSIXt` values, only the date portion is used.

Numeric inputs follow R conventions and are interpreted as days since 1970-01-01. The default width is 7. A trailing dot (".") is optional.

Output always occupies the specified width. If the date value is shorter than the specified width, it is left-padded with spaces.

"time" Format

The "TIMEw.d" format is a time-display format that replicates the behavior of the SAS "TIMEw.d" family of formats. It converts time values into fixed-width character representations of the form `h:mm:ss`, optionally including fractional seconds.

The format accepts numeric values (interpreted as seconds), `POSIXt` objects, `hms` objects or `difftime` objects. For `POSIXt` inputs, only the time-of-day component is used. Numeric `hms` and `difftime` inputs may be negative or exceed 24 hours; this is not applicable to `POSIXt` values.

The width `W` controls the total output width, while `D` specifies the number of decimal places for seconds. If omitted, `W` defaults to 8 and `D` defaults to 0. A trailing dot (".") is optional.

Output always occupies the specified width and is left-padded with spaces if needed. Missing values return `NA`. This format resolves known rounding differences between R and SAS and supports up to 12 digits of fractional seconds, exceeding R's default precision.

"datetime" Format

The "DATETIMEw.d" format is a datetime-display format that replicates the behavior of the SAS "DATETIMEw.d" family of formats. It converts date-time values into fixed-width character representations of the form `ddMMMyy:hh:mm:ss` or `ddMMMyyyy:hh:mm:ss`, optionally including fractional seconds.

The format accepts numeric values and `POSIXt` objects. Numeric inputs follow R conventions and are interpreted as seconds since `1970-01-01 00:00:00` in the local time zone, by default. Negative numeric values are allowed and represent datetimes before 1970. For `POSIXt` inputs, the full date and time components are used.

The width `w` controls the total output width, while `d` specifies the number of decimal places for seconds. If omitted, `W` defaults to 16 and `D` defaults to 0. A trailing dot (".") is optional.

Output always occupies the specified width and is left-padded with spaces if needed. Missing values return `NA`. This format supports display of up to 23 digits after the decimal point. Precision is maintained through 15 digits; beyond that, additional decimal places are padded with zeros.

Numeric Formatting

Below are some commonly used formatting codes for other data types:

- %s = string
- %d = integer
- %f = floating point number

See the sprintf function for additional codes and examples of formatting other data types.

"best" Numeric Format

The "best" format is a special numeric format that replicates the "best" style of formatting from SAS. The "best" format automatically determines the best way to format a number within a specified with.

For example, if you have a number like 123.45678, and format it with "best6", the result will be "123.46". Note that the decimal point counts as one of the digits. Also note that the rightmost digit will be rounded according to SAS rounding rules.

The same number above formatted with "best8" will be "123.4567". The default width is 12. That means, if you simply send the format "best" with no width, it will be interpreted as "best12".

See Also

fapply for formatting vectors, and fdata for formatting data frames.

Examples

```
# Examples of R-Style date and time formats
t <- Sys.time()
fapply(t, "%d/%m/%Y")              # Day/Month/Year
fapply(t, "%d%b%Y")                # Day abbreviated month year
fapply(t, "%y-%m")                 # Two digit year - month
fapply(t, "%A, %B %d")             # Weekday, unabbreviated month and date
fapply(t, "%Y-%Q")                 # Year and Quarter
fapply(t, "%Y-%m%-%d %H:%M:%S %p") # Common timestamp format

# Examples of SAS-Style date and time formats
d <- Sys.Date()
fapply(d, "date5")                 # Month Year (mmmyy)
fapply(d, "date7")                 # Day Month Year (ddmmmyy)
fapply(d, "date9")                 # Day Month Year (ddmmmyyyy)
fapply(d, "date11")                # Day Month Year (dd-mmm-yyyy)
t <- Sys.time()
fapply(t, "time5")                 # Hour and Minute (HH:MM)
fapply(t, "time8")                 # Hour, Minute, Second (HH:MM:SS)
fapply(t, "datetime14")            # Datetime format (ddmmyy:HH:MM)
fapply(t, "datetime19")            # Datetime format (ddmmyyyy:HH:MM:SS)

# Examples for formatting numbers
a <- 1234.56789
fapply(a, "%f")                    # Floating point number
fapply(a, "%.1f")                  # One decimal place
fapply(a, "%8.1f")                 # Fixed width
fapply(a, "%-8.1f")                # Fixed width left justified
fapply(a, "%08.1f")                # Zero padded
```

```
fapply(a, "%+.1f")                # Forced sign
fapply(-a, "%+.1f")               # Negative
fapply(a, "%.1f%%")               # Percentage
fapply(a, "$%.2f")                # Currency
fapply(a, "The number is %f.")    # Interpolation

# "best" formatting
fapply(a, "best6")                # Total width of 6 (####.#)
fapply(a, "best8")                # Total width of 8 (####.###)
```

`is.fcat` *Class test for a format catalog*

Description

This function tests whether an object is a format catalog. The format catalog has a class of "fcat".

Usage

```
is.fcat(x)
```

Arguments

x	The object to test.

Value

TRUE or FALSE, depending on whether or not the object is a format catalog.

See Also

Other fcat: `as.data.frame.fcat()`, `as.fcat()`, `as.fcat.data.frame()`, `as.fcat.fmt_lst()`, `as.fcat.list()`, `fcat()`, `print.fcat()`, `read.fcat()`, `write.fcat()`

Examples

```
# Create format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d%b%Y")

# Test for "fcat" class
is.fcat(c1)
# [1] TRUE

is.fcat(Sys.Date())
# [1] FALSE
```

is.flist *Is object a formatting list*

Description

Determines if object is a formatting list of class 'fmt_lst'.

Usage

```
is.flist(x)
```

Arguments

x	Object to test.

Value

TRUE or FALSE, depending on class of object.

See Also

Other flist: as.data.frame.fmt_lst(), as.flist(), as.flist.data.frame(), as.flist.fcat(), as.flist.list(), as.flist.tbl_df(), flist(), print.fmt_lst(), read.flist(), write.flist()

Examples

```
# Create flist
flst <- flist("%d%b%Y", "%.1f")
is.flist(flst)
is.flist("A")
```

is.format *Determine whether an object is a user-defined format*

Description

The is.format function can be used to determine if an object is a user-defined format of class "fmt".

Usage

```
is.format(x)
```

Arguments

x	A user-defined format of class "fmt".

Details

The `is.format` function returns TRUE if the object passed is a user-defined format. User-defined formats are defined using the `value` function. See the `value` function help for further details.

Value

A logical value or TRUE or FALSE.

See Also

`value` to define a format, `condition` to define the conditions for a format, and `fapply` to apply the format to a vector.

Other fmt: `as.data.frame.fmt()`, `as.fmt()`, `as.fmt.data.frame()`, `condition()`, `labels.fmt()`, `print.fmt()`, `value()`

Examples

```
# Define format
fmt1 <- value(condition(x == "A", "Label A"),
              condition(x == "B", "Label B"),
              condition(TRUE, "Other"))

# Check for format
is.format(fmt1)
# [1] TRUE

is.format("A")
# [1] FALSE
```

---

justification *Get or set justification for data frame columns*

Description

The `justification` function extracts all assigned justify attributes from a data frame, and returns them in a named list. The function also assigns justify attributes from a named list.

Usage

```
justification(x)

justification(x) <- value
```

Arguments

`x`	A data frame or tibble
`value`	A named list of justification values.

Details

If justification values are assigned to the "justify" attributes of the data frame columns, the `justification` function will extract those values. The function will return the justification values in a named list, where the names correspond to the name of the column that the justification was assigned to. If a column does not have a justify attribute assigned, that column will not be included in the list.

When used on the receiving side of an assignment, the function will assign justification to a data frame. The justification values should be in a named list, where each name corresponds to the name of the data frame column to assign values to.

Finally, if you wish to clear out the justification attributes, assign a NULL value to the `justification` function.

Value

A named list of justification values.

See Also

`fdata` to display formatted data, `value` to create user-defined formats, and `fapply` to apply formatting to a vector.

Examples

```
# Take subset of data
df1 <- mtcars[1:5, c("mpg", "cyl") ]

# Print current state
print(df1)
#                    mpg cyl
# Mazda RX4         21.0   6
# Mazda RX4 Wag     21.0   6
# Datsun 710        22.8   4
# Hornet 4 Drive    21.4   6
# Hornet Sportabout 18.7   8

# Assign justification
justification(df1) <- list(mpg = "left", cyl = "right")
widths(df1) <- list(mpg = 12, cyl = 10)

fdata(df1)
#                            mpg        cyl
# Mazda RX4         21                    6
# Mazda RX4 Wag     21                    6
# Datsun 710        22.8                  4
# Hornet 4 Drive    21.4                  6
# Hornet Sportabout 18.7                  8

# Display justification
justification(df1)
# $mpg
# [1] "left"
```

```
#
# $cyl
# [1] "right"

# Display widths
widths(df1)
# $mpg
# [1] 12
#
# $cyl
# [1] 10

# Clear justification
justification(df1) <- NULL

# Confirm justifications are cleared
justification(df1)
# list()
```

`labels.fmt` *Extract labels from a user-defined format*

Description

The `labels` function creates a vector of labels associated with a user-defined format.

Usage

```
## S3 method for class 'fmt'
labels(object, ...)
```

Arguments

`object`	A user-defined format of class "fmt".
`...`	Following arguments.

Details

The `condition` function creates a condition as part of a format definition. Each condition has a label as part of its definition. The `labels` function extracts the labels from the conditions and returns them as a vector. While the labels will typically be of type character, they can be of any data type. See the `condition` function help for further details.

Value

A vector of label values.

See Also

value to define a format, condition to define the conditions for a format, and fapply to apply the format to a vector.

Other fmt: as.data.frame.fmt(), as.fmt(), as.fmt.data.frame(), condition(), is.format(), print.fmt(), value()

Examples

```
# Define format
fmt1 <- value(condition(x == "A", "Label A"),
              condition(x == "B", "Label B"),
              condition(TRUE, "Other"))

# Extract labels
labels(fmt1)
# [1] "Label A" "Label B" "Other"
```

---

print.fcat *Print a format catalog*

Description

A class-specific instance of the print function for format catalogs. The function prints the format catalog in a tabular manner. Use verbose = TRUE to print the catalog as a list.

Usage

```
## S3 method for class 'fcat'
print(x, ..., verbose = FALSE)
```

Arguments

x	The format catalog to print.
...	Any follow-on parameters.
verbose	Whether or not to print the format catalog in verbose style. By default, the parameter is FALSE, meaning to print in tabular style.

Value

The object, invisibly.

See Also

Other fcat: as.data.frame.fcat(), as.fcat(), as.fcat.data.frame(), as.fcat.fmt_lst(), as.fcat.list(), fcat(), is.fcat(), read.fcat(), write.fcat()

Examples

```
#' # Create format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d%b%Y")

# Print the catalog
print(c1)
# # A format catalog: 3 formats
# - $num_fmt: type S, "%.1f"
# - $label_fmt: type U, 3 conditions
# - $date_fmt: type S, "%d%b%Y"
```

`print.fmt` *Print a format*

Description

Prints a format object. This function is a class-specific implementation of the the generic `print` method.

Usage

```
## S3 method for class 'fmt'
print(
  x,
  ...,
  name = deparse(substitute(x, env = environment())),
  verbose = FALSE
)
```

Arguments

`x`	An object of class "fmt".
`...`	Any follow-on parameters to the print function.
`name`	The name of the format to print. By default, the variable name that holds the format will be used.
`verbose`	Turn on or off verbose printing mode. Verbose mode will print object as a list. Otherwise, the object will be printed as a table.

See Also

Other fmt: `as.data.frame.fmt()`, `as.fmt()`, `as.fmt.data.frame()`, `condition()`, `is.format()`, `labels.fmt()`, `value()`

print.fmt_lst *Print a formatting list*

Description

Print a formatting list

Usage

```
## S3 method for class 'fmt_lst'
print(x, ..., verbose = FALSE)
```

Arguments

x	The formatting list to print
...	Follow-on parameters to the print function
verbose	Whether to print in summary or list-style.

See Also

Other flist: as.data.frame.fmt_lst(), as.flist(), as.flist.data.frame(), as.flist.fcat(), as.flist.list(), as.flist.tbl_df(), flist(), is.flist(), read.flist(), write.flist()

read.fcat *Read a format catalog from the file system*

Description

The read.fcat function reads a format catalog from the file system. The function accepts a path to the format catalog, reads the catalog, and returns it.

Usage

```
read.fcat(file_path)
```

Arguments

file_path	The path to the format catalog.

Value

The format catalog as an R object.

See Also

Other fcat: as.data.frame.fcat(), as.fcat(), as.fcat.data.frame(), as.fcat.fmt_lst(), as.fcat.list(), fcat(), is.fcat(), print.fcat(), write.fcat()

Examples

```
# Create format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
           date_fmt = "%d%b%Y")

# Get temp directory
tmp <- tempdir()

# Save catalog to file system
pth <- write.fcat(c1, dir_path = tmp)

# Read from file system
c2 <- read.fcat(pth)

# Use formats in the catalog
fapply(2, c1$num_fmt)
# [1] "2.0"

fapply(c("A", "B", "C", "B"), c1$label_fmt)
# [1] "Label A" "Label B" "Other"   "Label B"

fapply(Sys.Date(), c1$date_fmt)
# [1] "07Jan2024"
```

---

read.flist *Read a formatting list from the file system*

Description

The read.flist function reads a formatting list from the file system. The function accepts a path to the formatting list, reads the list, and returns it.

Usage

```
read.flist(file_path)
```

Arguments

file_path	The path to the formatting list.

Value

The formatting list as an R object.

See Also

Other flist: as.data.frame.fmt_lst(), as.flist(), as.flist.data.frame(), as.flist.fcat(), as.flist.list(), as.flist.tbl_df(), flist(), is.flist(), print.fmt_lst(), write.flist()

Examples

```
# Create formatting list
fl <- flist(f1 = "%5.1f",
            f2 = "%6.2f",
            type = "row")

# Get temp directory
tmp <- tempdir()

# Save formatting list to file system
pth <- write.flist(fl, dir_path = tmp)

# Read from file system
fr <- read.flist(pth)

# Create sample data
dat <- c(12.3844, 292.28432)

# Use formats in the catalog
fapply(dat, fr)
# [1] " 12.4"  "292.28"
```

---

value *Create a user-defined format*

Description

The value function creates a user-defined format.

Usage

```
value(..., log = TRUE, as.factor = FALSE)
```

Arguments

...	One or more condition functions.
log	Whether to log the creation of the format. Default is TRUE. This parameter is used internally.
as.factor	If TRUE, the fapply function will return the result as an ordered factor. Otherwise, the result will be returned as a vector. Default is FALSE.

Details

The `value` function creates a user defined format object, in a manner similar to a SAS® format. The `value` function accepts one or more `condition` arguments that define the format. The conditions map an R expression to a label. When applied, the format will return the label corresponding to the first true expression.

The format object is an S3 class of type "fmt". When the object is created, the **levels** attribute of the object will be set with a vector of values assigned to the **labels** property of the `condition` arguments. These labels may be accessed either from the `levels` function or the `labels` function. If no order has been assigned to the conditions, the labels will be returned in the order the conditions were passed to the `value` function. If an order has been assigned to the conditions, the labels will be returned in the order specified.

The format object may be applied to a vector using the `fapply` function. See `fapply` for further details.

Note that the label may also be a string format. That means a user-defined format can be used to apply string formats conditionally. This capability is useful when you want to conditionally format data values.

Value

The new format object.

See Also

`condition` to define a condition, `levels` or `labels.fmt` to access the labels, and `fapply` to apply the format to a vector.

Other fmt: `as.data.frame.fmt()`, `as.fmt()`, `as.fmt.data.frame()`, `condition()`, `is.format()`, `labels.fmt()`, `print.fmt()`

Examples

```
## Example 1: Character to Character Mapping ##
# Set up vector
v1 <- c("A", "B", "C", "B")

# Define format
fmt1 <- value(condition(x == "A", "Label A"),
              condition(x == "B", "Label B"),
              condition(TRUE, "Other"))

# Apply format to vector
fapply(v1, fmt1)
# [1] "Label A" "Label B" "Other"   "Label B"

## Example 2: Character to Integer Mapping ##
fmt2 <- value(condition(x == "A", 1),
              condition(x == "B", 2),
              condition(TRUE, 3))

# Apply format to vector
```

```
fapply(v1, fmt2)
# [1] 1 2 3 2

## Example 3: Categorization of Continuous Variable ##
# Set up vector
v2 <- c(1, 6, 11, 7)

# Define format
fmt3 <- value(condition(x < 5, "Low"),
              condition(x >= 5 & x < 10, "High"),
              condition(TRUE, "Out of range"))

# Apply format to vector
fapply(v2, fmt3)
# [1] "Low"          "High"         "Out of range" "High"

### Example 4: Conditional formatting
v3 <- c(10.398873, 12.98762, 0.5654, 11.588372)

fmt4 <- value(condition(x < 1, "< 1.0"),
              condition(TRUE, "%.2f"))

fapply(v3, fmt4)
# [1] "10.40" "12.99" "< 1.0" "11.59"
```

`widths` *Get or set column widths for a data frame*

Description

The `widths` function extracts all assigned widths from a data frame, and returns them in a named list. The function also assigns widths from a named list.

Usage

```
widths(x)

widths(x) <- value
```

Arguments

`x`	A data frame or tibble
`value`	A named list of widths. The widths must be positive integers greater than zero.

Details

If widths are assigned to the "width" attributes of the data frame columns, the `widths` function will extract those widths. The function will return the widths in a named list, where the names correspond to the name of the column that the width was assigned to. If a column does not have a width attribute assigned, that column will not be included in the list.

When used on the receiving side of an assignment, the function will assign widths to a data frame. The widths should be in a named list, where each name corresponds to the data frame column to assign the width to.

Finally, if you wish to clear out the width attributes, assign a NULL value to the `widths` function.

Value

A named list of widths. The widths must be positive integers greater than zero.

See Also

`fdata` to display formatted data, `value` to create user-defined formats, and `fapply` to apply formats to a vector.

Examples

```
# Take subset of data
df1 <- mtcars[1:5, c("mpg", "cyl") ]

# Print current state
print(df1)
#                   mpg cyl
# Mazda RX4         21.0   6
# Mazda RX4 Wag     21.0   6
# Datsun 710        22.8   4
# Hornet 4 Drive    21.4   6
# Hornet Sportabout 18.7   8

# Assign widths
widths(df1) <- list(mpg = 12, cyl = 10)

# Display formatted data
fdata(df1)
#                            mpg        cyl
# Mazda RX4                 21.0          6
# Mazda RX4 Wag             21.0          6
# Datsun 710                22.8          4
# Hornet 4 Drive            21.4          6
# Hornet Sportabout         18.7          8

# View assigned widths
widths(df1)
# $mpg
# [1] 12
#
```

```
# $cyl
# [1] 10

# Clear widths
widths(df1) <- NULL

# Confirm widths are cleared
widths(df1)
# list()
```

`write.fcat` *Write a format catalog to the file system*

Description

The `write.fcat` function writes a format catalog to the file system. By default, the catalog will be written to the current working directory, using the variable name as the file name. These defaults can be overridden using the appropriate parameters. The catalog will be saved with a file extension of ".fcat".

Usage

```
write.fcat(x, dir_path = getwd(), file_name = NULL)
```

Arguments

`x`	The format catalog to write.
`dir_path`	The directory path to write the catalog to. Default is the current working directory.
`file_name`	The name of the file to save the catalog as. Default is the name of the variable that contains the catalog. The ".fcat" file extension will be added automatically.

Value

The full path of the saved format catalog.

See Also

Other fcat: `as.data.frame.fcat()`, `as.fcat()`, `as.fcat.data.frame()`, `as.fcat.fmt_lst()`, `as.fcat.list()`, `fcat()`, `is.fcat()`, `print.fcat()`, `read.fcat()`

Examples

```
# Create format catalog
c1 <- fcat(num_fmt  = "%.1f",
           label_fmt = value(condition(x == "A", "Label A"),
                             condition(x == "B", "Label B"),
                             condition(TRUE, "Other")),
```

```
            date_fmt = "%d%b%Y")

# Get temp directory
tmp <- tempdir()

# Save catalog to file system
pth <- write.fcat(c1, dir_path = tmp)

# Read from file system
c2 <- read.fcat(pth)

# Use formats in the catalog
fapply(2, c1$num_fmt)
# [1] "2.0"

fapply(c("A", "B", "C", "B"), c1$label_fmt)
# [1] "Label A" "Label B" "Other"   "Label B"

fapply(Sys.Date(), c1$date_fmt)
# [1] "07Jan2024"
```

`write.flist` *Write a formatting list to the file system*

Description

The `write.flist` function writes a formatting list to the file system. By default, the formatting list will be written to the current working directory, using the variable name as the file name. These defaults can be overridden using the appropriate parameters. The catalog will be saved with a file extension of ".flist".

Usage

```
write.flist(x, dir_path = getwd(), file_name = NULL)
```

Arguments

`x`	The formatting list to write.
`dir_path`	The directory path to write the catalog to. Default is the current working directory.
`file_name`	The name of the file to save the catalog as. Default is the name of the variable that contains the formatting list. The ".flist" file extension will be added automatically.

Value

The full path of the saved formatting list.

See Also

Other flist: as.data.frame.fmt_lst(), as.flist(), as.flist.data.frame(), as.flist.fcat(), as.flist.list(), as.flist.tbl_df(), flist(), is.flist(), print.fmt_lst(), read.flist()

Examples

```
# Create formatting list
fl <- flist(f1 = "%5.1f",
            f2 = "%6.2f",
            type = "row")

# Get temp directory
tmp <- tempdir()

# Save formatting list to file system
pth <- write.flist(fl, dir_path = tmp)

# Read from file system
fr <- read.flist(pth)

# Create sample data
dat <- c(12.3844, 292.28432)

# Use formats in the catalog
fapply(dat, fr)
# [1] " 12.4"  "292.28"
```

Index