Package ‘glue’

November 7, 2021

Title Interpreted String Literals
Version 1.5.0
Description An implementation of interpreted string literals, inspired by
Python's Literal String Interpolation
<https://www.python.org/dev/peps/pep-0498/> and Docstrings
<https://www.python.org/dev/peps/pep-0257/> and Julia's Triple-Quoted
String Literals
#Triple-Quoted-String-Literals-1>.
License MIT + file LICENSE
BugReports https://github.com/tidyverse/glue/issues
Depends R (>= 3.2)
Imports methods
Suggests covr, crayon, DBI, dplyr, forcats, ggplot2, knitr, magrittr,
microbenchmark, R.utils, rmarkdown, rprintf, RSQLite, stringr,
testthat, vctrs (>= 0.3.0), waldo, withr
VignetteBuilder knitr
ByteCompile true
Config/Needs/website tidyverse/tidytemplate
Config/testthat/edition 3
Encoding UTF-8
RoxygenNote 7.1.2
NeedsCompilation yes
Author Jim Hester [aut, cre] (<https://orcid.org/0000-0002-2739-7082>),
RStudio [cph, fnd]
Maintainer Jim Hester <jim.hesters@rstudio.com>
Repository CRAN
Date/Publication 2021-11-07 00:50:07 UTC
R topics documented:

as_glue ................................. 2
glue ...................................... 2
glue_col ................................ 5
glue_collapse ......................... 6
glue_safe ................................ 7
glue_sql ................................ 8
identity_transformer .................. 11
quoting .................................. 11
trim ..................................... 12

Index

as_glue ................................. Coerce object to glue

Description

Coerce object to glue

Usage

as_glue(x, ...)

Arguments

x object to be coerced.
... further arguments passed to methods.

glue ................................... Format and interpolate a string

Description

Expressions enclosed by braces will be evaluated as R code. Long strings are broken by line and concatenated together. Leading whitespace and blank lines from the first and last lines are automatically trimmed.
Usage

```
 glue_data(
  .x,  # [listish]
  ...,  # [expressions]
  .sep = "",  # [character(1): ""]
  .envir = parent.frame(),  # [environment: parent.frame()]
  .open = "{",  # [character(1): '{']
  .close = "}",  # [character(1): '}']
  .na = "NA",  # Separator used to separate elements.
  .null = character(),  # Environment to evaluate each expression in. Expressions are evaluated from left
to right. If .x is an environment, the expressions are evaluated in that environment
and .envir is ignored. If NULL is passed it is equivalent to emptyenv().
  .comment = "#",  # The opening delimiter. Doubling the full delimiter escapes it.
  .transformer = identity_transformer,  # The closing delimiter. Doubling the full delimiter escapes it.
  .trim = TRUE
)
```

```
 glue(  # [expressions]
  ...,  # [expressions]
  .sep = "",  # [character(1): ""]
  .envir = parent.frame(),  # [environment: parent.frame()]
  .open = "{",  # [character(1): '{']
  .close = "}",  # [character(1): '}']
  .na = "NA",  # Separator used to separate elements.
  .null = character(),  # Environment to evaluate each expression in. Expressions are evaluated from left
to right. If .x is an environment, the expressions are evaluated in that environment
and .envir is ignored. If NULL is passed it is equivalent to emptyenv().
  .comment = "#",  # The opening delimiter. Doubling the full delimiter escapes it.
  .transformer = identity_transformer,  # The closing delimiter. Doubling the full delimiter escapes it.
  .trim = TRUE
)
```

Arguments

- `.x` [listish]
  An environment, list or data frame used to lookup values.

- `...` [expressions]
  Unnamed arguments are taken to be expressions string(s) to format. Multiple
inputs are concatenated together before formatting. Named arguments are taken
to be temporary variables available for substitution.

- `.sep` [character(1): ""]
  Separator used to separate elements.

- `.envir` [environment: parent.frame()]
  Environment to evaluate each expression in. Expressions are evaluated from left
to right. If `.x` is an environment, the expressions are evaluated in that environment
and `.envir` is ignored. If NULL is passed it is equivalent to `emptyenv()`.

- `.open` [character(1): '{']
  The opening delimiter. Doubling the full delimiter escapes it.

- `.close` [character(1): '}']
  The closing delimiter. Doubling the full delimiter escapes it.
.na [character(1): ‘NA’]
Value to replace NA values with. If NULL missing values are propagated, that is
an NA result will cause NA output. Otherwise the value is replaced by the value
of .na.

.null [character(1): ‘character()’]
Value to replace NULL values with. If character() whole output is character().
If NULL all NULL values are dropped (as in paste0()). Otherwise the value is
replaced by the value of .null.

.comment [character(1): ‘#’]
Value to use as the comment character.

.transformer [function]
A function taking three parameters code, envir and data used to transform the
output of each block before during or after evaluation. For example transformers
see vignette(“transformers”).

.trim [logical(1): ‘TRUE’]
Whether to trim the input template with trim() or not.

See Also
https://www.python.org/dev/peps/pep-0498/ and https://www.python.org/dev/peps/pep-0257/
upon which this is based.

Examples

name <- "Fred"
age <- 50
anniversary <- as.Date("1991-10-12")
glue("My name is {name}, 
  'my age next year is {age + 1},'
  'my anniversary is {format(anniversary, "%A, %B %d, %Y")}.
"")

# single braces can be inserted by doubling them

# Named arguments can be used to assign temporary variables.
glue("My name is {name}, 
  'my age next year is {age + 1},'
  'my anniversary is {format(anniversary, "%A, %B %d, %Y")}.
  name = "Joe",
  age = 40,
  anniversary = as.Date("2001-10-12")
"")

# `glue()` can also be used in user defined functions
intro <- function(name, profession, country){
  glue("My name is {name}, a {profession}, from {country}"
}
intro("Shelmith", "Senior Data Analyst", "Kenya")
intro("Cate", "Data Scientist", "Kenya")

# `glue_data()` is useful in magrittr pipes
if (require(magrittr)) {
  mtcars %>% glue_data("[rownames(.)] has {hp} hp")
  # Or within dplyr pipelines
  if (require(dplyr)) {
    head(iris) %>%
    mutate(description = glue("This {Species} has a petal length of (Petal.Length)"))
  }
  # Alternative delimiters can also be used if needed
  one <- "1"
  glue("The value of $e^{2\pi i}$ is $<<one>>$.", .open = "<<", .close = ">>")
}

### `glue_col`

#### Construct strings with color

**Description**

The `crayon` package defines a number of functions used to color terminal output. `glue_col()` and `glue_data_col()` functions provide additional syntax to make using these functions in `glue` strings easier.

Using the following syntax will apply the function `crayon::blue()` to the text 'foo bar'.

```
{blue foo bar}
```

If you want an expression to be evaluated, simply place that in a normal brace expression (these can be nested).

```
{blue 1 + 1 = {1 + 1}}
```

**Usage**

```
glue_col(..., .envir = parent.frame(), .na = "NA")
```

```
glue_data_col(.x, ..., .envir = parent.frame(), .na = "NA")
```

**Arguments**

- `...` [expressions]
  
  Unnamed arguments are taken to be expressions string(s) to format. Multiple inputs are concatenated together before formatting. Named arguments are taken to be temporary variables available for substitution.

- `.envir` [environment: parent.frame()]
  
  Environment to evaluate each expression in. Expressions are evaluated from left to right. If `.x` is an environment, the expressions are evaluated in that environment and `.envir` is ignored. If NULL is passed it is equivalent to `emptyenv()`.
glue_collapse

Description

`glue_collapse()` collapses a character vector of any length into a length 1 vector. `glue_sql_collapse()` does the same but returns a `[DBI::SQL()]` object rather than a glue object.

Usage

```r
glue_collapse(x, sep = "", width = Inf, last = "")
glue_sql_collapse(x, sep = "", width = Inf, last = ")
```

Arguments

- `x` The character vector to collapse.
- `sep` a character string to separate the terms. Not `NA_character_`.
- `width` The maximum string width before truncating with `...`
- `last` String used to separate the last two items if `x` has at least 2 items.

Examples

```r
if (require(crayon)) {
  glue_col("(blue foo bar)")
  glue_col("(blue 1 + 1 = (1 + 1))")
  white_on_grey <- bgBlack $ white
  glue_col("(white_on_grey
Roses are {red (colors()[[552]])}
Violets are {blue (colors()[[26]])}
\'glue_col\' can show {red c}{yellow o}{green l}{cyan o}{blue r}{magenta s}
and {bold bold} and {underline underline} too!
\)
}
```
Examples

```r
glue_collapse(glue("{1:10}"))
```

# Wide values can be truncated
```r
glue_collapse(glue("{1:10}"), width = 5)
```

```r
glue_collapse(1:4, ", ", last = " and ")
```

#> 1, 2, 3 and 4

---

**glue_safe**

*Safely interpolate strings*

Description

`glue_safe()` and `glue_data_safe()` differ from `glue()` and `glue_data()` in that the safe versions only look up symbols from an environment using `get()`; they do not execute any R code. This makes them suitable when used with untrusted input, such as inputs in a shiny application, where using the normal functions would allow an attacker to execute arbitrary code.

Usage

```r
glue_safe(..., .envir = parent.frame())
```

```r
glue_data_safe(.x, ..., .envir = parent.frame())
```

Arguments

...  
[expressions]
Unnamed arguments are taken to be expressions string(s) to format. Multiple inputs are concatenated together before formatting. Named arguments are taken to be temporary variables available for substitution.

.envir  
[environment: parent.frame()]
Environment to evaluate each expression in. Expressions are evaluated from left to right. If `.x` is an environment, the expressions are evaluated in that environment and `.envir` is ignored. If `NULL` is passed it is equivalent to `emptyenv()`.

.x  
[listish]
An environment, list or data frame used to lookup values.

Examples

```r
"1 + 1" <- 5
# glue actually executes the code
```

```r
glue("{1 + 1}"
```

```r
# glue_safe just looks up the value

glue_safe("{1 + 1}"
```

```r
rm("1 + 1")
```
**glue_sql**

*Interpolate strings with SQL escaping*

**Description**

SQL databases often have custom quotation syntax for identifiers and strings which make writing SQL queries error prone and cumbersome to do. `glue_sql()` and `glue_data_sql()` are analogs to `glue()` and `glue_data()` which handle the SQL quoting. `glue_sql_collapse()` can be used to collapse `DBI::SQL()` objects.

**Usage**

```r
glue_sql(..., .con, .envir = parent.frame(), .na = DBI::SQL("NULL"))

glue_data_sql(.x, ..., .con, .envir = parent.frame(), .na = DBI::SQL("NULL"))
```

**Arguments**

- `...` [expressions]
  Unnamed arguments are taken to be expressions string(s) to format. Multiple inputs are concatenated together before formatting. Named arguments are taken to be temporary variables available for substitution.

- `.con` [DBIConnection]: A DBI connection object obtained from `DBI::dbConnect()`.

- `.envir` [environment: parent.frame()]
  Environment to evaluate each expression in. Expressions are evaluated from left to right. If `.x` is an environment, the expressions are evaluated in that environment and `.envir` is ignored. If NULL is passed it is equivalent to `emptyenv()`.

- `.na` [character(1): ‘NA’]
  Value to replace NA values with. If NULL missing values are propagated, that is an NA result will cause NA output. Otherwise the value is replaced by the value of `.na`.

- `.x` [listish]
  An environment, list or data frame used to lookup values.

**Details**

They automatically quote character results, quote identifiers if the glue expression is surrounded by backticks `'```'` and do not quote non-characters such as numbers. If numeric data is stored in a character column (which should be quoted) pass the data to `glue_sql()` as a character.

Returning the result with `DBI::SQL()` will suppress quoting if desired for a given value.

Note *parameterized queries* are generally the safest and most efficient way to pass user defined values in a query, however not every database driver supports them.

If you place a `*` at the end of a glue expression the values will be collapsed with commas. This is useful for the SQL IN Operator for instance.
Value

A \texttt{DBI::SQL} object with the given query.

See Also

\texttt{glue_sql\_collapse} to collapse \texttt{DBI::SQL} objects.

Examples

c\texttt{on} \texttt{con} \texttt{\leftarrow} \texttt{DBI::\texttt{dbConnect(RSQLite::SQLite(), ":memory:\")}}
\texttt{iris2 \texttt{\leftarrow} iris}
colnames(iris2) \texttt{\leftarrow} gsub("[.]", ",\", tolower(colnames(iris)))
\texttt{DBI::\texttt{dbWriteTable(con, "iris", iris2)}
\texttt{var} \texttt{\leftarrow "sepal\_width"}
\texttt{tbl} \texttt{\leftarrow "iris"}
\texttt{num} \texttt{\leftarrow 2}
\texttt{val} \texttt{\leftarrow "setosa"}
\texttt{glue\_sql("}
\quad \texttt{SELECT \{\texttt{\backslash Var}\texttt{var}\}
\quad \texttt{FROM \{\texttt{\backslash Var}\texttt{tbl}\}}
\quad \texttt{WHERE \{\texttt{\backslash Var}\texttt{tbl}\}.sepal\_length > \{num\}}
\quad \texttt{AND \{\texttt{\backslash Var}\texttt{ tbl}\}.species = \{val\}}
\quad \texttt{\", .con = con\})
\texttt{# If sepal\_length is store on the database as a character explicitly convert}
\texttt{# the data to character to quote appropriately.}
\texttt{glue\_sql("}
\quad \texttt{SELECT \{\texttt{\backslash Var}\texttt{var}\}
\quad \texttt{FROM \{\texttt{\backslash Var}\texttt{tbl}\}}
\quad \texttt{WHERE \{\texttt{\backslash Var}\texttt{ tbl}\}.sepal\_length > \{as.character(num)\}}
\quad \texttt{AND \{\texttt{\backslash Var}\texttt{ tbl}\}.species = \{val\}}
\quad \texttt{\", .con = con\})
\texttt{# `glue\_sql()` can be used in conjuction with parameterized queries using}
\texttt{# `\texttt{DBI::dbBind()}` to provide protection for SQL Injection attacks}
\texttt{sql} \texttt{\leftarrow glue\_sql("}
\quad \texttt{SELECT \{\texttt{\backslash Var}\texttt{var}\}
\quad \texttt{FROM \{\texttt{\backslash Var}\texttt{tbl}\}}
\quad \texttt{WHERE \{\texttt{\backslash Var}\texttt{ tbl}\}.sepal\_length > ?}
\quad \texttt{\", .con = con\})
\texttt{query} \texttt{\leftarrow DBI::\texttt{dbSendQuery(con, sql)}
\texttt{DBI::\texttt{dbBind(query, list(num))}}
\texttt{DBI::\texttt{dbFetch(query, n = 4)}
\texttt{DBI::\texttt{dbClearResult(query)}
\texttt{# `glue\_sql()` can be used to build up more complex queries with}
\texttt{# interchangeable sub queries. It returns `\texttt{DBI::SQL()}` objects which are}
\texttt{# properly protected from quoting.}
\texttt{sub\_query} \texttt{\leftarrow glue\_sql("}
\quad \texttt{SELECT *}
\quad \texttt{FROM \{'tbl\'}
\texttt{)}}


```
".con = con)

glue_sql("SELECT s.(\'var\')
    FROM ((sub_query)) AS s
    ", .con = con)

# If you want to input multiple values for use in SQL IN statements put `\` at the end of the value and the values will be collapsed and quoted appropriately.

    glue_sql("SELECT * FROM (\{tbl\}) WHERE sepal_length IN ((vals*))",
        vals = 1:5, .con = con)

    glue_sql("SELECT * FROM (\{tbl\}) WHERE species IN ((vals*))",
        vals = c("setosa", "versicolor"), .con = con)

    glue_sql("SELECT * FROM (\{tbl\}) WHERE species IN ((vals*))",
        vals = c("setosa", "versicolor"), .con = con)

    glue_sql("SELECT * FROM (\{tbl\}) WHERE sepal_length IN ((vals*))",
        vals = 1:5, .con = con)

    glue_sql("SELECT * FROM (\{tbl\}) WHERE sepal_length IN ((vals*))",
        vals = c("setosa", "versicolor"), .con = con)

    glue_sql("SELECT * FROM (\{tbl\}) WHERE species IN ((vals*))",
        vals = c("setosa", "versicolor"), .con = con)

# If you need to reference variables from multiple tables use `DBI::Id()`.
# Here we create a new table of nicknames, join the two tables together and
# select columns from both tables. Using `DBI::Id()` and the special
# `\`glue_sql\` syntax ensures all the table and column identifiers are quoted
# appropriately.

iris_db <- "iris"

nicknames_db <- "nicknames"

nicknames <- data.frame(
    species = c("setosa", "versicolor", "virginica"),
    nickname = c("Beachhead Iris", "Harlequin Blueflag", "Virginia Iris"),
    stringsAsFactors = FALSE
)

DBI::dbWriteTable(con, nicknames_db, nicknames)

cols <- list(
    DBI::Id(table = iris_db, column = "sepal_length"),
    DBI::Id(table = iris_db, column = "sepal_width"),
    DBI::Id(table = nicknames_db, column = "nickname")
)

iris_species <- DBI::Id(table = iris_db, column = "species")
nicknames_species <- DBI::Id(table = nicknames_db, column = "species")

query <- glue_sql(""
    SELECT `\{cols\}`
    FROM `\{iris_db\}`
    JOIN `\{nicknames_db\}`
    ON `\{iris_species\}`=`\{nicknames_species\}`",
    .con = con
```

".con = con)
identity_transformer

) query
  DBI::dbGetQuery(con, query, n = 5)
  DBI::dbDisconnect(con)

identity_transformer   Parse and Evaluate R code

Description

This is a simple wrapper around eval(parse()), used as the default transformer.

Usage

identity_transformer(text, envir)

Arguments

text       Text (typically) R code to parse and evaluate.
envir      environment to evaluate the code in

See Also

vignette("transformers","glue") for documentation on creating custom glue transformers and some common use cases.

_____________________

quoting   Quoting operators

_____________________

Description

These functions make it easy to quote each individual element and are useful in conjunction with glue_collapse().

Usage

single_quote(x)

double_quote(x)

backtick(x)

Arguments

  x       A character to quote.
Examples

```r
x <- 1:5
glue('Values of x: {glueCollapse(backtick(x), sep = "", last = "and ")}')
```

Description

This trims a character vector according to the trimming rules used by glue. These follow similar rules to Python Docstrings, with the following features.

- Leading and trailing whitespace from the first and last lines is removed.
- A uniform amount of indentation is stripped from the second line on, equal to the minimum indentation of all non-blank lines after the first.
- Lines can be continued across newlines by using `\`.

Usage

```r
trim(x)
```

Arguments

- `x` A character vector to trim.

Examples

```r
glue("A formatted string
Can have multiple lines
with additional indentation preserved")
```

```r
glue("\ntrailing or leading newlines can be added explicitly\n"
"
```

```r
glue("A formatted string \n can also be on a \n single line"
"
```
Index

as_glue, 2
backtick (quoting), 11

 crayon, 5
 crayon::blue(), 5

 DBI::dbConnect(), 8
 DBI::SQL(), 8, 9
 double_quote (quoting), 11

eemptyenv(), 3, 5, 7, 8

 get(), 7
 glue, 2
 glue(), 7, 8
 glue_col, 5
 glueCollapse, 6
 glueCollapse(), 11
 glue_data (glue), 2
 glue_data(), 7, 8
 glue_data_col (glue_col), 5
 glue_data_safe (glue_safe), 7
 glue_data_sql (glue_sql), 8
 glue_safe, 7
 glue_sql, 8
 glue_sqlCollapse (glueCollapse), 6
 glue_sqlCollapse(), 9

 identity_transformer, 11

 NA_character_, 6

 quoting, 11

 single_quote (quoting), 11

 trim, 12
 trim(), 4