

Package ‘glue’

July 17, 2018

Title Interpreted String Literals

Version 1.3.0

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 <<https://docs.julialang.org/en/stable/manual/strings/#triple-quoted-string-literals>>.

Depends R (>= 3.1)

Imports methods

Suggests testthat, covr, magrittr, crayon, knitr, rmarkdown, DBI, RSQLite, R.utils, forcats, microbenchmark, rprintf, stringr, ggplot2, dplyr, withr

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

URL <https://github.com/tidyverse/glue>

BugReports <https://github.com/tidyverse/glue/issues>

VignetteBuilder knitr

ByteCompile true

NeedsCompilation yes

Author Jim Hester [aut, cre]

Maintainer Jim Hester <james.f.hester@gmail.com>

Repository CRAN

Date/Publication 2018-07-17 19:10:02 UTC

R topics documented:

as_glue	2
glue	2
glue_col	4
glue_collapse	5
glue_sql	6
identity_transformer	8
quoting	9
trim	9

Index **11**

as_glue	<i>Coerce object to glue</i>
---------	------------------------------

Description

Coerce object to glue

Usage

```
as_glue(x, ...)
```

Arguments

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

glue	<i>Format and interpolate a string</i>
------	--

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, ..., .sep = "", .envir = parent.frame(), .open = "{",
  .close = "}", .na = "NA", .transformer = identity_transformer,
  .trim = TRUE)
```

```
glue(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}",
  .na = "NA", .transformer = identity_transformer)
```

Arguments

<code>.x</code>	[listish] An environment, list or data frame used to lookup values.
<code>...</code>	[expressions] Expressions string(s) to format, multiple inputs are concatenated together before formatting.
<code>.sep</code>	[character(1): ""] Separator used to separate elements.
<code>.envir</code>	[environment: parent.frame()] Environment to evaluate each expression in. Expressions are evaluated from left to right. If <code>.x</code> is an environment, the expressions are evaluated in that environment and <code>.envir</code> is ignored.
<code>.open</code>	[character(1): '{'] The opening delimiter. Doubling the full delimiter escapes it.
<code>.close</code>	[character(1): '}'] The closing delimiter. Doubling the full delimiter escapes it.
<code>.na</code>	[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 <code>.na</code> .
<code>.transformer</code>	[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 <code>vignette("transformers")</code> .
<code>.trim</code>	[logical(1): 'TRUE'] Whether to trim the input template with <code>trim()</code> 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

```
glue("My name is {name}, not {{name}}.")
```

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_data()` is useful in magrittr pipes
library(magrittr)
mtcars %>% glue_data("{rownames(.)} has {hp} hp")

# Or within dplyr pipelines
library(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 blue function 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]
Expressions string(s) to format, multiple inputs are concatenated together before
formatting.
```

<code>.envir</code>	[environment: parent.frame()] Environment to evaluate each expression in. Expressions are evaluated from left to right. If <code>.x</code> is an environment, the expressions are evaluated in that environment and <code>.envir</code> is ignored.
<code>.na</code>	[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 <code>.na</code> .
<code>.x</code>	[listish] An environment, list or data frame used to lookup values.

Examples

```
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!
  }")
}
```

<code>glue_collapse</code>	<i>Collapse a character vector</i>
----------------------------	------------------------------------

Description

Collapses a character vector of any length into a length 1 vector.

Usage

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

Arguments

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

Examples

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

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

glue_collapse(1:4, ", ", last = " and ")
#> 1, 2, 3 and 4
```

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.

Usage

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

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

Arguments

<code>...</code>	[expressions] Expressions string(s) to format, multiple inputs are concatenated together before formatting.
<code>.con</code>	[DBIConnection]: A DBI connection object obtained from <code>DBI::dbConnect()</code> .
<code>.envir</code>	[environment: <code>parent.frame()</code>] Environment to evaluate each expression in. Expressions are evaluated from left to right. If <code>.x</code> is an environment, the expressions are evaluated in that environment and <code>.envir</code> is ignored.
<code>.na</code>	[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 <code>.na</code> .
<code>.x</code>	[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 `DBI::SQL()` object with the given query.

Examples

```
con <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
colnames(iris) <- gsub("[.]", "_", tolower(colnames(iris)))
DBI::dbWriteTable(con, "iris", iris)
var <- "sepal_width"
tbl <- "iris"
num <- 2
val <- "setosa"
glue_sql("
  SELECT `{var}`
  FROM `{tbl}`
  WHERE `{tbl}`.sepal_length > {num}
    AND `{tbl}`.species = {val}
", .con = con)

# If sepal_length is store on the database as a character explicitly convert
# the data to character to quote appropriately.
glue_sql("
  SELECT `{var}`
  FROM `{tbl}`
  WHERE `{tbl}`.sepal_length > {as.character(num)}
    AND `{tbl}`.species = {val}
", .con = con)

# `glue_sql()` can be used in conjunction with parameterized queries using
# `DBI::dbBind()` to provide protection for SQL Injection attacks
sql <- glue_sql("
  SELECT `{var}`
  FROM `{tbl}`
  WHERE `{tbl}`.sepal_length > ?
", .con = con)
query <- DBI::dbSendQuery(con, sql)
DBI::dbBind(query, list(num))
DBI::dbFetch(query, n = 4)
DBI::dbClearResult(query)
```

```

# `glue_sql()` can be used to build up more complex queries with
# interchangeable sub queries. It returns `DBI::SQL()` objects which are
# properly protected from quoting.
sub_query <- glue_sql("
  SELECT *
  FROM `{tbl}`
  ", .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, .con = con)

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 = "setosa", .con = con)

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

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

<code>text</code>	Text (typically) R code to parse and evaluate.
<code>envir</code>	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

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

`trim`*Trim a character vector*

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`trim(x)`**Arguments**

`x` A character vector to trim.

Examples

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

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

```
glue("
  A formatted string \\
  can also be on a \\
  single line
")
```

Index

`as_glue`, 2

`backtick` (quoting), 9

`crayon`, 4

`double_quote` (quoting), 9

`glue`, 2

`glue_col`, 4

`glue_collapse`, 5

`glue_data` (`glue`), 2

`glue_data_col` (`glue_col`), 4

`glue_data_sql` (`glue_sql`), 6

`glue_sql`, 6

`identity_transformer`, 8

`NA_character_`, 5

`quoting`, 9

`single_quote` (quoting), 9

`trim`, 9