

# Package ‘glue’

October 29, 2017

**Title** Interpreted String Literals

**Version** 1.2.0

**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)

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

**Imports** methods

**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](mailto:james.f.hester@gmail.com)>

**Repository** CRAN

**Date/Publication** 2017-10-29 20:03:53 UTC

**R topics documented:**

as_glue	2
collapse	2
evaluate	3
glue	3
glue_sql	5
quoting	7
trim	7

<b>Index</b>	<b>9</b>
--------------	----------

---

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.

---

collapse	<i>Collapse a character vector</i>
----------	------------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

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

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

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

---

 evaluate

*Evaluate R code*


---

**Description**

This is a simple wrapper around `eval(parse())` which provides a more consistent interface than the default functions. If data is NULL then the code is evaluated in the environment. If data is not NULL then the code is evaluated in the data object first, with the enclosing environment of `envir`.

**Usage**

```
evaluate(code, envir)
```

**Arguments**

<code>code</code>	R code to evaluate
<code>envir</code>	environment to evaluate the code in

**Details**

This function is designed to be used within transformers to evaluate the code in the glue block.

---

 glue

*Format and interpolate a string*


---

**Description**

Expressions enclosed by braces will be evaluated as R code. Single braces can be inserted by doubling them.

**Usage**

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

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

**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>code</code> , <code>envir</code> and <code>data</code> used to transform the output of each block before during or after evaluation. For example transformers see <code>vignette("transformers")</code> .

**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 also be supplied

```
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")

# Alternative delimiters can also be used if needed
one <- "1"
glue("The value of  $e^{2\pi i}$  is  $\$<<one>>\$.$ ", .open = "<<", .close = ">>")

```

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_sql_data()` are analogs to `glue()` and `glue_data()` which handle the SQL quoting.

**Usage**

```

glue_sql(..., .con, .envir = parent.frame())

glue_data_sql(.x, ..., .con, .envir = parent.frame())

```

**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>.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.

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)

# `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)
```

---

quoting

*Quoting operators*

---

### Description

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

### Usage

```
single_quote(x)
```

```
double_quote(x)
```

```
backtick(x)
```

### Arguments

`x` A character to quote.

### Examples

```
x <- 1:5
glue('Values of x: {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), 7

`collapse`, 2

`double_quote` (quoting), 7

`evaluate`, 3

`glue`, 3

`glue_data` (`glue`), 3

`glue_data_sql` (`glue_sql`), 5

`glue_sql`, 5

`NA_character_`, 2

quoting, 7

`single_quote` (quoting), 7

`trim`, 7