Package 'epoxy'

August 31, 2026

Title String Interpolation for Documents, Reports and Apps

Version 1.0.1

Description Extra strength 'glue' for data-driven templates. String interpolation for 'Shiny' apps or 'R Markdown' and 'knitr'-powered 'Quarto' documents, built on the 'glue' and 'whisker' packages.

License MIT + file LICENSE

URL https://pkg.garrickadenbuie.com/epoxy/, https://github.com/gadenbuie/epoxy

BugReports https://github.com/gadenbuie/epoxy/issues

Depends R (>= 3.6.0)

Imports and, glue (>= 1.5.0), htmltools, knitr (>= 1.37), lifecycle, purrr, rlang, rmarkdown, scales (>= 1.1.0), tools, whisker

Suggests cleanrmd, commonmark, dbplyr, debugme, dplyr, pandoc, shiny, shinytest2, testthat

VignetteBuilder cleanrmd, knitr, rmarkdown

Config/Needs/rcmdcheck RSQLite, rstudio/chromote

Config/Needs/website rstudio/rmarkdown, gadenbuie/grkgdown

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Garrick Aden-Buie [aut, cre] (ORCID: <https://orcid.org/0000-0002-7111-0077>), Kushagra Gour [ctb] (hint.css), The mustache.js community [ctb] (mustache.js)

Maintainer Garrick Aden-Buie <garrick@adenbuie.com>

Repository CRAN

Date/Publication 2026-08-31 18:40:13 UTC

Contents

bechdel 2
engine_pick 3
epoxy 4
epoxy_mustache 7
epoxy_transform 9
epoxy_transform_html 12
epoxy_transform_inline 14
epoxy_transform_one_shot 19
epoxy_use 21
render_epoxy 24
run_epoxy_example_app 25
ui_epoxy_html 27
ui_epoxy_markdown 31
ui_epoxy_mustache 34
use_epoxy_knitr_engines 37

Index **39**

`bechdel` *Top 10 Highest-Rated, Bechdel-Passing Movies*

Description

A small dataset for **epoxy** demonstrations with the top audience-rated movies that pass the Bechdel Test.

Usage

```
bechdel
```

Format

A data frame with 10 rows and 18 variables:

imdb_id IMDB Movie ID

bechdel_rating Rating (0-3): 0 = unscored; 1 = It has to have at least two (named) women in it; 2 = Who talk to each other; 3 = About something besides a man.

year Year

title Title of movie

budget Budget in $USD as of release year

domgross Domestic gross in $USD in release year

intgross International gross in $USD in release year

plot Plot of the movie

rated Moving rating, e.g. PG, PG-13, R, etc.

language Language of the movie

country Country where the movie was produced

imdb_rating IMDB rating of the movie, 0-10

director Director of the movie

actors Major actors appearing in the movie

genre Genre

awards Awards won by the movie, text description

runtime Movie runtime in minutes

poster URL of movie poster image, sourced from themoviedb.org. Poster images URLs ar provided from the TMDB API but **epoxy** is not endorsed or certified by TMDB.

Source

TidyTuesday (2021-03-09), FiveThirtyEight, bechdeltest.com, themoviedb.org

---

`engine_pick` *Pick an engine-specific value*

Description

Set different values that will be used based on the current epoxy or knitr engine (one of `md`, `html`, or `latex`). The engine-specific value will be used inside epoxy knitr chunks or epoxy functions matching the source syntax: `epoxy()` (`md`), `epoxy_html()` (`html`), or `epoxy_latex()` (`latex`).

Usage

```
engine_pick(md, html = md, latex = md)
```

Arguments

`md`, `html`, `latex` The value to use in a markdown, HTML, or LaTeX context.

Value

The value of `md`, `html` or `latex` depending on the epoxy or knitr currently being evaluated.

Examples

```
# Markdown and HTML are okay with bare `$` character,
# but we need to escape it in LaTeX.
engine_pick(md = "$", latex = "\\$")
```

epoxy *Epoxy string interpolation*

Description

These functions power the knitr chunk engines and are wrappers around `glue::glue()`, with a few extra conveniences provided by **epoxy**.

- `epoxy()` is super `glue::glue()`.
- `epoxy_html()` is super `glue::glue()` with HTML-specific defaults.
- `epoxy_latex()` is super `glue::glue()` with LaTeX-specific defaults.

Each of these functions can be called directly or used as a knitr chunk engine where the chunk text is handled as if it were a string passed into the function version. When used as a knitr chunk engine, the function arguments can be passed in as chunk options.

All of `epoxy()`, `epoxy_html()` and `epoxy_latex()` use `epoxy_transform_inline()` by default. This transformer brings a concise inline-formatting syntax that you can read more about in `?epoxy_transform_inline`.

`epoxy_html()` also includes an inline transformation syntax that makes it easier to wrap the expression text in HTML elements with a specific ID or a set of classes. Learn more about this syntax in `?epoxy_transform_html`.

Usage

```
epoxy(
  ...,
  .data = NULL,
  .sep = "",
  .envir = parent.frame(),
  .open = "{",
  .close = "}",
  .na = "",
  .null = "",
  .comment = character(),
  .literal = FALSE,
  .trim = FALSE,
  .transformer = NULL,
  .collapse = NULL,
  .style = lifecycle::deprecated()
)

epoxy_html(
  ...,
  .data = NULL,
  .sep = "",
  .envir = parent.frame(),
  .open = "{{",
```

```
  .close = "}}",
  .na = "",
  .null = "",
  .comment = "",
  .literal = FALSE,
  .trim = FALSE,
  .transformer = NULL,
  .collapse = NULL
)

epoxy_latex(
  ...,
  .data = NULL,
  .sep = "",
  .envir = parent.frame(),
  .open = "<<",
  .close = ">>",
  .na = "",
  .null = "",
  .comment = "",
  .literal = FALSE,
  .trim = FALSE,
  .transformer = NULL,
  .collapse = NULL
)
```

Arguments

`...`	[`expressions`] Unnamed arguments are taken to be expression string(s) to format. Multiple inputs are concatenated together before formatting. Named arguments are taken to be temporary variables available for substitution. For `glue_data()`, elements in `...` override the values in `.x`.
`.data`	A data set
`.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 around the template variable or expression. Doubling the full delimiter escapes it.
`.close`	[`character(1)`: `\}`] The closing delimiter around the template variable or expression. 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.
`.literal`	[`boolean(1)`: 'FALSE'] Whether to treat single or double quotes, backticks, and comments as regular characters (vs. as syntactic elements), when parsing the expression string. Setting `.literal = TRUE` probably only makes sense in combination with a custom `.transformer`, as is the case with `glue_col()`. Regard this argument (especially, its name) as experimental.
`.trim`	[`logical(1)`: 'TRUE'] Whether to trim the input template with `trim()` or not.
`.transformer`	A transformer function or transformer chain created with `epoxy_transform()`. Alternatively, a character vector of epoxy transformer names, e.g. `c("bold", "collapse")` or a list of epoxy transformers, e.g. `list(epoxy_transform_bold(), epoxy_transform_collapse())`. In **epoxy**, you'll most likely want to use the defaults or consult `epoxy_transform()` for more information. See also `glue::glue()` for more information on transformers.
`.collapse`	A character string used to collapse a vector result into a single value. If `NULL` (the default), the result is not collapsed.
`.style`	**[Deprecated]** Please use `.transformer` instead.

Value

Returns a transformed string, using `glue::glue()` but with the additional transformers provided to the `.transformer` argument of `epoxy()`.

See Also

- `use_epoxy_knitr_engines()` for knitr engines powered by these epoxy functions.
- `epoxy_mustache()` for more powerful templating needs when you don't need epoxy's inline formatting syntax.

Examples

```
movie <- bechdel[1, ]
movies <- bechdel[2:4, ]

# A basic example with a single row of data
epoxy("{.emph movie$title} ({movie$year}) was directed by {movie$director}.")
```

```
# Or vectorized over multiple rows of data
epoxy("* {.emph movies$title} ({movies$year}) was directed by {movies$director}.")

# You can provide the data frame to `.data` to avoid repeating `data$`
epoxy("{.emph title} ({year}) was directed by {director}.", .data = movie)
epoxy("* {.emph title} ({year}) was directed by {director}.", .data = movies)

# Inline transformers can be nested
epoxy("I'd be happy to watch {.or {.italic title}}.", .data = movies)
epoxy("They were directed by {.and {.bold director}}.", .data = movies)

# Learn more about inline transformers in ?epoxy_transform_inline
epoxy("The budget for {.emph title} was {.dollar budget}.", .data = movie)

# --------- HTML and LaTeX variants ---------
# There are also HTML and LaTeX variants of epoxy.
# Each uses default options that are most natural for the format.

# epoxy_html() uses `{{ expr }}` for delimiters
epoxy_html("I'd be happy to watch {{ title }}.", .data = movie)
# It also supports an HTML transformer syntax
epoxy_html("I'd be happy to watch {{em.movie-title title}}.", .data = movie)
# Or use the inline transformer syntax, which uses `@` instead of `.` in HTML
epoxy_html("I'd be happy to watch {{@or {{@emph title}} }}.", .data = movies)

# epoxy_latex() uses `<< expr >>` for delimiters
epoxy_latex("I'd be happy to watch <<.or <<.emph title >> >>.", .data = movies)
```

`epoxy_mustache` *Mustache-style string interpolation*

Description

[Experimental] A wrapper around the mustache templating language, provided by the whisker package. Under the hood, `epoxy_mustache()` uses `whisker::whisker.render()` to render the template, but adds a few conveniences:

- The template can be passed in `...` as a single string, several strings or as a vector of strings. If multiple strings are passed, they are collapsed with `.sep` (`"\n"` by default).
- `epoxy_mustache()` can be vectorized over the items in the `.data` argument. If `.data` is a data frame, vectorization is turned on by default so that you can iterate over the rows of the data frame. The output will be a character vector of the same length as the number of rows in the data frame.

Usage

```
epoxy_mustache(
  ...,
```

```
  .data = parent.frame(),
  .sep = "\n",
  .vectorized = inherits(.data, "data.frame"),
  .partials = list()
)
```

Arguments

`...`	A string or a vector of strings containing the template(s). Refer to the mustache documentation for an overview of the syntax. If multiple strings are passed, they are collapsed with `.sep` (`"\n"` by default).
`.data`	A data frame or a list. If `.data` is a data frame, `epoxy_mustache()` will transform the data frame so that the template can be applied to each row of the data frame. To avoid this transformation, wrap the `.data` value in `I()`.
`.sep`	The separator to use when collapsing multiple strings passed in `...` into a single template. Defaults to `"\n"`.
`.vectorized`	If `TRUE` , `epoxy_mustache()` will vectorize over the items in `.data`. In other words, each item or row of `.data` will be used to render the template once. By default, `.vectorized` is set to `TRUE` if `.data` is a data frame and `FALSE` otherwise.
`.partials`	A named list with partial templates. See `whisker::whisker.render()` or the mustache documentation for details.

Value

A character vector of length 1 if `.vectorized` is `FALSE` or a character vector of the same length as the number of rows or items in `.data` if `.vectorized` is `TRUE`.

See Also

Other Mustache-style template functions: `ui_epoxy_mustache()`

Examples

```
# The canonical mustache example
epoxy_mustache(
  "Hello {{name}}!",
  "You have just won {{value}} dollars!",
  "{{#in_ca}}",
  "Well, {{taxed_value}} dollars, after taxes.",
  "{{/in_ca}}",
  .data = list(
    name = "Chris",
    value = 10000,
    taxed_value = 10000 - (10000 * 0.4),
    in_ca = TRUE
  )
)
```

```
# Vectorized over the rows of .data
epoxy_mustache(
 "mpg: {{ mpg }}",
 "hp: {{ hp }}",
 "wt: {{ wt }}\n",
 .data = mtcars[1:2, ]
)

# Non-vectorized
epoxy_mustache(
 "mpg: {{ mpg }}",
 "hp: {{ hp }}",
 "wt: {{ wt }}",
 .data = mtcars[1:2, ],
  .vectorized = FALSE
)

# With mustache partials
epoxy_mustache(
  "Hello {{name}}!",
 "{{> salutation }}",
  "You have just won {{value}} dollars!",
  "{{#in_ca}}",
  "Well, {{taxed_value}} dollars, after taxes.",
  "{{/in_ca}}",
 .partials = list(
  salutation = c("Hope you are well, {{name}}.")
 ),
 .sep = " ",
  .data = list(
    name = "Chris",
    value = 10000,
    taxed_value = 10000 - (10000 * 0.4),
    in_ca = TRUE
  )
)
```

`epoxy_transform` *epoxy Transformers*

Description

These transformers provide additional automatic formatting for the template strings. They are designed to be used with the `.transformer` chunk option of in epoxy chunks. You can use `epoxy_transform()` to chain several transformers together. `epoxy_transform()` and individual **epoxy** transform functions can be used in `epoxy`, `epoxy_html` and `epoxy_latex` chunks and will choose the correct engine for each.

Usage

```
epoxy_transform(..., engine = NULL, syntax = lifecycle::deprecated())

epoxy_transform_get(engine = c("md", "html", "latex"), inline = FALSE)

epoxy_transform_set(..., engine = NULL, syntax = lifecycle::deprecated())
```

Arguments

`...`	Transformer functions, e.g. epoxy_transform_bold or the name of an **epoxy** transform function, e.g. `"bold"`, or a call to a transform function, e.g. `epoxy_transform_bold()`. `epoxy_transform()` chains the transformer functions together, applying the transformers in order from first to last. For example, `epoxy_transform("bold", "collapse")` results in replaced strings that are emboldened *and then* collapsed, e.g. `**a** and **b**`. On the other hand, `epoxy_transform("collapse", "bold")` will collapse the vector *and then* embolden the entire string. In `epoxy_transform_apply()`, the `...` are passed to the underlying call the underlying function call. In `epoxy_transform_collapse()`, the `...` are ignored.
`engine`	One of `"markdown"` (or `"md"`), `"html"`, or `"latex"`. The default is chosen based on the engine of the chunk where the transform function is called, or according to the option `epoxy.engine`. Caution: invalid options are silently ignored, falling back to `"markdown"`.
`syntax`	**[Deprecated]** Use `engine` instead.
`inline`	In `epoxy_transform_get()`, whether to return the session-specific inline formatting functions for `epoxy_transform_inline()`.

Value

A function of `text` and `envir` suitable for the `.transformer` argument of `glue::glue()`.

Functions

- `epoxy_transform()`: Construct a chained transformer using **epoxy** transformers for use as a glue transformer. The resulting transformers can be passed to the `.transformer` argument of `epoxy()` or `glue::glue()`.
- `epoxy_transform_get()`: Get the default epoxy `.transformer` for all epoxy engines or for a subset of engines.
- `epoxy_transform_set()`: Set the default epoxy `.transformer` for all epoxy engines or for a subset of engines.

Output-specific transformations

The `epoxy_transform_` functions will attempt to use the correct engine for transforming the replacement text for markdown, HTML and LaTeX. This choice is driven by the chunk engine where

the transformer function is used. The `epoxy` engine corresponds to markdown, `epoxy_html` to HTML, and `epoxy_latex` to LaTeX.

Automatic engine selection only works when the epoxy transform functions are used with epoxy knitr engines and during the knitr rendering process. When used outside of this context, you can choose the desired engine by setting the `engine` to one of `"markdown"`, `"html"` or `"latex"`.

Session-wide settings

To change the transformer used by `epoxy()` and the HTML and LaTeX variants, use `epoxy_transform_set()`. This function takes the same values as `epoxy_transform()`, but makes them the default transformer for any `epoxy()` calls that do not specify a transformer. By default, the setting is made for all engines, but you can specify a single engine with the `engine` argument.

Here's a small example that applies the bold and collapse transformers to all epoxy chunks:

```
epoxy_transform_set("bold", "collapse")
```

Most often, you'll want to to update the default transformer to customize the formatting functions used by the inline transformer. You can use `epoxy_transform_set()` to change settings of existing formatting functions or to add new one. Pass the new function to an argument with the dot-prefixed name.

In the next example I'm setting the `.dollar` transformation to use "K" and "M" to abbreviate large numbers. I'm also adding my own transformation that truncates long strings to fit in 8 characters.

```
epoxy_transform_set(
 .dollar = scales::label_dollar(
  accuracy = 0.01,
  scale_cut = scales::cut_short_scale()
 ),
 .trunc8 = function(x) glue::glue_collapse(x, width = 8)
)

epoxy("{.dollar 12345678}")
#> $12.34M
epoxy("{.trunc8 12345678}")
#> 12345...
```

Note that the `engine` argument can be used even with inline tranformations, e.g. to apply a change only for HTML you can use `engine = "html"`.

To unset the session defaults, you have two options:

1. Unset everything by passing `NULL` to `epoxy_transform_set()`:

   ```
   epoxy_transform_set(NULL)
   ```

2. Unset a single inline transformation by passing `rlang::zap()` to the named argument:

   ```
   epoxy_transform_set(.dollar = rlang::zap())
   ```

Or you can provide new values to overwrite the current settings. And as before, you can unset session defaults for a specific `engine`.

See Also

Other epoxy's glue transformers: epoxy_transform_html(), epoxy_transform_inline()

Examples

```
epoxy("{.strong {.and letters[1:3]}}")
epoxy("{.and {.strong letters[1:3]}}")

# If you used the development version of epoxy, the above is equivalent to:
epoxy("{letters[1:3]&}", .transformer = epoxy_transform("bold", "collapse"))
epoxy("{letters[1:3]&}", .transformer = epoxy_transform("collapse", "bold"))

# In an epoxy_html chunk...
epoxy_html("{{.strong {{.or letters[1:3]}} }}")

# Or in an epoxy_latex chunk...
epoxy_latex("<<.and <<.strong letters[1:3]>> >>")

# ---- Other Transformers ----

# Format numbers with an inline transformation
amount <- 123.4234234
epoxy("{.number amount}")
epoxy(
  "{.number amount}",
  .transformer = epoxy_transform_inline(
    .number = scales::label_number(accuracy = 0.01)
  )
)

# Apply _any_ function to all replacements
epoxy(
  "{amount} is the same as {amount}",
  .transformer = epoxy_transform_apply(round, digits = 0)
)

epoxy(
  "{amount} is the same as {amount}",
  .transformer = epoxy_transform(
    epoxy_transform_apply(~ .x * 100),
    epoxy_transform_apply(round, digits = 2),
    epoxy_transform_apply(~ paste0(.x, "%"))
  )
)
```

---

epoxy_transform_html *Concise syntax for expressions inside HTML elements*

Description

`epoxy_transform_html()` provides a pug-like syntax for expressions in HTML that are wrapped in HTML elements.

Syntax:

You can specify the HTML element and its `id` and `class` into which the text of the expression will be placed. The template is to specify the element using the syntax below, followed by the R expression, separated by a space:

```
{{ [<element>][#<id> | .<class>...] expr }}
```

For example, to place the expression in a `<li>` element with `id = "food"` and `class = "fruit"`, you could write

```
{{ li#food.fruit fruit_name }}
```

Each item in the HTML template is optional:

1. If a specific HTML element is desired, the element name must be first. If no element is specified, the default as set by the `element` argument of `epoxy_transform_html()` will be used.
2. IDs are specified using `#<id>` and only one ID may be present
3. Classes are written using `.<class>` and as many classes as desired are allowed.

If the expression is a vector, the same element container will be used for each item in the vector.

Finally, if the expression returns HTML, it will be escaped by default. You can either use `htmltools::HTML()` to mark it as safe HTML in R, or you can write `!!expr` in the inline markup: `{{ li#food.fruit !!fruit_name }}`.

Usage

```
epoxy_transform_html(
  class = NULL,
  element = "span",
  collapse = TRUE,
  transformer = glue::identity_transformer
)
```

Arguments

`class`	`[character()]` Additional classes to be added to the inline HTML element.
`element`	`[character()` The default HTML element tag name to be used when an element isn't specified in the expression.
`collapse`	`[logical(1)]` If `TRUE`, transformed HTML outputs will be collapsed into a single character string. This is helpful when you're including the value of a vector within an outer HTML tag. Use `collapse = FALSE` to return a vector of HTML character strings instead, which follows what you'd typically expect from `glue::glue()`, i.e. when you want to repeat the outer wrapping text for each element of the vector.

`transformer` The transformer to apply to the replacement string. This argument is used for chaining the transformer functions. By providing a function to this argument you can apply an additional transformation after the current transformation. In nearly all cases, you can let `epoxy_transform()` handle this for you. The chain ends when `glue::identity_transformer()` is used as the `transformer`.

Value

A function of `text` and `envir` suitable for the `.transformer` argument of `glue::glue()`.

See Also

Used by default in `epoxy_html()`

Other epoxy's glue transformers: `epoxy_transform()`, `epoxy_transform_inline()`

Examples

```
epoxy_html("<ul>{{ li letters[1:3] }}</ul>")
epoxy_html("<ul>{{ li.alpha letters[1:3] }}</ul>")
epoxy_html("<ul>{{ li#my-letter letters[7] }}</ul>")

# The default element is used if no element is directly requested
epoxy_html("My name starts with {{ .name-letter letters[7] }}")

epoxy_html(
 "{{ h3#title title }}",
 title = "Epoxy for HTML"
)

# If your replacement text contains HTML, it's escaped by default.
hello <- "<strong>Hi there!</strong>"
epoxy_html("{{ hello }}")

# You can use !! inline to mark the text as safe HTML...
epoxy_html("{{ !!hello }}")
epoxy_html("{{ button !!hello }}")

# ...or you can use htmltools::HTML() to mark it as safe HTML in R.
hello <- htmltools::HTML("<strong>Hi there!</strong>")
epoxy_html("{{ hello }}")
```

`epoxy_transform_inline` *Epoxy Inline Transformer*

Description

This epoxy transformer is heavily inspired by the inline formatters in the cli package. The syntax is quite similar, but **epoxy**'s syntax is slightly different to accommodate reporting use cases.

To transform a template expression inline, you include a keyword, prefixed with a dot (`.`) that is used to format the value of the template expression in place.

```
epoxy("It cost {.dollar 123456}.", .transformer = "inline")
#> It cost $123,456.
```

The formatters, e.g. `.dollar` in the example above, can be customized using the arguments of `epoxy_transform_inline()`. Pass a customized `scales::label_dollar()` to `.dollar` to achieve a different transformation.

```
dollars_nzd <- scales::label_dollar(suffix = " NZD")

epoxy(
  "It cost {.dollar 123456}.",
  .transformer = epoxy_transform_inline(.dollar = dollars_nzd)
)
#> It cost $123,456 NZD.
```

Note that, unlike inline markup with cli, the text within the template expression, other than the keyword, is treated as an R expression.

```
money <- 123456
epoxy("It cost {.dollar money}.", .transformer = "inline")
#> It cost $123,456.
```

You can also nest inline markup expressions.

```
money <- c(123.456, 234.567)
epoxy("It will cost either {.or {.dollar money}}.", .transformer = "inline")
#> It will cost either $123.46 or $234.57.
```

Finally, you can provide your own functions that are applied to the evaluated expression. In this example, I add a `.runif` inline formatter that generates `n` random numbers (taken from the template expression) and sorts them.

```
set.seed(4242)

epoxy(
  "Here are three random percentages: {.and {.pct {.runif 3}}}.",
  .transformer = epoxy_transform_inline(
    .runif = function(n) sort(runif(n))
  )
)
#> Here are three random percentages: 23%, 35%, and 99%.
```

Usage

```
epoxy_transform_inline(
  ...,
  transformer = glue::identity_transformer,
  .and = and::and,
  .or = and::or,
  .incr = sort,
  .decr = function(x) sort(x, decreasing = TRUE),
  .bytes = scales::label_bytes(),
  .date = function(x) format(x, format = "%F"),
  .time = function(x) format(x, format = "%T"),
  .datetime = function(x) format(x, format = "%F %T"),
  .dollar = scales::label_dollar(prefix = engine_pick("$", "$", "\\$")),
  .number = scales::label_number(),
  .comma = scales::label_comma(),
  .ordinal = scales::label_ordinal(),
  .percent = scales::label_percent(suffix = engine_pick("%", "%", "\\%")),
  .pvalue = scales::label_pvalue(),
  .scientific = scales::label_scientific(),
  .uppercase = toupper,
  .lowercase = tolower,
  .titlecase = function(x) tools::toTitleCase(as.character(x)),
  .sentence = function(x) `substr<-`(x, 1, 1, toupper(substr(x, 1, 1))),
  .squote = function(x) sQuote(x, q = getOption("epoxy.fancy_quotes", FALSE)),
  .dquote = function(x) dQuote(x, q = getOption("epoxy.fancy_quotes", FALSE)),
  .strong = NULL,
  .emph = NULL,
  .code = NULL
)
```

Arguments

`...`	Additional named inline transformers as functions taking at least one argument. The evaluated expression from the template expression is passed as the first argument to the function. The argument names must include the leading `.` to match the syntax used inline.
`transformer`	The transformer to apply to the replacement string. This argument is used for chaining the transformer functions. By providing a function to this argument you can apply an additional transformation after the current transformation. In nearly all cases, you can let `epoxy_transform()` handle this for you. The chain ends when `glue::identity_transformer()` is used as the `transformer`.
`.and`	The function to apply to `x` when the template is `{.and x}`. Default is `and::and()`.
`.or`	The function to apply to `x` when the template is `{.or x}`. Default is `and::or()`.
`.incr`	The function to apply to `x` when the template is `{.incr x}`. Default is `sort()`.
`.decr`	The function to apply to `x` when the template is `{.decr x}`. Default is `function(x) sort(x, decreasing = TRUE)`.
`.bytes`	The function to apply to `x` when the template is `{.bytes x}`. Default is `scales::label_bytes()`.

`.date` The function to apply to `x` when the template is `{.date x}`. Default is `function(x) format(x, format = "%F")`.

`.time` The function to apply to `x` when the template is `{.time x}`. Default is `function(x) format(x, format = "%T")`.

`.datetime` The function to apply to `x` when the template is `{.datetime x}` or `{.dttm x}`. Default is `function(x) format(x, format = "%F %T")`.

`.dollar` The function to apply to `x` when the template is `{.dollar x}`. Default is `scales::label_dollar(prefix = engine_pick("$", "$", "\\$"))`.

`.number` The function to apply to `x` when the template is `{.number x}` or `{.num x}`. Default is `scales::label_number()`.

`.comma` The function to apply to `x` when the template is `{.comma x}`. Default is `scales::label_comma()`.

`.ordinal` The function to apply to `x` when the template is `{.ordinal x}`. Default is `scales::label_ordinal()`.

`.percent` The function to apply to `x` when the template is `{.percent x}` or `{.pct x}`. Default is `scales::label_percent(suffix = engine_pick("%", "%", "\\%"))`.

`.pvalue` The function to apply to `x` when the template is `{.pvalue x}`. Default is `scales::label_pvalue()`.

`.scientific` The function to apply to `x` when the template is `{.scientific x}`. Default is `scales::label_scientific()`.

`.uppercase` The function to apply to `x` when the template is `{.uppercase x}` or `{.uc x}`. Default is `toupper()`.

`.lowercase` The function to apply to `x` when the template is `{.lowercase x}` or `{.lc x}`. Default is `tolower()`.

`.titlecase` The function to apply to `x` when the template is `{.titlecase x}` or `{.tc x}`. Default is `function(x) tools::toTitleCase(as.character(x))`.

`.sentence` The function to apply to `x` when the template is `{.sentence x}` or `{.sc x}`. Default is `function(x) `substr<-`(x, 1, 1, toupper(substr(x, 1, 1)))`.

`.squote` The function to apply to `x` when the template is `{.squote x}`. Default is `function(x) sQuote(x, q = getOption("epoxy.fancy_quotes", FALSE))`.

`.dquote` The function to apply to `x` when the template is `{.dquote x}`. Default is `function(x) dQuote(x, q = getOption("epoxy.fancy_quotes", FALSE))`.

`.strong` The function to apply to `x` when the template is `{.strong x}` or `{.bold x}`. Default is chosen internally based on the output format.

`.emph` The function to apply to `x` when the template is `{.emph x}` or `{.italic x}`. Default is chosen internally based on the output format.

`.code` The function to apply to `x` when the template is `{.code x}`. Default is chosen internally based on the output format.

Value

A function of `text` and `envir` suitable for the `.transformer` argument of `glue::glue()`.

See Also

epoxy_transform(), epoxy_transform_set()

Other epoxy's glue transformers: epoxy_transform(), epoxy_transform_html()

Examples

```
revenue <- 0.2123
sales <- 42000.134

# ---- Basic Example with Inline Formatting --------------------------------
epoxy(
  '{.pct revenue} of revenue generates {.dollar sales} in profits.'
)

# With standard {glue} (`epoxy_transform_inline()` is a glue transformer)
glue::glue(
  '{.pct revenue} of revenue generates {.dollar sales} in profits.',
  .transformer = epoxy_transform_inline()
)

# ---- Setting Inline Formatting Options -----------------------------------
# To set inline format options, provide `scales::label_*()` to the supporting
# epoxy_transform_inline arguments.
epoxy(
  '{.pct revenue} of revenue generates {.dollar sales} in profits.',
  .transformer = epoxy_transform_inline(
    .percent = scales::label_percent(accuracy = 0.1),
    .dollar = scales::label_dollar(accuracy = 10)
  )
)

glue::glue(
  '{.pct revenue} of revenue generates {.dollar sales} in profits.',
  .transformer = epoxy_transform_inline(
    .percent = scales::label_percent(accuracy = 0.1),
    .dollar = scales::label_dollar(accuracy = 10)
  )
)

# ---- Custom Inline Formatting ----------------------------------------------
# Add your own formatting functions
search <- "why are cats scared of cucumbers"

epoxy_html(
  "https://example.com?q={{ .url_encode search }}>",
  .transformer = epoxy_transform_inline(
    .url_encode = utils::URLencode
  )
)
```

epoxy_transform_one_shot *One-shot epoxy transformers*

Description

These transformers are useful for applying the same transformation to every replacement in the template.

Usage

```
epoxy_transform_wrap(
  before = "**",
  after = before,
  engine = NULL,
  transformer = glue::identity_transformer,
  syntax = lifecycle::deprecated()
)

epoxy_transform_bold(engine = NULL, transformer = glue::identity_transformer)

epoxy_transform_italic(engine = NULL, transformer = glue::identity_transformer)

epoxy_transform_apply(
  .f = identity,
  ...,
  transformer = glue::identity_transformer
)

epoxy_transform_code(engine = NULL, transformer = glue::identity_transformer)

epoxy_transform_collapse(
  sep = ", ",
  last = sep,
  language = NULL,
  ...,
  transformer = glue::identity_transformer
)
```

Arguments

before, after	In `epoxy_transform_wrap()`, the characters to be added before and after variables in the template string.
engine	One of `"markdown"` (or `"md"`), `"html"`, or `"latex"`. The default is chosen based on the engine of the chunk where the transform function is called, or according to the option `epoxy.engine`. Caution: invalid options are silently ignored, falling back to `"markdown"`.

transformer	The transformer to apply to the replacement string. This argument is used for chaining the transformer functions. By providing a function to this argument you can apply an additional transformation after the current transformation. In nearly all cases, you can let epoxy_transform() handle this for you. The chain ends when glue::identity_transformer() is used as the transformer.
syntax	**[Deprecated]** Use engine instead.
.f	A function, function name or purrr::map()-style inline function.
...	Transformer functions, e.g. epoxy_transform_bold or the name of an **epoxy** transform function, e.g. "bold", or a call to a transform function, e.g. epoxy_transform_bold(). epoxy_transform() chains the transformer functions together, applying the transformers in order from first to last. For example, epoxy_transform("bold", "collapse") results in replaced strings that are emboldened *and then* collapsed, e.g. **a** and **b**. On the other hand, epoxy_transform("collapse", "bold") will collapse the vector *and then* embolden the entire string. In epoxy_transform_apply(), the ... are passed to the underlying call the underlying function call. In epoxy_transform_collapse(), the ... are ignored.
sep, last	The separator to use when joining the vector elements when the expression ends with a *. Elements are separated by sep, except for the last two elements, which use last.
language	In epoxy_transform_collapse(), language is passed to and::and() or and::or() to choose the correct and/or phrase and spacing for the language. By default, will follow the system language. See and::and_languages for supported languages.

Value

A function of text and envir suitable for the .transformer argument of glue::glue().

Functions

- epoxy_transform_wrap(): Wrap variables with text added before or after the inline expression.
- epoxy_transform_bold(): Embolden variables using ** in markdown, in HTML, or \textbf{} in LaTeX.
- epoxy_transform_italic(): Italicize variables using _ in markdown, in HTML, or \emph{} in LaTeX.
- epoxy_transform_apply(): Apply a function to all replacement expressions.
- epoxy_transform_code(): Code format variables using `` in markdown, <code> in HTML, or \texttt{} in LaTeX.
- epoxy_transform_collapse(): Collapse vector variables with a succinct syntax (but see epoxy_transform_inline() for a more readable option).

Examples

```
abc <- c("a", "b", "c")

epoxy("{abc}", .transformer = epoxy_transform_wrap("'"))

epoxy("{abc}", .transformer = epoxy_transform_bold())

epoxy("{abc}", .transformer = epoxy_transform_italic())

epoxy("{abc}", .transformer = epoxy_transform_code())

epoxy("{abc}", .transformer = epoxy_transform_apply(toupper))
```

---

epoxy_use *Reuse a Template Chunk*

Description

Reuse a template from another chunk or file. By calling `epoxy_use_chunk()` in an R chunk or inline R expression, you can reuse a template defined in another chunk in your document.

Alternatively, you can store the template in a separate file and use `epoxy_use_file()` to reuse it. When stored in a file, the template file can contain YAML front matter (following the same rules as pandoc documents) with options that should be applied when calling an epoxy function. The specific function called by `epoxy_use_file()` can be set via the `engine` option in the YAML front matter; the default is `epoxy()`.

Usage

```
epoxy_use_chunk(.data = NULL, label, ...)

epoxy_use_file(.data = NULL, file, ...)
```

Arguments

`.data`	A data set
`label`	The chunk label, i.e. the human-readable name, of the chunk containing the template string. This chunk should be an `epoxy`, `epoxy_html` or other epoxy-provided chunk type and it must have a label. `epoxy_use_chunk()` will apply the options from this chunk to the template, giving preference to arguments in `epoxy_use_chunk()` or the chunk options where it is called. See the "Template Options" section for more details.
`...`	Arguments passed on to `epoxy`
	`.transformer` A transformer function or transformer chain created with `epoxy_transform()`. Alternatively, a character vector of epoxy transformer names, e.g. `c("bold", "collapse")` or a list of epoxy transformers, e.g. `list(epoxy_transform_bold(), epoxy_transform_collapse())`.

In **epoxy**, you'll most likely want to use the defaults or consult `epoxy_transform()` for more information. See also `glue::glue()` for more information on transformers.

`.style` **[Deprecated]** Please use `.transformer` instead.

`.open` [`character(1)`: '\{']
The opening delimiter around the template variable or expression. Doubling the full delimiter escapes it.

`.close` [`character(1)`: '\}']
The closing delimiter around the template variable or expression. Doubling the full delimiter escapes it.

`.collapse` A character string used to collapse a vector result into a single value. If `NULL` (the default), the result is not collapsed.

`.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()`.

`.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.

`.literal` [`boolean(1)`: 'FALSE']
Whether to treat single or double quotes, backticks, and comments as regular characters (vs. as syntactic elements), when parsing the expression string. Setting `.literal = TRUE` probably only makes sense in combination with a custom `.transformer`, as is the case with `glue_col()`. Regard this argument (especially, its name) as experimental.

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

`file` The template file, i.e. a plain text file, containing the template. An `.md` or `.txt` file extension is recommended. In addition to the template, the file may also contain YAML front matter containing options that are used when rendering the template via `epoxy()`.

Value

A character string of the rendered template based on the `label` chunk. The results are marked as `"asis"` output so that they are treated as regular text rather than being displayed as code results.

Use in R Markdown or Quarto

````
```{epoxy movie-release}
{.emph title} was released in {year}.
```

```{r}
# Re-using the template we defined above
epoxy_use_chunk(bechdel[1, ], "movie-release")
```

```{r}
# Using in a dplyr pipeline
bechdel |>
  dplyr::filter(year == 1989) |>
  epoxy_use_chunk("movie-release")
```
````

Or you can even use it inline:

```
It's hard to believe that
`r epoxy_use_chunk(bechdel[2, ], "movie-release")`.
```

It's hard to believe that *Back to the Future Part II* was released in 1989.

The same template could also be stored in a file, e.g. `movie-release.md`:

```
---
engine: epoxy
---

{.emph title} was released in {year}.
```

The YAML front matter is used in template files to set options for the template. You can use the `engine` option to choose the epoxy function to be applied to the template, e.g. `engine: epoxy_html` or `engine: epoxy_latex`. By default, `engine: epoxy` is assumed unless otherwise specified.

Template Options

When rendering a template, `epoxy_use_chunk()` and `epoxy_use_file()` will inherit the options set in a number of different ways. The final template options are determined in the following order, ranked by importance. Options set in a higher-ranked location will override options set in a lower-ranked location.

1. The arguments passed to `epoxy_use_chunk()`, such as `.data` or any arguments passed in the `...`. These options always have preference over options set anywhere else.
2. The chunk options from the chunk where `epoxy_use_chunk()` or `epoxy_use_file()` is called.
3. The chunk options from the template chunk or file. These options typically are relevant to the template itself, such as the engine used or the opening and closing delimiters.

4. Global knitr chunk options for the document. You can set these with `knitr::opts_chunk$set()`, see `?knitr::opts_chunk` for more information.

`render_epoxy` *Render Epoxy Output*

Description

Server-side render function used to provide values for template items. Use named values matching the template variable names in the associated `ui_epoxy_html()` or `ui_epoxy_mustache()`. When the values are updated by the app, `render_epoxy()` will update the values shown in the app's UI.

Usage

```
render_epoxy(
  ...,
  .list = NULL,
  env = parent.frame(),
  outputFunc = ui_epoxy_html,
  outputArgs = list()
)

renderEpoxyHTML(..., env = parent.frame())
```

Arguments

`...`	Named values corresponding to the template variables created with the associated `ui_epoxy_html()` UI element.
`.list`	A named list or a `shiny::reactiveValues()` list with names corresponding to the template variables created with the associated `ui_epoxy_html()` UI element.
`env`	The environment in which to evaluate the `...`
`outputFunc`	Either `ui_epoxy_html()` or `ui_epoxy_mustache()`, i.e. the UI function to be paired with this output. This is only used when calling `render_epoxy()` in an Shiny runtime R Markdown document and when you are only providing the output without an explicit, corresponding UI element.
`outputArgs`	A list of arguments to be passed through to the implicit call to `ui_epoxy_html()` when `render_epoxy` is used in an interactive R Markdown document.

Value

A server-side Shiny render function that should be assigned to Shiny's `output` object and named to match the `.id` of the corresponding `ui_epoxy_html()` call.

Functions

- `renderEpoxyHTML()`: **[Deprecated]** Deprecated alias, please use `render_epoxy()`.

See Also

ui_epoxy_html(), ui_epoxy_mustache()

Examples

```
# This small app shows the current time using `ui_epoxy_html()`
# to provide the HTML template and `render_epoxy()` to
# update the current time every second.

ui <- shiny::fluidPage(
 shiny::h2("Current Time"),
 ui_epoxy_html(
  "time",
  shiny::p("The current time is {{strong time}}.")
 )
)

server <- function(input, output, session) {
 current_time <- shiny::reactive({
  shiny::invalidateLater(1000)
  strftime(Sys.time(), "%F %T")
 })

 output$time <- render_epoxy(time = current_time())
}

if (rlang::is_interactive()) {
 shiny::shinyApp(ui, server)
}


run_epoxy_example_app("render_epoxy")
```

---

run_epoxy_example_app *Example epoxy Shiny apps*

Description

Run an example epoxy Shiny app showcasing the Shiny UI and server components provided by epoxy.

Usage

```
run_epoxy_example_app(
 name = c("ui_epoxy_html", "ui_epoxy_markdown", "ui_epoxy_mustache", "render_epoxy"),
  display.mode = "showcase",
  ...
)
```

Arguments

`name` Name of the example, currently one of `"ui_epoxy_html"`, `"ui_epoxy_markdown"`, `"ui_epoxy_mustache"`, or `"render_epoxy"`.

`display.mode` The mode in which to display the application. If set to the value `"showcase"`, shows application code and metadata from a `DESCRIPTION` file in the application directory alongside the application. If set to `"normal"`, displays the application normally. Defaults to `"auto"`, which displays the application in the mode given in its `DESCRIPTION` file, if any.

`...` Arguments passed on to `shiny::runApp`

`appDir` The application to run. Should be one of the following:

- A directory containing `server.R`, plus, either `ui.R` or a `www` directory that contains the file `index.html`.
- A directory containing `app.R`.
- An `.R` file containing a Shiny application, ending with an expression that produces a Shiny app object.
- A list with `ui` and `server` components.
- A Shiny app object created by `shinyApp()`.

`port` The TCP port that the application should listen on. If the `port` is not specified, and the `shiny.port` option is set (with `options(shiny.port = XX)`), then that port will be used. Otherwise, use a random port between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried.

`launch.browser` If true, the system's default web browser will be launched automatically after the app is started. Defaults to true in interactive sessions only. The value of this parameter can also be a function to call with the application's URL.

`host` The IPv4 address that the application should listen on. Defaults to the `shiny.host` option, if set, or `"127.0.0.1"` if not. See Details.

`workerId` Can generally be ignored. Exists to help some editions of Shiny Server Pro route requests to the correct process.

`quiet` Should Shiny status messages be shown? Defaults to FALSE.

`test.mode` Should the application be launched in test mode? This is only used for recording or running automated tests. Defaults to the `shiny.testmode` option, or FALSE if the option is not set.

Value

Runs the Shiny example app interactively. Nothing is returned.

See Also

`ui_epoxy_html()`, `ui_epoxy_markdown()`, `ui_epoxy_mustache()`, `render_epoxy()`

Examples

```
# List examples by passing `name = NULL`
run_epoxy_example_app(name = NULL)
```

`ui_epoxy_html` *Epoxy HTML Output for Shiny*

Description

A glue-like output for Shiny. `ui_epoxy_html()` lets you use placeholders in your HTML such as `"{{first_name}}"`, that are provided values from the server by giving `render_epoxy()` a `first_name` value. Unlike `ui_epoxy_mustache()`, updates are highly targeted: only the regions where the server-side data have changed are updated in `ui_epoxy_html()`.

Usage

```
ui_epoxy_html(
  .id,
  ...,
  .class = NULL,
  .style = NULL,
  .item_tag = "span",
  .item_class = NULL,
  .placeholder = "",
  .sep = "",
  .open = "{{",
  .close = "}}",
  .na = "",
  .null = "",
  .literal = FALSE,
  .trim = FALSE,
  .aria_live = c("polite", "off", "assertive"),
  .aria_atomic = TRUE,
  .class_item = deprecated(),
  .container = deprecated(),
  .container_item = deprecated()
)

epoxyHTML(.id, ...)
```

Arguments

`.id`	The output id
`...`	UI elements or text (that will be treated as HTML), containing template variables. Use named values to provide initial placeholder values.

`.class`, `.style` Classes and inline style directives added to the `<epoxy-html>` container into which the elements in `...` are placed.

`.item_tag`, `.item_class`
The HTML element tag name and classes used to wrap each template variable. By default, each template is wrapped in a `<span>`.

`.placeholder` Default placeholder if a template variable placeholder isn't provided.

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

`.open` [`character(1)`: '\{']
The opening delimiter around the template variable or expression. Doubling the full delimiter escapes it.

`.close` [`character(1)`: '\}']
The closing delimiter around the template variable or expression. 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`.

`.literal` [`boolean(1)`: 'FALSE']
Whether to treat single or double quotes, backticks, and comments as regular characters (vs. as syntactic elements), when parsing the expression string. Setting `.literal = TRUE` probably only makes sense in combination with a custom `.transformer`, as is the case with `glue_col()`. Regard this argument (especially, its name) as experimental.

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

`.aria_live`, `.aria_atomic`
The aria-live and aria-atomic attribute values for the entire template region. By default, with `"polite"`, any updates within the region will be announced via screen readers.

If your template includes changes in lots of disparate areas, it would be better to set `"aria-live" = "polite"` and `"aria-atomic" = "true"` on specific regions that should be announced together. Otherwise, the default is to announce the entire region within the `ui_epoxy_html()` whenever any of the values within change. In other words, set `.aria_live = "off"` and `.aria_atomic = NULL` on the `ui_epoxy_html()` parent item and then set `"aria-live" = "polite"` and `"aria-atomic" = "true"` on the parent containers of each region in the app that receives updates. `ui_epoxy_html()` does targeted updates, changing only the parts of the UI that have changed.

`.class_item` **[Deprecated]** Deprecated in **epoxy** v1.0.0, please use `.item_class` instead.

`.container` **[Deprecated]** Deprecated in **epoxy** v1.0.0, where the container is now always `<epoxy-html>`.

`.container_item`
: **[Deprecated]** Deprecated in **epoxy** v1.0.0, please use `.item_tag` instead.

Value

An HTML object.

Functions

- `epoxyHTML()`: **[Deprecated]** Deprecated alias, please use `ui_epoxy_html()`.

HTML Markup

By default, placeholders are inserted into a `<span>` element in your UI, with the classes specified in `.class_item`.

`ui_epoxy_html()` also supports an HTML markup syntax similar to pug (an HTML preprocessor). As an example, the markup syntax

```
"{{h3.example.basic#basic-three demo}}"
```

creates a `demo` placeholder inside the following tag.

```
<h3 id="basic-three" class="example basic"></h3>
```

The placeholder template string follows the pattern `{{<markup> <name>}}`. The markup syntax comes first, separated from the placeholder name by a space. The HTML element is first, followed by classes prefixed with `.` or and ID prefixed with `#`. The template markup can contain only one element and one ID, but many classes can be specified.

By default, the placeholder is assumed to be text content and any HTML in the sent to the placeholder will be escaped — in other words if you sent `"<strong>word</strong>"`, you'd see that exact literal text in your app, rather than an emboldened **word**. To mark a placeholder as safe to accept HTML, use `!!` before the placeholder, e.g. `{{<markup> !!<name>}}`. So `{{h3 !!demo}}` will create an `<h3>` tag that accepts HTML within it.

See Also

`ui_epoxy_mustache()`, `render_epoxy()`

Examples

```
library(shiny)

ui <- fluidPage(
  h2("ui_epoxy_html demo"),
  ui_epoxy_html(
    .id = "example",
    .item_class = "inner",
    fluidRow(
      tags$div(
        class = "col-xs-4",
```

```
        selectInput(
          inputId = "thing",
          label = "What is this {{color}} thing?",
          choices = c("apple", "banana", "coconut", "dolphin")
        )
      ),
      tags$div(
        class = "col-xs-4",
        selectInput(
          inputId = "color",
          label = "What color is the {{thing}}?",
          c("red", "blue", "black", "green", "yellow")
        )
      ),
      tags$div(
        class = "col-xs-4",
        sliderInput(
          inputId = "height",
          label = "How tall is the {{color}} {{thing}}?",
          value = 5,
          min = 0,
          max = 10,
          step = 0.1,
          post = "ft"
        )
      )
    ),
    tags$p(class = "big", "The {{color}} {{thing}} is {{height}} feet tall."),
    # Default values for placeholders above.
    thing = "THING",
    color = "COLOR",
    height = "HEIGHT"
  ),
  tags$style(HTML(
    ".big { font-size: 1.5em; }
     .inner { background-color: rgba(254, 233, 105, 0.5);}
     .epoxy-item__placeholder { color: #999999; background-color: unset; }"
  ))
)

server <- function(input, output, session) {
  output$example <- render_epoxy(
    thing = input$thing,
    color = input$color,
    height = input$height
  )
}

if (interactive()) {
  shinyApp(ui, server)
}
```

```
run_epoxy_example_app("ui_epoxy_html")
```

`ui_epoxy_markdown` *Epoxy Markdown Template for Shiny*

Description

Create reactive HTML from a Markdown template. `ui_epoxy_markdown()` uses the same template syntax as `ui_epoxy_html()`, but rather than requiring HTML inputs, you can write in markdown. The template is first rendered from markdown to HTML using `pandoc::pandoc_convert()` (if **pandoc** is available) or `commonmark::markdown_html()` otherwise.

Usage

```
ui_epoxy_markdown(
  .id,
  ...,
  .markdown_fn = NULL,
  .markdown_args = list(),
  .class = NULL,
  .style = NULL,
  .item_tag = "span",
  .item_class = NULL,
  .placeholder = "",
  .sep = "",
  .open = "{{",
  .close = "}}",
  .na = "",
  .null = "",
  .literal = FALSE,
  .trim = FALSE,
  .aria_live = c("polite", "off", "assertive"),
  .aria_atomic = TRUE,
  .class_item = deprecated(),
  .container = deprecated(),
  .container_item = deprecated()
)
```

Arguments

`.id`	The output id
`...`	Unnamed arguments are treated as lines of markdown text, and named arguments are treated as initial values for templated variables.
`.markdown_fn`	The function used to convert the markdown to HTML. This function is passed the markdown text as a character vector for the first argument and any additional arguments from the list `.markdown_args`. By default, we use `pandoc::pandoc_convert()` if **pandoc** is available, otherwise we use `commonmark::markdown_html()`.

`.markdown_args` A list of arguments to pass to `commonmark::markdown_html()`.

`.class`, `.style` Classes and inline style directives added to the `<epoxy-html>` container into which the elements in `...` are placed.

`.item_tag`, `.item_class`
The HTML element tag name and classes used to wrap each template variable. By default, each template is wrapped in a `<span>`.

`.placeholder` Default placeholder if a template variable placeholder isn't provided.

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

`.open` [`character(1)`: '\{']
The opening delimiter around the template variable or expression. Doubling the full delimiter escapes it.

`.close` [`character(1)`: '\}']
The closing delimiter around the template variable or expression. 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`.

`.literal` [`boolean(1)`: 'FALSE']
Whether to treat single or double quotes, backticks, and comments as regular characters (vs. as syntactic elements), when parsing the expression string. Setting `.literal = TRUE` probably only makes sense in combination with a custom `.transformer`, as is the case with `glue_col()`. Regard this argument (especially, its name) as experimental.

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

`.aria_live`, `.aria_atomic`
The aria-live and aria-atomic attribute values for the entire template region. By default, with `"polite"`, any updates within the region will be announced via screen readers.

If your template includes changes in lots of disparate areas, it would be better to set `"aria-live" = "polite"` and `"aria-atomic" = "true"` on specific regions that should be announced together. Otherwise, the default is to announce the entire region within the `ui_epoxy_html()` whenever any of the values within change. In other words, set `.aria_live = "off"` and `.aria_atomic = NULL` on the `ui_epoxy_html()` parent item and then set `"aria-live" = "polite"` and `"aria-atomic" = "true"` on the parent containers of each region in the app that receives updates. `ui_epoxy_html()` does targeted updates, changing only the parts of the UI that have changed.

`.class_item` **[Deprecated]** Deprecated in **epoxy** v1.0.0, please use `.item_class` instead.

`.container` **[Deprecated]** Deprecated in **epoxy** v1.0.0, where the container is now always `<epoxy-html>`.

`.container_item` **[Deprecated]** Deprecated in **epoxy** v1.0.0, please use `.item_tag` instead.

Value

An HTML object.

See Also

`ui_epoxy_html()`, `ui_epoxy_mustache()`, `render_epoxy()`

Examples

```
library(shiny)

# Shiny epoxy template functions don't support inline transformations,
# so we still have to do some prep work ourselves.
bechdel <- epoxy::bechdel

as_dollars <- scales::label_dollar(
  scale_cut = scales::cut_short_scale()
)
bechdel$budget <- as_dollars(bechdel$budget)
bechdel$domgross <- as_dollars(bechdel$domgross)

vowels <- c("a", "e", "i", "o", "u")
bechdel$genre  <- paste(
  ifelse(substr(tolower(bechdel$genre), 1, 1) %in% vowels, "an", "a"),
  tolower(bechdel$genre)
)

movie_ids <- rlang::set_names(
  bechdel$imdb_id,
  bechdel$title
)

ui <- fixedPage(
  fluidRow(
    column(
      width = 3,
      selectInput("movie", "Movie", movie_ids),
      uiOutput("poster")
    ),
    column(
      width = 9,
      ui_epoxy_markdown(
        .id = "about_movie",
        "
## {{title}}
```

```
**Released:** {{ year }} \\
**Rated:** {{ rated }} \\
**IMDB Rating:** {{ imdb_rating }}

_{{ title }}_ is {{ genre }} film released in {{ year }}.
It was filmed in {{ country }} with a budget of {{ budget }}
and made {{ domgross }} at the box office.
_{{ title }}_ recieved a Bechdel rating of **{{ bechdel_rating }}**
for the following plot:

> {{ plot }}
"
      )
    )
  )
)

server <- function(input, output, session) {
  movie <- reactive({
    bechdel[bechdel$imdb_id == input$movie, ]
  })

  output$about_movie <- render_epoxy(.list = movie())
  output$poster <- renderUI(
    img(
      src = movie()$poster,
      alt = paste0("Poster for ", movie()$title),
      style = "max-height: 400px; max-width: 100%; margin: 0 auto; display: block;"
    )
  )
}

if (interactive()) {
  shinyApp(ui, server)
}


run_epoxy_example_app("ui_epoxy_markdown")
```

`ui_epoxy_mustache` *Epoxy HTML Mustache Template*

Description

A Shiny output that uses mustache templating to render HTML. Mustache is a powerful template language with minimal internal logic. The advantage of `ui_epoxy_mustache()` is that all parts of the HTML can be templated – including element attributes – whereas `ui_epoxy_html()` requires that the dynamic template variables appear in the text portion of the UI. The downside is that the entire template is re-rendered (in the browser), each time that updated data is sent from the server – unlike `ui_epoxy_html()`, whose updates are specific to the parts of the data that have changed.

Usage

```
ui_epoxy_mustache(
  id,
  ...,
  .file = NULL,
  .sep = "",
  .container = "epoxy-mustache"
)

ui_epoxy_whisker(
  id,
  ...,
  .file = NULL,
  .sep = "",
  .container = "epoxy-mustache"
)
```

Arguments

id	The ID of the output.
...	Character strings of HTML or htmltools::tags. All elements should be unnamed.
.file	A path to a template file. If provided, no other template lines should be included in ....
.sep	The separator used to concatenate elements in ....
.container	A character tag name, e.g. "div" or "span", or a function that returns an htmltools::tag().

Value

Returns a Shiny output UI element.

Functions

- ui_epoxy_whisker(): An alias for ui_epoxy_mustache(), provided because R users are more familiar with this syntax via the **whisker** package.

See Also

ui_epoxy_html(), render_epoxy()

Other Mustache-style template functions: epoxy_mustache()

Examples

```
library(shiny)

ui <- fluidPage(
  fluidRow(
    style = "max-width: 600px; margin: 0 auto",
```

```
    column(
      width = 6,
      ui_epoxy_mustache(
        id = "template",
        h2(class = "{{heading_class}}", "Hello, {{name}}!"),
        "{{#favorites}}",
        p("Your favorite fruits are..."),
        tags$ul(HTML("{{#fruits}}<li>{{.}}</li>{{/fruits}}")),
        "{{/favorites}}",
        "{{^favorites}}<p>Do you have any favorite fruits?</p>{{/favorites}}"
      )
    ),
    column(
      width = 6,
      h2("Inputs"),
      textInput("name", "Your name"),
      textInput("fruits", "Favorite fruits", placeholder = "apple, banana"),
      helpText("Enter a comma-separated list of fruits.")
    )
  )
)

server <- function(input, output, session) {
  user_name <- reactive({
    if (!nzchar(input$name)) return("user")
    input$name
  })

  favorites <- reactive({
    if (identical(input$fruits, "123456")) {
      # Errors are equivalent to "empty" values,
      # the rest of the template will still render.
      stop("Bad fruits, bad!")
    }

    if (!nzchar(input$fruits)) return(NULL)
    list(fruits = strsplit(input$fruits, "\\s*,\\s*")[[1]])
  })

  output$template <- render_epoxy(
    name = user_name(),
    heading_class = if (user_name() != "user") "text-success",
    favorites = favorites()
  )
}

if (interactive()) {
  shiny::shinyApp(ui, server)
}


run_epoxy_example_app("ui_epoxy_mustache")
```

`use_epoxy_knitr_engines` *Use the epoxy knitr engines*

Description

Sets **epoxy**'s **knitr** engines for use by **knitr** in R Markdown and other document formats powered by **knitr**. These engines are also set up when loading **epoxy** with `library()`, so in general you will not need to call this function explicitly.

epoxy provides four **knitr** engines:

- `epoxy` uses default **glue** syntax, e.g. `{var}` for markdown outputs
- `epoxy_html` uses double brace syntax, e.g. `{{var}}` for HTML outputs
- `epoxy_latex` uses double angle brackets syntax, e.g. `<<var>>` for LaTeX outputs
- `whisker` uses the **whisker** package which provides an R-based implementation of the mustache templating language.

For historical reasons, aliases for the HTML and LaTeX engines are also created: `glue_html` and `glue_latex`. You may opt into a third alias — `glue` for the `epoxy` engine — by calling `use_epoxy_glue_engine()`, but note that this will most likely overwrite the `glue` engine provided by the **glue** package.

Usage

```
use_epoxy_knitr_engines(
  use_glue_engine = "glue" %in% include,
  include = c("md", "html", "latex", "mustache")
)

use_epoxy_glue_engine()
```

Arguments

`use_glue_engine`	If `TRUE` (default `FALSE`), uses **epoxy**'s `glue` engine, most likely overwriting the `glue` engine provided by **glue**.
`include`	The epoxy knitr engines to include. Defaults to all engines except for the `glue` engine (which is just an alias for the `epoxy` engine).

Value

Silently sets **epoxy**'s knitr engines and invisible returns knitr::knit_engines as they were prior to the function call.

Functions

- `use_epoxy_glue_engine()`: Use **epoxy**'s `epoxy` engine as the `glue` engine.

See Also

epoxy(), epoxy_html(), epoxy_latex(), and epoxy_mustache() for the functions that power these knitr engines.

Examples

```
use_epoxy_knitr_engines()
```

Index

∗ **Mustache-style template functions**
epoxy_mustache, 7
ui_epoxy_mustache, 34
∗ **Templating functions**
epoxy_use, 21
∗ **datasets**
bechdel, 2
∗ **epoxy's glue transformers**
epoxy_transform, 9
epoxy_transform_html, 12
epoxy_transform_inline, 14

and::and(), *16*, *20*
and::and_languages, *20*
and::or(), *16*, *20*

bechdel, 2
bold, *11*

collapse, *11*
commonmark::markdown_html(), *31*, *32*

emptyenv(), *5*, *22*
engine_pick, 3
epoxy, 4, *21*
epoxy(), *3*, *10*, *11*, *21*, *22*, *38*
epoxy_html *(*epoxy*)*, 4
epoxy_html(), *3*, *14*, *38*
epoxy_latex *(*epoxy*)*, 4
epoxy_latex(), *3*, *38*
epoxy_mustache, 7
epoxy_mustache(), *6*, *35*, *38*
epoxy_transform, 9
epoxy_transform(), *6*, *11*, *14*, *18*, *21*, *22*
epoxy_transform_apply *(*epoxy_transform_one_shot*)*, 19
epoxy_transform_bold, *10*, *20*
epoxy_transform_bold *(*epoxy_transform_one_shot*)*, 19
epoxy_transform_bold(), *10*, *20*
epoxy_transform_code *(*epoxy_transform_one_shot*)*, 19
epoxy_transform_collapse *(*epoxy_transform_one_shot*)*, 19
epoxy_transform_get *(*epoxy_transform*)*, 9
epoxy_transform_html, 12
epoxy_transform_html(), *12*, *13*, *18*
epoxy_transform_inline, 14
epoxy_transform_inline(), *4*, *10*, *12*, *14*, *20*
epoxy_transform_italic *(*epoxy_transform_one_shot*)*, 19
epoxy_transform_one_shot, 19
epoxy_transform_set *(*epoxy_transform*)*, 9
epoxy_transform_set(), *18*
epoxy_transform_wrap *(*epoxy_transform_one_shot*)*, 19
epoxy_use, 21
epoxy_use_chunk *(*epoxy_use*)*, 21
epoxy_use_file *(*epoxy_use*)*, 21
epoxyHTML *(*ui_epoxy_html*)*, 27

glue::glue(), *4*, *6*, *10*, *14*, *17*, *20*, *22*
glue::identity_transformer(), *14*, *16*, *20*

htmltools::HTML(), *13*
htmltools::tag(), *35*
htmltools::tags, *35*

inline transformer, *11*

knitr::knit_engines, *37*

pandoc::pandoc_convert(), *31*
purrr::map(), *20*

render_epoxy, 24
render_epoxy(), *26*, *29*, *33*, *35*
renderEpoxyHTML *(*render_epoxy*)*, 24
run_epoxy_example_app, 25

scales::label_bytes(), *16*
scales::label_comma(), *17*
scales::label_dollar(), *15*
scales::label_number(), *17*
scales::label_ordinal(), *17*
scales::label_pvalue(), *17*
scales::label_scientific(), *17*
shiny::reactiveValues(), *24*
shiny::runApp, *26*
shinyApp(), *26*
sort(), *16*

tolower(), *17*
toupper(), *17*
trim(), *6*, *22*, *28*, *32*

ui_epoxy_html, 27
ui_epoxy_html(), *24–26*, *31*, *33–35*
ui_epoxy_markdown, 31
ui_epoxy_markdown(), *26*
ui_epoxy_mustache, 34
ui_epoxy_mustache(), *8*, *24–27*, *29*, *33*
ui_epoxy_whisker *(*ui_epoxy_mustache*)*, 34
use_epoxy_glue_engine
        *(*use_epoxy_knitr_engines*)*, 37
use_epoxy_knitr_engines, 37
use_epoxy_knitr_engines(), *6*

whisker::whisker.render(), *7, 8*