

# Package ‘vegawidget’

October 20, 2022

**Version** 0.4.2

**Title** 'Htmlwidget' for 'Vega' and 'Vega-Lite'

**Description** 'Vega' and 'Vega-Lite' parse text in 'JSON' notation to render chart-specifications into 'HTML'. This package is used to facilitate the rendering. It also provides a means to interact with signals, events, and datasets in a 'Vega' chart using 'JavaScript' or 'Shiny'.

**SystemRequirements** To use image functions for MacOS: X11

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**ByteCompile** true

**URL** <https://vegawidget.github.io/vegawidget/>

**BugReports** <https://github.com/vegawidget/vegawidget/issues>

**RoxygenNote** 7.2.1

**VignetteBuilder** knitr

**Depends** R (>= 2.10)

**Imports** jsonlite, htmlwidgets, assertthat, rlang, glue, magrittr, htmltools, digest, utils

**Suggests** spelling, knitr, rmarkdown, listviewer, testthat (>= 3.0.0), yaml, fs, usethis (>= 1.5.0), readr, tibble, lubridate, V8 (>= 4.0), withr, learnr, rsvg, dplyr, png, conflicted, here, shiny, purrr, rsconnect

**Language** en-US

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Ian Lyttle [aut, cre] (<<https://orcid.org/0000-0001-9962-4849>>),  
Vega/Vega-Lite Developers [aut],  
Alicia Schep [ctb] (<<https://orcid.org/0000-0002-3915-0618>>),  
Stuart Lee [ctb],  
Kanit Wongsuphasawat [ctb] (Vega/Vega-Lite library),

Dominik Moritz [ctb] (Vega/Vega-Lite library),  
 Arvind Satyanarayan [ctb] (Vega/Vega-Lite library),  
 Jeffrey Heer [ctb] (Vega/Vega-Lite library),  
 Mike Bostock [ctb] (D3 library)

**Maintainer** Ian Lyttle <ijlyttle@me.com>

**Repository** CRAN

**Date/Publication** 2022-10-20 06:40:02 UTC

## R topics documented:

add-listeners . . . . .	3
as_vegaspec . . . . .	4
data_category . . . . .	5
data_seattle_daily . . . . .	6
data_seattle_hourly . . . . .	7
glue_js . . . . .	7
image . . . . .	8
knit_print.vegaspec . . . . .	10
renderVegawidget . . . . .	11
shiny-getters . . . . .	11
shiny-setters . . . . .	13
spec_mtcars . . . . .	14
use_vegawidget . . . . .	14
vegawidget . . . . .	16
vegawidgetOutput . . . . .	18
vega_embed . . . . .	18
vega_schema . . . . .	20
vega_version . . . . .	21
vw_as_json . . . . .	22
vw_autosize . . . . .	23
vw_examine . . . . .	24
vw_handler_add_effect . . . . .	25
vw_handler_signal . . . . .	26
vw_rename_datasets . . . . .	28
vw_serialize_data . . . . .	28
vw_set_base_url . . . . .	30
vw_shiny_demo . . . . .	31
vw_spec_version . . . . .	32
vw_to_vega . . . . .	32

<b>Index</b>	<b>34</b>
--------------	-----------

---

add-listeners	<i>Add JavaScript listeners</i>
---------------	---------------------------------

---

### Description

Listeners are how we get information out of a Vega chart and into the JavaScript environment. To do this, we specify handler-functions to run whenever a certain signal changes or an event fires.

### Usage

```
vw_add_signal_listener(x, name, handler_body)
```

```
vw_add_data_listener(x, name, handler_body)
```

```
vw_add_event_listener(x, event, handler_body)
```

### Arguments

x	vegawidget object to be monitored
name	character, name of the signal or dataset to be monitored
handler_body	character or JS_EVAL, text of the body of the JavaScript handler-function to be called when the signal or dataset changes, or the event fires
event	character, name of the type of event to be monitored, e.g. "click"

### Details

The handler\_body can be the text of the *body* of a JavaScript function; the arguments to this function will vary according to the type of listener you are adding:

- signal-handler and data-handler arguments: name, value
- event-handler arguments: event, item

This package offers some functions to make it easier to build JavaScript handler functions from R: [vw\\_handler\\_signal\(\)](#), [vw\\_handler\\_data\(\)](#), and [vw\\_handler\\_event\(\)](#). You can pipe one of these functions to [vw\\_handler\\_add\\_effect\(\)](#) to perform side-effects on the result.

### Value

modified copy of vegawidget object x

### See Also

[vw\\_handler\\_signal\(\)](#), [vw\\_handler\\_data\(\)](#), [vw\\_handler\\_event\(\)](#), [vw\\_handler\\_add\\_effect\(\)](#), [vega-view](#)

---

`as_vegaspec`*Coerce to vegaspec*

---

## Description

Vega and Vega-Lite use JSON as their specification-format. Within R, it seems natural to work with these specifications as lists. Accordingly, a `vegaspec` is also a list. This family of functions is used to coerce lists, JSON, and character strings to `vegaspec`.

## Usage

```
as_vegaspec(spec, ...)  
  
## Default S3 method:  
as_vegaspec(spec, ...)  
  
## S3 method for class 'vegaspec'  
as_vegaspec(spec, ...)  
  
## S3 method for class 'list'  
as_vegaspec(spec, ...)  
  
## S3 method for class 'json'  
as_vegaspec(spec, ...)  
  
## S3 method for class 'character'  
as_vegaspec(spec, encoding = "UTF-8", ...)  
  
## S3 method for class 'vegawidget'  
as_vegaspec(spec, ...)
```

## Arguments

<code>spec</code>	An object to be coerced to <code>vegaspec</code> , a Vega/Vega-Lite specification
<code>...</code>	Other arguments (attempt to future-proof)
<code>encoding</code>	character, if <code>spec</code> is a file or a URL, specifies the encoding.

## Details

The character method for this function will take:

- JSON string.
- A path to a local JSON file.
- A URL that returns a JSON file.

For Vega and Vega-Lite, the translation between lists and JSON is a little bit particular. This function, `as_vegaspec()`, can be used to translate from JSON; `vw_as_json()` can be used to translate to JSON.

You can use the function `vw_spec_version()` to determine if a vegaspec is built for Vega-Lite or Vega. You can use `vw_to_vega()` to translate a Vega-Lite spec to Vega.

### Value

An object with S3 class `vegaspec`

### See Also

[Vega](#), [Vega-Lite](#), `vw_as_json()`, `vw_spec_version()`, `vw_to_vega()`

### Examples

```
spec <- list(
  `schema` = vega_schema(),
  data = list(values = mtcars),
  mark = "point",
  encoding = list(
    x = list(field = "wt", type = "quantitative"),
    y = list(field = "mpg", type = "quantitative"),
    color = list(field = "cyl", type = "nominal")
  )
)

as_vegaspec(spec)

## Not run:
# requires network-access
as_vegaspec("https://vega.github.io/vega-lite/examples/specs/bar.vl.json")

## End(Not run)
```

---

data\_category

*Example dataset: Categorical data*

---

### Description

This is a toy dataset; the numbers are generated randomly.

### Usage

data\_category

**Format**

A data frame with ten observations of two variables

**category** character, representative of a nominal variable

**number** double, representative of a quantitative variable

---

data\_seattle\_daily      *Example dataset: Seattle daily weather*

---

**Description**

This dataset contains daily weather-observations from Seattle for the years 2012-2015, inclusive.

**Usage**

data\_seattle\_daily

**Format**

A data frame with 1461 observations of six variables

**date** Date, date of the observation

**precipitation** double, amount of precipitation (mm)

**temp\_max** double, maximum temperature (°C)

**temp\_min** double, minimum temperature (°C)

**wind** double, average wind-speed (m/s)

**weather** character, description of weather

**Source**

<https://vega.github.io/vega-datasets/data/seattle-weather.csv>

---

data\_seattle\_hourly     *Example dataset: Seattle hourly temperatures*

---

### Description

This dataset contains hourly temperature observations from Seattle for the year 2010.

### Usage

```
data_seattle_hourly
```

### Format

A data frame with 8759 observations of two variables

**date** POSIXct, instant of the observation, uses "America/Los\_Angeles"

**temp** double, temperature (°C)

### Source

<https://vega.github.io/vega-datasets/data/seattle-weather-hourly-normals.csv>

---

glue\_js     *Interpolate into a JavaScript string*

---

### Description

Uses JavaScript notation to interpolate R variables into a string intended to be interpreted as JS.

### Usage

```
glue_js(..., .open = "${", .envir = parent.frame())
```

### Arguments

...	character vectors as the JavaScript source code (all arguments will be pasted into one character string)
.open	character, opening delimiter used by <code>glue::glue()</code>
.envir	environment, tells <code>glue::glue()</code> where to find the variables to be interpolated

### Details

This is a wrapper to `glue::glue()`, but it uses the notation used by [JavaScript's template-literals](#), `${}`.

**Value**

glue::glue() object

**Examples**

```
x <- 123
glue_js("function(){return({x});}") %>% print()
```

---

image

*Create or write image*

---

**Description**

If you have **V8**, **withr**, and **fs** installed, you can use these functions can to create or write images as PNG or SVG, using a vegaspec or vegawidget. To convert to a bitmap, or write a PNG file, you will additionally need the **rsvg** and **png** packages.

**Usage**

```
vw_to_svg(spec, width = NULL, height = NULL, base_url = NULL, seed = NULL)
```

```
vw_to_bitmap(spec, scale = 1, width = NULL, height = NULL, ...)
```

```
vw_write_svg(spec, path, width = NULL, height = NULL, ...)
```

```
vw_write_png(spec, path, scale = 1, width = NULL, height = NULL, ...)
```

**Arguments**

spec	An object to be coerced to vegaspec, a Vega/Vega-Lite specification
width	integer, if specified, the total rendered width (in pixels) of the chart - valid only for single-view charts and layered charts; the default is to use the width in the chart specification
height	integer, if specified, the total rendered height (in pixels) of the chart - valid only for single-view charts and layered charts; the default is to use the height in the chart specification
base_url	character, the base URL for a data file, useful for specifying a local directory; defaults to an empty string
seed	integer, the random seed for a Vega specification, defaults to a "random" integer
scale	numeric, useful for specifying larger images supporting the increased-resolution of retina displays
...	additional arguments passed to vw_to_svg()
path	character, local path to which to write the file

## Details

These functions can be called using (an object that can be coerced to) a vegaspec.

The scripts used are adapted from the Vega [command line utilities](#).

## Value

`vw_to_svg()` character, SVG string

`vw_to_bitmap()` array, bitmap array

`vw_write_svg()` invisible vegaspec or vegawidget, called for side-effects

`vw_write_png()` invisible vegaspec or vegawidget, called for side-effects

## See Also

[vega-view library](#)

## Examples

```
# call any of these functions using either a vegaspec or a vegawidget
svg <- vw_to_svg(vegawidget(spec_mtcars))
bmp <- vw_to_bitmap(spec_mtcars)
vw_write_png(spec_mtcars, file.path(tempdir(), "temp.png"))
vw_write_svg(spec_mtcars, file.path(tempdir(), "temp.svg"))

# To specify the path to a local file, use base_url
spec_precip <-
  list(
    `schema` = vega_schema(),
    data = list(url = "seattle-weather.csv"),
    mark = "tick",
    encoding = list(
      x = list(field = "precipitation", type = "quantitative")
    )
  ) %>%
  as_vegaspec()

data_dir <- system.file("example-data/", package = "vegawidget")
vw_write_png(
  spec_precip,
  file.path(tempdir(), "temp-local.png"),
  base_url = data_dir
)
```

---

knit\_print.vegaspec     *Knit-print method*

---

## Description

If you are knitting to an HTML-based format, the only supported options are `vega.width`, `vega.height` (as pixels) and `vega.embed` (as a list). If you are knitting to a non-HTML-based format, you additionally have the options `dev`, `out.width` and `out.height` available.

## Usage

```
knit_print.vegaspec(spec, ..., options = NULL)
```

## Arguments

<code>spec</code>	An object to be coerced to <code>vegaspec</code> , a Vega/Vega-Lite specification
<code>...</code>	other arguments
<code>options</code>	list, knitr options

## Details

The biggest thing to keep in mind about a Vega visualization is that very often, the chart tells you how much space it needs, rather than than you tell it how much space it has available. In the future, it may reveal itself how to manage better this "conversation".

## HTML-based

When knitting to an HTML-based format, the `spec` is rendered as normal, it calls `vegawidget()` using the options `vega.width`, `vega.height` and `vega.embed`:

- `vega.width` and `vega.height` are passed to `vegawidget()` as width and height, respectively. These values are coerced to numeric, so it is ineffective to specify a percentage. They are passed to `vw_autosize()` to resize the chart, if **possible**.
- `vega.embed` is passed to `vegawidget()` as `embed`. The function `vega_embed()` can be useful to set `vega.embed`.

## Non-HTML-based

When knitting to a non-HTML-based format, e.g. `github_document` or `pdf_document`, this function will convert the chart to an image, then knitr will incorporate the image into your document. You have the additional knitr options `dev`, `out.width`, and `out.height`:

- The supported values of `dev` are "png", "svg", and "pdf". If you are knitting to a LaTeX format (e.g. `pdf_document`) and you specify `dev` as "svg", it will be implemented as "pdf".
- To scale the image within your document, you can use `out.width` or `out.height`. Because the image will already have an aspect ratio, it is recommended to specify no more than one of these.

**See Also**

[vw\\_autosize\(\)](#), [vega\\_embed\(\)](#)

---

renderVegawidget	<i>Render shiny-output for vegawidget</i>
------------------	---

---

**Description**

Use this function in the server part of your Shiny app.

**Usage**

```
renderVegawidget(expr, env = parent.frame(), quoted = FALSE)
```

**Arguments**

expr	expression that generates a vegawidget. This can be a vegawidget or a vegaspec.
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

---

shiny-getters	<i>Get information from a Vega chart into Shiny</i>
---------------	---

---

**Description**

There are three types of information you can get from a Vega chart, a *signal*, *data* (i.e. a dataset), and information associated with an *event*. A dataset or a signal must first be defined and **named** in the vegaspec.

**Usage**

```
vw_shiny_get_signal(outputId, name, body_value = "value")
```

```
vw_shiny_get_data(outputId, name, body_value = "value")
```

```
vw_shiny_get_event(outputId, event, body_value = "datum")
```

**Arguments**

outputId	character, shiny outputId for the vegawidget
name	character, name of the signal (defined in Vega specification) being monitored
body_value	character or JS_EVAL, the <b>body</b> of a JavaScript function that Vega will use to handle the signal or event; this function must return a value
event	character, type of the event being monitored, e.g. "click", for list of supported events, please see <a href="#">Vega Event-Stream reference</a>

## Details

These getter-functions are called from within a Shiny `server()` function, where they act like `shiny::reactive()`, returning a reactive expression.

To see these functions in action, you can run a shiny-demo:

- `vw_shiny_get_signal()`: call `vw_shiny_demo("signal-set-get")`
- `vw_shiny_get_data()`: call `vw_shiny_demo("data-set-get")`
- `vw_shiny_get_event()`: call `vw_shiny_demo("event-get")`

In addition to the chart `outputId`, you will need to provide:

- `vw_shiny_get_signal()`: the name of the signal, as defined in the Vega specification
- `vw_shiny_get_data()`: the name of the dataset, as defined in the Vega specification
- `vw_shiny_get_event()`: the event type, as defined in the [Vega Event-Stream reference](#)

When the signal or data changes, or when the event fires, Vega needs to know which information you want returned to Shiny. To do this, you provide a JavaScript handler-function:

- `vw_shiny_get_signal()`: the default handler, `vw_handler_signal("value")`, specifies that the value of the signal be returned.
- `vw_shiny_get_data()`: the default handler, `vw_handler_data("value")`, specifies that the entire dataset be returned.
- `vw_shiny_get_event()`: the default handler, `vw_handler_event("datum")`, specifies that the single row of data associated with graphical mark be returned. For example, if you are monitoring a "click" event, Vega would return the row of data that backs any mark (like a point) that you click.

If you need to specify a different behavior for the handler, there are a couple of options. This package provides a library of handler-functions; call `vw_handler_signal()`, `vw_handler_data()`, or `vw_handler_event()` without arguments to list the available handlers.

If the library does not contain the handler you need, the `body_value` argument will also accept a character string which will be used as the **body** of the handler function.

For example, these calls are equivalent:

- `vw_shiny_get_signal(..., body_value = "value")`
- `vw_shiny_get_signal(..., body_value = vw_handler_signal("value"))`
- `vw_shiny_get_signal(..., body_value = "return value;")`

If you use a custom-handler that you think may be useful for the handler-function library, please [file an issue](#).

## Value

`shiny::reactive()` function that returns the value returned by `body_value`

## See Also

`vw_handler_signal()`, `vw_handler_event()`, vega-view: [addSignalListener\(\)](#), [addEventListener\(\)](#)

---

`shiny-setters`*Set information in a Vega chart from Shiny*

---

## Description

There are two ways to change a Vega chart: by setting a *signal* or by setting a *dataset*; you can also direct a Vega chart to re-run itself. Any signal or dataset you set must first be defined and **named** in the vegaspec. These functions are called from within a Shiny `server()` function, where they act like `shiny::observe()` or `shiny::observeEvent()`.

## Usage

```
vw_shiny_set_signal(outputId, name, value, run = TRUE, ...)
```

```
vw_shiny_set_data(outputId, name, value, run = TRUE, ...)
```

```
vw_shiny_run(outputId, value, ...)
```

## Arguments

<code>outputId</code>	character, shiny outputId for the vegawidget
<code>name</code>	character, name of the signal or dataset being set, as defined in the vegaspec
<code>value</code>	reactive expression, e.g. <code>input\$slider</code> or <code>dataset()</code> , that returns the value to which to set the signal or dataset
<code>run</code>	logical indicates if the chart is to be run immediately
<code>...</code>	other arguments passed on to <code>shiny::observeEvent()</code>

## Details

To see these functions in action, you can run a shiny-demo:

- `vw_shiny_set_signal()`: call `vw_shiny_demo("signal-set-get")`
- `vw_shiny_set_data()`: call `vw_shiny_demo("data-set-get")`
- `vw_shiny_run()`: call `vw_shiny_demo("data-set-swap-run")`

For the signal and data setters, in addition to the chart outputId, you will need to provide:

- the name of the signal or dataset you wish to keep updated
- the value to which you want to set the signal or dataset; this should be a reactive expression like `input$slider` or `rct_dataset()`
- whether or not you want to run the Vega view again immediately after setting this value

If you do not set `run = TRUE` in the setter-function, you can use the `vw_shiny_run()` function to control when the chart re-runs. One possibility is to set its value to a reactive expression that refers to, for example, a `shiny::actionButton()`.

**Value**

`shiny::observeEvent()` function that responds to changes in the reactive-expression value

---

spec\_mtcars

*Example vegaspec: mtcars scatterplot*

---

**Description**

A Vega-Lite specification to create a scatterplot for mtcars.

**Usage**

```
spec_mtcars
```

**Format**

S3 object of class vegaspec

**See Also**

`as_vegaspec()`

---

use\_vegawidget

*Add vegawidget functions to your package*

---

**Description**

These functions are offered to help you import and re-export vegawidget functions in your package. For more detail, please see [this article](#).

**Usage**

```
use_vegawidget(s3_class_name = NULL)
```

```
use_vegawidget_interactive()
```

**Arguments**

`s3_class_name` character, name of an S3 class for object to be coerced to a vegaspec; default (NULL) implies no additional class

## Details

use\_vegawidget():

Adds vegawidget functions:

- `as_vegaspec()`, `vw_as_json()`
- `format()`, `print()`, `knit_print()`
- `vegawidget()`, `vega_embed()`, `vw_set_base_url()`
- `vw_to_svg()` and other image functions
- `vegawidgetOutput()`, `renderVegawidget()`

In practical terms:

- adds **vegawidget** to Imports in your package's DESCRIPTION file.
- adds **V8**, **withr**, **fs**, **rsvg**, and **png** to Suggests in your package's DESCRIPTION file.
- creates `R/utils-vegawidget.R`
- you can delete references to functions you do not want to re-export.

If you have your own S3 class for a spec, specify the `s3_class_name` argument. You will have to edit `R/utils-vegawidget-<s3_class_name>.R`:

- add the code within your class's method for to coerce your object to a vegaspec.

To permit knit-printing of your custom class, you will have to add some code to your package's `.onLoad()` function.

use\_vegawidget\_interactive():

If you want to add the JavaScript and Shiny functions, use this after running `use_vegawidget()`. It adds:

- `vw_add_data_listener()` and other listener-functions.
- `vw_handler_data()` and other handler functions.
- `vw_shiny_get_data()` and other Shiny getters.
- `vw_shiny_set_data()` and other Shiny setters.

In practical terms:

- adds **shiny**, **dplyr**, to Suggests.
- creates `R/utils-vegawidget-interactive.R`.
- at your discretion, delete references to functions you do not want to re-export.

## Value

invisible NULL, called for side effects

vegawidget

*Create a Vega/Vega-Lite htmlwidget***Description**

The main use of this package is to render a `vegawidget`, which is also an `htmlwidget`. This function builds a `vegawidget` using a `vegaspec`.

**Usage**

```
vegawidget(
  spec,
  embed = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL,
  base_url = NULL,
  ...
)
```

**Arguments**

<code>spec</code>	An object to be coerced to <code>vegaspec</code> , a Vega/Vega-Lite specification
<code>embed</code>	list to specify <code>vega-embed</code> options, see <b>Details</b> on how this is set if NULL.
<code>width</code>	integer, if specified, the total rendered width (in pixels) of the chart - valid only for single-view charts and layered charts; the default is to use the width in the chart specification
<code>height</code>	integer, if specified, the total rendered height (in pixels) of the chart - valid only for single-view charts and layered charts; the default is to use the height in the chart specification
<code>elementId</code>	character, explicit element ID for the <code>vegawidget</code> , useful if you have other JavaScript that needs to explicitly discover and interact with a specific <code>vegawidget</code>
<code>base_url</code>	character, the base URL to prepend to data-URL elements in the <code>vegaspec</code> . This could be the path to a local directory that contains a local file referenced in the spec. It could be the base for a remote URL. Please note that by specifying the <code>base_url</code> here, you will override any loader that you specify using <code>vega_embed()</code> . Please note that this does not work with <code>kni tr</code> . See examples.
<code>...</code>	other arguments passed to <code>htmlwidgets::createWidget()</code>

**Details**

If `embed` is NULL, `vegawidget()` uses:

- `getOption("vega.embed")`, if that is NULL:

- an empty call to `vega_embed()`

The most-important arguments to `vega_embed()` are:

- `renderer`, to specify "canvas" (default) or "svg"
- `actions`, to specify action-links for export, source, compiled, and editor

If either width or height is specified, the `autosize()` function is used to override the width and height of the spec. There are some important provisions:

- Specifying width and height is **effective only for single-view charts and layered charts**. It will not work for concatenated, faceted, or repeated charts.
- In the spec, the default interpretation of width and height is to describe the dimensions of the **plotting rectangle**, not including the space used by the axes, labels, etc. Here, width and height describe the dimensions of the **entire** rendered chart, including axes, labels, etc.

Please note that if you are using a remote URL to refer to a dataset in your vegaspec, it may not render properly in the RStudio IDE, due to a security policy set by RStudio. If you open the chart in a browser, it should render properly.

## Value

S3 object of class `vegawidget` and `htmlwidget`

## See Also

[vega-embed options](#), `vega_embed()`, `vw_autosize()`

## Examples

```
vegawidget(spec_mtcars, width = 350, height = 350)

# vegaspec with a data URL
spec_precip <-
  list(
    `schema` = vega_schema(),
    data = list(url = "seattle-weather.csv"),
    mark = "tick",
    encoding = list(
      x = list(field = "precipitation", type = "quantitative")
    )
  ) %>%
  as_vegaspec()

# define local path to file
path_local <- system.file("example-data", package = "vegawidget")

# render using local path (does not work with knitr)
vegawidget(spec_precip, base_url = path_local)

## Not run:
# requires network-access
```

```
# define remote path to file
url_remote <- "https://vega.github.io/vega-datasets/data"

# render using remote path
# note: does not render in RStudio IDE; open using browser
vegawidget(spec_precip, base_url = url_remote)

## End(Not run)
```

---

vegawidgetOutput      *Shiny-output for vegawidget*

---

### Description

Use this function in the UI part of your Shiny app.

### Usage

```
vegawidgetOutput(outputId, width = "auto", height = "auto", widget = NULL)
```

### Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended. For vegawidgets, "auto" is useful because, as of now, the spec determines the size of the widget, then the widget determines the size of the container.
widget	character, indicating which version of libraries to use, e.g. "v15". Normally, you should not need to set this. See <code>vega_version_all()</code> for more information.

---

vega\_embed      *Vega embed options*

---

### Description

Helper-function to specify the embed argument to `vegawidget()`. These arguments reflect the options to the **vega-embed** library, which ultimately renders the chart specification as HTML.

**Usage**

```
vega_embed(
  renderer = c("canvas", "svg"),
  actions = NULL,
  defaultStyle = TRUE,
  config = NULL,
  patch = NULL,
  bind = NULL,
  ...
)
```

**Arguments**

renderer	character the renderer to use for the view. One of "canvas" (default) or "svg". See <a href="#">Vega docs</a> for details.
actions	logical or named vector of logicals, determines if action links ("Export as PNG/SVG", "View Source", "Open in Vega Editor") are included with the embedded view. If the value is TRUE (default), all action links will be shown and none if the value is FALSE. This property can be a named vector of logicals that maps keys (export, source, compiled, editor) to logical values for determining if each action link should be shown. By default, export, source, and editor are TRUE and compiled is FALSE, but these defaults can be overridden. For example, if actions is <code>list(export = FALSE, source = TRUE)</code> , the embedded visualization will have two links – "View Source" and "Open in Vega Editor".
defaultStyle	logical or character default stylesheet for embed actions. If set to TRUE (default), the embed actions are shown in a menu. Set to FALSE to use simple links. Provide a character string to set the style sheet.
config	character or list, a URL string from which to load a Vega/Vega-Lite or Vega-Lite configuration file, or a list of Vega/Vega-Lite configurations to override the default configuration options. If config is a URL, it will be subject to standard browser security restrictions. Typically this URL will point to a file on the same host and port number as the web page itself.
patch	JS function, list or character, A function to modify the Vega specification before it is parsed. Alternatively, an list that, when compiled to JSON, will meet <a href="#">JSON-Patch RFC6902</a> . If you use Vega-Lite, the compiled Vega will be patched. Alternatively to the function or the list, a URL string from which to load the patch can be provided. This URL will be subject to standard browser security restrictions. Typically this URL will point to a file on the same host and port number as the web page itself.
bind	character
...	other named items, outlined in <a href="#">vega-embed</a> options.

**Details**

The most important arguments are `renderer`, `actions`, and `defaultStyle`:

- The default renderer is "canvas".
- The default for actions is NULL, which means that the export, source, and editor links are shown, but the compiled link is not shown.
  - To suppress all action links, call with `actions = FALSE`.
  - To change from the default for a given action link, call with a list: `actions = list(editor = FALSE)`.
- The default for `defaultStyle` is TRUE, which means that action-links are rendered in a widget at the upper-right corner of the rendered chart.

The `vega-embed` library has a lot more options, you can supply these as names arguments using `...`

For example, it is ineffective to set the width and height parameters here when embedding a Vega-Lite specification, as they will be overridden by the value in the chart specification.

### Value

`list` to be used with `vega-embed` JavaScript library

### See Also

[vega-embed library](#), [vegawidget\(\)](#)

### Examples

```
vega_embed(renderer = "svg")
```

---

vega\_schema

*Create string for schema-URL*

---

### Description

Useful if you are creating a `vegaspec` manually.

### Usage

```
vega_schema(
  library = c("vega_lite", "vega"),
  version = NULL,
  major = is.null(version)
)
```

### Arguments

<code>library</code>	character, either "vega" or "vega_lite"
<code>version</code>	character, version of library, e.g. "5.2.0"; if version is provided, major defaults to FALSE.
<code>major</code>	logical return major version-tags rather than the tags for the specific versions supported by this package

**Value**

character URL for schema

**Examples**

```
vega_schema()
vega_schema("vega", major = FALSE)
vega_schema("vega_lite", version = "5.2.0")

# creating a spec by hand
spec <-
  list(
    `schema` = vega_schema(),
    width = 300,
    height = 300
    # and so on
  ) %>%
  as_vegaspec()
```

---

vega\_version

*Get Vega JavaScript versions*

---

**Description**

Use these functions to get which versions of Vega JavaScript libraries are available. `vega_version_all()` returns a data frame showing all versions included in this package, `vega_version_available()` returns all versions available - subject to locking, `vega_version()` shows the default version.

**Usage**

```
vega_version(major = FALSE)

vega_version_all(major = FALSE)

vega_version_available(major = FALSE)
```

**Arguments**

`major` logical return major version-tags rather than the tags for the specific versions supported by this package

**Details**

This package offers multiple widgets, each corresponding to a major version of Vega-Lite. Only one of these widgets can be used for a given loading of this package. When `vegawidget()` is first called, the widget is "locked" according to the `$schema` in the `vegaspec` used, or the default - the most-recent version.

`is_locked` indicates if `vegawidget()` is has locked the version.

`widget` indicates which version of the widget would be used.

### Value

**`vega_version()`** list with elements: `is_locked`, `widget`, `vega_lite`, `vega`, `vega_embed`.

**`vega_version_all()`** `data.frame` with elements: `widget`, `vega_lite`, `vega`, `vega_embed`.

**`vega_version_available()`** `data.frame` with elements: `widget`, `vega_lite`, `vega`, `vega_embed`.

### Examples

```
vega_version()
vega_version(major = TRUE)
vega_version_all()
vega_version_available()
```

---

vw\_as\_json

*Coerce vegaspec to JSON*

---

### Description

For Vega and Vega-Lite, the translation between lists and JSON is a little bit particular. This function, [vw\\_as\\_json\(\)](#), can be used to translate to JSON; [as\\_vegaspec\(\)](#) can be used to translate from JSON.

### Usage

```
vw_as_json(spec, pretty = TRUE)
```

### Arguments

<code>spec</code>	An object to be coerced to <code>vegaspec</code> , a Vega/Vega-Lite specification
<code>pretty</code>	logical indicates to use pretty (vs. minified) formatting

### Value

`jsonlite::json` object

### See Also

[as\\_vegaspec\(\)](#)

### Examples

```
vw_as_json(spec_mtcars)
```

---

vw_autosize	<i>Autosize vegaspec</i>
-------------	--------------------------

---

### Description

The arguments `width` and `height` are used to override the width and height of the provided `spec`, if the `spec` does not have multiple views. The dimensions you provide describe the overall width and height of the rendered chart, including axes, labels, legends, etc.

### Usage

```
vw_autosize(spec, width = NULL, height = NULL)
```

### Arguments

<code>spec</code>	An object to be coerced to <code>vegaspec</code> , a Vega/Vega-Lite specification
<code>width</code>	integer, if specified, the total rendered width (in pixels) of the chart - valid only for single-view charts and layered charts; the default is to use the width in the chart specification
<code>height</code>	integer, if specified, the total rendered height (in pixels) of the chart - valid only for single-view charts and layered charts; the default is to use the height in the chart specification

### Details

In a Vega or Vega-Lite specification, the default interpretation of `width` and `height` is to describe the dimensions of the **data rectangle**, not including the space used by the axes, labels, legends, etc. When `width` and `height` are specified using `autosize`, the meanings of `width` and `height` change to describe the dimensions of the **entire chart**, including axes, labels, legends, etc.

There is an important limitation: specifying `width` and `height` is **effective only for single-view and layered specifications**. It will not work for specifications with multiple views (e.g. `hconcat`, `vconcat`, `facet`, `repeat`); this will issue a warning that there will be no effect on the specification when rendered.

### Value

S3 object with class `vegaspec`

### See Also

[Article on vegaspec \(sizing\)](#), [Vega documentation on sizing](#)

### Examples

```
vw_autosize(spec_mtcars, width = 350, height = 350)
```

vw\_examine

*Examine vegaspec***Description**

This is a thin wrapper to `listviewer::jsonedit()`, use to interactively examine a Vega or Vega-Lite specification.

**Usage**

```
vw_examine(
  spec,
  mode = "view",
  modes = c("view", "code", "form", "text", "tree"),
  ...,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

**Arguments**

spec	An object to be coerced to vegaspec, a Vega/Vega-Lite specification
mode	string for the initial view from modes. 'view' is the default.
modes	string c('view', 'code', 'form', 'text', 'tree') will be the default, since these are all the modes currently supported by jsoneditor.
...	list of other options for jsoneditor. This is a temporary way of trying other options in jsoneditor. In the future, this will be eliminated in favor of specific, more self-documenting and helpful arguments.
width	integer in pixels defining the width of the div container.
height	integer in pixels defining the height of the div container.
elementId	character to specify valid CSS id of the htmlwidget for special situations in which you want a non-random identifier.

**Value**

S3 object of class jsonedit and htmlwidget

**Examples**

```
vw_examine(spec_mtcars)

spec_mtcars_autosize <-
  spec_mtcars %>%
  vw_autosize(width = 300, height = 300)

vw_examine(spec_mtcars_autosize)
```

---

vw\_handler\_add\_effect *Add a side-effect to a JavaScript handler*

---

## Description

With a JavaScript handler, once you have calculated a value based on the handler's arguments (e.g. name, value) you will likely want to produce a side-effect based on that calculated value. This function helps you do that.

## Usage

```
vw_handler_add_effect(vw_handler, body_effect, ...)
```

## Arguments

vw_handler	vw_handler created using <a href="#">vw_handler_signal()</a> or <a href="#">vw_handler_event()</a>
body_effect	character, the name of a defined handler-body, or the text of the body of a handler-function
...	additional <i>named</i> parameters to be interpolated into the text of the handler_body

## Details

The calculation of a value is meant to be separate from the production of a side-effect. This way, the code for a side-effect can be used for any type of handler.

You are supplying the body\_effect to an effect-handler. This takes a single argument, x, representing the calculated value. Doing this allows us to chain side-effects together; be careful not to modify x in any of the code you provide.

To see what side-effects are available in this package's handler-library, call `vw_handler_add_effect()` without any arguments. You may notice that some of the effects, like "element\_text", require additional parameters, in this case, selector.

Those parameters with a default value of NULL require you to supply a value; those with sensible defaults are optional.

To provide the parameters, call `vw_handler_add_effect()` with *named* arguments corresponding to the names of the parameters. See the examples for details.

## Value

modified copy of vw\_handler

## See Also

[vw\\_handler\\_signal\(\)](#)

## Examples

```
# list all the available effect-handlers
vw_handler_add_effect()

# build a signal handler that prints some text,
# then the value, to the console
vw_handler_signal("value") %>%
  vw_handler_add_effect("console", label = "signal value:")
```

---

vw\_handler\_signal      *Construct a JavaScript handler*

---

## Description

A Vega listener needs a JavaScript handler-function to call when the object-being-listened-to changes. For instance, [shiny-getters](#) and [add-listeners](#) functions each have an argument called `body_value`, which these functions help you build.

## Usage

```
vw_handler_signal(body_value)
```

```
vw_handler_data(body_value)
```

```
vw_handler_event(body_value)
```

## Arguments

`body_value`      character, the name of a defined handler-body, or the text of the body of a handler-function

## Details

There are two types of handlers defined in this package's handler-library. To see the handlers that are defined for each, call the function without any arguments:

- `vw_handler_signal()`
- `vw_handler_data()`
- `vw_handler_event()`

With a JavaScript handler, you are trying to do two types of things:

- calculate a value based on the handler's arguments
- produce a side-effect based on that calculated value

Let's look at a concrete example. A *signal handler* will take arguments name and value. Let's say that we want to return the value. We could do this two ways:

- `vw_handler_signal("value")`: use this package's handler library
- `vw_handler_signal("return value;")`: supply the body of the handler-function yourself

In the list above, the two calls do exactly the same thing, they build a JavaScript function that returns the value provided by whatever is calling the signal-handler. This will be a valid signal-handler, however, we will likely want a signal-handler to *do* something with that value, which is why we may wish to add a side-effect.

Let's say we want the handler to print the value to the JavaScript console. We would create the signal-handler, then add an effect to print the result to the console.

```
vw_handler_signal("value") %>% vw_handler_add_effect("console")
```

We can add as many effects as we like; for more information, please see the documentation for [vw\\_handler\\_add\\_effect\(\)](#).

Please be aware that these functions do *not* check for the correctness of JavaScript code you supply - any errors you make will not be apparent until your visualization is rendered in a browser.

One last note, if `body_value` is already a `vw_handler`, these functions are no-ops; they will return the `body_value` unchanged.

## Value

object with S3 class `vw_handler`

## See Also

[vw\\_handler\\_add\\_effect\(\)](#), [vega-view](#)

## Examples

```
# list all the available signal-handlers
vw_handler_signal()

# list all the available data-handlers
vw_handler_data()

# list all the available event-handlers
vw_handler_event()

# use a defined signal-handler
vw_handler_signal("value")

# define your own signal-handler
vw_handler_signal("return value;")
```

---

vw\_rename\_datasets      *Rename datasets in a vegaspec*

---

### Description

If a vegaspec has named datasets, it may be useful to rename them. This function will return a vegaspec with datasets named data\_001, data\_002, and so on. It will go through the spec and replace the references to the names. A future version of this function may give you the more control over the names used.

### Usage

```
vw_rename_datasets(spec)
```

### Arguments

spec                      An object to be coerced to vegaspec, a Vega/Vega-Lite specification

### Value

S3 object of class vegaspec

---

vw\_serialize\_data      *Serialize data-frame time-columns*

---

### Description

**Please think of this as an experimental function**

### Usage

```
vw_serialize_data(data, iso_dttm = FALSE, iso_date = TRUE)
```

### Arguments

data                      data.frame, data to be serialized

iso\_dttm                  logical, indicates if datetimes (POSIXct) are to be formatted using ISO-8601

iso\_date                  logical, indicates if dates (Date) are to be formatted using ISO-8601

## Details

In Vega, for now, there are only two time-zones available: the local time-zone of the browser where the spec is rendered, and UTC. This differs from R, where a time-zone attribute is available to POSIXct vectors. Accordingly, when designing a vegaspec that uses time, you have to make some compromises. This function helps you to implement your compromise in a principled way, as explained in the opinions below.

Let's assume that your POSIXct data has a time-zone attached. There are three different scenarios for rendering this data:

- using the time-zone of the browser
- using UTC
- using the time-zone of the data

If you intend to display the data using the **time-zone of the browser**, or using **UTC**, you should serialize datetimes using ISO-8601, i.e. `iso_dttm = TRUE`. In the rest of your vegaspec, you should choose local or UTC time-scales accordingly. However, in either case, you should use local time-units. No compromise is necessary.

If you intend to display the data using the **time-zone of the browser**, this is where you will have to compromise. In this case, you should serialize using `iso_dttm = FALSE`. By doing this, your datetimes will be serialized using a non-ISO-8601 format, and notably, **using the time-zone** of the datetime. When you design your vegaspec, you should treat this as if it were a UTC time. You should direct Vega to parse this data as UTC, i.e. `{"foo": "utc: '%Y-%m-%d %H:%M:%S'"}`. In other words, Vega should interpret your local timestamp as if it were a UTC timestamp. As in the first UTC case, you should use UTC time-scales and local time-units.

The compromise you are making is this: the internal representation of the instants in time will be different in Vega than it will be in R. You are losing information because you are converting from a POSIXct object with a time-zone to a timestamp without a time-zone. It is also worth noting that the time information in your Vega plot should not be used anywhere else - this should be the last place this serialized data should be used because it is no longer trustworthy. For this, you will gain the ability to show the data in the context of its time-zone.

Dates can be different creatures than datetimes. I think that can be "common currency" for dates. I think this is because it is more common to compare across different locations using dates as a common index. For example, you might compare daily stock-market data from NYSE, CAC-40, and Hang Seng. To maintain a common time-index, you might choose UTC to represent the dates in all three locations, despite the time-zone differences.

This is why the default for `iso_date` is `TRUE`. In this scenario, you need not specify to Vega how to parse the date; because of its ISO-8601 format, it will parse to UTC. As with the other UTC cases, you should use UTC time-scales and local time-units.

## Value

object with the same type as data

## See Also

[Vega-Lite Time Unit \(UTC\)](#)

**Examples**

```
# datetimes
data_seattle_hourly %>% head()
data_seattle_hourly %>% head() %>% vw_serialize_data(iso_dttm = TRUE)
data_seattle_hourly %>% head() %>% vw_serialize_data(iso_dttm = FALSE)

# dates
data_seattle_daily %>% head()
data_seattle_daily %>% head() %>% vw_serialize_data(iso_date = TRUE)
data_seattle_daily %>% head() %>% vw_serialize_data(iso_date = FALSE)
```

---

vw_set_base_url	<i>Set base URL</i>
-----------------	---------------------

---

**Description**

This is useful for specs where data is specified using a URL. Using this function to set the base URL, you can specify the data URL in specs using the relative path from the base.

For example, this [Vega-Lite example](#) uses the base URL `https://cdn.jsdelivr.net/npm/vega-datasets@2`. In a spec, instead of specifying:

```
data = "https://cdn.jsdelivr.net/npm/vega-datasets@2/data/cars.json"
```

You can call:

```
vw_set_base_url("https://cdn.jsdelivr.net/npm/vega-datasets@2")
```

Then specify:

```
data = "data/cars.json"
```

This function sets the value of `getOption("vega-embed")$loader$baseURL`. You need set it only once in a session or RMarkdown file.

**Usage**

```
vw_set_base_url(url)
```

**Arguments**

`url` character URL to use as the base URL.

**Value**

character called for side effects, it returns the previous value invisibly.

## Examples

```
# this is the URL used for Vega datasets
previous <- vw_set_base_url("https://cdn.jsdelivr.net/npm/vega-datasets@2")

# reset to previous value
vw_set_base_url(previous)
```

---

vw_shiny_demo	<i>Run Shiny demonstration-apps</i>
---------------	-------------------------------------

---

## Description

Run Shiny demonstration-apps

## Usage

```
vw_shiny_demo(example = NULL, ...)
```

## Arguments

example	character, name of the example to run; if NULL (default), prints out a list of available examples
...	additional arguments passed to <a href="#">shiny::runApp()</a>

## Value

invisible NULL, called for side-effects

## Examples

```
vw_shiny_demo() # returns available examples

# Run only in interactive R sessions
if (interactive()) {
  vw_shiny_demo("data-set-get")
}
```

---

vw_spec_version	<i>Determine vegaspec version</i>
-----------------	-----------------------------------

---

**Description**

Use this function to determine the library and version of a vegaspec.

**Usage**

```
vw_spec_version(spec)
```

**Arguments**

spec	An object to be coerced to vegaspec, a Vega/Vega-Lite specification
------	---

**Details**

Returns a list with two elements:

library character, either "vega" or "vega\_lite"

version character, version tag

**Value**

list with elements library, version

**Examples**

```
vw_spec_version(spec_mtcars)
# vw_to_vega() requires the V8 package
vw_spec_version(vw_to_vega(spec_mtcars))
```

---

vw_to_vega	<i>Convert to Vega specification</i>
------------	--------------------------------------

---

**Description**

If you have **V8** installed, you can use this function to compile a Vega-Lite specification into a Vega specification.

**Usage**

```
vw_to_vega(spec)
```

**Arguments**

spec	An object to be coerced to vegaspec, a Vega/Vega-Lite specification
------	---

**Value**

S3 object of class `vegaspec_vega` and `vegaspec`

**Examples**

```
vw_spec_version(spec_mtcars)
vw_spec_version(vw_to_vega(spec_mtcars))
```

# Index

- \* **datasets**
  - data\_category, 5
  - data\_seattle\_daily, 6
  - data\_seattle\_hourly, 7
  - spec\_mtcars, 14
- add-listeners, 3, 26
- as\_vegaspec, 4
- as\_vegaspec(), 5, 14, 15, 22
- data\_category, 5
- data\_seattle\_daily, 6
- data\_seattle\_hourly, 7
- glue::glue(), 7
- glue\_js, 7
- htmlwidgets::createWidget(), 16
- image, 8
- knit\_print.vegaspec, 10
- listviewer::jsonedit(), 24
- renderVegawidget, 11
- renderVegawidget(), 15
- shiny-getters, 11, 26
- shiny-setters, 13
- shiny::actionButton(), 13
- shiny::observe(), 13
- shiny::observeEvent(), 13, 14
- shiny::reactive(), 12
- shiny::runApp(), 31
- spec\_mtcars, 14
- use\_vegawidget, 14
- use\_vegawidget\_interactive  
(use\_vegawidget), 14
- vega\_embed, 18
- vega\_embed(), 10, 11, 15, 17
- vega\_schema, 20
- vega\_version, 21
- vega\_version\_all (vega\_version), 21
- vega\_version\_available (vega\_version),  
21
- vegawidget, 16
- vegawidget(), 10, 15, 20
- vegawidgetOutput, 18
- vegawidgetOutput(), 15
- vw\_add\_data\_listener (add-listeners), 3
- vw\_add\_data\_listener(), 15
- vw\_add\_event\_listener (add-listeners), 3
- vw\_add\_signal\_listener (add-listeners),  
3
- vw\_as\_json, 22
- vw\_as\_json(), 5, 15, 22
- vw\_autosize, 23
- vw\_autosize(), 10, 11, 17
- vw\_examine, 24
- vw\_handler\_add\_effect, 25
- vw\_handler\_add\_effect(), 3, 27
- vw\_handler\_data (vw\_handler\_signal), 26
- vw\_handler\_data(), 3, 12, 15
- vw\_handler\_event (vw\_handler\_signal), 26
- vw\_handler\_event(), 3, 12, 25
- vw\_handler\_signal, 26
- vw\_handler\_signal(), 3, 12, 25
- vw\_rename\_datasets, 28
- vw\_serialize\_data, 28
- vw\_set\_base\_url, 30
- vw\_set\_base\_url(), 15
- vw\_shiny\_demo, 31
- vw\_shiny\_get\_data (shiny-getters), 11
- vw\_shiny\_get\_data(), 15
- vw\_shiny\_get\_event (shiny-getters), 11
- vw\_shiny\_get\_signal (shiny-getters), 11
- vw\_shiny\_run (shiny-setters), 13
- vw\_shiny\_set\_data (shiny-setters), 13

`vw_shiny_set_data()`, [15](#)  
`vw_shiny_set_signal` (shiny-setters), [13](#)  
`vw_spec_version`, [32](#)  
`vw_spec_version()`, [5](#)  
`vw_to_bitmap`(image), [8](#)  
`vw_to_svg` (image), [8](#)  
`vw_to_svg()`, [15](#)  
`vw_to_vega`, [32](#)  
`vw_to_vega()`, [5](#)  
`vw_write_png`(image), [8](#)  
`vw_write_svg`(image), [8](#)