

Package ‘myIO’

October 3, 2026

Type Package

Title Interactive Data Visualizations Using 'd3.js'

Version 1.4.0

Description Create interactive 'd3.js' visualizations from R with built-in statistical transforms. Computes confidence intervals, regression fits, LOESS smoothing, moving averages, error bars, and uncertainty visualizations (quantile dot plots and fan charts) in R and renders them as composable chart layers via 'htmlwidgets'. Supports 36 chart types including boxplots, violin plots, Q-Q diagnostic plots, calendar heatmaps, survival curves, and group comparisons with pairwise significance testing. Also provides a machine-readable chart specification schema with validators so that large language model agents can author and verify charts. Works in 'RStudio', 'Shiny', and 'R Markdown'.

License MIT + file LICENSE

URL <https://mortonanalytics.github.io/myIO/>,
<https://github.com/mortonanalytics/myIO>

BugReports <https://github.com/mortonanalytics/myIO/issues>

Depends R (>= 4.1.0)

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, shiny, crosstalk (>= 1.2.0), dplyr, DT, reactable, htmltools, arrow, base64enc, cli, curl, DBI, duckdb, openssl, later, mockery, withr

Config/testthat/edition 3

VignetteBuilder knitr

Imports htmlwidgets, jsonlite, stats

NeedsCompilation no

Author Ryan Morton [aut, cre, cph],
 Mike Bostock [cph] (D3.js, d3-hexbin, and d3-sankey libraries),
 James Hall [cph] (jsPDF library),
 yWorks GmbH [cph] (jsPDF library)

Maintainer Ryan Morton <morton@myma.us>

Repository CRAN

Date/Publication 2026-10-03 20:00:02 UTC

Contents

addIoLayer	3
addKeyframe	4
clear_duckdb_wasm_cache	5
defineCategoricalAxis	5
dragPoints	6
duckdb_wasm_status	7
flipAxis	7
install_duckdb_wasm	8
linkCharts	9
myIO	10
myIO-shiny	11
myIOProxy	12
myio_chart_schema	13
myio_function_signature	13
myIO_last_error	14
myio_list_chart_types	15
myio_list_functions	15
myio_validate_call	16
myio_validate_spec	16
setAnnotation	17
setAxisFormat	18
setAxisLimits	19
setBigData	19
setBrush	21
setColorScheme	21
setExportOptions	22
setFacet	23
setKeyframe	24
setLayerOpacity	25
setLegendTitle	25
setLinked	26
setLinkedCursor	28
setMargin	29
setReferenceLines	29
setSlider	30
setTheme	31
setTitle	32

addIoLayer 3

setToggle	33
setToolTipOptions	34
setTransition	34
setTransitionSpeed	35
suppressAxis	36
suppressLegend	37

Index 38

addIoLayer	<i>Add a Layer to a myIO Chart</i>
------------	------------------------------------

Description

Adds individual layer to a myIO widget

Usage

```
addIoLayer(  
  myIO,  
  type,  
  color = NULL,  
  label,  
  data = NULL,  
  mapping,  
  transform = "identity",  
  options = list(barSize = "large", toolTipOptions = list(suppressY = FALSE))  
)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
type	chart type
color	optional CSS color string or vector for grouped layers. For type = "regression" the first element colors the raw scatter and the optional second element colors the fitted line and confidence band; supply the same color twice to render data and model in a single hue.
label	unique layer label
data	data frame backing the layer
mapping	named aesthetic mapping list
transform	transform name applied before serialization
options	layer options passed through to the widget config

Value

A modified myIO htmlwidget object with the new layer appended to the configuration.

Examples

```
myIO(data = mtcars) |>
  addIoLayer(
    type = "point", label = "points",
    mapping = list(x_var = "wt", y_var = "mpg")
  )
```

addKeyframe

Add a Named Data Keyframe

Description

Registers a named data state for sequential chart storytelling. A chart with one serialized layer accepts a data frame directly. Multi-layer charts use a named list of data frames keyed by existing layer labels; omitted layers retain their data from the previous keyframe.

Usage

```
addKeyframe(myIO, data, label)
```

Arguments

myIO	A widget created by <code>myIO()</code> with at least one layer.
data	A data frame for a single-layer chart, or a named list of data frames keyed by layer label for a multi-layer chart.
label	A unique, non-empty keyframe label.

Value

A modified myIO widget with the keyframe appended.

Examples

```
start <- data.frame(x = 1:3, y = c(2, 4, 3))
finish <- data.frame(x = 1:3, y = c(5, 3, 7))
myIO(start) |>
  addIoLayer("line", label = "series",
    mapping = list(x_var = "x", y_var = "y")) |>
  addKeyframe(start, "Start") |>
  addKeyframe(finish, "Finish")
```

clear_duckdb_wasm_cache

Remove DuckDB-WASM cache entries

Description

Remove DuckDB-WASM cache entries

Usage

```
clear_duckdb_wasm_cache(version = NULL)
```

Arguments

version	Character scalar naming a specific version to remove. If NULL, removes all cached versions.
---------	---

Value

Number of removed entries, invisibly.

Examples

```
## Not run:
# Removes cached DuckDB-WASM binaries from the user cache.
clear_duckdb_wasm_cache()

## End(Not run)
```

defineCategoricalAxis *Define Categorical Axis*

Description

Function to define the x variable as categorical

Usage

```
defineCategoricalAxis(myIO, xAxis = TRUE, yAxis = FALSE)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
xAxis	a logical argument (TRUE) for defining the x axis as categorical
yAxis	a logical argument (TRUE) for defining the y axis as categorical

Value

A modified myIO htmlwidget object with categorical axis configured.

Examples

```
# Define x axis as categorical
myIO() |> defineCategoricalAxis(xAxis = TRUE)

# Define both axes as categorical
myIO() |> defineCategoricalAxis(xAxis = TRUE, yAxis = TRUE)
```

dragPoints

Enable Draggable Points

Description

Function to make points draggable

Usage

```
dragPoints(myIO, dragPoints = TRUE)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
dragPoints	a logical argument (TRUE) for creating draggable points

Value

A modified myIO htmlwidget object with drag interaction enabled.

Examples

```
# Enable draggable points
myIO() |>
  addIoLayer(
    type = "point", color = "red", label = "pts",
    data = mtcars, mapping = list(x_var = "wt", y_var = "mpg")
  ) |>
  dragPoints()
```

duckdb_wasm_status	<i>DuckDB-WASM cache status</i>
--------------------	---------------------------------

Description

DuckDB-WASM cache status

Usage

duckdb_wasm_status()

Value

A list with class ‘myIO_duckdb_wasm_status’ and fields ‘installed’ (logical), ‘version’ (chr or NA), ‘cache_dir’ (chr), ‘size_bytes’ (numeric).

Examples

duckdb_wasm_status()

flipAxis	<i>Flip Chart Axes</i>
----------	------------------------

Description

Function to flip the x and y axes

Usage

flipAxis(myIO, flipAxis = TRUE)

Arguments

- | | |
|----------|---|
| myIO | an htmlwidget object created by the myIO() function |
| flipAxis | a logical argument (TRUE) for flipping the x and y axes |

Value

A modified myIO htmlwidget object with axes flipped.

Examples

```
# Flip the axes for a horizontal bar chart
myIO() |>
  addIoLayer(
    type = "bar", color = "steelblue", label = "bars",
    data = mtcars, mapping = list(x_var = "wt", y_var = "mpg")
  ) |>
  flipAxis()
```

install_duckdb_wasm	<i>Install the DuckDB-WASM binary for large-dataset virtualization</i>
---------------------	--

Description

Downloads and verifies the DuckDB-WASM runtime into a user-local cache so the in-browser big-data engine is available for subsequent widgets. This follows the keras3 / torch / reticulate pattern: the binary is NOT bundled with the R package and is not required for small-data use.

Usage

```
install_duckdb_wasm(
  version = NULL,
  from = NULL,
  force = FALSE,
  quiet = !interactive()
)
```

Arguments

version	Character scalar naming the version to install. Defaults to the latest row in ‘inst/duckdb-wasm-manifest.csv’.
from	Optional local directory containing pre-downloaded binaries (‘duckdb-mvp.wasm’ and ‘duckdb-browser-mvp.worker.js’). Airgap path.
force	Logical. If TRUE, overwrite existing cached binaries.
quiet	Logical. Suppress progress output. Defaults to ‘!interactive()’.

Value

Invisibly returns the cache path where the binary was installed.

Examples

```
## Not run:
# Downloads ~22 MB from the upstream mirror into the user cache.
install_duckdb_wasm()

## End(Not run)
```

linkCharts

*Link Charts for Cross-Selection***Description**

A convenience wrapper that sets matching link config on multiple widgets so that brush selections propagate between them. Unlike [setLinked](#), this does not require Crosstalk — it uses a shared group identifier and key column to coordinate selections across charts rendered in the same page.

Usage

```
linkCharts(..., on, group = NULL, cursor = FALSE, cursorAxis = "x")
```

Arguments

<code>...</code>	myIO widget objects to link.
<code>on</code>	Character. Column name to match rows across charts.
<code>group</code>	Character. Group identifier. Default auto-generated.
<code>cursor</code>	Logical. When TRUE, a hover in any linked chart draws a synchronized crosshair (and optional tooltip) on every sibling chart in the same group. Default FALSE.
<code>cursorAxis</code>	Character. Which axis to sync: "x" (default) draws a vertical rule at the hovered x value, "y" draws a horizontal rule at the hovered y value, and "xy" draws both. A rule is only drawn when the sibling chart can map the incoming value through its own scale.

Details

Cross-selection is driven by the brush, so at least one linked chart must also call [setBrush](#); brushing that chart propagates the selected on values to every other chart in the group, which dim their non-matching marks.

Value

A list of modified myIO widgets with matching link config.

See Also

[setLinked](#) for Crosstalk SharedData-based linking (use that path inside Shiny or with reactive filters).

Examples

```
w1 <- myIO() |>
  addIoLayer(type = "point", label = "scatter",
    data = mtcars, mapping = list(x_var = "wt", y_var = "mpg"))
w2 <- myIO() |>
  addIoLayer(type = "bar", label = "bars",
```

```
data = mtcars, mapping = list(x_var = "cyl", y_var = "mpg"))
linked <- linkCharts(w1, w2, on = "cyl")
```

myIO

*Create a myIO Chart Widget***Description**

Create an interactive D3.js chart widget

Usage

```
myIO(
  data = NULL,
  width = "100%",
  height = "400px",
  elementId = NULL,
  title = NULL,
  sparkline = FALSE,
  engine = "auto",
  webgl_threshold = 50000L,
  unify_data_path = FALSE
)
```

Arguments

data	an optional point of entry for the data frame or vector
width	a string of either pixel width or a percentage width
height	a string of pixel height
elementId	a unique id for the htmlwidget object
title	Optional chart title rendered inside the widget SVG.
sparkline	Logical. If TRUE, renders a compact sparkline suitable for embedding in table cells. Strips axes, legend, interactions, and the floating action button. Only "line", "bar", and "area" layer types are supported. Default FALSE.
engine	one of "auto" (default), "server", "wasm", or "svg". Only consulted when big-data features are attached via setBigData() . With no big-data attachment, charts render identically to the small-data SVG path regardless of this argument. See vignette("large-data-linking").
webgl_threshold	Positive integer row-count threshold for the big-data WebGL render path. Use Inf to disable WebGL. With unify_data_path = TRUE, disabled WebGL uses the SVG coordinator path; otherwise the existing inline SVG path is preserved. Default 50000.
unify_data_path	Logical. If TRUE, coordinator results also replace SVG layer data below webgl_threshold. Default FALSE preserves the inline small-data SVG render path.

Value

An htmlwidget object of class myIO.

Examples

```
myIO(data = mtcars) |>
  setMargin(top = 40, bottom = 80, left = 60, right = 10)
```

myIO-shiny

Shiny Bindings for myIO

Description

Shiny Bindings for myIO

Usage

```
myIOOutput(outputId, width = "100%", height = "400px")

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

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit or a number.
expr	An expression that generates a myIO
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression?

Value

myIOOutput returns a Shiny UI element for placement in a UI definition. renderMyIO returns a Shiny render function for use in a server definition.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(myIOOutput("chart"))
  server <- function(input, output) {
    output$chart <- renderMyIO({
      myIO(data = mtcars) |>
        addIoLayer(type = "point", label = "scatter",
          mapping = list(x_var = "wt", y_var = "mpg"))
    })
  }
}
```

```

    shinyApp(ui, server)
  }

```

myIOProxy

Update a myIO chart in place from the Shiny server

Description

‘myIOProxy()’ creates a lightweight handle to an already-rendered myIO widget, and ‘updateMyIOData()’ swaps the data of one or more existing layers without re-rendering the whole widget. Unlike re-executing ‘renderMyIO()’ (which destroys and recreates the chart on every reactive invalidation, dropping brush/zoom/toggle state and flickering), a proxy update reuses the existing data-join path: only the changed marks transition, and interaction state is preserved.

Usage

```

myIOProxy(outputId, session = NULL)

updateMyIOData(proxy, ...)

```

Arguments

outputId	The output id of the ‘myIOOutput()’ whose chart to update.
session	The Shiny session object. Defaults to the current reactive domain.
proxy	A ‘myIO_proxy’ object from ‘myIOProxy()’.
...	One or more ‘label = data.frame’ updates, where ‘label’ is an existing layer label and the data frame carries that layer’s mapped columns.

Details

Layers are matched by their ‘label’. Unknown labels are ignored client-side. The supplied data frame replaces the layer’s data as-is (the identity data path); statistical transforms attached at ‘addIoLayer()’ time are not re-applied, so pass already-transformed data for transformed layers.

Value

‘myIOProxy()’ returns a ‘myIO_proxy’ object; ‘updateMyIOData()’ returns the proxy invisibly.

Examples

```

## Not run:
library(shiny)
ui <- fluidPage(myIOOutput("chart"), actionButton("go", "Resample"))
server <- function(input, output, session) {
  output$chart <- renderMyIO({
    myIO(data = data.frame(x = 1:50, y = rnorm(50))) |>
      addIoLayer("line", label = "series", mapping = list(x_var = "x", y_var = "y"))
  })
}

```

```

    })
    observeEvent(input$go, {
      myIOProxy("chart") |>
        updateMyIOData(series = data.frame(x = 1:50, y = rnorm(50)))
    })
  }
  shinyApp(ui, server)

## End(Not run)

```

myio_chart_schema	<i>Get myIO Chart Schema for LLM Tool Calling</i>
-------------------	---

Description

Get myIO Chart Schema for LLM Tool Calling

Usage

```
myio_chart_schema(type = NULL)
```

Arguments

type Optional chart type. When NULL, returns every type schema.

Value

A list containing one chart schema or all chart schemas.

Examples

```
myio_chart_schema("boxplot")
```

myio_function_signature	<i>Get a myIO Function Signature for LLM Tool Calling</i>
-------------------------	---

Description

Get a myIO Function Signature for LLM Tool Calling

Usage

```
myio_function_signature(fn = NULL)
```

Arguments

fn Optional exported function name. When NULL, returns all signatures.

Value

A character vector of argument names or a named list of signatures.

Examples

```
myio_function_signature("setAxisFormat")
```

myIO_last_error	<i>Diagnose myIO Rendering Errors</i>
-----------------	---------------------------------------

Description

Prints guidance on how to find the most recent JavaScript error from a myIO widget. In Shiny, errors are available as reactive inputs. Outside Shiny, errors appear in the browser's developer console.

Usage

```
myIO_last_error(outputId = NULL)
```

Arguments

outputId optional Shiny output ID (character string). If provided, prints the exact Shiny input key to read.

Value

Invisibly returns NULL. Called for its side effect (printing diagnostic guidance).

Examples

```
myIO_last_error()  
myIO_last_error("chart1")
```

myio_list_chart_types *List myIO Chart Types for LLM Tool Calling*

Description

Returns chart type names from the generated myIO schema.

Usage

```
myio_list_chart_types()
```

Value

A character vector of chart type names.

Examples

```
head(myio_list_chart_types())
```

myio_list_functions *List myIO Functions for LLM Tool Calling*

Description

List myIO Functions for LLM Tool Calling

Usage

```
myio_list_functions()
```

Value

A character vector of exported myIO function names.

Examples

```
head(myio_list_functions())
```

myio_validate_call	<i>Validate a myIO Function Call</i>
--------------------	--------------------------------------

Description

Validate a myIO Function Call

Usage

```
myio_validate_call(fn, args = list())
```

Arguments

fn	Exported myIO function name.
args	Named list of proposed arguments.

Value

A list with valid and errors. Errors use stable machine-readable code values.

Examples

```
myio_validate_call("setAxisFormat", list(axis_x = ".0f"))
```

myio_validate_spec	<i>Validate a myIO Chart Specification</i>
--------------------	--

Description

Validates a proposed chart specification against the generated myIO schema.

Usage

```
myio_validate_spec(spec, columns = NULL)
```

Arguments

spec	A list with type, mapping, optional transform, and optional columns.
columns	Optional named column type map. When supplied, overrides spec\$columns and enables missing-column and numeric-column checks.

Value

A list with valid and errors. Errors use stable machine-readable code values.

Examples

```
myio_validate_spec(list(
  type = "boxplot",
  mapping = list(column_var = "Species", value_var = "Sepal.Width")
))
```

setAnnotation

Enable Click-to-Annotate

Description

Enables annotation mode where clicking a data point opens a label input. Annotations are stored as data and can be exported or accessed as a Shiny reactive input.

Usage

```
setAnnotation(myIO, labels = NULL, colors = NULL, mode = "click")
```

Arguments

myIO	an htmlwidget object created by the myIO() function
labels	character vector of preset label options (optional). If provided, shows a drop-down instead of free-text input.
colors	named character vector of category colors (optional). Names are category labels, values are CSS colors.
mode	"click" (default) to annotate individual points.

Value

A modified myIO htmlwidget object with annotation enabled.

Examples

```
myIO(data = mtcars) |>
  addIoLayer(
    type = "point", label = "pts",
    mapping = list(x_var = "wt", y_var = "mpg")
  ) |>
  setAnnotation(labels = c("outlier", "normal"))
```

setAxisFormat	<i>Set Axis Format</i>
---------------	------------------------

Description

Sets axis for x axis, y axis, and/or tool tip

Usage

```
setAxisFormat(
  myIO,
  xAxis = NULL,
  yAxis = NULL,
  toolTip = NULL,
  xLabel = NULL,
  yLabel = NULL
)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
xAxis	Optional string indicating a d3.js format for the x axis. When NULL, leaves the existing setting unchanged.
yAxis	Optional string indicating a d3.js format for the y axis. When NULL, leaves the existing setting unchanged.
toolTip	Optional string indicating a d3.js format for tooltips. When NULL, leaves the existing setting unchanged.
xLabel	Optional string label for the x axis. When NULL, leaves the existing setting unchanged.
yLabel	Optional string label for the y axis. When NULL, leaves the existing setting unchanged.

Value

A modified myIO htmlwidget object with updated axis format configuration. with options set for the tooltip formats

Examples

```
# Set axis formats using d3.js format strings
myIO() |> setAxisFormat(xAxis = ".0f", yAxis = ".1f")

# Set axis labels
myIO() |> setAxisFormat(xLabel = "Weight (lbs)", yLabel = "MPG")

# Label-only calls preserve previously configured formats
myIO() |> setAxisFormat(yAxis = ".2f") |> setAxisFormat(yLabel = "Rate")
```

setAxisLimits

Set Axis Limits

Description

Sets the minimum and maximum values for chart axes

Usage

```
setAxisLimits(
  myIO,
  xlim = list(min = NULL, max = NULL),
  ylim = list(min = NULL, max = NULL)
)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
xlim	a list of min and max values
ylim	a list of min and max values

Value

A modified myIO htmlwidget object with updated axis limits.

Examples

```
# Set x axis limits
myIO() |> setAxisLimits(xlim = list(min = 0, max = 100))

# Set both axis limits
myIO() |> setAxisLimits(
  xlim = list(min = 0, max = 50),
  ylim = list(min = -10, max = 200)
)
```

setBigData

Attach a big-data source to a myIO widget

Description

‘setBigData()’ declares the data source that a myIO widget should use for large-dataset rendering and linked-selection coordination.

Usage

```
setBigData(widget, source, rowkeyCol = NULL, ...)
```

Arguments

<code>widget</code>	A myIO htmlwidget object.
<code>source</code>	A supported big-data source: a <code>'data.frame'</code> , an Arrow table or record-batch reader, a single file path or URL ending in <code>'parquet'</code> , <code>'arrow'</code> , <code>'feather'</code> , or <code>'csv'</code> , or a <code>'DBIConnection'</code> .
<code>rowkeyCol</code>	Optional name of the column that uniquely identifies rows for Crosstalk-compatible linked selections.
<code>...</code>	Additional source-specific options. Also accepts the deprecated <code>rowkey_col</code> alias for <code>rowkeyCol</code> . For DBI sources, pass <code>'table = "name"'</code> so myIO can query the table schema. For file path or URL sources, pass <code>'schema = c("col1", "col2", ...)'</code> or a schema field list.

Details

This function writes the `'x.bigdata.*'` payload fields consumed by the widget's large-dataset virtualization path and follows its row-key contract. DBI sources are stored as an internal session-scoped marker in `'x.bigdata$dbi_handle_internal'`; render-time source registration is handled by the source registry phase.

Value

A modified myIO htmlwidget object.

Examples

```
myIO(data = mtcars) |>
  setBigData(mtcars)

myIO() |>
  setBigData("data/large.parquet", schema = c("id", "x", "y"), rowkeyCol = "id")

if (requireNamespace("duckdb", quietly = TRUE)) {
  con <- DBI::dbConnect(duckdb::duckdb())
  obs <- data.frame(id = seq_len(nrow(mtcars)), mpg = mtcars$mpg)
  DBI::dbWriteTable(con, "observations", obs)
  myIO() |>
    setBigData(con, table = "observations", rowkeyCol = "id")
  DBI::dbDisconnect(con, shutdown = TRUE)
}
```

setBrush	<i>Enable Brush Selection</i>
----------	-------------------------------

Description

Enables rectangle brush selection on the chart. When a user drags to select a region, the selected data points are available as a reactive input in Shiny or exportable in static HTML.

Usage

```
setBrush(myIO, direction = "xy", onSelect = NULL, ...)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
direction	brush direction: "xy" (default), "x", or "y"
onSelect	behavior in static mode: "highlight" (default) or "export" (scopes CSV download to selected points)
...	reserved; accepts the deprecated on_select alias for onSelect.

Value

A modified myIO htmlwidget object with brush interaction enabled.

Examples

```
myIO(data = mtcars) |>
  addIoLayer(
    type = "point", label = "pts",
    mapping = list(x_var = "wt", y_var = "mpg")
  ) |>
  setBrush()
```

setColorScheme	<i>Set Color Scheme</i>
----------------	-------------------------

Description

Sets color scheme for a chart and the category names (optional)

Usage

```
setColorScheme(myIO, colorScheme = NULL, setCategories = NULL)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
colorScheme	a vector of colors in the order you want them used
setCategories	an optional vector of names that will be mapped to the corresponding color in the colorScheme

Value

A modified myIO htmlwidget object with updated color scheme.

Examples

```
# Set a custom color scheme
myIO() |> setColorScheme(colorScheme = list("red", "blue", "green"))

# Set colors with category labels
myIO() |> setColorScheme(
  colorScheme = list("steelblue", "orange"),
  setCategories = c("Group A", "Group B")
)
```

setExportOptions	<i>Configure Export Options</i>
------------------	---------------------------------

Description

Controls which export buttons appear in the chart toolbar.

Usage

```
setExportOptions(
  myIO,
  png = TRUE,
  svg = TRUE,
  pdf = TRUE,
  clipboard = TRUE,
  csv = TRUE,
  title = NULL
)
```

Arguments

myIO	A myIO widget object.
png	Logical. Show PNG download button. Default TRUE.
svg	Logical. Show SVG download button. Default TRUE.

pdf	Logical. Show PDF download button. Default TRUE.
clipboard	Logical. Show clipboard copy button. Default TRUE.
csv	Logical. Show CSV export button. Default TRUE.
title	Character or NULL. Chart title for PDF metadata.

Value

Modified myIO widget.

Examples

```
myIO(iris) |>
  addIoLayer("point", label = "pts",
    mapping = list(x_var = "Sepal.Length", y_var = "Sepal.Width")) |>
  setExportOptions(svg = TRUE, clipboard = TRUE, pdf = FALSE)
```

setFacet	<i>Set Faceting (Small Multiples)</i>
----------	---------------------------------------

Description

Splits the chart into a grid of panels, one per unique value of the faceting variable. Each panel shows the same layers filtered to that subset of the data.

Usage

```
setFacet(
  myIO,
  var,
  ncol = NULL,
  minWidth = NULL,
  scales = "fixed",
  labelPosition = NULL,
  ...
)
```

Arguments

myIO	A myIO widget object.
var	Character. Column name to facet by. Must exist in at least one layer's data.
ncol	Integer or NULL. Number of columns in the grid. If NULL, auto-computes from minWidth and container width.
minWidth	Numeric. Minimum panel width in pixels when ncol is NULL. Default 200.
scales	Character. Scale sharing mode: • "fixed" – all panels share x and y scales (default)

- "free_x" – independent x scales per panel
- "free_y" – independent y scales per panel
- "free" – independent x and y scales per panel

labelPosition Character. Where to show panel labels: "top" (default) or "bottom".

... reserved; accepts the deprecated min_width and label_position aliases for minWidth and labelPosition.

Value

Modified myIO widget.

Examples

```
myIO(iris) |>
  addIoLayer("point", label = "pts",
    mapping = list(x_var = "Sepal.Length", y_var = "Sepal.Width")) |>
  setFacet("Species", ncol = 3)
```

setKeyframe	<i>Control Keyframes in Shiny</i>
-------------	-----------------------------------

Description

Select a named or numbered keyframe, or step an existing myIO widget without re-rendering the widget.

Usage

```
setKeyframe(proxy, frame)

stepKeyframe(proxy, direction = c("next", "previous"))
```

Arguments

proxy	A myIO_proxy object returned by <code>myIOProxy()</code> .
frame	A unique keyframe label or positive one-based keyframe index.
direction	Either "next" or "previous".

Value

The proxy, invisibly.

Examples

```
## Not run:
myIOProxy("chart") |> setKeyframe("Forecast")
myIOProxy("chart") |> stepKeyframe("next")

## End(Not run)
```

setLayerOpacity	<i>Set Layer Opacity</i>
-----------------	--------------------------

Description

Set Layer Opacity

Usage

```
setLayerOpacity(myIO, label, opacity)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
label	character. The layer label to target.
opacity	numeric. Opacity value between 0 (transparent) and 1 (opaque).

Value

A modified myIO htmlwidget object.

Examples

```
myIO(iris) |>  
  addIoLayer("point", label = "pts",  
    mapping = list(x_var = "Sepal.Length", y_var = "Sepal.Width")) |>  
  setLayerOpacity("pts", 0.5)
```

setLegendTitle	<i>Set the Legend Title</i>
----------------	-----------------------------

Description

Adds a title to the chart legend naming the variable its entries come from. The title renders on whichever legend surface is active – the compact in-plot legend or the panel opened from the chart's legend button – and on exported SVG, PNG and PDF output.

Usage

```
setLegendTitle(myIO, title = NULL)
```

Arguments

<code>myIO</code>	an <code>htmlwidget</code> object created by the <code>myIO()</code> function
<code>title</code>	a single character string used as the legend title, or <code>TRUE</code> to derive it from the name of the grouping column supplied as <code>addIoLayer(mapping = list(group = ...))</code> . <code>NULL</code> or <code>FALSE</code> clears the title, which is the default for every chart.

Details

The title is omitted when `suppressLegend()` is set. The derived form (`title = TRUE`) is omitted unless the legend has at least two entries and every entry comes from the same grouping column, so a chart that mixes grouped series with a standalone fitted line stays untitled unless a literal string is supplied.

Value

A modified `myIO` `htmlwidget` object carrying the legend title.

Examples

```
df <- data.frame(x = rep(1:5, 2), y = runif(10),
                 Month = rep(c("May", "June"), each = 5))

myIO() |>
  addIoLayer(type = "line", label = "Temp", data = df,
             mapping = list(x_var = "x", y_var = "y", group = "Month")) |>
  setLegendTitle("Month")

# Derive the title from the grouping column name
myIO() |>
  addIoLayer(type = "line", label = "Temp", data = df,
             mapping = list(x_var = "x", y_var = "y", group = "Month")) |>
  setLegendTitle(TRUE)
```

setLinked

Enable Linked Brushing via Crosstalk

Description

Connects the chart to a Crosstalk `SharedData` object so that brush selections propagate to other linked widgets.

Usage

```
setLinked(
  myIO,
  shared_data,
  mode = "both",
```

```

    filter = FALSE,
    key = NULL,
    group = NULL,
    cursor = FALSE,
    cursorAxis = "x"
  )

```

Arguments

myIO	an htmlwidget object created by the myIO() function
shared_data	a crosstalk::SharedData object
mode	"source", "target", or "both" (default)
filter	if TRUE, Crosstalk filter operations hide non-matching points. Default FALSE (dim only).
key	Optional character vector of row keys. When supplied, overrides the keys extracted from shared_data. Useful when the SharedData keys do not match the column used for cross-chart matching.
group	Optional character string. When supplied, overrides the Crosstalk group name from shared_data, allowing manual control over which widgets share selections.
cursor	Logical. When TRUE, a hover in any linked chart draws a synchronized crosshair on every sibling chart in the same group. Default FALSE.
cursorAxis	Character. Which axis to sync: "x" (default) draws a vertical rule at the hovered x value, "y" draws a horizontal rule at the hovered y value, and "xy" draws both. A rule is only drawn when the sibling chart can map the incoming value through its own scale.

Details

Selections travel on the Crosstalk key space, so a myIO chart matches rows against sibling widgets (**DT**, **plotly**, **leaflet**) by the same keys they use. The keys are matched to the chart's rows by position, which requires the data passed to addIoLayer() to be shared_data\$data() in its original row order. If a layer's row count does not match the number of keys – for example after re-filtering the frame, or after updateMyIOData() replaced the rows – the chart falls back to matching within its own widget only, rather than pairing keys with the wrong rows.

Value

A modified myIO htmlwidget with Crosstalk linking.

See Also

[linkCharts](#) for group-identifier linking that does not require Crosstalk (e.g. static R Markdown / Quarto HTML).

Examples

```
if (interactive() && requireNamespace("crosstalk", quietly = TRUE)) {
  shared <- crosstalk::SharedData$new(mtcars, key = ~rownames(mtcars))
  myIO() |>
    addIoLayer(
      type = "point", label = "scatter",
      data = shared$data(), mapping = list(x_var = "wt", y_var = "mpg")
    ) |>
    setLinked(shared)
}
```

setLinkedCursor

Toggle Linked Cursor Sync on a myIO Widget

Description

Enables or disables synchronized hover crosshair across linked charts. Composable with [linkCharts](#) and [setLinked](#) via the pipe: call after linking to opt into cursor sync without re-linking. Preserves any existing `interactions$linked` configuration.

Usage

```
setLinkedCursor(myIO, enabled = TRUE, axis = "x")
```

Arguments

<code>myIO</code>	A myIO htmlwidget.
<code>enabled</code>	Logical. TRUE to turn cursor sync on, FALSE to turn it off. Default TRUE.
<code>axis</code>	Character. Which axis to sync: "x" (default) draws a vertical rule at the hovered x value, "y" draws a horizontal rule at the hovered y value, and "xy" draws both. A rule is only drawn when the sibling chart can map the incoming value through its own scale, so a value outside the sibling's domain draws nothing on that axis.

Value

A modified myIO htmlwidget.

Examples

```
w1 <- myIO() |>
  addIoLayer(type = "point", label = "a",
    data = mtcars, mapping = list(x_var = "wt", y_var = "mpg"))
w2 <- myIO() |>
  addIoLayer(type = "point", label = "b",
    data = mtcars, mapping = list(x_var = "hp", y_var = "mpg"))
linked <- linkCharts(w1, w2, on = "cyl")
```

```
linked[[1]] <- setLinkedCursor(linked[[1]])
linked[[2]] <- setLinkedCursor(linked[[2]])
```

setMargin

Set Chart Margins

Description

Sets margins for the top, bottom, left, and right sides of the chart

Usage

```
setMargin(myIO, top = 20, bottom = 40, left = 50, right = 50)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
top	a numeric value representing in pixels the top margin
bottom	a numeric value representing in pixels the bottom margin
left	a numeric value representing in pixels the left margin
right	a numeric value representing in pixels the right margin

Value

A modified myIO htmlwidget object with updated margin configuration.

Examples

```
# Set custom margins
myIO() |> setMargin(top = 50, bottom = 80, left = 60, right = 20)
```

setReferenceLines

Set Reference Lines

Description

Sets x and y reference lines

Usage

```
setReferenceLines(myIO, xRef = 0, yRef = 0)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
xRef	a list of the reference line value of x
yRef	a list of the reference line value of y

Value

A modified myIO htmlwidget object with reference lines added.

Examples

```
# Add reference lines at x=5 and y=20
myIO() |> setReferenceLines(xRef = 5, yRef = 20)
```

setSlider	<i>Add a Parameter Slider (Shiny Only)</i>
-----------	--

Description

Adds a slider control below the chart. Read its Shiny input in the render expression and pass the value to the transform option to recompute the chart.

Usage

```
setSlider(myIO, param, label, min, max, value, step = NULL, debounce = 200)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
param	Slider input name (e.g., "ci_level", "degree"). Values are sent to input\$`myIO-{outputId}-slider-`
label	display label for the slider
min	minimum value
max	maximum value
value	default value
step	step size (default: NULL for auto)
debounce	debounce delay in milliseconds (default: 200)

Value

A modified myIO htmlwidget with slider config attached.

Examples

```
## Not run:
# In a Shiny server function:
output$chart <- renderMyIO({
  level <- input$`myIO-chart-slider-ci_level`
  if (is.null(level)) level <- 0.95
  myIO(data = mtcars) |>
    addIoLayer(
      type = "regression", label = "fit",
      mapping = list(x_var = "wt", y_var = "mpg"),
      options = list(level = level)
    ) |>
    setSlider("ci_level", "Confidence level", 0.80, 0.99, level, 0.01)
})

## End(Not run)
```

setTheme

Set Chart Theme

Description

Sets chart theme tokens using CSS custom properties

Usage

```
setTheme(
  myIO,
  textColor = NULL,
  gridColor = NULL,
  bg = NULL,
  font = NULL,
  mode = NULL,
  preset = NULL,
  overrides = list(),
  ...
)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
textColor	text and label color
gridColor	grid line color
bg	background color
font	font family

mode	Character or NULL. Theme mode: "light", "dark", or "auto". Default NULL (no mode, manual CSS vars only).
preset	Character or NULL. Named theme preset applied as a complete palette. One of "light", "dark", "midnight", "ocean", "forest", "sunset", "monochrome", "neon", "corporate", "academic", "nature", "minimal", "retro", or "warm". Unrecognized values are ignored. Default NULL.
overrides	Named list of CSS custom property overrides (e.g., list("--chart-tooltip-bg" = "#222")).
...	additional CSS custom property overrides; only names with a -- prefix are applied. Also accepts the deprecated text_color and grid_color aliases for textColor and gridColor. Other names are ignored with a warning.

Value

A modified myIO htmlwidget object with updated theme configuration.

Examples

```
myIO() |>
  setTheme(textColor = "#222222", gridColor = "#d9d9d9")

myIO() |>
  setTheme(mode = "dark", bg = "#1a1a2e")
```

setTitle	<i>Set Chart Title</i>
----------	------------------------

Description

Sets the title rendered inside the myIO widget SVG.

Usage

```
setTitle(myIO, title = NULL)
```

Arguments

myIO	A myIO widget object.
title	Character title or NULL to remove the title.

Value

A modified myIO htmlwidget object.

Examples

```
myIO() |>
  setTitle("Miles per gallon") |>
  addIoLayer("point", label = "cars",
    data = mtcars,
    mapping = list(x_var = "wt", y_var = "mpg"))
```

setToggle

Set Toggle Interaction

Description

Sets toggle options for y_var and adds a toggle button for chart

Usage

```
setToggle(myIO, variable, format = NULL)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
variable	a string indicating the variable name in data for toggle
format	a string indicating the format for the toggled variable

Value

A modified myIO htmlwidget object with toggle interaction configured.

Examples

```
# Add a toggle button to switch y variable
myIO() |> setToggle(variable = "Percent", format = ".0%")
```

setToolTipOptions	<i>Set Tooltip Options</i>
-------------------	----------------------------

Description

Generic function for setting tool tip options for a chart

Usage

```
setToolTipOptions(myIO, suppressY = NULL)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
suppressY	a boolean

Value

A modified myIO htmlwidget object with updated tooltip options.

Examples

```
# Suppress the y value in tooltips
myIO() |> setToolTipOptions(suppressY = TRUE)
```

setTransition	<i>Configure Chart Transitions</i>
---------------	------------------------------------

Description

Sets the duration, easing, and per-element stagger for the enter/update/exit animations used across chart layers. All arguments are optional; any left NULL keeps the chart's current behavior, so this setter is fully additive and backward compatible.

Usage

```
setTransition(myIO, duration = NULL, easing = NULL, stagger = NULL)
```

Arguments

myIO	an htmlwidget object created by the myIO() function
duration	transition duration in milliseconds (≥ 0). Set to 0 to disable animation.
easing	easing function name, one of "linear", "quad", "cubic", "sin", "exp", "circle", "back", "bounce", or "elastic" (mapped to the corresponding 'd3.js' easing). When NULL, each renderer keeps its built-in default easing.
stagger	per-element delay in milliseconds (≥ 0) applied across a layer's joined elements, creating a cascade. 0 disables stagger.

Details

Animation remains fully opt-out-able: `duration = 0` (or `setTransitionSpeed(0)`) disables animation, and easing and stagger automatically no-op whenever the effective duration is 0 — including when the viewer's system requests `prefers-reduced-motion: reduce`.

Value

A modified `myIO` `htmlwidget` object with updated transition settings.

See Also

[setTransitionSpeed](#) for the duration-only shorthand.

Examples

```
# Slower transitions with a bouncing ease and a 25ms cascade
myIO() |> setTransition(duration = 1200, easing = "bounce", stagger = 25)

# Change only the easing
myIO() |> setTransition(easing = "cubic")

# Disable animation entirely
myIO() |> setTransition(duration = 0)
```

setTransitionSpeed	<i>Set Transition Speed</i>
--------------------	-----------------------------

Description

Sets transition speeds across the chart (set to 0 to suppress)

Usage

```
setTransitionSpeed(myIO, speed)
```

Arguments

<code>myIO</code>	an <code>htmlwidget</code> object created by the <code>myIO()</code> function
<code>speed</code>	a number indicating the speed of transition in milliseconds

Value

A modified `myIO` `htmlwidget` object with updated transition speed.

See Also

[setTransition](#) to also configure easing and stagger.

Examples

```
# Set transition speed to 500ms
myIO() |> setTransitionSpeed(speed = 500)

# Disable transitions
myIO() |> setTransitionSpeed(speed = 0)
```

suppressAxis*Suppress Axis Display*

Description

Suppresses axes from printing

Usage

```
suppressAxis(myIO, xAxis = NULL, yAxis = NULL)
```

Arguments

<code>myIO</code>	an htmlwidget object created by the <code>myIO()</code> function
<code>xAxis</code>	a logical operator defining whether the x axis should be printed or not
<code>yAxis</code>	a logical operator defining whether the y axis should be printed or not

Value

A modified `myIO` htmlwidget object with axis display suppressed.

Examples

```
# Suppress both axes
myIO() |> suppressAxis(xAxis = TRUE, yAxis = TRUE)

# Suppress only the x axis
myIO() |> suppressAxis(xAxis = TRUE)
```

suppressLegend	<i>Suppress Legend Display</i>
----------------	--------------------------------

Description

Suppresses legend from printing

Usage

```
suppressLegend(myIO, suppressLegend = TRUE)
```

Arguments

myIO an htmlwidget object created by the myIO() function

suppressLegend a logical operator defining whether the legend should be printed or not

Value

A modified myIO htmlwidget object with legend display suppressed.

Examples

```
# Hide the legend  
myIO() |> suppressLegend()
```

Index

addIoLayer, [3](#)
addKeyframe, [4](#)

clear_duckdb_wasm_cache, [5](#)

defineCategoricalAxis, [5](#)
dragPoints, [6](#)
duckdb_wasm_status, [7](#)

flipAxis, [7](#)

install_duckdb_wasm, [8](#)

linkCharts, [9](#), [27](#), [28](#)

myIO, [4](#), [10](#), [34](#)
myIO-shiny, [11](#)
myio_chart_schema, [13](#)
myio_function_signature, [13](#)
myIO_last_error, [14](#)
myio_list_chart_types, [15](#)
myio_list_functions, [15](#)
myio_validate_call, [16](#)
myio_validate_spec, [16](#)
myIOOutput (myIO-shiny), [11](#)
myIOProxy, [12](#), [24](#)

renderMyIO (myIO-shiny), [11](#)

setAnnotation, [17](#)
setAxisFormat, [18](#)
setAxisLimits, [19](#)
setBigData, [10](#), [19](#)
setBrush, [9](#), [21](#)
setColorScheme, [21](#)
setExportOptions, [22](#)
setFacet, [23](#)
setKeyframe, [24](#)
setLayerOpacity, [25](#)
setLegendTitle, [25](#)
setLinked, [9](#), [26](#), [28](#)

setLinkedCursor, [28](#)
setMargin, [29](#)
setReferenceLines, [29](#)
setSlider, [30](#)
setTheme, [31](#)
setTitle, [32](#)
setToggle, [33](#)
setToolTipOptions, [34](#)
setTransition, [34](#), [35](#)
setTransitionSpeed, [35](#), [35](#)
stepKeyframe (setKeyframe), [24](#)
suppressAxis, [36](#)
suppressLegend, [37](#)

updateMyIOData (myIOProxy), [12](#)