Package 'jgd'

July 1, 2026

Title JSON Graphics Device

Version 0.1.1

Description A graphics device that translates R plotting operations into JSON and streams them over a local connection to an external display application. The device acts as a pure recorder with no rendering dependencies; all rendering occurs in that application (e.g. a 'VS Code' extension or a web browser). Official display applications are available from the project homepage.

License MIT + file LICENSE

Copyright file inst/COPYRIGHTS

Encoding UTF-8

NeedsCompilation yes

Suggests callr, ggplot2, jsonlite, processx, testthat (>= 3.0.0), withr

URL https://github.com/grantmcdermott/jgd

BugReports https://github.com/grantmcdermott/jgd/issues

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Author Grant McDermott [aut, cre], Tatsuya Shima [aut], Dave Gamble [cph] (cJSON library in src/cjson/), cJSON contributors [cph] (cJSON library in src/cjson/)

Maintainer Grant McDermott <contact@grantmcdermott.com>

Repository CRAN

Date/Publication 2026-06-30 23:40:02 UTC

Contents

`jgd` *JSON Graphics Device*

Description

Opens a graphics device that streams plot operations as JSON to an external renderer (e.g. VS Code extension or browser) over a Unix domain socket.

Usage

```
jgd(width = 8, height = 6, dpi = 96, socket = NULL)
```

Arguments

`width`	Device width in inches (default 8).
`height`	Device height in inches (default 6).
`dpi`	Resolution in dots per inch (default 96).
`socket`	Socket address for the rendering server. Supports URI formats (`tcp://host:port`, `unix:///path/to/socket`) or raw Unix socket paths. If `NULL` (default), use the `jgd.socket` R option, falling back to the `JGD_SOCKET`environment variable. If `JGD_SOCKET` environment variable is also unset, the device discovers the socket via the discovery file.

Value

Invisible `NULL`. The device is opened as a side effect.

Displaying plots with `jgd`

It is important to note that `jgd()` does not display any plots; it only streams them (i.e., converts them to a format that a JSON renderer understands). To actually *display* your plots with `jgd`, you'll need an appropriate frontend. Two official renderers are available:

- **VS Code.** Native `jgd` support is built into the VS Code R extension (`https://github.com/REditorSupport/vscode-R`), which handles device activation for you.

- **Deno server.** A standalone browser-based renderer, available from the project repository: https://github.com/grantmcdermott/jgd.

Users aren't limited to these two options. The jgd protocol is deliberately frontend-agnostic; you can render plots with any client that reads JSONL (JSON Lines). Again, please see the project repository for full documentation: https://github.com/grantmcdermott/jgd

Debugging

Set `options(jgd.debug = TRUE)` before opening the device to enable frame-level diagnostic output on stderr (via `REprintf`). This logs details about `newPage`, `flush_frame`, and `poll_resize` events, which is useful for diagnosing resize/replay issues.

Protocol specification

The jgd protocol is a simple, versioned JSONL wire format designed to be frontend-agnostic. You can use it to build your own renderer (e.g., for Neovim, Emacs, or a custom web app). See `jgd_spec` for the complete specification covering transports, message schemas, drawing operations, the resize protocol, font metrics, and multi-session routing.

See Also

`jgd_spec` for the protocol specification; `jgd_ext()` and `with_jgd_ext()` for renderer extensions.

Examples

```
# Requires a running renderer (e.g., VS Code extension or Deno server).
# See the "Displaying plots" section above.
library(jgd)
jgd()
plot(1:10)
lines(1:10, col = "red", lwd = 3)
hist(rnorm(1000), col = "steelblue")
dev.off()
```

---

jgd_begin_group *Begin a drawing group (experimental)*

Description

Emits a `beginGroup` operation into the drawing stream. All subsequent drawing operations until the matching `jgd_end_group()` are part of this group. The renderer may use the group's extension fields to apply effects to the group as a whole.

Usage

```
jgd_begin_group(ext = NULL)
```

Arguments

`ext`	A single JSON string with extension fields for this group, or `NULL` for a group without extension fields.

Value

Called for its side effect; returns `NULL` invisibly.

Lifecycle

Experimental. This API may change in future versions.

`jgd_discover` *Discover a running jgd server*

Description

Reads the jgd discovery file from the platform cache directory (`~/.cache/jgd` on Linux, `~/Library/Caches/jgd` on macOS, `%LOCALAPPDATA%/jgd` on Windows) and returns its contents. This does not require an open jgd device — it simply reads the file that a running server has written.

Usage

```
jgd_discover()
```

Value

A named list with `server_name` (character), `socket_path` (character), `pid` (integer), and `server_info` (named character vector), or `NULL` if no discovery file is found.

`jgd_end_group` *End a drawing group (experimental)*

Description

Emits an `endGroup` operation into the drawing stream, closing the most recently opened group from `jgd_begin_group()`.

Usage

```
jgd_end_group()
```

Value

Called for its side effect; returns `NULL` invisibly.

Lifecycle

Experimental. This API may change in future versions.

`jgd_ext` *Set extended graphics context (experimental)*

Description

Sets extension fields that are included in every subsequent drawing operation's graphics context (`gc.ext` in the JSON protocol). This is an experimental, low-level API for injecting renderer-specific properties (e.g. blend modes, shadows, opacity) that go beyond R's standard graphics parameters.

Usage

```
jgd_ext(json = NULL)
```

Arguments

`json`	A single JSON string representing the extension object, or `NULL` to clear. The string must be valid JSON (validated on the C side via cJSON); an error is raised otherwise. Packages built on top of jgd (using e.g. jsonlite) should provide user-friendly wrappers.

Value

Called for its side effect; returns `NULL` invisibly.

Supported extension fields

The Deno reference server and VS Code renderer currently support:

Field	Canvas2D property	Example
`blendMode`	`globalCompositeOperation`	`"multiply"`
`opacity`	`globalAlpha`	`0.5`
`shadow.blur`	`shadowBlur`	`10`
`shadow.color`	`shadowColor`	`"rgba(0,0,0,0.5)"`
`shadow.offsetX`	`shadowOffsetX`	`5`
`shadow.offsetY`	`shadowOffsetY`	`5`
`filter`	`filter`	`"blur(3px)"`

Custom renderers may support additional fields. Unknown fields are silently ignored, so extensions are forward-compatible.

Design for extension packages

`jgd_ext()` is intentionally low-level — it accepts a raw JSON string. Higher-level packages built on top of jgd can provide user-friendly wrappers with proper argument checking, e.g.:

```
jgd_shadow = function(blur = 0, color = "black",
                      offsetX = 0, offsetY = 0) {
  jgd_ext(jsonlite::toJSON(
    list(shadow = list(blur = blur, color = color,
                       offsetX = offsetX, offsetY = offsetY)),
    auto_unbox = TRUE
  ))
}
```

jgd itself has no dependency on jsonlite or any serialization library; upstream packages choose their own.

Lifecycle

Experimental. This API may change in future versions.

See Also

`with_jgd_ext()`, `jgd_frame_ext()`, `jgd_begin_group()`, `jgd_spec`

Examples

```
jgd()

# Drop shadow (scoped -- automatically cleared after the block)
with_jgd_ext(
  '{"shadow":{"blur":15,"color":"rgba(0,0,0,0.5)","offsetX":5,"offsetY":5}}',
  plot(1:10, pch = 19, cex = 3, col = "steelblue")
)

# Semi-transparent overlay
with_jgd_ext('{"opacity":0.3}', {
  plot(1:10, pch = 19, cex = 5, col = "red")
})

# Manual set/clear
jgd_ext('{"blendMode":"multiply"}')
plot(1:10)
jgd_ext(NULL)
```

jgd_frame_ext *Set frame-level extension fields (experimental)*

Description

Sets extension fields that are included once per frame in the JSON protocol (at the top level of the frame message, not per drawing operation). This is useful for frame-wide properties such as post-processing effects.

Usage

```
jgd_frame_ext(json = NULL)
```

Arguments

json	A single JSON string representing the extension object, or NULL or "" to clear. Non-empty strings must be valid JSON; an error is raised otherwise.

Value

Called for its side effect; returns NULL invisibly.

Lifecycle

Experimental. This API may change in future versions.

jgd_server_info *Get server information*

Description

Returns metadata about the jgd server. When a jgd device is open and connected, returns the welcome message information with connected = TRUE. Otherwise, falls back to reading the discovery file and returns information with connected = FALSE. Returns NULL if no information is available from either source.

Usage

```
jgd_server_info()
```

Details

The discovery fallback applies regardless of whether the current device is a jgd device. This means jgd_server_info() can return a non-NULL result even when no jgd device is open, as long as a valid discovery file exists.

Value

A named list, or NULL.

When connected:

- connected: TRUE
- server_name: Server name (character)
- protocol_version: Protocol version (integer)
- transport: Transport protocol (character)
- server_info: Named character vector of key-value pairs from the server's serverInfo object (e.g. c(httpUrl = "http://...")); empty if absent

When not connected (discovery file fallback):

- connected: FALSE
- server_name: Server name (character)
- socket_path: Socket URI (character)
- pid: Server process ID (integer)
- server_info: Named character vector (as above)

---

jgd_spec *jgd JSONL Protocol Specification*

Description

The jgd device communicates with a rendering server over JSONL (JSON Lines). Messages are exchanged over a persistent connection using one of three transport protocols: Unix domain sockets (Linux/macOS), Windows named pipes, or TCP.

This document specifies the wire protocol so that third-party servers can implement a compatible rendering backend.

Transport protocols

The client connects to the server using one of the following URI schemes:

- unix:///path/to/socket – Unix domain socket (Linux/macOS default)
- npipe:////./pipe/name – Windows named pipe (Windows default, Docker-standard 4-slash form)
- tcp://host:port – TCP socket (any platform)

Raw Unix socket paths (without a URI scheme) are also accepted.

Message format

All messages are single-line JSON objects terminated by \n (JSONL). Each message contains a "type" field identifying the message kind. Encoding is always UTF-8.

Receivers should ignore unknown top-level fields in any message (forward-compatible). Unknown "type" values should be silently discarded rather than treated as errors.

Coordinate system

All coordinates in drawing operations are in **device pixels** (i.e., `inches * dpi`). The origin `(0, 0)` is the **top-left** corner of the device surface. The X axis increases to the right and the Y axis increases downward.

Connection handshake

The welcome message is **deferred**: the server waits until it receives the first message from R before sending it. This avoids a race condition on Windows named pipes where writing before the first read completes can cause data loss.

```
R -> Server:  {"type":"ping"}
Server -> R:  {"type":"server_info", ...}
```

The server should also tolerate receiving a `frame` message before `ping` (e.g., if a future client skips the ping). The first received message of any type should trigger the deferred welcome.

Discovery file

The discovery file is an **optional** JSON file that allows the client to find the server without an explicit socket address. It is a hint for auto-connection only; the welcome message is the single source of truth.

Location (platform-specific):

- Linux: `$XDG_CACHE_HOME/jgd/discovery.json` or `~/.cache/jgd/discovery.json`
- macOS: `~/Library/Caches/jgd/discovery.json`
- Windows: `%LOCALAPPDATA%/jgd/discovery.json`

Schema:

```
{
  "serverName": "jgd-http-server",
  "socketPath": "tcp://127.0.0.1:9000",
  "pid": 12345,
  "serverInfo": {
    "httpUrl": "http://127.0.0.1:8080/"
  }
}
```

- `serverName` (string, required): Human-readable server name.
- `socketPath` (string, required): Socket URI where the client should connect.
- `pid` (integer, required): Process ID of the server.
- `serverInfo` (object, optional): Flat key-value pairs with string values. Canonical key: `httpUrl` (HTTP endpoint URL).

Lifecycle:

- Written atomically (temp file + rename) after all listeners are ready.

- Server should remove the file on graceful shutdown, but only after confirming it still owns the file (PID check).
- Clients should verify liveness (e.g., PID check) before using stale files, since `~/.cache` is not cleared on reboot.
- Multiple server instances may coexist; the last writer wins.
- Server implementors may omit discovery file support entirely. Clients can always connect directly via an explicit socket URI.
- The discovery file has no explicit version field. Readers should ignore unknown fields for forward compatibility.

server_info message

```
{
  "type": "server_info",
  "serverName": "jgd-http-server",
  "protocolVersion": 1,
  "transport": "unix",
  "serverInfo": {
    "httpUrl": "http://127.0.0.1:8080/"
  }
}
```

- `type`: `"server_info"` (string, always present)
- `serverName`: Human-readable server name (string)
- `protocolVersion`: Protocol version number, currently `1` (integer). Receivers should ignore messages with an unknown protocol version rather than raising an error.
- `transport`: Transport in use: `"tcp"`, `"unix"`, or `"npipe"` (string)
- `serverInfo`: A flat JSON object whose values are all strings (optional). Canonical key: `httpUrl`.

See `jgd_server_info()` for how the R client represents this data.

R-to-server messages

ping – Heartbeat; triggers the deferred welcome on first send.

```
{"type": "ping"}
```

frame – A complete or incremental set of drawing operations. See the Frame message section for the full schema.

metrics_request – Requests font metrics from the renderer.

```
{"type": "metrics_request", "id": 1, "kind": "strWidth",
 "str": "Hello",
 "gc": {"font": {"family": "sans", "face": 1,
                 "size": 12}}}
```

```
{"type": "metrics_request", "id": 2, "kind": "metricInfo",
 "c": 77,
 "gc": {"font": {"family": "sans", "face": 1,
                 "size": 12}}}
```

- `id`: Request identifier (integer); the response must echo it.
- `kind`: `"strWidth"` (string width) or `"metricInfo"` (glyph metrics).
- `str` (string, `strWidth` only): The string to measure.
- `c` (integer, `metricInfo` only): Unicode code point of the character to measure (e.g., 77 for `"M"`).
- gc: Graphics context with a `font` object containing `family` (string), `face` (integer), and `size` (font size in points).

close – Signals device shutdown.

```
{"type": "close"}
```

Server-to-R messages

server_info – Welcome message (see above).

resize – Renderer viewport change.

```
{"type": "resize", "width": 800, "height": 600}
```

- `width`, `height`: New viewport dimensions in device pixels (positive integers). These become the device's width and height directly (no DPI scaling is applied).
- `plotIndex` (integer, optional): If present, replay the historical plot identified by its R-assigned plot number (the `plotNumber` from earlier frames) instead of the current plot.

metrics_response – Font metrics from the renderer.

```
{"type": "metrics_response", "id": 1, "width": 48.5,
 "ascent": 10.2, "descent": 2.8}
```

- `id`: Must match the request `id`.
- Servers should respond promptly. Clients handle their own timeouts and may fall back to local computation. Servers are not required to synthesize fallback responses.

Frame message

The frame message carries drawing operations from R to the server.

New plot example:

```
{
  "type": "frame",
  "incremental": false,
  "newPage": true,
```

```
  "plotNumber": 0,
  "ext": {},
  "plot": {
    "version": 1,
    "sessionId": "r-1234-1",
    "device": {
      "width": 768,
      "height": 576,
      "dpi": 96,
      "bg": "rgba(255,255,255,1)"
    },
    "ops": []
  }
}
```

(Minimal example; real frames typically start with a `clip` op.)

Historical resize replay example:

```
{
  "type": "frame",
  "incremental": false,
  "resizeReplay": true,
  "plotIndex": 0,
  "plot": { "..." }
}
```

Top-level fields:

- `type`: `"frame"` (always present).
- `incremental` (boolean, always present): If `true`, `ops` contains only operations added since the last flush (delta). If `false`, `ops` contains the complete drawing for the page.
- `newPage` (boolean, optional): Present and `true` when this is a fresh plot (not a delta, not a resize replay).
- `resizeReplay` (boolean, optional): Present and `true` when this frame is a replay triggered by a resize.
- `plotIndex` (integer, optional): Present during `resizeReplay` when a historical plot (not the current one) was replayed. This is the absolute R-side plot number (the same 0-based value previously sent as `plotNumber` when the plot was created). It may diverge from the renderer's current history array index after deletions or evictions. See also the Resize protocol section for the full resize flow.
- `plotNumber` (integer, optional): Absolute 0-based sequence number for plots (e.g., 0 for the first, 1 for the second). Present on all frames for the current plot (including incremental and resize replay frames). Omitted only on historical resize replays where `plotIndex` is present.
- `ext` (object, optional): Frame-level extension data. When unset, the field is omitted (never sent as `null`); when set, it may be any JSON object including an empty `{}`. Servers should preserve and forward it to renderers.

plot object:

- `version` (integer): Protocol version, currently `1`.
- `sessionId` (string): Identifies the R session/device. Used for routing resize requests to the correct R process.
- `device` (object):
  - `width`, `height`: Device dimensions in pixels.
  - `dpi`: Dots per inch.
  - `bg`: Background color as an RGBA string (see the Color format section).
- `ops` (array): Drawing operations (see the Drawing operations section).

Color format

Colors are represented as CSS-style RGBA strings:

`"rgba(R,G,B,A)"`

- R, G, B: integers 0–255.
- A: decimal 0.0–1.0 (e.g., `"rgba(0,0,0,0.502)"`).

Transparent or `NA` colors are represented as JSON `null`.

Graphics context

Most drawing operations include a `"gc"` object (exceptions are noted per operation):

```
{
  "col": "rgba(0,0,0,1)",
  "fill": null,
  "lwd": 1.0,
  "lty": [],
  "lend": "round",
  "ljoin": "round",
  "lmitre": 10.0,
  "font": {
    "family": "sans",
    "face": 1,
    "size": 12.0,
    "lineheight": 1.2
  },
  "ext": {}
}
```

- `col`: Stroke color (RGBA string or `null`).
- `fill`: Fill color (RGBA string or `null`).
- `lwd`: Line width in pixels (number).

- `lty`: Line type as an array of dash lengths (in pixels). Solid lines and blank (invisible) lines both produce an empty array `[]`. When the line is blank, `col` is `null`, so renderers can distinguish via the color.
- `lend`: Line end cap: `"round"`, `"butt"`, or `"square"`.
- `ljoin`: Line join: `"round"`, `"miter"`, or `"bevel"`.
- `lmitre`: Miter limit (number).
- `font`:
  - `family`: Font family name (string; empty string if default).
  - `face`: Font face: 1 = plain, 2 = bold, 3 = italic, 4 = bold italic, 5 = symbol.
  - `size`: Font size in points (number).
  - `lineheight`: Line height multiplier (number).
- `ext` (object, optional): Per-operation extension data. Present only when set. Free-form JSON.

Drawing operations

Each element of the `ops` array is a JSON object with an `"op"` field. Most drawing operations include a `"gc"` field (see the Graphics context section). Exceptions are noted per operation.

All coordinates are in device pixels with a top-left origin (see the Coordinate system section).

clip – Set the clipping rectangle. No `gc`.

```
{"op": "clip", "x0": 0, "y0": 0, "x1": 768, "y1": 576}
```

line – A single line segment.

```
{"op": "line", "x1": 100, "y1": 200,
 "x2": 300, "y2": 400, "gc": {}}
```

polyline – Connected line segments (not closed).

```
{"op": "polyline", "x": [1, 2, 3],
 "y": [4, 5, 6], "gc": {}}
```

polygon – Closed polygon (filled and/or stroked).

```
{"op": "polygon", "x": [1, 2, 3],
 "y": [4, 5, 6], "gc": {}}
```

rect – Rectangle.

```
{"op": "rect", "x0": 10, "y0": 20,
 "x1": 100, "y1": 80, "gc": {}}
```

circle – Circle.

```
{"op": "circle", "x": 50, "y": 50, "r": 25, "gc": {}}
```

text – Text string.

```
{"op": "text", "x": 100, "y": 200, "str": "Hello",
 "rot": 0, "hadj": 0.5, "gc": {}}
```

- `str`: The text content (string).
- `rot`: Rotation angle in degrees (counter-clockwise).
- `hadj`: Horizontal adjustment (0 = left-aligned, 0.5 = centered, 1 = right-aligned).

path – Complex path with subpaths and a fill rule.

```
{"op": "path", "winding": "nonzero",
 "subpaths": [[[10, 20], [30, 40], [50, 20]]],
 "gc": {}}
```

- `winding`: Fill rule: `"nonzero"` or `"evenodd"`.
- `subpaths`: Array of subpaths. Each subpath is an array of `[x, y]` coordinate pairs.

raster – Raster image. No `gc`.

```
{"op": "raster", "x": 0, "y": 576, "w": 100, "h": -80,
 "rot": 0, "interpolate": true,
 "pw": 200, "ph": 160,
 "data": "..."}
```

- `x`, `y`: Bottom-left corner of the destination rectangle in device coordinates.
- `w`, `h`: Displayed width and height. May be **negative** to indicate a horizontal or vertical flip; renderers should use the absolute value for sizing and adjust the anchor point accordingly.
- `pw`, `ph`: Pixel width and height of the source image.
- `rot`: Rotation angle in degrees.
- `interpolate`: Whether to interpolate when scaling.
- `data`: Base64-encoded PNG as a data URI.

beginGroup – Start a drawing group (experimental). No `gc`.

```
{"op": "beginGroup",
 "ext": {"filter": "blur(5px)", "opacity": 0.8}}
```

- `ext` (object, optional): Group-level extension data. Present only when set. Free-form JSON. Common keys: `filter` (CSS filter string), `opacity` (number 0–1), `blendMode` (CSS blend mode string).

endGroup – End the most recently opened group. No `gc`, no fields other than `"op"`.

```
{"op": "endGroup"}
```

Groups nest arbitrarily.

Resize protocol

The server receives resize messages from the renderer and forwards them to R. R replays the drawing at the new dimensions and sends back a frame message.

Normal resize flow:

```
Renderer -> Server:  {"type":"resize","width":800,
                      "height":600}
Server   -> R:       {"type":"resize","width":800,
                      "height":600}
R        -> Server:  {"type":"frame",
                      "resizeReplay":true,
                      "incremental":false,...}
```

History resize flow (replay a historical plot):

```
Renderer -> Server:  {"type":"resize","width":800,
                      "height":600,"plotIndex":2,
                      "sessionId":"r-1234-1"}
Server   -> R:       {"type":"resize","width":800,
                      "height":600,"plotIndex":2}
R        -> Server:  {"type":"frame",
                      "resizeReplay":true,
                      "plotIndex":2,
                      "incremental":false,...}
```

Note: The server strips `sessionId` before forwarding to R. History resizes are routed only to the R session that owns the target plot.

Resize deduplication:

Servers should deduplicate consecutive normal resizes with identical dimensions for each R session. However, if the previous resize was a `plotIndex` resize, the next normal resize at the same dimensions must NOT be deduplicated, because they target different contexts (historical snapshot vs. current plot).

Multiple R sessions

A server may accept connections from multiple R processes simultaneously. Each R connection has its own `sessionId` and independent state. Servers should:

- Route `metrics_request` messages from each R connection to the renderer, and route the matching `metrics_response` back to the originating R session. The `id` field is only unique within a single R process, so servers must scope routing by the originating connection (e.g. keyed by session + `id`). If a server remaps IDs when forwarding to a shared renderer, it must restore the original `id` when relaying the response back to R.
- Route `plotIndex` resizes to the R session that owns the target plot (identified by `sessionId` in the resize message).
- Broadcast normal resizes to all connected R sessions.
- Broadcast `frame` and `close` messages to all connected renderers.

Session ID management

The `sessionId` in frame messages identifies the R device instance. Servers should treat it as an **opaque string**. Do not parse it or make assumptions about its format.

Session ID reuse is unlikely but possible (e.g., after process restart). As a defensive measure, servers should disambiguate if a previously retired `sessionId` reappears, to prevent `plotIndex` resizes for old plots from reaching the new connection. The server's (possibly remapped) `sessionId` is what the renderer sees; `plotIndex` resizes use it for routing.

Connection lifecycle

Graceful close: The client sends `{"type":"close"}` on device shutdown. The server should forward this to renderers and clean up routing state for that session.

Ungraceful disconnect: If the connection drops without a `close` message (e.g., process crash), the server should detect the broken connection (EOF or socket error), clean up the session, and optionally notify renderers.

Incomplete lines: If a connection drops mid-line (no trailing `\n`), the partial data should be discarded.

Extension fields

Extension fields (`ext`) appear at three levels:

- **Frame-level**: Top-level `ext` on the frame message. Applies to the entire frame.
- **Graphics context level**: `ext` inside the `gc` object. Applies to all drawing operations while active.
- **Group level**: `ext` on `beginGroup` operations. Applies only to that group.

All `ext` fields are free-form JSON objects. When unset, the field is omitted from the message (never sent as `null`). When set, it may contain any JSON object, including an empty `{}`. Servers should preserve and forward them to renderers without validation. Renderers should ignore unknown keys.

Extension fields are preserved across resize replays, so historical plot snapshots retain their `ext` data.

See `jgd_ext()`, `jgd_frame_ext()`, and `jgd_begin_group()` for the R API to set these fields.

Implementing a server

A minimal server implementation needs to:

1. Listen on a Unix socket, named pipe, or TCP port.
2. Accept R connections and read JSONL lines.
3. Send a deferred `server_info` welcome after receiving the first message from R (not before).
4. Forward `frame` messages to connected renderers.
5. Forward `resize` messages from renderers to R.
6. Handle `metrics_request`/`metrics_response` routing between R and the renderer. Clients handle their own timeouts; servers are not required to synthesize fallback responses.

7. Forward `close` messages to renderers and clean up the session state (remove metrics routing entries, etc.).

Optional:

- Write a discovery file on startup, remove on shutdown.
- Implement resize deduplication.
- Implement session ID remapping for reused IDs.
- Serve an HTTP endpoint for the renderer UI.

See Also

`jgd()`, `jgd_server_info()`, `jgd_ext()`, `jgd_frame_ext()`, `jgd_begin_group()`

`with_jgd_ext` *Scoped extended graphics context (experimental)*

Description

Temporarily sets extension fields for the duration of `expr`, then clears ext on exit (sets to `NULL`).

Usage

```
with_jgd_ext(json, expr)
```

Arguments

`json`	A single JSON string representing the extension object. Must be valid JSON; an error is raised otherwise. Unlike `jgd_ext()`, `NULL` is not accepted (use `jgd_ext(NULL)` to clear ext explicitly).
`expr`	Expression to evaluate with the extension active.

Value

The result of evaluating `expr`.

Lifecycle

Experimental. This API may change in future versions.

with_jgd_frame_ext *Scoped frame-level extension fields (experimental)*

Description

Temporarily sets frame-level extension fields for the duration of `expr`, then clears them on exit.

Usage

```
with_jgd_frame_ext(json, expr)
```

Arguments

`json`	A single JSON string representing the extension object.
`expr`	Expression to evaluate with the frame extension active.

Value

The result of evaluating `expr`.

Lifecycle

Experimental. This API may change in future versions.

with_jgd_group *Scoped drawing group (experimental)*

Description

Opens a drawing group with extension fields, evaluates `expr`, then closes the group on exit.

Usage

```
with_jgd_group(ext, expr)
```

Arguments

`ext`	A single JSON string with extension fields for this group, or `NULL` for a group without extension fields.
`expr`	Expression to evaluate within the group.

Value

The result of evaluating `expr`.

Lifecycle

Experimental. This API may change in future versions.

Index