Package: fastgeojson (via r-universe)

May 19, 2026

Title High-Performance 'GeoJSON' and 'JSON' Serialization for R

Version 0.2.2

Description A high-performance 'JSON' encoder implemented in 'Rust'. It provides a universal 'as_json()' function capable of serializing most R objects -- including 'sf' spatial data, data frames, and nested lists. Leveraging a performant 'Rust' backend, it generates compliant 'JSON' strings optimized for direct use in web frameworks like 'Shiny', 'Leaflet', and 'Highcharter'.

License MIT + file LICENSE

URL https://github.com/firstzeroenergy/fastgeojson

BugReports https://github.com/firstzeroenergy/fastgeojson/issues

SystemRequirements Cargo (Rust's package manager), rustc

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0), jsonlite, sf, leaflet

Config/testthat/edition 3

Config/rextendr/version 0.4.2

Config/pak/sysreqs libclang-dev

Repository https://firstzeroenergy.r-universe.dev

Date/Publication 2026-01-18 23:58:58 UTC

RemoteUrl https://github.com/firstzeroenergy/fastgeojson

RemoteRef HEAD

RemoteSha a00a2548745d3e7fb818faa02e957829f14d5212

Contents

fastgeojson *Fast JSON and GeoJSON serialization for R*

Description

fastgeojson provides a high-performance serialization backend for converting R data structures into JSON strings. The core encoders are implemented in Rust using the **extendr** framework and are designed to efficiently handle large spatial datasets, tabular data, and generic R objects.

Recommended Usage: All users should use the as_json() function. It acts as a universal "omnivore" that automatically detects the input type (sf object, data frame, list, or vector) and dispatch it to the correct high-performance Rust encoder.

The resulting JSON is returned as a character string with an appropriate class ("geojson" / "json"), allowing it to be passed directly to client-side JavaScript libraries or web frameworks (like Shiny or Plumber) without additional serialization steps.

Usage

```
as_json(
  x,
  auto_unbox = FALSE,
  dataframe = c("rows", "columns"),
  na = NULL,
  null = c("list", "null")
)

sf_geojson_str(
  x,
  auto_unbox = FALSE,
  na = c("null", "string"),
  null = c("list", "null")
)

df_json_str(
  x,
  auto_unbox = FALSE,
  dataframe = c("rows", "columns"),
  na = c("null", "string"),
  null = c("list", "null")
)
```

Arguments

x	An input object (e.g., a data.frame, sf object, list, or vector) to serialize.
auto_unbox	Logical. If TRUE, atomic vectors of length 1 are automatically unboxed into scalar JSON values (e.g., [1] becomes 1). If FALSE (default), they remain as single-element arrays (e.g., [1]).

`dataframe` Character. Defines the output structure for data frames: `"rows"` (default) output as an array of objects (`[{...}]`), `"columns"` output as an object of arrays (`{...}`).

`na` Character. Controls how `NA` values are serialized:

- **Default:** Uses "smart" logic (see Type Handling section).
- **"null":** Forces all `NA` values to be serialized as JSON `null`.
- **"string":** Forces all `NA` values to be serialized as `"NA"`.

`null` Character. Controls how `NULL` values (in lists) are serialized: `"list"` (default) maps to `[]` (empty array) or `{}` (empty object) depending on context. `"null"` maps to JSON `null`.

Details

For sufficiently large inputs (specifically Data Frames and Spatial objects), encoding is performed in parallel using multiple CPU cores via the Rust **rayon** library.

Performance Note: There is **no material performance penalty** for using `as_json()` compared to the specialized underlying functions. The internal dispatch mechanism has negligible overhead. Users are strongly encouraged to use `as_json()` exclusively.

Value

A length-one character vector containing the JSON string with class `"json"`. If the input is an `sf` object, the class is `c("geojson", "json")`.

Type handling

- **Numeric:** Written as JSON numbers. Infinite/NaN values are determined by the `na` argument.
- **Logical:** Written as JSON booleans.
- **Character:** Written as JSON strings with UTF-8 escaping.
- **Matrix:** Serialized row-major as an array of arrays.
- **Factor:** Encoded using their character levels.

Missing Value (NA) Handling

`fastgeojson` employs "Smart" defaults to match standard R JSON conventions (specifically `jsonlite`), while offering strict overrides via the `na` argument:

- **Default (Smart):**
  - In **Column Mode** (`dataframe="columns"`), numeric `NA`s are converted to `"NA"` strings to maintain array type homogeneity. Non-numeric `NA`s become `null`.
  - In **Row Mode** (`dataframe="rows"`), `NA` values are usually **omitted** from the object to reduce payload size.
- **Explicit (`na="null"`):** Forces all `NA` values (numeric or otherwise) to be serialized as JSON `null`.
- **Explicit (`na="string"`):** Forces all `NA` values to be serialized as `"NA"`.

See Also

as_json() - The primary function for all serialization tasks.

Examples

```
# 1. Generic Objects
as_json(list(a = 1, b = "foo", c = NA))

# 2. Auto-unbox
as_json(list(val = 5), auto_unbox = TRUE)  # {"val":5}
as_json(list(val = 5), auto_unbox = FALSE) # {"val":[5]}

# 3. Data Frames (Row vs Column orientation)
df <- data.frame(x = 1:2, y = c("a", NA))
as_json(df, dataframe = "rows")    # [{"x":1,"y":"a"},{"x":2}]  (NA omitted)
as_json(df, dataframe = "columns") # {"x":[1,2],"y":["a",null]}

# 4. Controlling NA serialization
# Force NA to null in all contexts
as_json(c(1, NA, 3), na = "null") # [1, null, 3]
# Force NA to string
as_json(c(1, NA, 3), na = "string") # [1, "NA", 3]

# 5. Spatial Data (sf)
if (requireNamespace("sf", quietly = TRUE)) {
     nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
     # automatically detects sf and outputs GeoJSON FeatureCollection
     geo_str <- as_json(nc[1:3, ])
}
```

Index