Package ‘fastgeojson’

January 23, 2026

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

Version 0.1.3

Description Converts R data frames and 'sf' spatial objects into 'JSON' and 'GeoJSON' strings. The core encoders are implemented in 'Rust' using the 'extendr' framework and are designed to efficiently serialize large tabular and spatial datasets. Returns serialized 'JSON' text, allowing applications such as 'shiny' or web APIs to transfer data to client-side 'JavaScript' libraries without additional encoding overhead.

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

RoxygenNote 7.3.3

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

Config/testthat/edition 3

Config/rextendr/version 0.4.2

NeedsCompilation yes

Author Alex Jorion [aut, cre],
The authors of the dependency Rust crates [ctb] (see inst/AUTHORS file for details)

Maintainer Alex Jorion <alex.jorion@firstzeroenergy.com>

Repository CRAN

Date/Publication 2026-01-23 19:50:23 UTC

Contents

`fastgeojson` *High-Performance 'GeoJSON' and 'JSON' Serialization*

Description

`fastgeojson` provides a high-performance serialization backend for converting common 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 and tabular datasets.

The package focuses on two primary use cases:

- Converting `sf` objects into 'GeoJSON' FeatureCollections.
- Converting rectangular `data.frame` objects into 'JSON' arrays.

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 without additional serialization steps.

Usage

```
sf_geojson_str(x)

df_json_str(x)
```

Arguments

`x`	An input object (e.g., a data.frame or sf object) to serialize.

Details

For sufficiently large inputs, encoding may be performed in parallel using multiple CPU cores via the 'rayon' library in 'Rust'. Parallel execution is enabled automatically based on input size and geometry type.

By returning pre-serialized 'JSON' strings, these functions allow frameworks such as 'Shiny' and 'Plumber' to avoid redundant re-encoding during data transfer, which can significantly reduce server-side overhead in interactive or high-throughput applications.

Value

For `sf_geojson_str()`, a length-one character vector with class `c("geojson", "json")` containing a 'GeoJSON' FeatureCollection string. For `df_json_str()`, a length-one character vector with class `"json"` containing a 'JSON' array string.

Type handling

- **Numeric:** Written as JSON numbers.
- **Logical:** Written as JSON booleans.
- **Character:** Written as JSON strings with UTF-8 escaping.
- **Factor:** Encoded using their character levels.
- **Missing values:** Missing (`NA`) values may be omitted from output objects to reduce payload size.
- **Geometries:** Supports common `sf` geometry types including POINT, LINESTRING, POLYGON, and their MULTI variants.

See Also

`sf_geojson_str()` for spatial data, `df_json_str()` for tabular data.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
   nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
   geo_str <- sf_geojson_str(nc)
}

df <- data.frame(x = runif(10), y = runif(10))
json_str <- df_json_str(df)
```

Index