Package 'fastgeojson'

September 15, 2026

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

Version 0.3.0

Description Converts R objects such as data frames, lists and vectors into 'JSON' strings, and 'sf' spatial objects into 'GeoJSON'. 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

Depends R (>= 4.5)

SystemRequirements Cargo (Rust's package manager), rustc

Encoding UTF-8

Language en-US

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

Config/testthat/edition 3

Config/rextendr/version 0.4.2

Config/roxygen2/version 8.1.0

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 2026-09-15 01:50:02 UTC

Contents

fastgeojson 2
fastgeojson_threads 4

Index **6**

`fastgeojson` *Fast Serialization of R Objects to JSON and GeoJSON*

Description

`as_json()` serializes R objects to JSON, and `sf` objects to GeoJSON, in parallel and with lossless numbers. The encoders are implemented in Rust via **extendr**.

It takes the arguments of `jsonlite::toJSON()`, in the same order, and follows jsonlite's output conventions: jsonlite's own test suite runs against it. Two defaults differ: numbers are written losslessly (`digits = Inf`, where `toJSON()` rounds to 4 decimal places) and `sf` objects become GeoJSON (`sf = "geojson"`, where `toJSON()` writes a record array).

Usage

```
as_json(
  x,
  dataframe = c("rows", "columns", "values"),
  matrix = c("rowmajor", "columnmajor"),
  Date = c("ISO8601", "epoch"),
  POSIXt = c("string", "ISO8601", "epoch", "mongo"),
  factor = c("string", "integer"),
  complex = c("string", "list"),
  raw = c("base64", "hex", "mongo", "int", "js"),
  null = c("list", "null"),
  na = c("null", "string"),
  auto_unbox = FALSE,
  digits = Inf,
  pretty = FALSE,
  force = FALSE,
  ...
)
```

Arguments

`x`	The object to serialize.
`dataframe`	How to encode data frames: `"rows"` (default, `[{"a":1},{"a":2}]`), `"columns"` (`{"a":[1,2]}`) or `"values"` (`[[1],[2]]`, one array per row without names).
`matrix`	How to encode matrices: `"rowmajor"` (default) or `"columnmajor"`.
`Date`	How to encode `Date`: `"ISO8601"` (default, `"2015-01-01"`) or `"epoch"` (days since 1970-01-01).

POSIXt	How to encode date-times: "string" (default, R's own format()), "ISO8601", "epoch" (milliseconds) or "mongo".
factor	How to encode factors: "string" (default) or "integer".
complex	How to encode complex numbers: "string" (default, "1+2i") or "list" ({"real":[..],"imaginary":
raw	How to encode raw vectors: "base64" (default), "hex", "mongo", "int" or "js".
null	How to encode NULL inside lists: "list" (default, {}) or "null".
na	How to encode missing values: "null" or "string". When not supplied, type-specific defaults apply – in particular, row-oriented data frame output omits the key entirely, matching toJSON().
auto_unbox	If TRUE, length-one atomic vectors become JSON scalars rather than length-one arrays. Defaults to FALSE.
digits	Precision of numeric values. The default, Inf, writes the shortest decimal that reads back as the same double, so nothing is lost. A number is a count of decimal places (toJSON()'s default is 4); wrap it in base::I() to count *significant* digits instead; NA is toJSON()'s 15 significant digits.
pretty	If TRUE, indent the output by two spaces; a number sets the indent width.
force	If TRUE, strip S3 classes that would otherwise raise an error. sf objects are exempt so geometry handling is preserved.
...	Further arguments passed to the encoder, mirroring jsonlite::toJSON(). Recognised here: keep_vec_names, rownames, json_verbatim, UTC, time_format, always_decimal, use_signif, indent, sf and as_bytes. as_bytes = TRUE, which is not a toJSON() argument, returns the same bytes as a **raw vector**, skipping R's string interning – most of a large call. Use it when the JSON is leaving R: writeBin() to a file or connection, an HTTP response body. It cannot be combined with pretty.

Details

Coming from jsonlite. Code that calls toJSON() with named arguments can call as_json() with the same ones; pass digits = 4 (and sf = "dataframe" for sf input) to reproduce toJSON()'s defaults. Unknown arguments are accepted through ..., as toJSON() does.

Encoding strategy. as_json() inspects x and dispatches to the appropriate Rust encoder:

- **Simple features** (sf)**:** a GeoJSON FeatureCollection.
- **Data frames:** row- or column-oriented, processed in parallel in row chunks for large inputs.
- **Atomic vectors and lists:** JSON arrays and objects, recursively.

Type mapping.

- **Numeric:** JSON numbers, exact by default; see digits. NA, NaN, Inf and -Inf follow the na argument.
- **Logical:** true / false.
- **Character:** JSON strings. latin1- and native-encoded inputs are translated to UTF-8; bytes that are not valid UTF-8 pass through unchanged, as toJSON() passes them, and a "bytes"-marked string raises the error toJSON() raises.

- **Factor:** labels, or integer codes when `factor = "integer"`.
- **Date / POSIXt:** controlled by `Date` and `POSIXt`.
- **complex / raw:** controlled by `complex` and `raw`.

Threads. See `fastgeojson_threads()` to control parallelism.

Value

A length-one character vector of class `"json"`, or `c("geojson", "json")` for `sf` input – unless `as_bytes = TRUE`, which returns an unclassed raw vector holding the same bytes.

See Also

`fastgeojson_threads()` to control parallelism.

Examples

```
as_json(list(a = 1, b = "foo", c = NA))
as_json(list(val = 5), auto_unbox = TRUE)

df <- data.frame(x = c(1.5, 2.5), y = c("a", "b"))
as_json(df)
as_json(df, dataframe = "columns")

# Straight out to a file, without interning the result as an R string.
f <- tempfile()
con <- file(f, "wb")
writeBin(as_json(df, as_bytes = TRUE), con)
close(con)
unlink(f)

if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geo <- as_json(nc[1:2, ])
}
```

`fastgeojson_threads` *Get or set the number of worker threads*

Description

Controls how many threads the Rust backend uses for parallel serialization.

Normal use gets the whole machine. The count is resolved in this order: an explicit value set here; then `FASTGEOJSON_NUM_THREADS`, `RAYON_NUM_THREADS`, `OMP_NUM_THREADS` or `OMP_THREAD_LIMIT`; then two if `R CMD check` is detected (CRAN's policy caps *checking* at two cores, since the check farm is shared – it places no limit on ordinary use); then the number of available cores.

Usage

```
fastgeojson_threads(n = NULL)
```

Arguments

n	Integer. The number of threads to use. `n = 1` disables parallelism entirely, which is useful for reproducible benchmarking. `n <= 0` restores the automatic behaviour described above. `n = NULL` (default) changes nothing and reports the current value.

Value

An integer scalar, the number of worker threads in use after the call. Returned invisibly when `n` is supplied; visibly when it is not.

Examples

```
fastgeojson_threads()
old <- fastgeojson_threads(2)
fastgeojson_threads(0)
```

Index

as_json *(*fastgeojson*)*, 2

base::I(), *3*

fastgeojson, 2
fastgeojson_threads, 4
fastgeojson_threads(), *4*

jsonlite::toJSON(), *2*