Package 'jsonify'

June 2, 2020

Type Package

Title Convert Between 'R' Objects and Javascript Object Notation (JSON)

Version 1.2.1

Date 2020-06-02

Description Conversions between 'R' objects and Javascript Object Notation (JSON) using the 'rapidjsonr' library <https://CRAN.R-project.org/package=rapidjsonr>.

License MIT + file LICENSE

Depends R (>= 3.3.0)

SystemRequirements C++11

Imports Rcpp (>= 0.12.18)

LinkingTo rapidjsonr (>= 1.2.0), Rcpp

RoxygenNote 7.1.0

Suggests covr, testthat, knitr, rmarkdown

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation yes

Author David Cooley [aut, cre],
Chris Muir [ctb],
Brendan Knapp [ctb] (<https://orcid.org/0000-0003-3284-4972>)

Maintainer David Cooley <dcooley@symbolix.com.au>

Repository CRAN

Date/Publication 2020-06-02 08:50:21 UTC

R topics documented:

`as.json` *Coerce string to JSON*

Description

Coerce string to JSON

Usage

```
as.json(x)
```

Arguments

`x`	string to coerce to JSON

Examples

```
js <- '{"x":1,"y":2}'
as.json(js)
```

`from_json` *From JSON*

Description

Converts JSON to an R object.

Usage

```
from_json(json, simplify = TRUE, fill_na = FALSE, buffer_size = 1024)
```

Arguments

`json`	JSON to convert to R object. Can be a string, url or link to a file.
`simplify`	logical, if `TRUE`, coerces JSON to the simplest R object possible. See Details
`fill_na`	logical, if `TRUE` and `simplify` is `TRUE`, data.frames will be na-filled if there are missing JSON keys. Ignored if `simplify` is `FALSE`. See details and examples.
`buffer_size`	size of buffer used when reading a file from disk. Defaults to 1024

Details

When `simplify = TRUE`

- single arrays are coerced to vectors
- array of arrays (all the same length) are coerced to matrices
- objects with the same keys are coerced to data.frames

When `simplify = TRUE` and `fill_na = TRUE`

- objects are coerced to data.frames, and any missing values are filled with NAs

Examples

```
from_json('{"a":[1, 2, 3]}')
from_json('{"a":8, "b":99.5, "c":true, "d":"cats", "e":[1, "cats", 3]}')
from_json('{"a":8, "b":{"c":123, "d":{"e":456}}}')

lst <- list("a" = 5L, "b" = 1.43, "c" = "cats", "d" = FALSE)
js <- jsonify::to_json(lst, unbox = TRUE)
from_json( js )

## Return a data frame
from_json('[{"id":1,"val":"a"},{"id":2,"val":"b"}]')

## Return a data frame with a list column
from_json('[{"id":1,"val":"a"},{"id":2,"val":["b","c"]}]')

## Without simplifying to a data.frame
from_json('[{"id":1,"val":"a"},{"id":2,"val":["b","c"]}]', simplify = FALSE )

## Missing JSON keys
from_json('[{"x":1},{"x":2,"y":"hello"}]')

## Missing JSON keys - filling with NAs
from_json('[{"x":1},{"x":2,"y":"hello"}]', fill_na = TRUE )

## Duplicate object keys
from_json('[{"x":1,"x":"a"},{"x":2,"x":"b"}]')

from_json('[{"id":1,"val":"a","val":1},{"id":2,"val":"b"}]', fill_na = TRUE )
```

`from_ndjson` *from ndjson*

Description

Converts ndjson into R objects

Usage

```
from_ndjson(ndjson, simplify = TRUE, fill_na = FALSE)
```

Arguments

ndjson	new-line delimited JSON to convert to R object. Can be a string, url or link to a file.
simplify	logical, if TRUE, coerces JSON to the simplest R object possible. See Details
fill_na	logical, if TRUE and simplify is TRUE, data.frames will be na-filled if there are missing JSON keys. Ignored if simplify is FALSE. See details and examples.

Examples

```
js <- to_ndjson( data.frame( x = 1:5, y = 6:10 ) )
from_ndjson( js )
```

---

minify_json *Minify Json*

Description

Removes indentiation from a JSON string

Usage

```
minify_json(json, ...)
```

Arguments

json	string of JSON
...	other argments passed to to_json

Examples

```
df <- data.frame(id = 1:10, val = rnorm(10))
js <- to_json( df )
jsp <- pretty_json(js)
minify_json( jsp )
```

pretty_json *Pretty Json*

Description

Adds indentiation to a JSON string

Usage

```
pretty_json(json, ...)
```

Arguments

json	string of JSON
...	other argments passed to to_json

Examples

```
df <- data.frame(id = 1:10, val = rnorm(10))
js <- to_json( df )
pretty_json(js)

## can also use directly on an R object
pretty_json( df )
```

to_json *To JSON*

Description

Converts R objects to JSON

Usage

```
to_json(
  x,
  unbox = FALSE,
  digits = NULL,
  numeric_dates = TRUE,
  factors_as_string = TRUE,
  by = "row"
)
```

Arguments

x	object to convert to JSON
unbox	logical indicating if single-value arrays should be 'unboxed', that is, not contained inside an array.
digits	integer specifying the number of decimal places to round numerics. Default is NULL - no rounding
numeric_dates	logical indicating if dates should be treated as numerics. Defaults to TRUE for speed. If FALSE, the dates will be coerced to character in UTC time zone
factors_as_string	logical indicating if factors should be treated as strings. Defaults to TRUE.
by	either "row" or "column" indicating if data.frames and matrices should be processed row-wise or column-wise. Defaults to "row"

Examples

```
to_json(1:3)
to_json(letters[1:3])

## factors treated as strings
to_json(data.frame(x = 1:3, y = letters[1:3], stringsAsFactors = TRUE ))
to_json(data.frame(x = 1:3, y = letters[1:3], stringsAsFactors = FALSE ))

to_json(list(x = 1:3, y = list(z = letters[1:3])))
to_json(seq(as.Date("2018-01-01"), as.Date("2018-01-05"), length.out = 5))
to_json(seq(as.Date("2018-01-01"), as.Date("2018-01-05"), length.out = 5), numeric_dates = FALSE)

psx <- seq(
  as.POSIXct("2018-01-01", tz = "Australia/Melbourne"),
  as.POSIXct("2018-02-01", tz = "Australia/Melbourne"),
  length.out = 5
  )
to_json(psx)
to_json(psx, numeric_dates = FALSE)

## unbox single-value arrays
to_json(list(x = 1), unbox = TRUE)
to_json(list(x = 1, y = c("a"), z = list(x = 2, y = c("b"))), unbox = TRUE)

## rounding numbers using the digits argument
to_json(1.23456789, digits = 2)
df <- data.frame(x = 1L:3L, y = rnorm(3), z = letters[1:3], stringsAsFactors = TRUE )
to_json(df, digits = 0 )

## keeping factors
to_json(df, digits = 2, factors_as_string = FALSE )
```

`to_ndjson` *To ndjson*

Description

Converts R objects to ndjson

Usage

```
to_ndjson(
  x,
  unbox = FALSE,
  digits = NULL,
  numeric_dates = TRUE,
  factors_as_string = TRUE,
  by = "row"
)
```

Arguments

`x`	object to convert to JSON
`unbox`	logical indicating if single-value arrays should be 'unboxed', that is, not contained inside an array.
`digits`	integer specifying the number of decimal places to round numerics. Default is `NULL` - no rounding
`numeric_dates`	logical indicating if dates should be treated as numerics. Defaults to TRUE for speed. If FALSE, the dates will be coerced to character in UTC time zone
`factors_as_string`	logical indicating if factors should be treated as strings. Defaults to TRUE.
`by`	either "row" or "column" indicating if data.frames and matrices should be processed row-wise or column-wise. Defaults to "row"

Details

Lists are converted to ndjson non-recursively. That is, each of the objects in the list at the top level are converted to a new-line JSON object. Any nested sub-elements are then contained within that JSON object. See examples

Examples

```
to_ndjson( 1:5 )
to_ndjson( letters )

mat <- matrix(1:6, ncol = 2)
```

```
to_ndjson( x = mat )
to_ndjson( x = mat, by = "col" )

df <- data.frame(
  x = 1:5
  , y = letters[1:5]
  , z = as.Date(seq(18262, 18262 + 4, by = 1 ), origin = "1970-01-01" )
  )

to_ndjson( x = df )
to_ndjson( x = df, numeric_dates = FALSE )
to_ndjson( x = df, factors_as_string = FALSE )
to_ndjson( x = df, by = "column" )
to_ndjson( x = df, by = "column", numeric_dates = FALSE )

## Lists are non-recurisve; only elements `x` and `y` are converted to ndjson
lst <- list(
  x = 1:5
  , y = list(
    a = letters[1:5]
    , b = data.frame(i = 10:15, j = 20:25)
  )
)

to_ndjson( x = lst )
to_ndjson( x = lst, by = "column")
```

`validate_json` *validate JSON*

Description

Validates JSON

Usage

```
validate_json(json)
```

Arguments

`json`	character or json object

Value

logical vector

Examples

```
validate_json('[]')
df <- data.frame(id = 1:5, val = letters[1:5])
validate_json( to_json(df) )

validate_json('{"x":1,"y":2,"z":"a"}')

validate_json( c('{"x":1,"y":2,"z":"a"}', to_json(df) ) )
validate_json( c('{"x":1,"y":2,"z":a}', to_json(df) ) )
```

Index