Package 'rjsoncons'

December 4, 2023

Title 'C++' Header-Only 'jsoncons' Library for 'JSON' Queries

Version 1.0.1

Description The 'jsoncons' <https://danielaparker.github.io/jsoncons/> 'C++' header-only library constructs representations from a 'JSON' character vector, and provides extensions for flexible queries and other operations on 'JSON' objects. This package has simple 'R' wrappers to support 'JSONpath' and 'JMESpath' queries into 'JSON' strings or 'R' objects. The 'jsoncons' library is also be easily linked to other packages for direct access to 'C++' functionality.

Imports jsonlite

Suggests tinytest, BiocStyle, knitr, rmarkdown

License BSL-1.0

LinkingTo cpp11

NeedsCompilation yes

Encoding UTF-8

BugReports https://github.com/mtmorgan/rjsoncons/issues

RoxygenNote 7.2.3

VignetteBuilder knitr

URL https://mtmorgan.github.io/rjsoncons/

Author Martin Morgan [aut, cre] (<https://orcid.org/0000-0002-5874-8148>), Marcel Ramos [aut] (<https://orcid.org/0000-0002-3242-0582>), Daniel Parker [aut, cph] (jsoncons C++ library maintainer)

Maintainer Martin Morgan <mtmorgan.xyz@gmail.com>

Repository CRAN

Date/Publication 2023-12-03 23:40:02 UTC

R topics documented:

version 2

Index **5**

version *Query the jsoncons C++ library*

Description

`version()` reports the version of the C++ jsoncons library in use.

`jsonpath()` executes a query against a JSON string using the 'jsonpath' specification

`jmespath()` executes a query against a JSON string using the 'jmespath' specification.

`as_r()` transforms a JSON string to an *R* object.

Usage

```
version()

jsonpath(data, path, object_names = "asis", as = "string", ...)

jmespath(data, path, object_names = "asis", as = "string", ...)

as_r(data, object_names = "asis", ...)
```

Arguments

`data`	an *R* object. If `data` is a scalar (length 1) character vector, it is treated as a single JSON string. Otherwise, it is parsed to a JSON string using `jsonlite::toJSON()`. Use `I()` to treat a scalar character vector as an *R* object rather than JSON string, e.g., `I("A")` will be parsed to `["A"]` before processing.
`path`	character(1) jsonpath or jmespath query string.
`object_names`	character(1) order `data` object elements `"asis"` (default) or `"sort"` before filtering on `path`.
`as`	character(1) return type. `"string"` returns a single JSON string; `"R"` returns an *R* object following the rules outlined below.
`...`	arguments passed to `jsonlite::toJSON` when `data` is not a scalar character vector. For example, use `auto_unbox = TRUE` to automatically 'unbox' vectors of length 1 to JSON scalar values.

Details

The `as = "R"` argument to `jsonpath()` and `jmespath()` and `as_r()` transform a JSON string representation to an *R* object. Main rules are:

- JSON arrays of a single type (boolean, integer, double, string) are transformed to *R* vectors of the same length and corresponding type. A JSON scalar and a JSON vector of length 1 are represented in the same way in *R*.
- If a JSON 64-bit integer array contains a value larger than *R*'s 32-bit integer representation, the array is transformed to an *R* numeric vector. NOTE that this results in loss of precision for 64-bit integer values greater than `2^53`.

- JSON arrays mixing integer and double values are transformed to *R* numeric vectors.
- JSON objects are transformed to *R* named lists.

The vignette reiterates this information and provides additional details.

Value

`version()` returns a character(1) major.minor.patch version string .

`jsonpath()` and `jmespath()` return a character(1) JSON string (`as = "string"`, default) or *R* object (`as = "R"`) representing the result of the query.

`as_r()` returns an *R* object.

Examples

```
version()

json <- '{
  "locations": [
    {"name": "Seattle", "state": "WA"},
    {"name": "New York", "state": "NY"},
    {"name": "Bellevue", "state": "WA"},
    {"name": "Olympia", "state": "WA"}
  ]
 }'

## return a JSON string
jsonpath(json, "$..name") |>
    cat("\n")

## return an R object
jsonpath(json, "$..name", as = "R")

## create a list with state and name as scalar vectors
lst <- jsonlite::fromJSON(json, simplifyVector = FALSE)

## objects other than scalar character vectors are automatically
## coerced to JSON; use `auto_unbox = TRUE` to represent R scalar
## vectors in the object as JSON scalar vectors
jsonpath(lst, "$..name", auto_unbox = TRUE) |>
    cat("\n")

## a scalar character vector like "Seattle" is not valid JSON...
try(jsonpath("Seattle", "$"))
## ...but a double-quoted string is
jsonpath('"Seattle"', "$")

## use I("Seattle") to coerce to a JSON object ["Seattle"]
jsonpath(I("Seattle"), "$[0]")      |> cat("\n")

## different ordering of object names -- 'asis' (default) or 'sort'
json_obj <- '{"b": "1", "a": "2"}'
jsonpath(json_obj, "$")             |> cat("\n")
```

```
jsonpath(json_obj, "$.*")           |> cat("\n")
jsonpath(json_obj, "$", "sort")   |> cat("\n")
jsonpath(json_obj, "$.*", "sort") |> cat("\n")

path <- "locations[?state == 'WA'].name | sort(@)"
jmespath(json, path) |>
    cat("\n")

## original filter always fails, e.g., '["WA"] != 'WA'
jmespath(lst, path)  # empty result set, '[]'

## filter with unboxed state, and return unboxed name
jmespath(lst, "locations[?state[0] == 'WA'].name[0] | sort(@)") |>
    cat("\n")

## automatically unbox scalar values when creating the JSON string
jmespath(lst, path, auto_unbox = TRUE) |>
    cat("\n")

## as_r()
as_r('[1, 2, 3]')       # JSON integer array -> R integer vector
as_r('[1, 2.0, 3]')     # JSON intger and double array -> R numeric vector
as_r('[1, 2.0, "3"]')   # JSON mixed array -> R list
as_r('[1, 2147483648]') # JSON integer > R integer max -> R numeric vector

json = '{"b": 1, "a": ["c", "d"], "e": true, "f": [true], "g": {}}'
as_r(json) |> str()     # parsing complex objects
identical(              # JSON scalar and length 1 array identical in R
    as_r('{"a": 1}'), as_r('{"a": [1]}')
)
```

Index

as_r *(*version*)*, 2

jmespath *(*version*)*, 2
jsonpath *(*version*)*, 2

version, 2