

R documentation

of all in 'RJSONIO/man/'

November 27, 2009

R topics documented:

asJSVars	1
Bob	2
fromJSON	2
toJSON	5

Index	7
--------------	----------

asJSVars	<i>Serialize R objects as Javascript/ActionScript variables</i>
----------	---

Description

This function takes R objects and serializes them as Javascript/ActionScript values. It uses the specified names in the R call as Javascript variable names. One can also specify qualifiers ('public', 'protected', 'private') and also types. These are optional, but useful, in ActionScript.

Usage

```
asJSVars(..., .vars = list(...), qualifier = character(), types = character())
```

Arguments

...	name = value pairs where the value is an R object that is converted to JSON format and name is the name of the corresponding Javascript variable.
.vars	this is an alternative to ... as a way to specify a collection of name = value pairs that is already in a list.
qualifier	a character vector (recycled as necessary) which is used as qualifiers for the individual ActionScript variables. The values should be public, protected or private.
types	either a logical value or a character vector (which is recycled if necessary). If this is TRUE, then we compute the Javascript type for each of the R objects (using the non-exported function jsType)

Value

A character vector of length 1 giving the variable declarations and initializations.

Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

See Also

[toJSON](#)

Examples

```
cat(asJSVars( a = 1:10, myMatrix = matrix(1:15, 3, 5)))
cat(asJSVars( a = 1:10, myMatrix = matrix(1:15, 3, 5), types = TRUE))
cat(asJSVars( a = 1:10, myMatrix = matrix(1:15, 3, 5),
            qualifier = "protected", types = TRUE))
```

Bob

Symbolic constants identifying the type of a JSON value.

Description

These constants are used by handler functions that are called when a JSON value is encountered by the JSON parser. These identify the type of the JSON value. The values will already have been converted, but the start and end array and object events won't have a type.

Format

A collection of integer values.

Source

JSON_parser.h code from <http://www.json.org>.

References

<http://www.json.org>.

`fromJSON`*Convert JSON content to R objects*

Description

This function and its methods read content in JSON format and de-serializes it into R objects. JSON content is made up of logicals, integers, real numbers, strings, arrays of these and associative arrays/hash tables using `key: value` pairs. These map very naturally to R data types (logical, integer, numeric, character, and named lists).

Usage

```
fromJSON(content, handler = basicJSONHandler(default.size),
         default.size = 100, depth = 150L, allowComments = TRUE,
         asText = isContent(content), data = NULL, ...)
```

Arguments

<code>content</code>	the JSON content. This can be the name of a file or the content itself as a character string. We will add support for connections in the near future.
<code>handler</code>	an R object that is responsible for processing each individual token/element within the JSON content. This can be an R function, a list of functions with class "JSONParserHandler" having <code>update</code> and <code>value</code> elements, or the address of a native (C) routine. In the case of the latter, the <code>data</code> parameter can be used to specify an object that is passed to the C routine each time it is called. This will commonly be an <code>externalptr</code> object.
<code>default.size</code>	a number giving the default buffer size to use for arrays and objects in an effort to avoid reallocating each time we add a new element.
<code>depth</code>	the maximum number of nested JSON levels, i.e. arrays and objects within arrays and objects.
<code>allowComments</code>	a logical value indicating whether to allow C-style comments within the JSON content or to raise an error if they are encountered.
<code>asText</code>	a logical value indicating whether the value of the <code>content</code> argument should be treated as the JSON content, i.e. read directly rather than considered the name of a file.
<code>data</code>	a value that is only used when the value of <code>handler</code> is a native (C) routine. In this case, the value is passed in each call to that C routine by the JSON tokenizer.
<code>...</code>	additional parameters for methods.

Value

An R object created by mapping the JSON content to its R equivalent.

Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

References

<http://www.json.org>

See Also

[toJSON](#) the non-exported collector function {RJSONIO:::basicJSONHandler}.

Examples

```

fromJSON(I(toJSON(1:10)))

fromJSON(I(toJSON(1:10 + .5)))

fromJSON(I(toJSON(c(TRUE, FALSE, FALSE, TRUE))))

x = fromJSON('{ "ok":true,"id":"x123","rev":"1-1794908527" }')

# Reading from a connection. It is a text connection so we could
# just read the text directly, but this could be a dynamic connection.
m = matrix(1:27, 9, 3)
txt = toJSON(m)
con = textConnection(txt)
identical(m, fromJSON(con))

# Use a connection and move the cursor ahead to skip over some lines.
f = system.file("sampleData", "obj1.json", package = "RJSONIO")
con = file(f, "r")
readLines(con, 1)
fromJSON(con)
close(con)

f = system.file("sampleData", "embedded.json", package = "RJSONIO")
con = file(f, "r")
readLines(con, 1) # eat the first line
fromJSON(con, maxNumLines = 4)
close(con)

## Not run:
if(require(rjson)) {
  # We see an approximately a factor of 3.9 speed up when we use
  # this approach that mixes C-level tokenization and an R callback
  # function to gather the results into objects.

  f = system.file("sampleData", "usaPolygons.as", package = "RJSONIO")
  t1 = system.time(a <- RJSONIO:::fromJSON(f))
  t2 = system.time(b <- fromJSON(paste(readLines(f), collapse = "\n")))
}
## End(Not run)
# Use a C routine
fromJSON(I("[1, 2, 3, 4]"),
        getNativeSymbolInfo("R_json_testNativeCallback", "RJSONIO"))

# Use a C routine that populates an R integer vector with the
# elements read from the JSON array. Note that we must ensure
# that the array is big enough.

```

```

fromJSON(I("[1, 2, 3, 4]"),
         getNativeSymbolInfo("R_json_IntegerArrayCallback", PACKAGE = "RJSONIO"),
         data = rep(1L, 5))

x = fromJSON(I("[1.1, 2.2, 3.3, 4.4]"),
            getNativeSymbolInfo("R_json_RealArrayCallback", PACKAGE = "RJSONIO"),
            data = rep(1, 5))
length(x) = 4

# This illustrates a "specialized" handler which knows what it is
# expecting and pre-allocates the answer
# This then populates the answer with the values.
# The speed improvement is 1.8 versus "infinity"!

x = rnorm(1000000)
str = toJSON(x, digits = 6)

fromJSON(I(str),
         getNativeSymbolInfo("R_json_RealArrayCallback", PACKAGE = "RJSONIO"),
         data = numeric(length(x)))

# This is another example of very fast reading of specific JSON.
x = matrix(rnorm(1000000), 1000, 1000)
str = toJSON(x, digits = 6)

v = fromJSON(I(str),
            getNativeSymbolInfo("R_json_RealArrayCallback", PACKAGE = "RJSONIO"),
            data = matrix(0, 1000, 1000))

```

toJSON

Convert an R object to a string in Javascript Object Notation

Description

This function and its methods convert an R object into a string that represents the object in Javascript Object Notation (JSON).

Usage

```
toJSON(x, container = length(x) > 1 || length(names(x)) > 0, ...)
```

Arguments

<code>x</code>	the R object to be converted to JSON format
<code>...</code>	additional arguments controlling the formatting of the JSON.
<code>container</code>	a logical value indicating whether to treat the object as a vector/container or a scalar and so represent it as an array or primitive in JavaScript.

Value

A string containing the JSON content.

Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

References

<http://www.json.org>

See Also

[fromJSON](#)

Examples

```
toJSON(1:10)
toJSON(rnorm(3))
toJSON(rnorm(3), digits = 4)

toJSON(c("Duncan", "Temple Lang"))

toJSON(c(FALSE, FALSE, TRUE))

# List of elements
toJSON(list(1L, c("a", "b"), c(FALSE, FALSE, TRUE), rnorm(3)))
# with digits controlling formatting of sub-elements
toJSON(list(1L, c("a", "b"), c(FALSE, FALSE, TRUE), rnorm(3)),
        digits = 10)

# nested lists
toJSON(list(1L, c("a", "b"), list(c(FALSE, FALSE, TRUE), rnorm(3))))

# with names
toJSON(list(a = 1L, c("a", "b"), c(FALSE, FALSE, TRUE), rnorm(3)))

setClass("TEMP", representation(a = "integer", xyz = "logical"))
setClass("TEMP1", representation(one = "integer", two = "TEMP"))

new("TEMP1", one = 1:10, two = new("TEMP", a = 4L, xyz = c(TRUE, FALSE)))
```

Index

*Topic IO

asJSVars, 1
fromJSON, 2
toJSON, 5

*Topic datasets

Bob, 2

*Topic programming

asJSVars, 1
fromJSON, 2
toJSON, 5

asJSVars, 1

Bob, 2

fromJSON, 2, 5
fromJSON, AsIs-method (*fromJSON*), 2
fromJSON, character-method
 (*fromJSON*), 2
fromJSON, connection-method
 (*fromJSON*), 2

JSON_T_ARRAY_BEGIN (*Bob*), 2
JSON_T_ARRAY_END (*Bob*), 2
JSON_T_FALSE (*Bob*), 2
JSON_T_FLOAT (*Bob*), 2
JSON_T_INTEGER (*Bob*), 2
JSON_T_KEY (*Bob*), 2
JSON_T_MAX (*Bob*), 2
JSON_T_NONE (*Bob*), 2
JSON_T_NULL (*Bob*), 2
JSON_T_OBJECT_BEGIN (*Bob*), 2
JSON_T_OBJECT_END (*Bob*), 2
JSON_T_STRING (*Bob*), 2
JSON_T_TRUE (*Bob*), 2

toJSON, 2, 3, 5
toJSON, ANY-method (*toJSON*), 5
toJSON, character-method (*toJSON*),
 5
toJSON, hexmode-method (*toJSON*), 5
toJSON, integer, missing-method
 (*toJSON*), 5
toJSON, integer-method (*toJSON*), 5
toJSON, list-method (*toJSON*), 5

toJSON, logical-method (*toJSON*), 5
toJSON, matrix-method (*toJSON*), 5
toJSON, name-method (*toJSON*), 5
toJSON, NULL-method (*toJSON*), 5
toJSON, numeric-method (*toJSON*), 5