json Documentation

Release 0.1.4

Recep Aslantas

Apr 09, 2022

Getting Started:

1 Features **3**

2 Build json **5**
- 2.1 Unix (Autotools): . . . 5
- 2.2 Windows (MSBuild): . . . 5
- 2.3 Documentation (Sphinx): . . . 6

3 Getting Started **7**
- 3.1 Allocations: . . . 7
- 3.2 Design and Data Structure: . . . 7

4 API documentation **9**
- 4.1 parse json . . . 9
- 4.2 utils / helpers . . . 11
- 4.3 print . . . 13

5 Indices and tables **15**

Index **17**

json is lighweight JSON parser written in C99 (compatible with C89).

CHAPTER 1

Features

- header-only or optional compiled library
- option to store members and arrays as reverse order or normal
- doesn't alloc memory for keys and values only for tokens
- creates DOM-like data structure to make it easy to iterate though
- simple api
- provides some util functions to print json, get int32, int64, float, double...
- very small library
- and other...

CHAPTER 2

Build json

json library does not have external dependencies.

NOTE: If you only need to inline versions, you don't need to build **json**, you don't need to link it to your program. Just import cglm to your project as dependency / external lib by copy-paste then use it as usual

2.1 Unix (Autotools):

```
$ sh autogen.sh
$ ./configure
$ make
$ make check            # run tests (optional)
$ [sudo] make install   # install to system (optional)
```

make will build json to **.libs** sub folder in project folder. If you don't want to install **json** to your system's folder you can get static and dynamic libs in this folder.

2.2 Windows (MSBuild):

Windows related build files, project files are located in *win* folder, make sure you are inside in json/win folder.

Code Analysis are enabled, it may take awhile to build.

```
$ cd win
$ .\build.bat
```

if *msbuild* is not worked (because of multi versions of Visual Studio) then try to build with *devenv*:

```
$ devenv json.sln /Build Release
```

Currently tests are not available on Windows.

2.3 Documentation (Sphinx):

json uses sphinx framework for documentation, it allows lot of formats for documentation. To see all options see sphinx build page:

https://www.sphinx-doc.org/en/master/man/sphinx-build.html

Example build:

```
$ cd json/docs
$ sphinx-build source build

or

$ cd json/docs
$ sh ./build-docs.sh
```

CHAPTER 3

Getting Started

json uses **json_** prefix for all functions e.g. json_parse(). There are only a few types which represents json document, json object, json array and json value (as string).

- **json_doc_t** represents JSON document. It stores root JSON node and allocated memory.
- **json_t** represents JSON object. Arrays also are json object.
- **json_array_t** represents JSON array. It inherits **json_t**, so you can cast array to json object.
- **json_type_t** represents JSON type.

3.1 Allocations:

json doesn't alloc any memory for JSON contents, keys and values... It ONLY allocs memory for DOM-tree (json tokens), that's it.

It creates pointers to actual data, so you must retain JSON data until you have finished to process json.

After you finished to parse JSON, this is the order that you must use to free-ing things:

1. free original JSON data
2. free json_doc

actually the order doesn't matter but you must free the json doc which is returned from json_parse().

3.2 Design and Data Structure:

json creates a TREE to traverse JSON. Every json object's child node has **key** pointer. A value of **json_t** may be one of these:

- Child node
- String contents

you must use **type** member of json object to identify the value type. If you need to integer, float or boolean values, then you can use util functions to get these values. These functions will be failed if the value is not string.

VERY IMPORTANT:

key and **value** ARE JUST POINTERS to original data. Because of this, you will see that json object has **keySize** and **valueSize** members. When comparing two keys, you must use *keySize*. Instead of *strcmp()* you could use *strncmp()* and its friends, because it has *size* parameter which is our *keySize*

You can also use built-in helper to compare two keys: **json_key_eq()**

Also when copying values you must also use *valueSize*. You could use *json_string_dup()* to duplicate strings. It is better to not copy contents as possible as much.

UTILITIES / HELPERS:

json library also provides some inline utiliy functions to make things easier while handling json data.

CHAPTER 4

API documentation

4.1 parse json

Header: json/json.h

4.1.1 JSON Document

JSON document is returned when parsing json contents is done. This object stores root JSON object and allocated memories.

It creates pointers to actual data, so you must retain JSON data until you have finished to process json.

You After you processed the parsed JSON, then you must free this document.

4.1.2 Table of contents (click to go):

Functions:

1. *json_parse()*
2. *json_free()*
3. *json_get()*
4. *json_array()*

4.1.3 Functions documentation

json_doc_t* **json_parse**(const char * __restrict *contents*)

parse json string

this function parses JSON string and retuns a document which contains:

- JSON object

- allocated memory

after JSON is processed, the object must be freed with json_free()

this library doesn't alloc any memory for JSON itsef and doesn't copy JSON contents into a data structure. It only allocs memory for tokens. So don't free 'contents' parameter until you finished to process JSON.

Desired order:

1. Read JSON file
2. Pass the contents to json_parse()
3. Process/Handle JSON
4. free JSON document with json_free()
5. free *contents*

Parameters:

[in] **contents** JSON string

Returns:

json document which contains json object as root object

void **json_free** (json_doc_t * __restrict *jsondoc*)
frees json document and its allocated memory

Parameters:

[in] **jsondoc** JSON document

const json_t* **json_get** (const json_t * __restrict *object*, const char * __restrict *key*)
gets value for key

You should only use this for DEBUG or if you only need to only specific key. Desired usage is iterative way:

You must iterate through json's next and value links.

Parameters:

[in] **object** json object
[in] **key** key to find value

Returns:

value found for the key or NULL

const json_array_t* **json_array** (const json_t * __restrict *object*)
contenient function to cast object's child/value to array

Parameters:

[in] **object** json object

Returns:

json array or NULL

4.2 utils / helpers

Header: json/util.h

Inline helpers to make things easier while process JSON. Most of uitl functions expects default value, so if it fails to convert string to a number or boolean then that default value will be returned.

4.2.1 Table of contents (click to go):

Functions:

1. *json_int32()*
2. *json_uint32()*
3. *json_int64()*
4. *json_uint64()*
5. *json_float()*
6. *json_double()*
7. *json_bool()*
8. *json_string()*
9. *json_string_dup()*
10. *json_key_eq()*

4.2.2 Functions documentation

int32_t **json_int32** (const json_t * __restrict *object*, int32_t *defaultValue*)

creates number (int32) from string value

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: number

uint32_t **json_uint32** (const json_t * __restrict *object*, uint32_t *defaultValue*)

creates number (uint32) from string value

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: number

int64_t **json_int64** (const json_t * __restrict *object*, int64_t *defaultValue*)

creates number (int64) from string value

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: number

int64_t **json_uint64** (const json_t * __restrict *object*, uint64_t *defaultValue*)

creates number (uint64) from string value

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: number

float **json_float** (const json_t * __restrict *object*, float *defaultValue*)

creates number (float) from string value

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: number

double **json_double** (const json_t * __restrict *object*, double *defaultValue*)

creates number (double) from string value

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: number

int **json_bool** (const json_t * __restrict *object*, int *defaultValue*)

creates boolean from string value

it returns integer to separate default value from true or false

Parameters:

[in] **object** json object
[in] **defaultValue** default value if operation fails

Returns: boolean values as integer: 1 true, 0 false, error: defaultValue

const char* **json_string** (const json_t * __restrict *object*)

return non-NULL terminated string value
you must use object->valSize to copy, compare ... string

Parameters:

[in] **object** json object

Returns: non-NULL terminated string value (pointer only)

char* **json_string_dup** (const json_t * __restrict *object*)

return non-NULL terminated string value
you must use object->valSize to copy, compare ... string

Parameters:

[in] **object** json object

Returns: NULL terminated duplicated string value

bool **json_key_eq** (const json_t * __restrict *obj*, const char * __restrict *str*)

compares json key with a string|

Parameters:

[in] **obj** json object
[in] **str** string to compare

Returns: true if str is equals to json's key

4.3 print

Header: json/print.h

Print functions

4.3.1 Table of contents (click to go):

Functions:

1. *json_print()*
2. *json_print_pad()*

4.3.2 Functions documentation

void **json_print** (const json_t * __restrict *json*)

print json

Parameters:

[in] **json** json object with title and zero padding

void **json_print_pad**(const json_t * __restrict *json*, int *pad*)

print json

Parameters:

[in] **json** json object
[in] **pad** padding

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

J

`json_array` (*C function*), 10
`json_bool` (*C function*), 12
`json_double` (*C function*), 12
`json_float` (*C function*), 12
`json_free` (*C function*), 10
`json_get` (*C function*), 10
`json_int32` (*C function*), 11
`json_int64` (*C function*), 11
`json_key_eq` (*C function*), 13
`json_parse` (*C function*), 9
`json_print` (*C function*), 13
`json_print_pad` (*C function*), 14
`json_string` (*C function*), 12
`json_string_dup` (*C function*), 13
`json_uint32` (*C function*), 11
`json_uint64` (*C function*), 12