LwJSON

Tilen MAJERLE

Dec 10, 2020

CONTENTS

1 Features **3**

2 Requirements **5**

3 Contribute **7**

4 Example code **9**

5 License **11**

6 Table of contents **13**

6.1 Getting started 13

6.2 User manual 16

6.3 API reference 20

Index **25**

Welcome to the documentation for version v1.1.0.

LwJSON is a generic JSON parser library optimized for embedded systems.

Download library Getting started Open Github Donate

CHAPTER

ONE

FEATURES

- Written in ANSI C99, compatible with `size_t` for size data types
- RFC 4627 compliant
- Based on static token allocation with optional application dynamic pre-allocation
- No recursion during parse operation
- Re-entrant functions
- Zero-copy, no `malloc` or `free` functions used
- Advanced find algorithm for tokens
- Testscoverage is available
- User friendly MIT license

CHAPTER

TWO

REQUIREMENTS

- C compiler

CHAPTER

THREE

CONTRIBUTE

Fresh contributions are always welcome. Simple instructions to proceed:

1. Fork Github repository
2. Respect C style & coding rules used by the library
3. Create a pull request to `develop` branch with new features or bug fixes

Alternatively you may:

1. Report a bug
2. Ask for a feature request

CHAPTER

FOUR

EXAMPLE CODE

Listing 1: Example code

```c
#include <stdio.h>
#include "lwjson/lwjson.h"

/* LwJSON instance and tokens */
static lwjson_token_t tokens[128];
static lwjson_t lwjson;

/* Parse JSON */
void
example_minimal_run(void) {
    lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));
    if (lwjson_parse(&lwjson, "{\"mykey\":\"myvalue\"}") == lwjsonOK) {
        const lwjson_token_t* t;
        printf("JSON parsed..\r\n");

        /* Find custom key in JSON */
        if ((t = lwjson_find(&lwjson, "mykey")) != NULL) {
            printf("Key found with data type: %d\r\n", (int)t->type);
        }
    }
}
```

CHAPTER

FIVE

LICENSE

```
MIT License

Copyright (c) 2020 Tilen MAJERLE

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

SIX

TABLE OF CONTENTS

6.1 Getting started

6.1.1 Download library

Library is primarly hosted on Github.

- Download latest release from releases area on Github
- Clone *develop* branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

- Download and install `git` if not already
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available `3` options
  - Run `git clone --recurse-submodules https://github.com/MaJerle/lwjson` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwjson` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch master https://github.com/MaJerle/lwjson` to clone *latest stable* branch, including submodules
- Navigate to `examples` directory and run favourite example

Update cloned to latest version

- Open console and navigate to path in the system where your resources repository is. Use command `cd your_path`
- Run `git pull origin master --recurse-submodules` command to pull latest changes and to fetch latest changes from submodules
- Run `git submodule foreach git pull origin master` to update & merge all submodules

Note: This is preferred option to use when you want to evaluate library and run prepared examples. Repository consists of multiple submodules which can be automatically downloaded when cloning and pulling changes from root repository.

6.1.2 Add library to project

At this point it is assumed that you have successfully download library, either cloned it or from releases page.

- Copy `lwjson` folder to your project
- Add `lwjson/src/include` folder to *include path* of your toolchain
- Add source files from `lwjson/src/` folder to toolchain build
- Copy `lwjson/src/include/lwjson/lwjson_opts_template.h` to project folder and rename it to `lwjson_opts.h`
- Build the project

6.1.3 Configuration file

Library comes with template config file, which can be modified according to needs. This file shall be named `lwjson_opts.h` and its default template looks like the one below.

Note: Default configuration template file location: `lwjson/src/include/lwjson/lwjson_opts_template.h`. File must be renamed to `lwjson_opts.h` first and then copied to the project directory (or simply renamed in-place) where compiler include paths have access to it by using `#include "lwjson_opts.h"`.

Tip: Check *Configuration* section for possible configuration settings

Listing 1: Template options file

```
/**
 * \file            lwjson_opts_template.h
 * \brief           Template config file
 */

/*
 * Copyright (c) 2020 Tilen MAJERLE
 *
```

(continues on next page)

(continued from previous page)

```
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwJSON - Lightweight JSON format parser.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v1.1.0
 */
#ifndef LWJSON_HDR_OPTS_H
#define LWJSON_HDR_OPTS_H

/* Rename this file to "lwjson_opts.h" for your application */

/*
 * Open "include/lwjson/lwjson_opt.h" and
 * copy & replace here settings you want to change values
 */

#endif /* LWJSON_HDR_OPTS_H */
```

6.1.4 Minimal example code

Run below example to test and verify library

Listing 2: Minimal example code

```
#include <stdio.h>
#include "lwjson/lwjson.h"

/* LwJSON instance and tokens */
static lwjson_token_t tokens[128];
static lwjson_t lwjson;

/* Parse JSON */
void
example_minimal_run(void) {
    lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));
    if (lwjson_parse(&lwjson, "{\"mykey\":\"myvalue\"}") == lwjsonOK) {
```

(continues on next page)

(continued from previous page)

```
        const lwjson_token_t* t;
        printf("JSON parsed..\r\n");

        /* Find custom key in JSON */
        if ((t = lwjson_find(&lwjson, "mykey")) != NULL) {
            printf("Key found with data type: %d\r\n", (int)t->type);
        }
    }
}
```

6.2 User manual

6.2.1 How it works

LwJSON fully complies with *RFC 4627* memo.

LwJSON accepts input string formatted as JSON as per *RFC 4627*. Library parses each character and creates list of tokens that are understood by C language for easier further processing.

When JSON is successfully parsed, there are several tokens used, one for each JSON data type. Each token consists of:

- Token type
- Token parameter name (*key*) and its length
- Token value or pointer to first child (in case of *object* or *array* types)

As an example, JSON text `{"mykey":"myvalue"}` will be parsed into `2` tokens:

- First token is the opening bracket and has type *object* as it holds children tokens
- Second token has name `"mykey"`, its type is *string* and value is `myvalue`

> **Warning:** When JSON input string is parsed, create tokens use input string as a reference. This means that until JSON parsed tokens are being used, original text must stay as-is.

6.2.2 Token design

Every element of LwJSON is a token. There are different set of token types:

- *Object*: Type that has nested key-value pairs, eg `{"key":{"sub-key":"value"}}`
- *Array*: Type that holds nested values, eg `{"key":[1,2,3,4,5]}`
- *String*: Regular string, quoted sequence of characters, eg `{"key":"my_string"}`
- *Number*: Integer or real number, eg `{"intnum":123,"realnum":4.3}`
- *Boolean true*: Boolean type true, eg `{"key":true}`
- *Boolean false*: Boolean type false, eg `{"key":false}`
- *Null*: Null indicator, eg `{"key":null}`

When parsed, input string is not copied to token, every token uses input string as a reference and points to the beginning of strings/values. This is valid for all string data types and for parameter names.

Note: Input string is not modified therefore all strings contain additional parameter with string length.

6.2.3 Access to data

Once application successfully parses input JSON string, LwJSON creates set of tokens in hierarchical order with tree for children tokens.

To simplify data extraction and to quickly find/access-to given object and token, LwJSON implements simple *find* algorithm based on path formatting.

Traverse object

Valid JSON input starts with *object* (`{`) or *array* (`[`). Anything else is invalid JSON object. When `lwjson_parse()` successfully processes input data, application may access JSON tokens with simple loop. Every token is part of linked list and has *tree* organization for children objects.

Note: Children objects are only available for *object* and *array* types

To traverse through all elements, application must first get top object. It can then loop through all in linked list until it reaches zero.

If the token type is *object* or *array*, application must check children nodes for more token data.

Listing 3: Traverse through all tokens

```
#include <stdio.h>
#include "lwjson/lwjson.h"

/* LwJSON instance and tokens */
static lwjson_token_t tokens[128];
static lwjson_t lwjson;

/* Parse JSON */
void
example_traverse_run(void) {
    lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));
    if (lwjson_parse(&lwjson, "{\"mykey\":\"myvalue\",\"num\":1,\"obj\":{},\"arr\":[1,
↪2,3,4]}") == lwjsonOK) {
        lwjson_token_t* t;
        printf("JSON parsed..\r\n");

        /* Get very first token as top object */
        t = lwjson_get_first_token(&lwjson);
        if (t->type == LWJSON_TYPE_ARRAY) {
            printf("JSON starts with array..\r\n");
        } else if (t->type == LWJSON_TYPE_OBJECT) {
            printf("JSON starts with object..\r\n");
        } else {
            printf("This should never happen..\r\n");
        }
```

(continues on next page)

(continued from previous page)

```
        /* Now print all keys in the object */
        for (const lwjson_token_t* tkn = lwjson_get_first_child(t); tkn != NULL; tkn␣
↪= tkn->next) {
            printf("Token: %.*s", (int)tkn->token_name_len, tkn->token_name);
            if (tkn->type == LWJSON_TYPE_ARRAY || tkn->type == LWJSON_TYPE_OBJECT) {
                printf(": Token is array or object...check children tokens if any, in␣
↪recursive mode..");
                /* Get first child of token */
                //lwjson_get_first_child(tkn);
            }
            printf("\n");
        }
    }
}
```

Tip: Check `lwjson_print_json()` to print data on stream output

Find token in JSON tree

Instead of manually traversing through all tokens, LwJSON implements simple search algorithm to quickly get token from application standpoint.

Let's consider following JSON as input:

```
{
    "name":"John",
    "born": {
            "city": "Munich",
            "year": 1993
    }
    "cars":[
            {
                    "brand":"Porsche",
                    "year":2018
            },
            {
                    "brand":"Range",
                    "year":2020,
                    "crashed":1
            }
    ]
}
```

There is one *John*, born in *Munich* in `1993` and has `2` cars, *Porsche* and *Range*.

- `name` is string with value `John`
- `born` is object with `2` fields
- `cars` is array of `2` objects
  - object 1
    - `brand` is set to *Porsche*

    * `year` is set to `2018`
  - object 2
    * `brand` is set to *Range*
    * `year` is set to `2020`
    * `repainted` is set to `true` as this car was recently repainted

To find the person's name, application must first `name` key and print its value. This can be done by scanning entire object and check which token matches `name` keyword.

LwJSON implements *find* functionality to find the token in simple way. This is done by providing full path to token in JSON tree, separated by *dot* `.` character.

To find person name, application would then simply call `lwjson_find()` and pass `name` as path parameter. If token exists, function will return handle of the token where application can print its value.

If application is interested in *city of birth*, it will set path as `born.city` and search algorithm will:

- First search for `born` token and check if it is *object*
- It will enter the object and search for `city` token and return it on match
- It will return `NULL` of object is not found

Tip: Application shall use `lwjson_find()` to get the token based on input path.

When JSON contains arrays (these do not have keys), special character `#` may be used, indicating *any* element in array to be checked until first match is found.

- `cars.#.brand` will return first token matching path, the one with string value `Porsche` in first object
- `cars.#.repainted` will return first token matching path, the one with boolean value `true` in second object

In first case, `brand` keyword exists already in first object, so find function will get the match immediately. Because in second case, `repainted` only exists in second object, function will return value from second object.

Access array index

It is possible to access specific array index by adding decimal number after hash character, in format `#[0-9]+`

- `cars.#0.brand` will return `brand` token from *first* object in array (`index = 0`) with value set to *Porsche*
- `cars.#1.brand` will return `brand` token from *second* object in array (`index = 1`) with value set to *Range*

To retrieve full object of the array, application may only apply `#[0.9]+` in search pattern.

- `cars.#0` will return first object token in array
- `cars.#1` will return second object token in array
- `cars.#` will return error as there is no valid index. Use `cars` to retrieve full array

Warning: Passing path in format `path.to.cars.#` (hashtag as last element without index number) will always return `NULL` as this is considered invalid path. To retrieve full array, pass path to array `path.to.cars` only, without trailling `#`.

6.3 API reference

List of all the modules:

6.3.1 LwJSON

group **LWJSON**
LwJSON - Lightweight JSON format parser.

Defines

LWJSON_ARRAYSIZE(*x*)
Get size of statically allocated array.

Return Number of elements in array

Parameters

- `[in] x`: Object to get array size of

lwjson_get_tokens_used(*lw*)
Get number of tokens used to parse JSON.

Return Number of tokens used to parse JSON

Parameters

- `[in] lw`: Pointer to LwJSON instance

lwjson_get_first_token(*lw*)
Get top token on a list.

Return Pointer to first token

Parameters

- `[in] lw`: Pointer to LwJSON instance

lwjson_get_val_int(*token*)
Get token value for *LWJSON_TYPE_NUM_INT* type.

Return Int number if type is integer, `0` otherwise

Parameters

- `[in] token`: token with integer type

lwjson_get_val_real(*token*)
Get token value for *LWJSON_TYPE_NUM_REAL* type.

Return Real numbeer if type is real, `0` otherwise

Parameters

- `[in] token`: token with real type

lwjson_get_first_child(*token*)
Get for child token for *LWJSON_TYPE_OBJECT* or *LWJSON_TYPE_ARRAY* types.

Return Pointer to first child

Parameters

- `[in] token`: token with integer type

Typedefs

typedef LWJSON_CFG_REAL_TYPE **lwjson_real_t**
Real data type.

typedef LWJSON_CFG_INT_TYPE **lwjson_int_t**
Integer data type.

Enums

enum lwjson_type_t
List of supported JSON types.

Values:

enumerator LWJSON_TYPE_STRING
String/Text format. Everything that has beginning and ending quote character

enumerator LWJSON_TYPE_NUM_INT
Number type for integer

enumerator LWJSON_TYPE_NUM_REAL
Number type for real number

enumerator LWJSON_TYPE_OBJECT
Object data type

enumerator LWJSON_TYPE_ARRAY
Array data type

enumerator LWJSON_TYPE_TRUE
True boolean value

enumerator LWJSON_TYPE_FALSE
False boolean value

enumerator LWJSON_TYPE_NULL
Null value

enum lwjsonr_t
JSON result enumeration.

Values:

enumerator lwjsonOK
Function returns successfully

enumerator lwjsonERR

enumerator lwjsonERRJSON
Error JSON format

enumerator lwjsonERRMEM
Memory error

Functions

lwjsonr_t **lwjson_init** (*lwjson_t *lw*, *lwjson_token_t *tokens*, size_t *tokens_len*)
Setup LwJSON instance for parsing JSON strings.

Return *lwjsonOK* on success, member of *lwjsonr_t* otherwise

Parameters

- `[inout] lw`: LwJSON instance
- `[in] tokens`: Pointer to array of tokens used for parsing
- `[in] tokens_len`: Number of tokens

lwjsonr_t **lwjson_parse** (*lwjson_t *lw*, **const** char **json_str*)
Parse input JSON format JSON format must be complete and must comply with RFC4627.

Return *lwjsonOK* on success, member of *lwjsonr_t* otherwise

Parameters

- `[inout] lw`: LwJSON instance
- `[in] json_str`: JSON string to parse

lwjsonr_t **lwjson_reset** (*lwjson_t *lw*)
Reset token instances and prepare for new parsing.

Return *lwjsonOK* on success, member of *lwjsonr_t* otherwise

Parameters

- `[inout] lw`: LwJSON instance

const *lwjson_token_t* ***lwjson_find** (*lwjson_t *lw*, **const** char **path*)
Find first match in the given path for JSON entry JSON must be valid and parsed with *lwjson_parse* function.

Return Pointer to found token on success, `NULL` if token cannot be found

Parameters

- `[in] lw`: JSON instance with parsed JSON string
- `[in] path`: Path with dot-separated entries to search for the JSON key to return

lwjsonr_t **lwjson_free** (*lwjson_t *lw*)

void **lwjson_print_token** (**const** *lwjson_token_t *token*)
Prints and outputs token data to the stream output.

Note This function is not re-entrant

Parameters

- `[in] token`: Token to print

void **lwjson_print_json**(**const** *lwjson_t* **lw*)
Prints and outputs full parsed LwJSON instance.

Note This function is not re-entrant

Parameters

- `[in] lw`: LwJSON instance to print

const char ***lwjson_get_val_string**(*lwjson_token_t* **token*, size_t **str_len*)
Get string value from JSON token.

Return Pointer to string

Parameters

- `[in] token`: Token with string type
- `[out] str_len`: Pointer to variable holding length of string

struct lwjson_token_t
#include <lwjson.h> JSON token.

Public Members

struct lwjson_token ***next**
Next token on a list

struct lwjson_token ***parent**
Parent token (think about optimization and remove this one?)

lwjson_type_t **type**
Token type

const char ***token_name**
Token name (if exists)

size_t **token_name_len**
Length of token name (this is needed to support const input strings to parse)

const char ***token_value**
Value if type is not *LWJSON_TYPE_OBJECT* or *LWJSON_TYPE_ARRAY*

size_t **token_value_len**
Length of token value (this is needed to support const input strings to parse)

struct *lwjson_token_t*::**[anonymous]**::**[anonymous] str**
String data

lwjson_real_t **num_real**
Real number format

lwjson_int_t **num_int**
Int number format

struct lwjson_token ***first_child**
First children object

union *lwjson_token_t*::**[anonymous] u**
Union with different data types

struct lwjson_t
#include <lwjson.h> LwJSON instance.

Public Members

lwjson_token_t ***tokens**
Pointer to array of tokens

size_t **tokens_len**
Size of all tokens

size_t **next_free_token_pos**
Position of next free token instance

lwjson_token_t **first_token**
First token on a list

uint8_t **parsed**
Flag indicating JSON parsing has finished successfully

struct *lwjson_t*::**[anonymous] flags**
List of flags

6.3.2 Configuration

This is the default configuration of the middleware. When any of the settings shall be modified, it shall be done in dedicated application config `lwjson_opts.h` file.

Note: Check *Getting started* for guidelines on how to create and use configuration file.

group **LWJSON_OPT**
LwJSON options.

Defines

LWJSON_CFG_REAL_TYPE
Real data type used to parse numbers with floating point number.

This is used for numbers in *LWJSON_TYPE_NUM_REAL* token data type.

Note Data type must be signed, normally `float` or `double`

LWJSON_CFG_INT_TYPE
Integer type used to parse numbers.

This is used for numbers in *LWJSON_TYPE_NUM_INT* token data type.

Note Data type must be signed integer

INDEX

L

LWJSON_ARRAYSIZE (*C macro*), 20
LWJSON_CFG_INT_TYPE (*C macro*), 24
LWJSON_CFG_REAL_TYPE (*C macro*), 24
lwjson_find (*C++ function*), 22
lwjson_free (*C++ function*), 22
lwjson_get_first_child (*C macro*), 20
lwjson_get_first_token (*C macro*), 20
lwjson_get_tokens_used (*C macro*), 20
lwjson_get_val_int (*C macro*), 20
lwjson_get_val_real (*C macro*), 20
lwjson_get_val_string (*C++ function*), 23
lwjson_init (*C++ function*), 22
lwjson_int_t (*C++ type*), 21
lwjson_parse (*C++ function*), 22
lwjson_print_json (*C++ function*), 22
lwjson_print_token (*C++ function*), 22
lwjson_real_t (*C++ type*), 21
lwjson_reset (*C++ function*), 22
lwjson_t (*C++ struct*), 23
lwjson_t::first_token (*C++ member*), 24
lwjson_t::flags (*C++ member*), 24
lwjson_t::next_free_token_pos (*C++ member*), 24
lwjson_t::parsed (*C++ member*), 24
lwjson_t::tokens (*C++ member*), 24
lwjson_t::tokens_len (*C++ member*), 24
lwjson_token_t (*C++ struct*), 23
lwjson_token_t::first_child (*C++ member*), 23
lwjson_token_t::next (*C++ member*), 23
lwjson_token_t::num_int (*C++ member*), 23
lwjson_token_t::num_real (*C++ member*), 23
lwjson_token_t::parent (*C++ member*), 23
lwjson_token_t::str (*C++ member*), 23
lwjson_token_t::token_name (*C++ member*), 23
lwjson_token_t::token_name_len (*C++ member*), 23
lwjson_token_t::token_value (*C++ member*), 23
lwjson_token_t::token_value_len (*C++ member*), 23
lwjson_token_t::type (*C++ member*), 23
lwjson_token_t::u (*C++ member*), 23
lwjson_type_t (*C++ enum*), 21
lwjson_type_t::LWJSON_TYPE_ARRAY (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_FALSE (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_NULL (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_NUM_INT (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_NUM_REAL (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_OBJECT (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_STRING (*C++ enumerator*), 21
lwjson_type_t::LWJSON_TYPE_TRUE (*C++ enumerator*), 21
lwjsonr_t (*C++ enum*), 21
lwjsonr_t::lwjsonERR (*C++ enumerator*), 21
lwjsonr_t::lwjsonERRJSON (*C++ enumerator*), 21
lwjsonr_t::lwjsonERRMEM (*C++ enumerator*), 21
lwjsonr_t::lwjsonOK (*C++ enumerator*), 21