LwJSON

Tilen MAJERLE

May 26, 2021

CONTENTS

Welcome to the documentation for version branch-5b1d37e.

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 and RFC 8259 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
- Optional support for inline comments with /* *comment…* */ syntax between any *blank* region of input string
- Advanced find algorithm for tokens
- Tests coverage is available
- User friendly MIT license

CHAPTER

TWO

REQUIREMENTS

- C compiler
- Few kB of ROM memory

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);
        }

        /* Call this when not used anymore */
        lwjson_free(&lwjson);
    }
}
```

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

Getting started may be the most challenging part of every new library. This guide is describing how to start with the library quickly and effectively

6.1.1 Download library

Library is primarly hosted on Github.

You can get it with:

- Downloading latest release from releases area on Github
- Cloning `master` branch for latest stable version
- Cloning `develop` branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

This is used when you do not have yet local copy on your machine.

- Make sure `git` is installed.
- 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 on `master` branch
- Run `git pull origin develop --recurse-submodules` command to pull latest changes and to fetch latest changes from submodules on `develop` branch
- 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. Next step is to add the library to the project, by means of source files to compiler inputs and header files in search path

- Copy `lwjson` folder to your project, it contains library files
- Add `lwjson/src/include` folder to *include path* of your toolchain. This is where *C/C++* compiler can find the files during compilation process. Usually using `-I` flag
- Add source files from `lwjson/src/` folder to toolchain build. These files are built by *C/C++* compiler
- 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

Configuration file is used to overwrite default settings defined for the essential use case. Library comes with template config file, which can be modified according to needs. and it should be copied (or simply renamed in-place) and named `lwjson_opts.h`

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 where compiler include paths have access to it by using `#include "lwjson_opts.h"`.

List of configuration options are available in the *Configuration* section. If any option is about to be modified, it should be done in configuration file

Listing 1: Template configuration file

```
1 /**
2  * \file            lwjson_opts_template.h
3  * \brief           Template config file
4  */
```

(continues on next page)

(continued from previous page)

```

/*
 * 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.
 *
 * This file is part of LwJSON - Lightweight JSON format parser.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v1.5.0
 */
#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 */
```

Note: If you prefer to avoid using configuration file, application must define a global symbol `LWJSON_IGNORE_USER_OPTS`, visible across entire application. This can be achieved with `-D` compiler option.

6.1.4 Minimal example code

To verify proper library setup, minimal example has been prepared. Run it in your main application file to verify its proper execution

Listing 2: Absolute minimum example

```
#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);
        }

        /* Call this when not used anymore */
        lwjson_free(&lwjson);
    }
}
```

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

```
1 #include <stdio.h>
2 #include "lwjson/lwjson.h"
3
```

(continues on next page)

(continued from previous page)

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

/* Parse JSON */
void
example_traverse_run(void) {
    /* Initialize and pass statically allocated tokens */
    lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));

    /* Try to parse input string */
    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");
        }

        /* 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");
        }

        /* Call this when not used anymore */
        lwjson_free(&lwjson);
    }
}
```

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,
            "repainted":true
            }
    ]
}
```

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.

Parameters

- **x** – **[in]** Object to get array size of

Returns Number of elements in array

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

Parameters

- **lw** – **[in]** Pointer to LwJSON instance

Returns Number of tokens used to parse JSON

lwjson_get_first_token(lw)
Get very first token of LwJSON instance.

Parameters

- **lw** – **[in]** Pointer to LwJSON instance

Returns Pointer to first token

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

Parameters

- **token** – **[in]** token with integer type

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

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

Parameters

- **token** – **[in]** token with real type

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

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

Parameters

- **token** – **[in]** token with integer type

Returns Pointer to first child or `NULL` if parent token is not object or array

lwjson_get_val_string_length(token)
Get length of string for *LWJSON_TYPE_STRING* token type.

Parameters

- **token** – **[in]** token with string type

Returns Length of string in units of bytes

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**
Generic error message

enumerator **lwjsonERRJSON**
Error JSON format

enumerator **lwjsonERRMEM**
Memory error

enumerator **lwjsonERRPAR**
Parameter error

Functions

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

Parameters

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

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

lwjsonr_t **lwjson_parse_ex**(*lwjson_t* *lw, const void *json_data, size_t len)
Parse JSON data with length parameter JSON format must be complete and must comply with RFC4627.

Parameters

- **lw** – **[inout]** LwJSON instance
- **json_data** – **[in]** JSON string to parse
- **jsonČlen** – **[in]** JSON data length

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

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.

Parameters

- **lw** – **[inout]** LwJSON instance
- **json_str** – **[in]** JSON string to parse

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

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.

Parameters

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

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

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

Parameters

- **lw** – **[in]** JSON instance with parsed JSON string
- **token** – **[in]** Root token to start search at. Token must be type *LWJSON_TYPE_OBJECT* or *LWJSON_TYPE_ARRAY*. Set to NULL to use root token of LwJSON object
- **path** – **[in]** path with dot-separated entries to search for JSON key

Returns Pointer to found token on success, NULL if token cannot be found

lwjsonr_t **lwjson_free**(*lwjson_t* *lw)
Free token instances (specially used in case of dynamic memory allocation)

Parameters **lw** – **[inout]** LwJSON instance

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

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 **token** – **[in]** 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 **lw** – **[in]** LwJSON instance to print

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

Parameters

- **token** – **[in]** Token with string type
- **str_len** – **[out]** Pointer to variable holding length of string. Set to NULL if not used

Returns Pointer to string or NULL if invalid token type

static inline uint8_t **lwjson_string_compare**(const *lwjson_token_t* *token, const char *str)
Compare string token with user input string for a case-sensitive match.

Parameters

- **token** – **[in]** Token with string type
- **str** – **[out]** String to compare

Returns 1 if equal, 0 otherwise

static inline uint8_t **lwjson_string_compare_n**(const *lwjson_token_t* *token, const char *str, size_t len)
Compare string token with user input string for a case-sensitive match.

Parameters

- **token** – **[in]** Token with string type
- **str** – **[out]** String to compare

Returns 1 if equal, 0 otherwise

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

Public Members

struct lwjson_token *__next__
Next token on a list

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__
Pointer to the beginning of the string

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 for object or array type

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

`LWJSON_CFG_COMMENTS`
Enables `1` or disables `0` support for inline comments.

Default set to `0` to be JSON compliant

INDEX

L