
LwJSON

Tilen MAJERLE

Apr 02, 2024

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 17
 - 6.3 API reference 23
 - 6.4 Changelog 35
 - 6.5 Authors 37
- Index 39**

Welcome to the documentation for version latest-develop.

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

[Download library](#) [Getting started](#) [Open Github](#) [Donate](#)

FEATURES

- Written in C (C11), 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
- Supports streaming parsing as secondary option
- 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

REQUIREMENTS

- C compiler
- Few kB of ROM memory

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

EXAMPLE CODE

Listing 1: Example code

```
1  #include <stdio.h>
2  #include "lwjson/lwjson.h"
3
4  /* LwJSON instance and tokens */
5  static lwjson_token_t tokens[128];
6  static lwjson_t lwjson;
7
8  /* Parse JSON */
9  void
10 example_minimal_run(void) {
11     lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));
12     if (lwjson_parse(&lwjson, "{\"mykey\":\"myvalue\"}") == lwjsonOK) {
13         const lwjson_token_t* t;
14         printf("JSON parsed.\r\n");
15
16         /* Find custom key in JSON */
17         if ((t = lwjson_find(&lwjson, "mykey")) != NULL) {
18             printf("Key found with data type: %d\r\n", (int)t->type);
19         }
20
21         /* Call this when not used anymore */
22         lwjson_free(&lwjson);
23     }
24 }
```


LICENSE**MIT License**

Copyright (c) 2024 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.

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 primarily hosted on [Github](#).

You can get it by:

- Downloading latest release from [releases area](#) on Github
- Cloning `main` 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 options below
 - 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 main 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 repository is located. Use command `cd your_path`
- Run `git pull origin main` command to get latest changes on main branch
- Run `git pull origin develop` command to get latest changes on develop branch
- Run `git submodule update --init --remote` to update submodules to latest version

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.

CMake is the main supported build system. Package comes with the `CMakeLists.txt` and `library.cmake` files, both located in the `lwjson` directory:

- `CMakeLists.txt`: Is a wrapper and only includes `library.cmake` file. It is used if target application uses `add_subdirectory` and then uses `target_link_libraries` to include the library in the project
- `library.cmake`: It is a fully configured set of variables. User must use `include(path/to/library.cmake)` to include the library and must manually add files/includes to the final target

Tip: Open `library.cmake` file and manually analyze all the possible variables you can set for full functionality.

If you do not use the *CMake*, you can do the following:

- 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. *CMake* configuration comes with the library, allows users to include library in the project as **subdirectory** and **library**.
- 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 the application 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"`.

Tip: If you are using *CMake* build system, define the variable `LWJSON_OPTS_FILE` before adding library's directory to the *CMake* project. Variable must contain the path to the user options file. If not provided and to avoid build error, one will be generated in the build directory.

Configuration options list is available 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   */
5
6  /**
7   * Copyright (c) 2024 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwJSON - Lightweight JSON format parser.
30  *
31  * Author:          Tilen MAJERLE <tilen@majerle.eu>
32  * Version:         v1.6.1
33  */
34  #ifndef LWJSON_OPTS_HDR_H
35  #define LWJSON_OPTS_HDR_H
36
37  /* Rename this file to "lwjson_opts.h" for your application */
38
39  /**
40   * Open "include/lwjson/lwjson_opt.h" and

```

(continues on next page)

(continued from previous page)

```
41  * copy & replace here settings you want to change values
42  */
43
44 #endif /* LWJSON_OPTS_HDR_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

```
1  #include <stdio.h>
2  #include "lwjson/lwjson.h"
3
4  /* LwJSON instance and tokens */
5  static lwjson_token_t tokens[128];
6  static lwjson_t lwjson;
7
8  /* Parse JSON */
9  void
10 example_minimal_run(void) {
11     lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));
12     if (lwjson_parse(&lwjson, "{\"mykey\":\"myvalue\"}") == lwjsonOK) {
13         const lwjson_token_t* t;
14         printf("JSON parsed.\r\n");
15
16         /* Find custom key in JSON */
17         if ((t = lwjson_find(&lwjson, "mykey")) != NULL) {
18             printf("Key found with data type: %d\r\n", (int)t->type);
19         }
20
21         /* Call this when not used anymore */
22         lwjson_free(&lwjson);
23     }
24 }
```

6.2 User manual

6.2.1 How it works

LwJSON fully complies with *RFC 4627* memo and supports 2 types of parsing:

- Parsing with full data available as single linear memory (primary option)
- Stream parsing with partial available bytes at any given point of time - advanced state machine

When full data are available, standard parsing is used with tokens, that contain references to start/stop indexes of the strings and other primitives and provide full device tree - sort of custom hash-map value.

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* with value set as `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. Any modification of source JSON input may destroy references from the token tree and hence generate wrong output for the user

Tip: See [Stream parser](#) for implementation of streaming parser where full data do not need to be available at any given time.

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
4  /* LwJSON instance and tokens */
5  static lwjson_token_t tokens[128];
6  static lwjson_t lwjson;
7
8  /* Parse JSON */
9  void
10 example_traverse_run(void) {
11     /* Initialize and pass statically allocated tokens */
12     lwjson_init(&lwjson, tokens, LWJSON_ARRAYSIZE(tokens));
13
14     /* Try to parse input string */
15     if (lwjson_parse(&lwjson, "{\"mykey\":\"myvalue\",\"num\":1,\"obj\":{\"arr\":[1,2,
16 ↪ 3,4]}\"}") == lwjsonOK) {
17         lwjson_token_t* t;
18         printf("JSON parsed..\r\n");
19
20         /* Get very first token as top object */
21         t = lwjson_get_first_token(&lwjson);
22         if (t->type == LWJSON_TYPE_ARRAY) {
23             printf("JSON starts with array..\r\n");
24         } else if (t->type == LWJSON_TYPE_OBJECT) {
25             printf("JSON starts with object..\r\n");
26         } else {
27             printf("This should never happen..\r\n");
28         }
29
30         /* Now print all keys in the object */
```

(continues on next page)

(continued from previous page)

```

30     for (const lwjson_token_t* tkn = lwjson_get_first_child(t); tkn != NULL; tkn = tkn->next) {
31         printf("Token: %.*s", (int)tkn->token_name_len, tkn->token_name);
32         if (tkn->type == LWJSON_TYPE_ARRAY || tkn->type == LWJSON_TYPE_OBJECT) {
33             printf(": Token is array or object...check children tokens if any, in recursive mode..");
34             /* Get first child of token */
35             //lwjson_get_first_child(tkn);
36         }
37         printf("\n");
38     }
39
40     /* Call this when not used anymore */
41     lwjson_free(&lwjson);
42 }
43

```

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 if 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 trailing **#**.

6.2.4 Stream parser

Streaming parser implementation is alternative option versus standard tokenized one, in the sense that:

- There is no need to have full JSON available at one time to have successful parsing
- It can be utilized to parse very large JSON strings on very small systems with limited memory
- It allows users to *take* from the stream only necessary parts and store them to local more system-friendly variable

This type of parser does not utilize use of tokens, rather focuses on the callback function, where user is in charge to manually understand token structure and get useful data from it.

Stream parser introduces *stack* mechanism instead - to keep the track of depthness during parsing the process. 3 different element types are stored on *local stack*:

- Start of object, with { character
- Start of array, with [character
- Key from the *object* entry

Note: Stack is nested as long as JSON input stream is nested in the same way

Consider this input string: {"k1":"v1","k2":[true, false]}. During parsing procedure, at some point of time, these events will occur:

1. Start of *object* detected - *object* pushed to stack
 1. *key* element with name k1 detected and pushed to stack
 1. *string* v1 parsed as *string-value*
 2. *key* element with name k1 popped from stack
 3. *key* element with name k2 detected and pushed to stack
 1. Start of *array* detected - *array* pushed to stack
 1. *true* primitive detected
 2. *false* primitive detected
 2. End of *array* detected - *array* popped from stack
 4. *key* element with name k2 popped from stack
2. End of *object* detected - *object* popped from stack

Each of these events is reported to user in the callback function.

An example of the stream parsing:

Listing 4: Parse JSON data as a stream object

```

1 #include <stdio.h>
2 #include "lwjson/lwjson.h"
3
4 /* Test string to parser */
5 static const char* json_str = "{\"k1\":\"v1\",\"k2\":[true, false]}";
6
7 /* LwJSON stream parser */
8 static lwjson_stream_parser_t stream_parser;
```

(continues on next page)

(continued from previous page)

```

9
10 /**
11  * \brief      Callback function for various events
12  * \param      jsp: JSON stream parser object
13  * \param      type: Event type
14  */
15 static void
16 prv_example_callback_func(lwjson_stream_parser_t* jsp, lwjson_stream_type_t type) {
17     /* Get a value corresponding to "k1" key */
18     if (jsp->stack_pos >= 2 /* Number of stack entries must be high */
19         && lwjson_stack_seq_2(jsp, 0, OBJECT, KEY) && strcmp(jsp->stack[1].meta.name, "k1
↵") == 0) {
20         printf("Got key '%s' with value '%s'\r\n", jsp->stack[1].meta.name, jsp->data.
↵str.buff);
21     }
22     (void)type;
23 }
24
25 /* Parse JSON */
26 void
27 example_stream_run(void) {
28     lwjsonr_t res;
29     printf("\r\n\r\nParsing stream\r\n");
30     lwjson_stream_init(&stream_parser, prv_example_callback_func);
31
32     /* Demonstrate as stream inputs */
33     for (const char* c = json_str; *c != '\0'; ++c) {
34         res = lwjson_stream_parse(&stream_parser, *c);
35         if (res == lwjsonSTREAMINPROG) {
36         } else if (res == lwjsonSTREAMWAITFIRSTCHAR) {
37             printf("Waiting first character\r\n");
38         } else if (res == lwjsonSTREAMDONE) {
39             printf("Done\r\n");
40         } else {
41             printf("Error\r\n");
42             break;
43         }
44     }
45     printf("Parsing completed\r\n");
46 }

```

Example

For the purpose of example, the following JSON input...

Listing 5: JSON input for streaming

```

1 {
2   "test": "abc",
3   "array": [
4     "123",

```

(continues on next page)

(continued from previous page)

```

5  "def",
6  "ghi"
7  ],
8  "array_in_array": [
9    ["1", "2", "3"],
10   ["4", "5", "6"],
11   ["7", "8", "9"]
12 ]
13 }

```

... will output the log as:

6.3 API reference

List of all the modules:

6.3.1 LwJSON

group **LWJSON**

LwJSON - Lightweight JSON format parser.

LWJSON_STREAM_SEQ

Helper functions for stack analysis in a callback function

Note: Useful exclusively for streaming functions

lwjson_stack_seq_1(jsp, start_num, sp0)

Check the sequence of JSON stack, starting from start_number index.

Note: This applies only to one sequence element. Other macros, starting with `lwjson_stack_seq_X` (where X is the sequence length), provide more parameters for longer sequences.

Parameters

- **jsp** – [in] LwJSON stream instance
- **start_num** – [in] Start number in the stack. Typically starts with 0, but user may choose another number, if intention is to check partial sequence only
- **sp0** – [in] Stream stack type. Value of `lwjson_stream_type_t`, but only last part of the enum. If user is interested in the `LWJSON_STREAM_TYPE_OBJECT`, you should only write OBJECT as parameter. Idea behind is to make code smaller and easier to read, especially when using other sequence values with more parameters.

Returns

0 if sequence doesn't match, non-zero otherwise

lwjson_stack_seq_2(jsp, start_num, sp0, sp1)

lwjson_stack_seq_3(jsp, start_num, sp0, sp1, sp2)

lwjson_stack_seq_4(jsp, start_num, sp0, sp1, sp2, sp3)

lwjson_stack_seq_5(jsp, start_num, sp0, sp1, sp2, sp3, sp4)

lwjson_stack_seq_6(jsp, start_num, sp0, sp1, sp2, sp3, sp4, sp5)

lwjson_stack_seq_7(jsp, start_num, sp0, sp1, sp2, sp3, sp4, sp5, sp6)

lwjson_stack_seq_8(jsp, start_num, sp0, sp1, sp2, sp3, sp4, sp5, sp6, sp7)

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(lwobj)

Get number of tokens used to parse JSON.

Parameters

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

Returns

Number of tokens used to parse JSON

lwjson_get_first_token(lwobj)

Get very first token of LwJSON instance.

Parameters

- **lwobj** – [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 number 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.

```
typedef void (*lwjson_stream_parser_callback_fn)(struct lwjson_stream_parser *jsp,  
lwjson_stream_type_t type)
```

Callback function for various events.

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** = 0x00

Function returns successfully

enumerator **lwjsonERR**

Generic error message

enumerator **lwjsonERRJSON**

Error JSON format

enumerator **lwjsonERRMEM**

Memory error

enumerator **lwjsonERRPAR**

Parameter error

enumerator **lwjsonSTREAMWAITFIRSTCHAR**

Streaming parser did not yet receive first valid character indicating start of JSON sequence

enumerator **lwjsonSTREAMDONE**

Streaming parser is done, closing character matched the stream opening one

enumerator **lwjsonSTREAMINPROG**

Stream parsing is still in progress

enum **lwjson_stream_type_t**

Object type for streaming parser.

Values:

enumerator **LWJSON_STREAM_TYPE_NONE**

No entry - not used

enumerator **LWJSON_STREAM_TYPE_OBJECT**

Object indication

enumerator **LWJSON_STREAM_TYPE_OBJECT_END**

Object end indication

enumerator **LWJSON_STREAM_TYPE_ARRAY**

Array indication

enumerator **LWJSON_STREAM_TYPE_ARRAY_END**

Array end indication

enumerator **LWJSON_STREAM_TYPE_KEY**

Key string

enumerator **LWJSON_STREAM_TYPE_STRING**

String type

enumerator **LWJSON_STREAM_TYPE_TRUE**

True primitive

enumerator **LWJSON_STREAM_TYPE_FALSE**

False primitive

enumerator **LWJSON_STREAM_TYPE_NULL**

Null primitive

enumerator **LWJSON_STREAM_TYPE_NUMBER**

Generic number

enum **lwjson_stream_state_t**

Values:

enumerator **LWJSON_STREAM_STATE_WAITINGFIRSTCHAR** = 0x00

State to wait for very first opening character

enumerator **LWJSON_STREAM_STATE_PARSING**

In parsing of the first char state - detecting next character state

enumerator **LWJSON_STREAM_STATE_PARSING_STRING**

Parse string primitive

enumerator **LWJSON_STREAM_STATE_PARSING_PRIMITIVE**

Parse any primitive that is non-string, either “true”, “false”, “null” or a number

enumerator **LWJSON_STREAM_STATE_EXPECTING_COMMA_OR_END**

Expecting ‘,’ ‘}’ or ‘]’

enumerator **LWJSON_STREAM_STATE_EXPECTING_COLON**

Expecting ‘:’

Functions

lwjsonr_t **lwjson_init**(*lwjson_t* *lwobj, *lwjson_token_t* *tokens, size_t tokens_len)

Setup LwJSON instance for parsing JSON strings.

Parameters

- **lwobj** – [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* *lwobj, const void *json_data, size_t len)

Parse JSON data with length parameter JSON format must be complete and must comply with RFC4627.

Parameters

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

Returns

lwjsonOK on success, member of *lwjsonr_t* otherwise

lwjsonr_t **lwjson_parse**(*lwjson_t* *lwobj, const char *json_str)

Parse input JSON format JSON format must be complete and must comply with RFC4627.

Parameters

- **lwobj** – [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 *lwbj, const char *path)
```

Find first match in the given path for JSON entry JSON must be valid and parsed with *lwjson_parse* function.

Parameters

- **lwbj** – [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 *lwbj, 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

- **lwbj** – [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 *lwbj)
```

Free token instances (specially used in case of dynamic memory allocation)

Parameters

lwbj – [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 *lwbj)
```

Prints and outputs full parsed LwJSON instance.

Note: This function is not re-entrant

Parameters

lwbj – [in] LwJSON instance to print

```
lwjsonr_t lwjson_stream_init(lwjson_stream_parser_t *jsp, lwjson_stream_parser_callback_fn evt_fn)
```

Initialize LwJSON stream object before parsing takes place.

Parameters

jsp – [inout] Stream JSON structure

Returns

lwjsonOK on success, member of *lwjsonr_t* otherwise

lwjsonr_t **lwjson_stream_set_user_data**(*lwjson_stream_parser_t* *jsp, void *user_data)

Set user_data in stream parser.

Parameters

- **jsp** – [inout] LwJSON stream parser
- **user_data** – [in] user data

Returns

lwjsonOK on success, member of *lwjsonr_t* otherwise

void ***lwjson_stream_get_user_data**(*lwjson_stream_parser_t* *jsp)

Get user_data in stream parser.

Parameters

jsp – [in] LwJSON stream parser

Returns

pointer to user data

lwjsonr_t **lwjson_stream_reset**(*lwjson_stream_parser_t* *jsp)

Reset LwJSON stream structure.

Parameters

jsp – [inout] LwJSON stream parser

Returns

lwjsonOK on success, member of *lwjsonr_t* otherwise

lwjsonr_t **lwjson_stream_parse**(*lwjson_stream_parser_t* *jsp, char c)

Parse JSON string in streaming mode.

Parameters

- **jsp** – [inout] Stream JSON structure
- **chr** – [in] Character to parse

Returns

lwjsonSTREAMWAITFIRSTCHAR when stream did not start parsing since no valid start character has been received

Returns

lwjsonSTREAMINPROG if parsing is in progress and no hard error detected

Returns

lwjsonSTREAMDONE when valid JSON was detected and stack level reached back 0 level

Returns

One of enumeration otherwise

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** – [in] NULL-terminated 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** – [in] NULL-terminated string to compare
- **len** – [in] Length of the string in bytes

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

struct **lwjson_stream_stack_t**
#include <lwjson.h> Stream parsing stack object.

Public Members

lwjson_stream_type_t **type**

Streaming type - current value

char **name**[LWJSON_CFG_STREAM_KEY_MAX_LEN + 1]

Last known key name, used only for *LWJSON_STREAM_TYPE_KEY* type

uint16_t **index**

Current index when type is an array

union *lwjson_stream_stack_t*::[anonymous] **meta**

Meta information

struct **lwjson_stream_parser_t**

#include <lwjson.h> LwJSON streaming structure.

Public Members

lwjson_stream_stack_t **stack**[LWJSON_CFG_STREAM_STACK_SIZE]

Stack used for parsing. TODO: Add conditional compilation flag

size_t **stack_pos**

Current stack position

lwjson_stream_state_t **parse_state**

Parser state

lwjson_stream_parser_callback_fn **evt_fn**

Event function for user

void ***user_data**

User data for callback function

char **buff**[LWJSON_CFG_STREAM_STRING_MAX_LEN + 1]

Buffer to write temporary data. TODO: Size to be variable with define

Temporary write buffer

size_t **buff_pos**

Buffer position for next write (length of bytes in buffer)

Buffer position for next write

size_t **buff_total_pos**

Total buffer position used up to now (in several data chunks)

uint8_t **is_last**

Status indicates if this is the last part of the string

struct *lwjson_stream_parser_t*::[anonymous]::[anonymous] **str**

String structure. It is only used for keys and string objects. Use primitive part for all other options

struct *lwjson_stream_parser_t*::[anonymous]::[anonymous] **prim**

Primitive object. Used for all types, except key or string

union *lwjson_stream_parser_t*::[anonymous] **data**

Data union used to parse various

char **prev_c**

History of characters

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

LWJSON_MEMSET(dst, val, len)

Memory set function.

Note: Function footprint is the same as memset

LWJSON_MEMCPY(dst, src, len)

Memory copy function.

Note: Function footprint is the same as memcpy

JSON streaming configuration.

Defines**LWJSON_CFG_STREAM_KEY_MAX_LEN**

Max length of token key (object key name) to be available for stack storage.

LWJSON_CFG_STREAM_STACK_SIZE

Max stack size (depth) in units of *lwjson_stream_stack_t* structure.

LWJSON_CFG_STREAM_STRING_MAX_LEN

Max size of string for single parsing in units of bytes.

LWJSON_CFG_STREAM_PRIMITIVE_MAX_LEN

Max number of bytes used to parse primitive.

Primitives are all numbers and logical values (null, true, false)

6.4 Changelog

```
# Changelog

## Develop

- Add clang-tidy
- Add helper functions for sequence check in stream parsing
- Add support to discard invalid JSON stream

## 1.6.1

- Fix critical issue - missing correct return when waiting for first character. Should_
```

(continues on next page)

```
↪be `lwjsonSTREAMWAITFIRSTCHAR`

## 1.6.0

- Split CMakeLists.txt files between library and executable
- Change license year to 2022
- Fix GCC warning for incompatible comparison types
- Update code style with astyle
- Add support for stream parsing - first version
- Add `.clang-format`
- Add `lwjsonSTREAMDONE` return code when streamer well parsed some JSON and reached end_
↪of string
- Add option to reset stream state machine

## 1.5.0

- Add string compare feature
- Add string compare with custom length feature

## 1.4.0

- Add support with input string with length specifier
- Add VSCode project for Win32 compilation

## 1.3.0

- Added support for inline `/* */` comments

## v1.2.0

- Added `lwjson_find_ex` function to accept token pointer as starting reference
- Update of the docs for `*find*`
- Remove unused reset and add free function for future dynamic allocation support

## v1.1.0

- Improved find algorithm to match array index
- Added more test code

## v1.0.2

- Fix wrong parsing of hex in some corner cases
- Add more robust code to handle errorneous JSON input

## v1.0.1

- Added test code
- Fixed bug with improper string parsing

## v1.0.0

- First stable release
```

(continues on next page)

(continued from previous page)

- Compliant with RFC 4627 for JSON
- Full features JSON parser

6.5 Authors

List of authors and contributors to the library

```
Tilen Majerle <tilen.majerle@gmail.com>
Tilen Majerle <tilen@majerle.eu>
Tristen Pierson <tpierson@bitconcepts.tech>
erics <eric.sidorov@ayyeka.com>
Eric Sidorov <ericsidorov@gmail.com>
erics <ericsidorov@gmail.com>
Josef Salda <josef.salda@mujmail.cz>
```


INDEX

L

- LWJSON_ARRAYSIZE (*C macro*), 24
- LWJSON_CFG_COMMENTS (*C macro*), 34
- LWJSON_CFG_INT_TYPE (*C macro*), 34
- LWJSON_CFG_REAL_TYPE (*C macro*), 34
- LWJSON_CFG_STREAM_KEY_MAX_LEN (*C macro*), 35
- LWJSON_CFG_STREAM_PRIMITIVE_MAX_LEN (*C macro*), 35
- LWJSON_CFG_STREAM_STACK_SIZE (*C macro*), 35
- LWJSON_CFG_STREAM_STRING_MAX_LEN (*C macro*), 35
- lwjson_find (*C++ function*), 28
- lwjson_find_ex (*C++ function*), 29
- lwjson_free (*C++ function*), 29
- lwjson_get_first_child (*C macro*), 25
- lwjson_get_first_token (*C macro*), 24
- lwjson_get_tokens_used (*C macro*), 24
- lwjson_get_val_int (*C macro*), 24
- lwjson_get_val_real (*C macro*), 24
- lwjson_get_val_string (*C++ function*), 30
- lwjson_get_val_string_length (*C macro*), 25
- lwjson_init (*C++ function*), 28
- lwjson_int_t (*C++ type*), 25
- LWJSON_MEMCPY (*C macro*), 35
- LWJSON_MEMSET (*C macro*), 35
- lwjson_parse (*C++ function*), 28
- lwjson_parse_ex (*C++ function*), 28
- lwjson_print_json (*C++ function*), 29
- lwjson_print_token (*C++ function*), 29
- lwjson_real_t (*C++ type*), 25
- lwjson_stack_seq_1 (*C macro*), 23
- lwjson_stack_seq_2 (*C macro*), 24
- lwjson_stack_seq_3 (*C macro*), 24
- lwjson_stack_seq_4 (*C macro*), 24
- lwjson_stack_seq_5 (*C macro*), 24
- lwjson_stack_seq_6 (*C macro*), 24
- lwjson_stack_seq_7 (*C macro*), 24
- lwjson_stack_seq_8 (*C macro*), 24
- lwjson_stream_get_user_data (*C++ function*), 30
- lwjson_stream_init (*C++ function*), 29
- lwjson_stream_parse (*C++ function*), 30
- lwjson_stream_parser_callback_fn (*C++ type*), 25
- lwjson_stream_parser_t (*C++ struct*), 33
 - lwjson_stream_parser_t::buff (*C++ member*), 33
 - lwjson_stream_parser_t::buff_pos (*C++ member*), 33
 - lwjson_stream_parser_t::buff_total_pos (*C++ member*), 33
 - lwjson_stream_parser_t::data (*C++ member*), 34
 - lwjson_stream_parser_t::evt_fn (*C++ member*), 33
 - lwjson_stream_parser_t::is_last (*C++ member*), 33
 - lwjson_stream_parser_t::parse_state (*C++ member*), 33
 - lwjson_stream_parser_t::prev_c (*C++ member*), 34
 - lwjson_stream_parser_t::prim (*C++ member*), 34
 - lwjson_stream_parser_t::stack (*C++ member*), 33
 - lwjson_stream_parser_t::stack_pos (*C++ member*), 33
 - lwjson_stream_parser_t::str (*C++ member*), 34
 - lwjson_stream_parser_t::user_data (*C++ member*), 33
- lwjson_stream_reset (*C++ function*), 30
- lwjson_stream_set_user_data (*C++ function*), 30
- lwjson_stream_stack_t (*C++ struct*), 32
 - lwjson_stream_stack_t::index (*C++ member*), 33
 - lwjson_stream_stack_t::meta (*C++ member*), 33
 - lwjson_stream_stack_t::name (*C++ member*), 33
 - lwjson_stream_stack_t::type (*C++ member*), 33
- lwjson_stream_state_t (*C++ enum*), 27
 - lwjson_stream_state_t::LWJSON_STREAM_STATE_EXPECTING_COLON (*C++ enumerator*), 28
 - lwjson_stream_state_t::LWJSON_STREAM_STATE_EXPECTING_COMMA (*C++ enumerator*), 28
 - lwjson_stream_state_t::LWJSON_STREAM_STATE_PARSING (*C++ enumerator*), 27
 - lwjson_stream_state_t::LWJSON_STREAM_STATE_PARSING_PRIMITIVE (*C++ enumerator*), 28
 - lwjson_stream_state_t::LWJSON_STREAM_STATE_PARSING_STRING (*C++ enumerator*), 28
 - lwjson_stream_state_t::LWJSON_STREAM_STATE_WAITINGFIRSTCHAR (*C++ enumerator*), 27
- lwjson_stream_type_t (*C++ enum*), 27

`lwjson_stream_type_t::LWJSON_STREAM_TYPE_ARRAY` `lwjson_type_t::LWJSON_TYPE_NUM_REAL` (C++ enumerator), 27
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_ARRAY` `lwjson_type_t::LWJSON_TYPE_OBJECT` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_FALSE` `lwjson_type_t::LWJSON_TYPE_STRING` (C++ enumerator), 25
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_KEY` `lwjson_type_t::LWJSON_TYPE_TRUE` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_NONE` `lwjsonr_t` (C++ enum), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_NULL` `lwjsonr_t::lwjsonERR` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_NULL` `lwjsonr_t::lwjsonERRJSON` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_NULL` `lwjsonr_t::lwjsonERRMEM` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_NUMBER` `lwjsonr_t::lwjsonERRPAR` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_NUMBER` `lwjsonr_t::lwjsonOK` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_OBJECT` `lwjsonr_t::lwjsonSTREAMDONE` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_OBJECT` `lwjsonr_t::lwjsonSTREAMINPROG` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_STRING` `lwjsonr_t::lwjsonSTREAMWAITFIRSTCHAR` (C++ enumerator), 26
`lwjson_stream_type_t::LWJSON_STREAM_TYPE_TRUE`
(C++ enumerator), 27
`lwjson_string_compare` (C++ function), 31
`lwjson_string_compare_n` (C++ function), 31
`lwjson_t` (C++ struct), 32
`lwjson_t::first_token` (C++ member), 32
`lwjson_t::flags` (C++ member), 32
`lwjson_t::next_free_token_pos` (C++ member), 32
`lwjson_t::parsed` (C++ member), 32
`lwjson_t::tokens` (C++ member), 32
`lwjson_t::tokens_len` (C++ member), 32
`lwjson_token_t` (C++ struct), 31
`lwjson_token_t::first_child` (C++ member), 32
`lwjson_token_t::next` (C++ member), 31
`lwjson_token_t::num_int` (C++ member), 32
`lwjson_token_t::num_real` (C++ member), 32
`lwjson_token_t::str` (C++ member), 31
`lwjson_token_t::token_name` (C++ member), 31
`lwjson_token_t::token_name_len` (C++ member), 31
`lwjson_token_t::token_value` (C++ member), 31
`lwjson_token_t::token_value_len` (C++ member), 31
`lwjson_token_t::type` (C++ member), 31
`lwjson_token_t::u` (C++ member), 32
`lwjson_type_t` (C++ enum), 25
`lwjson_type_t::LWJSON_TYPE_ARRAY` (C++ enumerator), 26
`lwjson_type_t::LWJSON_TYPE_FALSE` (C++ enumerator), 26
`lwjson_type_t::LWJSON_TYPE_NULL` (C++ enumerator), 26
`lwjson_type_t::LWJSON_TYPE_NUM_INT` (C++ enumerator), 25