[![Issue Stats](http://issuestats.com/github/nlohmann/json/badge/pr?style=flat)](http://issuestats.com/github/nlohmann/json) [![Issue Stats](http://issuestats.com/github/nlohmann/json/badge/issue?style=flat)](http://issuestats.com/github/nlohmann/json)

# How to contribute

This project started as a little excuse to exercise some of the cool new C++11 features. Over time, people actually started to use the JSON library (yey!) and started to help improve it by proposing features, finding bugs, or even fixing my mistakes. I am really [thankful](https://github.com/nlohmann/json/blob/master/README.md#thanks) for this and try to keep track of all the helpers.

To make it as easy as possible for you to contribute and for me to keep an overview, here are a few guidelines which should help us avoid all kinds of unnecessary work or disappointment. And of course, this document is subject to discussion, so please [create an issue](https://github.com/nlohmann/json/issues/new/choose) or a pull request if you find a way to improve it!

## Private reports

Usually, all issues are tracked publicly on [GitHub](https://github.com/nlohmann/json/issues). If you want to make a private report (e.g., for a vulnerability or to attach an example that is not meant to be published), please send an email to <mail@nlohmann.me>.

## Prerequisites

Please [create an issue](https://github.com/nlohmann/json/issues/new/choose), assuming one does not already exist, and describe your concern. Note you need a [GitHub account](https://github.com/signup/free) for this.

## Describe your issue

Clearly describe the issue:

- If it is a bug, please describe how to \*\*reproduce\*\* it. If possible, attach a complete example which demonstrates the error. Please also state what you \*\*expected\*\* to happen instead of the error.

- If you propose a change or addition, try to give an \*\*example\*\* how the improved code could look like or how to use it.

- If you found a compilation error, please tell us which \*\*compiler\*\* (version and operating system) you used and paste the (relevant part of) the error messages to the ticket.

Please stick to the provided issue templates ([bug report](https://github.com/nlohmann/json/blob/develop/.github/ISSUE\_TEMPLATE/Bug\_report.md), [feature request](https://github.com/nlohmann/json/blob/develop/.github/ISSUE\_TEMPLATE/Feature\_request.md), or [question](https://github.com/nlohmann/json/blob/develop/.github/ISSUE\_TEMPLATE/question.md)) if possible.

## Files to change

:exclamation: Before you make any changes, note the single-header file [`single\_include/nlohmann/json.hpp`](https://github.com/nlohmann/json/blob/develop/single\_include/nlohmann/json.hpp) is \*\*generated\*\* from the source files in the [`include/nlohmann` directory](https://github.com/nlohmann/json/tree/develop/include/nlohmann). Please \*\*do not\*\* edit file `single\_include/nlohmann/json.hpp` directly, but change the `include/nlohmann` sources and regenerate file `single\_include/nlohmann/json.hpp` by executing `make amalgamate`.

To make changes, you need to edit the following files:

1. [`include/nlohmann/\*`](https://github.com/nlohmann/json/tree/develop/include/nlohmann) - These files are the sources of the library. Before testing or creating a pull request, execute `make amalgamate` to regenerate `single\_include/nlohmann/json.hpp`.

2. [`test/src/unit-\*.cpp`](https://github.com/nlohmann/json/tree/develop/test/src) - These files contain the [doctest](https://github.com/onqtam/doctest) unit tests which currently cover [100 %](https://coveralls.io/github/nlohmann/json) of the library's code.

If you add or change a feature, please also add a unit test to this file. The unit tests can be compiled and executed with

```sh

$ mkdir build

$ cd build

$ cmake ..

$ cmake --build .

$ ctest

```

The test cases are also executed with several different compilers on [Travis](https://travis-ci.org/nlohmann/json) once you open a pull request.

## Note

- If you open a pull request, the code will be automatically tested with [Valgrind](http://valgrind.org)'s Memcheck tool to detect memory leaks. Please be aware that the execution with Valgrind \_may\_ in rare cases yield different behavior than running the code directly. This can result in failing unit tests which run successfully without Valgrind.

- There is a Makefile target `make pretty` which runs [Artistic Style](http://astyle.sourceforge.net) to fix indentation. If possible, run it before opening the pull request. Otherwise, we shall run it afterward.

## Please don't

- The C++11 support varies between different \*\*compilers\*\* and versions. Please note the [list of supported compilers](https://github.com/nlohmann/json/blob/master/README.md#supported-compilers). Some compilers like GCC 4.7 (and earlier), Clang 3.3 (and earlier), or Microsoft Visual Studio 13.0 and earlier are known not to work due to missing or incomplete C++11 support. Please refrain from proposing changes that work around these compiler's limitations with `#ifdef`s or other means.

- Specifically, I am aware of compilation problems with \*\*Microsoft Visual Studio\*\* (there even is an [issue label](https://github.com/nlohmann/json/issues?utf8=?&q=label%3A%22visual+studio%22+) for these kind of bugs). I understand that even in 2016, complete C++11 support isn't there yet. But please also understand that I do not want to drop features or uglify the code just to make Microsoft's sub-standard compiler happy. The past has shown that there are ways to express the functionality such that the code compiles with the most recent MSVC - unfortunately, this is not the main objective of the project.

- Please refrain from proposing changes that would \*\*break [JSON](https://json.org) conformance\*\*. If you propose a conformant extension of JSON to be supported by the library, please motivate this extension.

- We shall not extend the library to \*\*support comments\*\*. There is quite some [controversy](https://www.reddit.com/r/programming/comments/4v6chu/why\_json\_doesnt\_support\_comments\_douglas\_crockford/) around this topic, and there were quite some [issues](https://github.com/nlohmann/json/issues/376) on this. We believe that JSON is fine without comments.

- We do not preserve the \*\*insertion order of object elements\*\*. The [JSON standard](https://tools.ietf.org/html/rfc7159.html) defines objects as "an unordered collection of zero or more name/value pairs". To this end, this library does not preserve insertion order of name/value pairs. (In fact, keys will be traversed in alphabetical order as `std::map` with `std::less` is used by default.) Note this behavior conforms to the standard, and we shall not change it to any other order. If you do want to preserve the insertion order, you can specialize the object type with containers like [`tsl::ordered\_map`](https://github.com/Tessil/ordered-map) or [`nlohmann::fifo\_map`](https://github.com/nlohmann/fifo\_map).

- Please do not open pull requests that address \*\*multiple issues\*\*.

## Wanted

The following areas really need contribution:

- Extending the \*\*continuous integration\*\* toward more exotic compilers such as Android NDK, Intel's Compiler, or the bleeding-edge versions Clang.

- Improving the efficiency of the \*\*JSON parser\*\*. The current parser is implemented as a naive recursive descent parser with hand coded string handling. More sophisticated approaches like LALR parsers would be really appreciated. That said, parser generators like Bison or ANTLR do not play nice with single-header files -- I really would like to keep the parser inside the `json.hpp` header, and I am not aware of approaches similar to [`re2c`](http://re2c.org) for parsing.

- Extending and updating existing \*\*benchmarks\*\* to include (the most recent version of) this library. Though efficiency is not everything, speed and memory consumption are very important characteristics for C++ developers, so having proper comparisons would be interesting.