build unknown

JSON schema validator for JSON for Modern C++

What is it?（JSON schema validator是什么？）

This is a C++ library for validating JSON documents based on a JSON Schema which itself should validate with draft-7 of JSON Schema Validation.（这是一个基于 JSON Schema 验证 JSON 文档的 C++ 库，该库本身应使用 JSON Schema 验证草案 7 进行验证。）

First a disclaimer: *It is work in progress and contributions or hints or discussions are welcome.*（首先声明：*这项工作仍在进行中，欢迎贡献、提示或讨论。*）

Niels Lohmann et al develop a great JSON parser for C++ called JSON for Modern C++. This validator is based on this library, hence the name.（Niels Lohmann 等人开发了一款出色的 C++ JSON 解析器，名为 JSON for Modern C++。此验证器基于此 库，因此得名。）

External documentation is missing as well. However the API of the validator is rather simple.（外部文档也缺失。不过验证器的 API 相当简单。）

New in version 2

Although significant changes have been done for the 2nd version (a complete rewrite) the API is compatible with the 1.0.0 release. Except for the namespace which is now `nlohmann::json_schema`.（尽管第二版进行了重大更改（完全重写），但 API 与 1.0.0 版本兼容。除了命名空间现在为"nlohmann::json_schema"。）

Version **2** supports JSON schema draft 7, whereas 1 was supporting draft 4 only. Please update your schemas.（版本 **2** 支持 JSON 架构草案 7，而版本 1 仅支持草案 4。请更新您的架构。）

The primary change in 2 is the way a schema is used. While in version 1 the schema was kept as a JSON-document and used again and again during validation, in version 2 the schema is parsed into compiled C++ objects which are then used during validation. There are surely still optimizations to be done, but validation speed has improved by factor 100 or more.（版本 2 的主要变化在于模式的使用方式。在版本 1 中，模式被保存为 JSON 文档并在验证过程中反复使用，而在版本 2 中，模式被解析为编译后的 C++ 对象，然后在验证过程中使用。当然，仍有优化需要完成，但验证速度已提高了 100 倍或更多。）

Design goals（设计目标）

The main goal of this validator is to produce *human-comprehensible* error messages if a JSON-document/instance does not comply to its schema.（此验证器的主要目标是，如果 JSON 文档/实例不符合其架构，则生成*人类可理解的*错误 消息。）

By default this is done with exceptions thrown at the users with a helpful message telling what's wrong with the document while validating.（默认情况下，这是通过向用户抛出异常来完成的，并显示一条有用的消息，告知验证过程中文档出了什么问题。）

Starting with **2.0.0** the user can pass a `json_schema::basic_error_handler`-derived object along with the instance to validate to receive a callback each time a validation error occurs and decide what to do (throwing, counting, collecting).（从 **2.0.0** 开始，用户可以传递一个 `json_schema::basic_error_handler` 派生的对象以及要验证的实例，以便在每次发生验证错误时接收回调并决定要做什么（抛出、计数、收集）。）

Another goal was to use Niels Lohmann's JSON-library. This is why the validator lives in his namespace.（另一个目标是使用 Niels Lohmann 的 JSON 库。这就是验证器存在于其命名空间中的原因。）

Thread-safety（线程安全）

Instance validation is thread-safe and the same validator-object can be used by different threads（实例验证是线程安全的，并且不同的线程可以使用相同的验证器对象）:

The validate method is `const` which indicates the object is not modified when being called（validate 方法是 const ，表示调用时对象不会被修改）:

```
json json_validator::validate(const json &) const;
```

Validator-object creation however is not thread-safe. A validator has to be created in one (main?) thread once.（但是验证器对象的创建不是线程安全的。验证器必须在一个（主？）线程中创建一次。）

Weaknesses（缺点）

Numerical validation uses nlohmann-json's integer, unsigned and floating point types, depending on if the schema type is "integer" or "number". Bignum (i.e. arbitrary precision and range) is not supported at this time.（数字验证使用 nlohmann-json 的整数、无符号和浮点类型，具体取决于架构类型是“整数”还是“数字”。目前不支持 Bignum（即任意精度和范围）。）

Building（构建）

This library is based on Niels Lohmann's JSON-library and thus has a build-dependency to it.（该库基于 Niels Lohmann 的 JSON 库，因此具有构建依赖性。）

Currently at least version **3.8.0** of NLohmann's JSON library is required.（目前至少需要 NLohmann 的 JSON 库的 **3.8.0** 版本。）

Various methods using CMake can be used to build this project.（可以使用多种使用 CMake 的方法来构建该项目。）

Build out-of-source

Do not run cmake inside the source-dir. Rather create a dedicated build-dir（不要在源目录中运行 cmake。而是创建一个专用的构建目录）:

```
git clone https://github.com/pboettch/json-schema-validator.git
cd json-schema-validator
mkdir build
cd build
cmake [..]
make
make install # if needed
ctest # run unit, non-regression and test-suite tests
```

Building as shared library（构建成一个共享库）

By default a static library is built. Shared libraries can be generated by using the `BUILD_SHARED_LIBS`-cmake variable（默认情况下会构建静态库。可以使用 `BUILD_SHARED_LIBS`-cmake 变量生成共享库）:

In your initial call to cmake simply add:

```
cmake [..] -DBUILD_SHARED_LIBS=ON [..]
```

nlohmann-json integration

As nlohmann-json is a dependency, this library tries find it.（由于 nlohmann-json 是一个依赖项，因此该库会尝试找到它。）

The cmake-configuration first checks if nlohmann-json is available as a cmake-target. This may be the case, because it is used as a submodule in a super-project which already provides and uses nlohmann-json. Otherwise, it calls `find_package` for nlohmann-json and requires nlohmann-json to be installed on the system.（cmake-configuration 首先检查 nlohmann-json 是否可用作 cmake-target。可能是这样，因为它被用作超级项目中的子模块，而超级项目已经提供并使用了 nlohmann-json。否则，它会调用 nlohmann-json 的 `find_package`，并要求在系统上安装 nlohmann-json。）

Building with Hunter package manager

To enable access to nlohmann json library, Hunter can be used. Just run with `JSON_VALIDATOR_HUNTER=ON` option. No further dependencies needed

```
cmake [..] -DJSON_VALIDATOR_HUNTER=ON [..]
```

Building as a CMake-subdirectory from within another project

Adding this library as a subdirectory to a parent project is one way of building it.

If the parent project already used `find_package()` to find the CMake-package of nlohmann_json or includes it as a submodule likewise.

Building directly, finding a CMake-package. (short)

When nlohmann-json has been installed, it provides files which allows CMake's `find_package()` to be used.

This library is using this mechanism if `nlohmann_json::nlohmann_json`-target does not exist.

Install

Since version 2.1.0 this library can be installed and CMake-package-files will be created accordingly. If the installation of nlohmann-json and this library is done into default unix-system-paths CMake will be able to find this library by simply doing:

```
find_package(nlohmann_json_schema_validator REQUIRED)
```

and

```
target_link_libraries(<your-target> [..] nlohmann_json_schema_validator)
```

to build and link.

Code

See also `app/json-schema-validate.cpp`.

```
#include <iostream>
#include <iomanip>

#include <nlohmann/json-schema.hpp>

using nlohmann::json;
using nlohmann::json_schema::json_validator;

// The schema is defined based upon a string literal
static json person_schema = R"(
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "A person",
    "properties": {
        "name": {
            "description": "Name",
            "type": "string"
        },
        "age": {
            "description": "Age of the person",
            "type": "number",
            "minimum": 2,
            "maximum": 200
        }
    },
    "required": [
                "name",
                "age"
                ],
    "type": "object"
}

)"_json;

// The people are defined with brace initialization
static json bad_person = {{"age", 42}};
static json good_person = {{"name", "Albert"}, {"age", 42}};

int main()
{
    /* json-parse the schema */

    json_validator validator; // create validator

    try {
        validator.set_root_schema(person_schema); // insert root-schema
    } catch (const std::exception &e) {
        std::cerr << "Validation of schema failed, here is why: " << e.what() <<
"\n";
        return EXIT_FAILURE;
    }

    /* json-parse the people - API of 1.0.0, default throwing error handler */

    for (auto &person : {bad_person, good_person}) {
        std::cout << "About to validate this person:\n"
                  << std::setw(2) << person << std::endl;
        try {
            validator.validate(person); // validate the document - uses the
default throwing error-handler
            std::cout << "Validation succeeded\n";
        } catch (const std::exception &e) {
            std::cerr << "Validation failed, here is why: " << e.what() << "\n";
        }
    }

    /* json-parse the people - with custom error handler */
    class custom_error_handler : public nlohmann::json_schema::basic_error_handler
    {
        void error(const nlohmann::json_pointer<nlohmann::basic_json<>> &pointer,
const json &instance,
            const std::string &message) override
        {
            nlohmann::json_schema::basic_error_handler::error(pointer, instance,
message);
            std::cerr << "ERROR: '" << pointer << "' - '" << instance << "': " <<
message << "\n";
        }
    };
    for (auto &person : {bad_person, good_person}) {
        std::cout << "About to validate this person:\n"
                  << std::setw(2) << person << std::endl;

        custom_error_handler err;
        validator.validate(person, err); // validate the document

        if (err)
            std::cerr << "Validation failed\n";
        else
            std::cout << "Validation succeeded\n";
    }

    return EXIT_SUCCESS;
}
```

Compliance

There is an application which can be used for testing the validator with the JSON-Schema-Test-Suite. In order to simplify the testing, the test-suite is included in the repository.

If you have cloned this repository providing a path the repository-root via the cmake-variable `JSON_SCHEMA_TEST_SUITE_PATH` will enable the test-target(s).

All required tests are **OK**.

Format

Optionally JSON-schema-validator can validate predefined or user-defined formats. Therefore a format-checker-function can be provided by the user which is called by the validator when a format-check is required (ie. the schema contains a format-field).

This is how the prototype looks like and how it can be passed to the validation-instance:

```
static void my_format_checker(const std::string &format, const std::string &value)
{
    if (format == "something") {
        if (!check_value_for_something(value))
            throw std::invalid_argument("value is not a good something");
    } else
        throw std::logic_error("Don't know how to validate " + format);
}

// when creating the validator

json_validator validator(nullptr, // or loader-callback
                         my_format_checker); // create validator
```

Default Checker

The library contains a default-checker, which does some checks. It needs to be provided manually to the constructor of the validator:

```
json_validator validator(loader, // or nullptr for no loader
                         nlohmann::json_schema::default_string_format_check);
```

Supported formats: `date-time, date, time, email, hostname, ipv4, ipv6, uuid, regex`

More formats can be added in `src/string-format-check.cpp`. Please contribute implementions for missing json schema draft formats.

Default value processing

As a result of the validation, the library returns a json patch including the default values of the specified schema.

```
#include <iostream>
#include <nlohmann/json-schema.hpp>

using nlohmann::json;
using nlohmann::json_schema::json_validator;

static const json rectangle_schema = R"(
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "A rectangle",
    "properties": {
        "width": {
            "$ref": "#/definitions/length",
            "default": 20
        },
        "height": {
            "$ref": "#/definitions/length"
        }
    },
    "definitions": {
        "length": {
            "type": "integer",
            "minimum": 1,
            "default": 10
        }
    }
})"_json;

int main()
{
    try {
        json_validator validator{rectangle_schema};
        /* validate empty json -> will be expanded by the default values defined
in the schema */
        json rectangle = "{}"_json;
        const auto default_patch = validator.validate(rectangle);
        rectangle = rectangle.patch(default_patch);
        std::cout << rectangle.dump() << std::endl; // {"height":10,"width":20}
    } catch (const std::exception &e) {
        std::cerr << "Validation of schema failed: " << e.what() << "\n";
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

The example above will output the specified default values `{"height":10,"width":20}` to stdout.

> Note that the default value specified in a `$ref` may be overridden by the current instance location. Also note that this behavior will break draft-7, but it is compliant to newer drafts (e.g. `2019-09` or `2020-12`).

Contributing

This project uses `pre-commit` to enforce style-checks. Please install and run it before creating commits and making pull requests.

```
$ pip install pre-commit
$ pre-commit install
```