config reader

v1.3.0

Table of contents

Overview

ConfigReader library designed to read/write config files or strings in JSON format. The library provides simple interface to read application configuration parameters from file/string or write configuration parameters to file/string. The library is a wrapper over the library JSON for Modern C++ (included as source code under MIT license) and provides simple interface to work with JSON config structures.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	31.01.2022	First version.

Version	Release date	What's new
1.1.0	13.02.2023	- Changed C++ version from C++11 to C++17.
1.2.0	26.04.2023	- CMake updated.
1.3.0	27.06.2023	- Useless methods excluded. - Added examples. - Added documentation. - Added license. - Repository made public.

ConfigReader class description

ConfigReader class declaration

ConfigReader.h file contains **ConfigReader** class declaration. Class declaration:

```
namespace cr
{
namespace utils
{
#define JSON_READABLE(Type, ...) \
    NLOHMANN_DEFINE_TYPE_INTRUSIVE(Type, __VA_ARGS__)
/**
 * @brief Config reader class.
 */
class ConfigReader
{
public:
    /**
     * @brief Get string of current class version.
     * @return String of current class version in format "Major.Minor.Patch".
     */
    static std::string getVersion();
    /**
     * @brief Class constructor.
     */
    ConfigReader();
    /**
     * @brief Class destructor.
     */
    ~ConfigReader();
    /**
     * @brief Read configuration from file.
     * @param fileName Configuration file name.
     * @return TRUE if the configuration was read or FALSE if not.
     */
    bool readFromFile(std::string fileName);
    /**
```

```
     * @brief Write configuration to file.
     * @param fileName File name to write configuration.
     * @return TRUE if the configuration was writed or FALSE if not.
     */
    bool writeToFile(std::string fileName);
    /**
     * @brief Read configuration from string.
     * @param json String to read configuration.
     * @return TRUE if the configuration was read or FALSE if not.
     */
    bool readFromString(std::string json);
    /**
     * @brief Write configuration to string.
     * @param json String name to write configuration.
     * @return TRUE if the configuration was writed or FALSE if not.
     */
    bool writeToString(std::string& json);
    /**
     * @brief Get object.
     * @param obj Object.
     * @param objName Name of object/property.
     * @return TRUE If the object was read or FALSE if not.
     */
    template<class T>
    bool get(T& obj, std::string objName = "");
    /**
     * @brief Set object.
     * @param obj Object.
     * @param objName Name of object/property.
     * @return TRUE If the object was set or FALSE if not.
     */
    template<class T>
    bool set(T& obj, std::string objName = "");
};
}
}
```

Basic principles

Before using **ConfigReader**, the user must define the parameters structure. The parameter structure is a class whose fields are marked with a special macro (**JSON_READABLE** macro) for reading and writing to JSON. Parameters are written and read according to the user-defined parameter class. Class example:

```
/// Custom parameter class.
class CustomParam
{
public:
    /// Parameter 1.
    int param1{1};
    /// Parameter 2.
    int param2{2};
```

```
    /// Declare params as readable for JSON.
    JSON_READABLE(CustomParam, param1, param2)
};

/// Parameters class.
class Params
{
public:
    /// Parameter.
    int param1{-10};
    /// Parameter.
    float param2{10.0f};
    /// Parameter.
    double param3{-10.0};
    /// Parameter.
    bool param4{false};
    /// Mass of parameters.
    float mass1[3];
    /// Mass of parameters.
    int mass2[3];
    /// List of parameters.
    vector<int> list1{vector<int>()};
    /// List of objects.
    vector<CustomParam> list2{vector<CustomParam>()};

    /// Declare params as readable for JSON.
    JSON_READABLE(Params, param1, param2, param3, mass1, list1, list2)
};
```

ConfigReader supports all basic types (int, float, double, string, bool, buffers) and list of other classes as well. To be readable/writable to json variables should be listed in **JSON_READABLE** macro. Some parameters can be excluded from JSON and user can use them as private fields. Example of JSON file according to given class (as you can see **param4** and **mass2** excluded from JSON):

```
{
    "Params": {
        "list1": [
            235,
            44,
            169,
            3
        ],
        "list2": [
            {
                "param1": 33,
                "param2": 187
            },
            {
                "param1": 239,
                "param2": 95
            },
            {
                "param1": 95,
                "param2": 76
```

```
            },
            {
                "param1": 252,
                "param2": 16
            }
        ],
        "mass1": [
            236.0,
            190.0,
            212.0
        ],
        "param1": 41,
        "param2": 107.0,
        "param3": 214.0
    }
}
```

ConfigReader reads and writes params only according to defined parameters class. During reading params from file/string if parameter has different structure as define params the library will return ERROR.

getVersion method

getVersion() method returns string of **ConfigReader** class version. Method declaration:

```
static std::string getVersion();
```

Method can be used without **ConfigReader** class instance. Example:

```
cout << "ConfigReader class version: " << ConfigReader::getVersion() << endl;
```

Console output:

```
ConfigReader class version: 1.3.0
```

readFromFile method

readFromFile(...) method designed to read config file. The method reads the contents of the config file and stores it in the internal **ConfigReader** class variables. Method declaration:

```
bool readFromFile(std::string fileName);
```

Parameter	Description
fileName	Configuration file name.

Returns: TRUE if config file was read or FALSE if not.

Example:

```
ConfigReader inConfig;
if(!inConfig.readFromFile("TestConfig.json"))
    cout << "ERROR: Can't read file" << endl;
```

writeToFile method

writeToFile(...) method designed to write config file. If config file already exists the method will rewrite it. To write config file user must call method **set(...)** before **writeToFile(...)** to put parameters structure into **ConfigReader** object. Method declaration:

```
bool writeToFile(std::string fileName);
```

Parameter	Description
fileName	File name store parameters structure.

Returns: TRUE if file was write or FALSE if not.

Example:

```
// Copy params to config reader.
ConfigReader outConfig;
if (!outConfig.set(outParams, "Params"))
    cout << "ERROR: Can't copy params" << endl;

// Create and write JSON file.
if (!outConfig.writeToFile("TestConfig.json"))
    cout << "ERROR: Can't write file" << endl;
```

readFromString method

readFromString(...) method designed to read parameters from string. The method reads the contents of the string and stores it in the internal **ConfigReader** class variables. Method declaration:

```
bool readFromString(std::string json);
```

Parameter	Description
json	String with parameters structure.

Returns: TRUE if string was read or FALSE if not.

Example:

```
ConfigReader inConfig;
if(!inConfig.readFromString(jsonData))
    cout << "ERROR: Can't read file" << endl;
```

writeToString method

writeToString(...) method designed to write params to string. To write parameters to string user must call method **set(...)** before **writeToString(...)** to put parameters structure into **ConfigReader** object. Method declaration:

```
bool writeToString(std::string& json);
```

Parameter	Description
json	String to store parameters structure.

Returns: TRUE if string was write or FALSE if not.

Example:

```
// Copy params to config reader.
ConfigReader outConfig;
if (!outConfig.set(outParams, "Params"))
    cout << "ERROR: Can't copy params" << endl;

// Write params to string.
string jsonData;
if (!outConfig.writeToString(jsonData))
    cout << "ERROR: Can't write file" << endl;
```

get method

get(...) method designed to parse parameters structure which was read from file or string. The method parser parameters according to given parameters structure (see **Basic principles**). Method declaration:

```
template<class T>
bool get(T& obj, std::string objName = "");
```

Parameter	Description
obj	Parameters class object (see **Basic principles**). This object (parameters initialization) will be initialized by method.
objName	Name of object/parameters. To be read successfully, the parameter structure (loaded from a file or string) must match a user-defined parameter class (see **Basic principles**).

Returns: TRUE if parameters was parsed FALSE if not.

Example:

```
// Read params from file.
ConfigReader inConfig;
if(!inConfig.readFromFile("TestConfig.json"))
    cout << "ERROR: Can't read file" << endl;

// Read values from "Params" section.
Params inParams;
if(!inConfig.get(inParams, "Params"))
    cout << "ERROR: Can't read params" << endl;
```

set method

set(...) method designed put parameters structure into **ConfigReader** class to write to file (**writeToFile(...)** method) or string (**writeToString(...)** method) (see **Basic principles**). Method declaration:

```
template<class T>
bool set(T& obj, std::string objName = "");
```

Parameter	Description
obj	Parameters class object (see **Basic principles**).
objName	Name of object/parameters.

Returns: TRUE if parameters was copied FALSE if not.

Example:

```
// Copy params to config reader.
ConfigReader outConfig;
if (!outConfig.set(outParams, "Params"))
    cout << "ERROR: Can't copy params" << endl;

// Create and write JSON file.
if (!outConfig.writeToFile("TestConfig.json"))
    cout << "ERROR: Can't write file" << endl;
```

Examples

Read and write config file

Example shows how to create parameters class, how to write parameters to file and how to read parameters from file. First we create parameters class and add necessary params to read/write (some parameters excluded from JSON manipulations). After we initialize parameters by random values. After we write parameters to file. After we read parameters from file and compare with initial values.

```
#include <iostream>
#include "ConfigReader.h"

/// Link namespaces.
using namespace std;
using namespace cr::utils;

/// Custom parameter class.
class CustomParam
{
public:
    /// Parameter 1.
    int param1{1};
    /// Parameter 2.
    int param2{2};

    /// Declare params as readable for JSON.
    JSON_READABLE(CustomParam, param1, param2)
};

/// Parameters class.
class Params
{
public:
    /// Parameter.
    int param1{-10};
    /// Parameter.
    float param2{10.0f};
    /// Parameter.
    double param3{-10.0};
    /// Parameter.
    bool param4{false};
    /// Mass of parameters.
    float mass1[3];
    /// Mass of parameters.
    int mass2[3];
    /// List of parameters.
    vector<int> list1{vector<int>()};
    /// List of objects.
    vector<CustomParam> list2{vector<CustomParam>()};

    /// Declare params as readable for JSON.
    JSON_READABLE(Params, param1, param2, param3, mass1, list1, list2)
};

// Entry point.
```

```cpp
int main(void)
{
    cout<< "==================================================" << endl;
    cout<< "ConfigReader v" << ConfigReader::getVersion()       << endl;
    cout<< "Write and read file example"                        << endl;
    cout<< "==================================================" << endl;
    cout<< endl;

    // Prepare parameters.
    Params outParams;
    outParams.param1 = rand() % 255;
    outParams.param2 = (float)(rand() % 255);
    outParams.param3 = (double)(rand() % 255);
    outParams.param4 = true; // Will not be saved.
    for (int i = 0; i < 4; ++i)
    {
        outParams.list1.push_back(rand() % 255);
    }
    for (int i = 0; i < 4; ++i)
    {
        CustomParam param;
        param.param1 = rand() % 255;
        param.param2 = rand() % 255;
        outParams.list2.push_back(param);
    }
    for (int i = 0; i < 4; ++i)
        outParams.mass1[i] = (float)(rand() % 255);
    for (int i = 0; i < 4; ++i)
        outParams.mass2[i] = (int)(rand() % 255);

    // Copy params to config reader.
    ConfigReader outConfig;
    if (!outConfig.set(outParams, "Params"))
    {
        cout << "ERROR: Can't copy params" << endl;
        return -1;
    }

    // Create and write JSON file.
    if (!outConfig.writeToFile("TestConfig.json"))
    {
        cout << "ERROR: Can't write file" << endl;
        return -1;
    }

    // Read params from file.
    ConfigReader inConfig;
    if(!inConfig.readFromFile("TestConfig.json"))
    {
        cout << "ERROR: Can't read file" << endl;
        return -1;
    }

    // Read values from "Params" section.
    Params inParams;
    if(!inConfig.get(inParams, "Params"))
```

```
    {
        cout << "ERROR: Can't read params" << endl;
        return -1;
    }

    // Compare params.
    if (outParams.param1 != inParams.param1)
        cout << "ERROR: param1 has wrong value" << endl;
    if (outParams.param2 != inParams.param2)
        cout << "ERROR: param2 has wrong value" << endl;
    if (outParams.param3 != inParams.param3)
        cout << "ERROR: param3 has wrong value" << endl;
    if (outParams.param4 != inParams.param4)
        cout << "OK: param4 not included to list for reading/writing" << endl;
    for (int i = 0; i < 3; ++i)
        if (outParams.mass1[i] != inParams.mass1[i])
            cout << "ERROR: mass1[" << i << "] has wrong value" << endl;
    for (int i = 0; i < 3; ++i)
        if (outParams.mass2[i] != inParams.mass2[i])
            cout << "OK: mass2[" << i << "] not included to list for" <<
                    "reading/writing" << endl;
    if (outParams.list1.size() == inParams.list1.size())
    {
        for (int i = 0; i < inParams.list1.size(); ++i)
            if (outParams.list1[i] != inParams.list1[i])
                cout << "ERROR: list1[" << i << "] has wrong value" << endl;
    }
    else
    {
        cout << "ERROR: list1 has wrong size" << endl;
    }
    if (outParams.list2.size() == inParams.list2.size())
    {
        for (int i = 0; i < inParams.list2.size(); ++i)
            if (outParams.list2[i].param1 != inParams.list2[i].param1 ||
                outParams.list2[i].param2 != inParams.list2[i].param2)
                cout << "ERROR: list1[" << i << "] has wrong value" << endl;
    }
    else
    {
        cout << "ERROR: list2 has wrong size" << endl;
    }
}
```

Result **TestConfig.json** files for this example:

```
{
    "Params": {
        "list1": [
            235,
            44,
            169,
            3
        ],
        "list2": [
```

```
            {
                "param1": 33,
                "param2": 187
            },
            {
                "param1": 239,
                "param2": 95
            },
            {
                "param1": 95,
                "param2": 76
            },
            {
                "param1": 252,
                "param2": 16
            }
        ],
        "mass1": [
            236.0,
            190.0,
            212.0
        ],
        "param1": 41,
        "param2": 107.0,
        "param3": 214.0
    }
}
```

Read and write parameters to string

Example shows how to create parameters class, how to write parameters to string and how to read parameters from string. First we create parameters class and add necessary params to read/write (some parameters excluded from JSON manipulations). After we initialize parameters by random values. After we write parameters to string. After we read parameters from string and compare with initial values.

```
#include <iostream>
#include "ConfigReader.h"

/// Link namespaces.
using namespace std;
using namespace cr::utils;

/// Custom parameter class.
class CustomParam
{
public:
    /// Parameter 1.
    int param1{1};
    /// Parameter 2.
    int param2{2};
```

```
    /// Declare params as readable for JSON.
    JSON_READABLE(CustomParam, param1, param2)
};

/// Parameters class.
class Params
{
public:
    /// Parameter.
    int param1{-10};
    /// Parameter.
    float param2{10.0f};
    /// Parameter.
    double param3{-10.0};
    /// Parameter.
    bool param4{false};
    /// Mass of parameters.
    float mass1[3];
    /// Mass of parameters.
    int mass2[3];
    /// List of parameters.
    vector<int> list1{vector<int>()};
    /// List of objects.
    vector<CustomParam> list2{vector<CustomParam>()};

    /// Declare params as readable for JSON.
    JSON_READABLE(Params, param1, param2, param3, mass1, list1, list2)
};

// Entry point.
int main(void)
{
    cout<< "==================================================" << endl;
    cout<< "ConfigReader v" << ConfigReader::getVersion()       << endl;
    cout<< "Write and read file example"                        << endl;
    cout<< "==================================================" << endl;
    cout<< endl;

    // Prepare parameters.
    Params outParams;
    outParams.param1 = rand() % 255;
    outParams.param2 = (float)(rand() % 255);
    outParams.param3 = (double)(rand() % 255);
    outParams.param4 = true; // Will not be saved.
    for (int i = 0; i < 4; ++i)
    {
        outParams.list1.push_back(rand() % 255);
    }
    for (int i = 0; i < 4; ++i)
    {
        CustomParam param;
        param.param1 = rand() % 255;
        param.param2 = rand() % 255;
        outParams.list2.push_back(param);
    }
    for (int i = 0; i < 4; ++i)
```

```
        outParams.mass1[i] = (float)(rand() % 255);
    for (int i = 0; i < 4; ++i)
        outParams.mass2[i] = (int)(rand() % 255);

    // Copy params to config reader.
    ConfigReader outConfig;
    if (!outConfig.set(outParams, "Params"))
    {
        cout << "ERROR: Can't copy params" << endl;
        return -1;
    }

    // Create and write JSON file.
    if (!outConfig.writeToFile("TestConfig.json"))
    {
        cout << "ERROR: Can't write file" << endl;
        return -1;
    }

    // Read params from file.
    ConfigReader inConfig;
    if(!inConfig.readFromFile("TestConfig.json"))
    {
        cout << "ERROR: Can't read file" << endl;
        return -1;
    }

    // Read values from "Params" section.
    Params inParams;
    if(!inConfig.get(inParams, "Params"))
    {
        cout << "ERROR: Can't read params" << endl;
        return -1;
    }

    // Compare params.
    if (outParams.param1 != inParams.param1)
        cout << "ERROR: param1 has wrong value" << endl;
    if (outParams.param2 != inParams.param2)
        cout << "ERROR: param2 has wrong value" << endl;
    if (outParams.param3 != inParams.param3)
        cout << "ERROR: param3 has wrong value" << endl;
    if (outParams.param4 != inParams.param4)
        cout << "OK: param4 not included to list for reading/writing" << endl;
    for (int i = 0; i < 3; ++i)
        if (outParams.mass1[i] != inParams.mass1[i])
            cout << "ERROR: mass1[" << i << "] has wrong value" << endl;
    for (int i = 0; i < 3; ++i)
        if (outParams.mass2[i] != inParams.mass2[i])
            cout << "OK: mass2[" << i << "] not included to list for" <<
                    "reading/writing" << endl;
    if (outParams.list1.size() == inParams.list1.size())
    {
        for (int i = 0; i < inParams.list1.size(); ++i)
            if (outParams.list1[i] != inParams.list1[i])
                cout << "ERROR: list1[" << i << "] has wrong value" << endl;
```

```
    }
    else
    {
        cout << "ERROR: list1 has wrong size" << endl;
    }
    if (outParams.list2.size() == inParams.list2.size())
    {
        for (int i = 0; i < inParams.list2.size(); ++i)
            if (outParams.list2[i].param1 != inParams.list2[i].param1 ||
                outParams.list2[i].param2 != inParams.list2[i].param2)
                cout << "ERROR: list1[" << i << "] has wrong value" << endl;
    }
    else
    {
        cout << "ERROR: list2 has wrong size" << endl;
    }
}
```

Build and connect to your project

Typical commands to build **ConfigReader** library:

```
git clone https://github.com/ConstantRobotics-Ltd/ConfigReader.git
cd ConfigReader
mkdir build
cd build
cmake ..
make
```

If you want connect **ConfigReader** library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

You can add repository **ConfigReader** as submodule by command:

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/ConfigReader.git
3rdparty/ConfigReader
```

In you repository folder will be created folder **3rdparty/ConfigReader** which contains files of **ConfigReader** repository. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    ConfigReader
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

###############################################################################
## 3RD-PARTY
## dependencies for the project
###############################################################################
project(3rdparty LANGUAGES CXX)

###############################################################################
## SETTINGS
## basic 3rd-party settings before use
###############################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

###############################################################################
## CONFIGURATION
## 3rd-party submodules configuration
###############################################################################
SET(${PARENT}_SUBMODULE_CONFIG_READER                   ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_CONFIG_READER)
    SET(${PARENT}_CONFIG_READER                         ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_CONFIG_READER_EXAMPLES                OFF CACHE BOOL "" FORCE)
endif()

###############################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
###############################################################################
if (${PARENT}_SUBMODULE_CONFIG_READER)
    add_subdirectory(ConfigReader)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **ConfigReader** to your project and excludes examples (ConfigReader class usage examples) from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    ConfigReader
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include ConfigReader library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} ConfigReader)
```

Done!