jsonToBatProject

Generated on Tue Feb 27 2024 21:32:50 for jsonToBatProject by Doxygen 1.9.8

Tue Feb 27 2024 21:32:50

Chapter 1

README

Doxygen Documentation

Sonar Cloud

1.1 README

Current workflows:

- build
  - build and test the application on:
    - windows with cl
    - ubunut with g++
    - ubuntu with clang++
- buildWithPrecompiled
  - Same as build but with the precompiled libraries
- CodeQl
  - Code security
- Doxygen Action
  - Generate Doxygen documentation
  - Deploys generated documentation to gh-pages
- Microsoft C++ Code Analysis
- pages-build-deployment
- SonarCloud
  - Static code analysis *For Scanning Alerts -> Security*

Regarding coding style (?):

- no classes in global namespace
- no "using NAMESPACE"
- 4 space indenting
- ? *setup astyle options?*

Git (?):

- no direct commits onto main (only via pull-requests)
-

Libraries

- jsoncpp
- Easyloggingpp
- Catch2

Libraries can be found in ./lib. They are subprojects and will be compiled when building the project for the first time. Alternatevly compiled versions can be found at ./lib/compiled. As is, this approach works on linux (gcc, clang) and Windows (Mingw). As steps found in the tutorial (checking for compiler in cmake) are not necessary.

1.1.1 Precompiled

By setting the flag `-DPRECOMPILED=ON` when initialising the cmake project, the precompiled versions of the libraries will be used. ***This does currently not work under windows***

Chapter 2

Todo List

Global main (int argc, char ∗argv[])

Github

- "Dev-Ops"
- Doxygen settings
- Template-Comment
- Template-Header-Comment

Global utils::StartupHandler::getOptions (int argc, char ∗argv[])

Implement functionality for the options. Implement/Add more options. Shorten function and outsource functionality to other functions.

Global utils::StartupHandler::initEasyLogging ()

Improve easylogging configuration

Chapter 3

Bug List

Global utils::StartupHandler::getOptions (int argc, char ∗argv[])

Global verbose flag is not working

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 globals Namespace Reference

Variables

- static int verbose = 0

  BUG Globals dont stay persistent when set in static funciton in statuphadnler

7.1.1 Variable Documentation

7.1.1.1 verbose

```
int globals::verbose = 0  [static]
```

\BUG Globals dont stay persistent when set in static funciton in statuphadnler

Definition at line 3 of file globals.cpp.

7.2 utils Namespace Reference

Namespace for utility functions.

Data Structures

- class StartupHandler

  StartupHandler.

7.2.1 Detailed Description

Namespace for utility functions.

This namespace contains utility functions for the application. Currently, it contains the StartupHandler class.

7.3 WIP Namespace Reference

Functions

- void exampleEasyLogging ()

7.3.1 Function Documentation

7.3.1.1 exampleEasyLogging()

```
void WIP::exampleEasyLogging ( )
```

Chapter 8

Data Structure Documentation

8.1 utils::StartupHandler Class Reference

StartupHandler.

`#include <StartupHandler.hpp>`

Static Public Member Functions

- static void initEasyLogging ()
  Initialize easylogging.
- static std::optional< std::string > getOptions (int argc, char *argv[])
  Get options from command line.

Private Member Functions

- StartupHandler ()=default
  Constructor (private)
- StartupHandler (const StartupHandler &)=delete
  Copy constructor (deleted)
- StartupHandler & operator= (const StartupHandler &)=delete
  Assignment operator (deleted)

8.1.1 Detailed Description

StartupHandler.

This class provides functionality for the startup of the application. Currently it initializes easylogging and parses given options.

Note

Should stay static?

Definition at line 23 of file StartupHandler.hpp.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 StartupHandler() [1/2]

```
utils::StartupHandler::StartupHandler ( )  [private], [default]
```

Constructor (private)

This class should not be instantiated.

8.1.2.2 StartupHandler() [2/2]

```
utils::StartupHandler::StartupHandler (
            const StartupHandler &  )  [private], [delete]
```

Copy constructor (deleted)

This class should not be instantiated.

8.1.3 Member Function Documentation

8.1.3.1 getOptions()

```
std::optional< std::string > utils::StartupHandler::getOptions (
            int argc,
            char * argv[] )  [static]
```

Get options from command line.

This function parses the command line options and returns the filename given as an argument. It can hadle short, long and "regular" arguments. Currently, the following options are supported:

- -h, –help: Show help
- -V, –version: Show version
- –verbose: Set verbose flag
- –brief: Unset verbose flag
- –test: Test

Todo Implement functionality for the options. Implement/Add more options. Shorten function and outsource functionality to other functions.

Bug Global verbose flag is not working

Parameters

argc	Number of arguments
argv	Arguments

Returns

Returns either the filename or nothing.

Exceptions

std::invalid_argument	If more than one filename is given.

Definition at line 19 of file StartupHandler.cpp.

References globals::verbose.

8.1.3.2 initEasyLogging()

```
void utils::StartupHandler::initEasyLogging ( )  [static]
```

Initialize easylogging.

This function initializes easylogging with the configuration file "$SOURCE/conf/easylogging.conf".

Todo Improve easylogging configuration

Definition at line 10 of file StartupHandler.cpp.

8.1.3.3 operator=()

```
StartupHandler & utils::StartupHandler::operator= (
            const StartupHandler &  )  [private], [delete]
```

Assignment operator (deleted)

This class should not be instantiated.

The documentation for this class was generated from the following files:

- src/headers/StartupHandler.hpp
- src/sources/StartupHandler.cpp

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/headers/StartupHandler.hpp File Reference

```
#include <getopt.h>
#include <optional>
#include <string>
```

Data Structures

- class utils::StartupHandler

  StartupHandler.

Namespaces

- namespace utils

  Namespace for utility functions.

9.3 StartupHandler.hpp

Go to the documentation of this file.

```
00001 #include <getopt.h>
00002 #include <optional>
00003 #include <string>
00004
00012 namespace utils {
00023 class StartupHandler {
00024   public:
00035     static void initEasyLogging();
00036
00065     static std::optional<std::string> getOptions(int argc, char* argv[]);
00066
00067   private:
00074     StartupHandler() = default;
00075
00082     StartupHandler(const StartupHandler &) = delete;
00083
00090     StartupHandler &operator=(const StartupHandler &) = delete;
00091
00092 };
00093 } // namespace utils
```

9.4 src/main.cpp File Reference

```
#include "StartupHandler.hpp"
#include "easylogging++.h"
#include "globals.cpp"
#include <getopt.h>
#include <iostream>
#include <stdexcept>
```

Namespaces

- namespace WIP

Functions

- void WIP::exampleEasyLogging ()
- int main (int argc, char ∗argv[ ])

  Main function.

9.4.1 Function Documentation

9.4.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main function.

Codeconvention:

- Formatter: astyle

  Todo

Definition at line 28 of file main.cpp.

References utils::StartupHandler::getOptions(), and utils::StartupHandler::initEasyLogging().

9.5 main.cpp

Go to the documentation of this file.

```
00001 #include "StartupHandler.hpp"
00002 #include "easylogging++.h"
00003 #include "globals.cpp"
00004 #include <getopt.h>
00005 #include <iostream>
00006 #include <stdexcept>
00007
00008 INITIALIZE_EASYLOGGINGPP
00009
00010 namespace WIP {
00011 void exampleEasyLogging();
00012 }
00013
00028 int main(int argc, char* argv[])
00029 {
00030     utils::StartupHandler::initEasyLogging();
00031
00032     if (argc <= 1) {
00033         LOG(WARNING) « "No arguments provided, exiting!";
00034         std::cout « "No arguments provided, exiting!\n";
00035         return 1;
00036     }
00037
00038     std::cout « "Hello, World!" « std::endl;
00039     std::optional<std::string> filename;
00040
00041     try {
00042         filename = utils::StartupHandler::getOptions(argc, argv);
00043     }
00044     catch (const std::invalid_argument &e) {
00045         LOG(WARNING) « "Caught invalid argument: " « e.what();
00046         std::cout « "Invalid argument: " « e.what() « std::endl;
00047     }
00048
00049     if (!filename.has_value()) {
00050         LOG(ERROR) « "No filename given! Exiting...";
00051         std::cerr « "No filename given!\nExiting...\n";
00052         return 1;
00053     }
00054
00055     LOG(INFO) « "Filename received: " « filename.value();
00056     std::cout « "Filename: " « filename.value() « std::endl;
00057     LOG(INFO) « "Further processing...";
00058     std::cout « "Further processing..." « std::endl;
00059     LOG(INFO) « "Application exiting!";
00060     return 0;
00061 }
```

9.6 src/sources/globals.cpp File Reference

Namespaces

- namespace globals

Variables

- static int globals::verbose = 0

  \BUG Globals dont stay persistent when set in static funciton in statuphadnler

9.7 globals.cpp

Go to the documentation of this file.

```
00001 namespace globals {
00003 static int verbose = 0;
00004 }
```

9.8 src/sources/StartupHandler.cpp File Reference

```
#include "StartupHandler.hpp"
#include "easylogging++.h"
#include "globals.cpp"
#include <getopt.h>
#include <memory>
#include <optional>
#include <stdexcept>
```

Namespaces

- namespace utils

  Namespace for utility functions.

9.9 StartupHandler.cpp

Go to the documentation of this file.

```
00001 #include "StartupHandler.hpp"
00002 #include "easylogging++.h"
00003 #include "globals.cpp"
00004 #include <getopt.h>
00005 #include <memory>
00006 #include <optional>
00007 #include <stdexcept>
00008
00009 namespace utils {
00010 void StartupHandler::initEasyLogging()
00011 {
00012     el::Configurations conf("conf/easylogging.conf");
00013     el::Loggers::reconfigureLogger("default", conf);
00014     el::Loggers::reconfigureAllLoggers(conf);
00015     LOG(INFO) « "Easylogging initialized!";
00016 }
00017
00018
00019 std::optional<std::string> StartupHandler::getOptions(int argc, char* argv[])
00020 {
00021     LOG(INFO) « "Parsing options...";
00022     static const struct option long_options[] = {
00023         /* These options set a flag. */
00024         {"verbose", no_argument, &globals::verbose, 1},
00025         {"brief", no_argument, &globals::verbose, 0},
00026         {"help", no_argument, 0, 'h'},
00027         {"version", no_argument, 0, 'V'},
00028         {"test", required_argument, 0, 0},
00029         0
00030     };
00031
00032     do {
00033         int optIndex = -1;
00034         std::unique_ptr<struct option> opt = 0;
00035         auto result = getopt_long(argc, argv, "hV", long_options, &optIndex);
00036
00037         if (result == -1) {
00038             break;
00039         }
00040
00041         switch (result) {
00042             case '?':
00043                 LOG(INFO) « "Unknown option given";
00044                 std::cout « "Not know\n";
00045                 break;
00046
00047             case 'h':
00048                 LOG(INFO) « "Help option given";
00049                 std::cout « "long h\n";
00050                 break;
00051
00052             case 'V':
```

```
00053                 LOG(INFO) « "Version option given";
00054                 std::cout « "long V\n";
00055
00056             case '0':
00057                 opt = std::make_unique<struct option>(long_options[optIndex]);
00058                 LOG(INFO) « "Option " « opt->name « " given";
00059
00060                 if (opt->has_arg == required_argument) {
00061                     LOG(INFO) « "Argument: " « optarg;
00062                 }
00063
00064                 break;
00065         }
00066     } while (1);
00067
00068     LOG(INFO) « "Parsing options done";
00069     std::optional<std::string> filename = {};
00070     LOG(INFO) « "Parsing other arguments...";
00071
00072     while (optind < argc) {
00073         if (filename.has_value()) {
00074             LOG(ERROR) « "Only one filename can be given!";
00075             throw std::invalid_argument("Only one filename can be given!\n");
00076         }
00077
00078         LOG(INFO) « "Filename set to: " « argv[optind];
00079         filename = std::string(argv[optind++]);
00080     }
00081
00082     return filename;
00083 }
00084 } // namespace utils
```

Index