jsonToBatProject

0.2.0

Generated on Wed Feb 28 2024 14:48:14 for jsonToBatProject by Doxygen 1.9.8

Wed Feb 28 2024 14:48:14

Chapter 1

Todo List

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

Implement functionality for the options.

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

Member utils::StartupHandler::initEasyLogging ()

Improve easylogging configuration

Chapter 2

Bug List

Member main (int argc, char ∗argv[])

Getopt is not working on Windows.

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

Global verbose flag is not working.

Member utils::StartupHandler::initEasyLogging ()

Easylogging conf only recognized when running application from source dir

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 utils Namespace Reference

Namespace for utility functions.

Classes

- class StartupHandler

  Handles startup task for the application.

Variables

- static int verbose = 0

6.1.1 Detailed Description

Namespace for utility functions.

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

6.1.2 Variable Documentation

6.1.2.1 verbose

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

Definition at line 11 of file StartupHandler.cpp.

Chapter 7

Class Documentation

7.1 utils::StartupHandler Class Reference

Handles startup task for the application.

`#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)

7.1.1 Detailed Description

Handles startup task for the application.

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

Note

I think this class should stay static - Simon

Definition at line 23 of file StartupHandler.hpp.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 StartupHandler() [1/2]

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

Constructor (private)

This class should not be instantiated.

7.1.2.2 StartupHandler() [2/2]

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

Copy constructor (deleted)

This class should not be instantiated.

7.1.3 Member Function Documentation

7.1.3.1 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

  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 21 of file StartupHandler.cpp.

References utils::verbose.

Here is the caller graph for this function:

7.1.3.2 initEasyLogging()

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

Initialize easylogging.

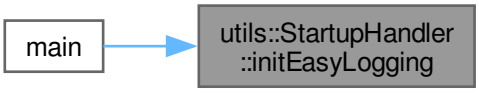
This function initializes easylogging with the configuration file "$SOURCE/conf/easylogging.conf".

Todo • Improve easylogging configuration

Bug Easylogging conf only recognized when running application from source dir

Definition at line 13 of file StartupHandler.cpp.

Here is the caller graph for this function:



7.1.3.3 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

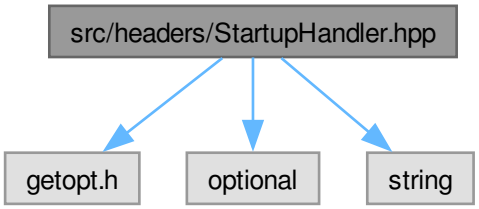
Chapter 8

File Documentation

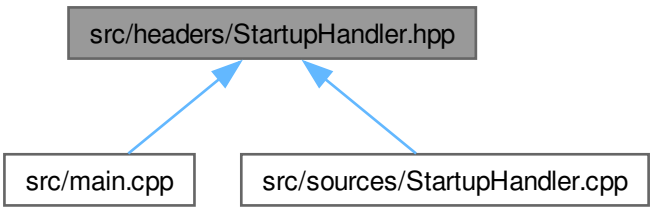
8.1 src/headers/StartupHandler.hpp File Reference

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

Include dependency graph for StartupHandler.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class utils::StartupHandler

  Handles startup task for the application.

Namespaces

- namespace utils

  Namespace for utility functions.

8.2 StartupHandler.hpp

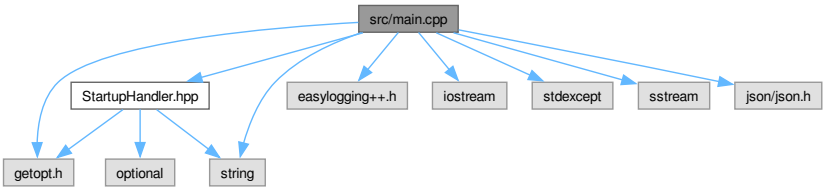
Go to the documentation of this file.

```
00001 #include <getopt.h>
00002 #include <optional>
00003 #include <string>
00004
00012 namespace utils {
00023 class StartupHandler {
00024   public:
00039     static void initEasyLogging();
00040
00069     static std::optional<std::string> getOptions(int argc, char* argv[]);
00070
00071   private:
00078     StartupHandler() = default;
00079
00086     StartupHandler(const StartupHandler &) = delete;
00087
00094     StartupHandler &operator=(const StartupHandler &) = delete;
00095
00096 };
00097 } // namespace utils
```

8.3 src/main.cpp File Reference

```
#include "StartupHandler.hpp"
#include "easylogging++.h"
#include <getopt.h>
#include <iostream>
#include <stdexcept>
#include <string>
#include <sstream>
#include <json/json.h>
```

Include dependency graph for main.cpp:



Functions

- INITIALIZE_EASYLOGGINGPP int main (int argc, char ∗argv[ ])

  Main function.

8.3.1 Function Documentation

8.3.1.1 main()

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

Main function.

This is the main function for the application, The application is designed to parse a json file and create a batch file from it. Further more it provides a CLI to help the user to interact with the application.

Bug Getopt is not working on Windows.

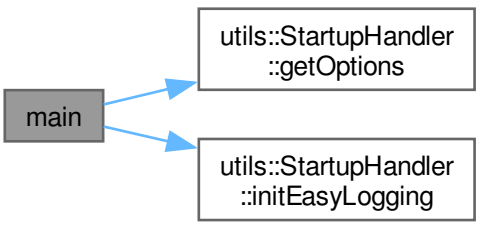
Note

json parsing seems simple edgecases? basically just treat as array/map

Definition at line 25 of file main.cpp.

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

Here is the call graph for this function:



8.4 main.cpp

Go to the documentation of this file.

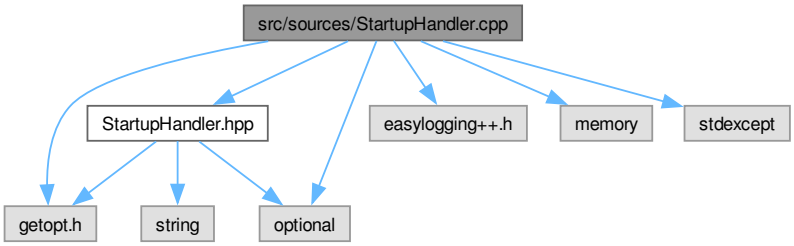
```
00001 #include "StartupHandler.hpp"
00002 #include "easylogging++.h"
00003
00004 #include <getopt.h>
00005 #include <iostream>
00006 #include <stdexcept>
00007 #include <string>
00008 #include <sstream>
00009 #include <json/json.h>
00010
00011 INITIALIZE_EASYLOGGINGPP
00012
00025 int main(int argc, char* argv[])
00026 {
00027     std::cout << "Starting Application..." << std::endl;
00028     utils::StartupHandler::initEasyLogging();
00029
00030     if (argc <= 1) {
00031         LOG(WARNING) << "No arguments provided, exiting!";
00032         std::cout << "No arguments provided, exiting!\n";
00033         return 1;
00034     }
00035
00036     std::optional<std::string> filename;
00037
00038     try {
00039         filename = utils::StartupHandler::getOptions(argc, argv);
00040     }
00041     catch (const std::invalid_argument &e) {
00042         LOG(WARNING) << "Caught invalid argument: " << e.what();
00043         std::cout << "Invalid argument: " << e.what() << std::endl;
00044     }
00045
00046     if (!filename.has_value()) {
00047         LOG(ERROR) << "No filename given! Exiting...";
00048         std::cerr << "No filename given!\nExiting...\n";
00049         return 1;
00050     }
00051
00052     LOG(INFO) << "Filename received: " << filename.value();
00053     std::cout << "Filename: " << filename.value() << std::endl;
00054     LOG(INFO) << "Further processing...";
00055     std::cout << "Further processing..." << std::endl;
00061     Json::Value root;
00062     std::ifstream file(filename.value());
00063     Json::Reader reader;
00064     reader.parse(file, root);
00065     auto memberNames = root.getMemberNames();
00066     std::cout << "Memebers: " << std::endl;
00067
00068     for (auto name : memberNames) {
00069         std::cout << "    \"" << name << "\" :" << "\n";
00070
00071         switch (root[name].type()) {
00072             case Json::ValueType::arrayValue:
00073                 std::cout << "          Type: array\n";
00074                 break;
00075
00076             case Json::ValueType::booleanValue:
00077                 std::cout << "          Type: boolean\n";
00078                 break;
00079
00080             case Json::ValueType::intValue:
00081                 std::cout << "          Type: int\n";
00082                 break;
00083
00084             case Json::ValueType::realValue:
00085                 std::cout << "          Type: real\n";
00086                 break;
00087
00088             case Json::ValueType::stringValue:
00089                 std::cout << "          Type: string\n";
00090                 break;
00091
00092             case Json::ValueType::uintValue:
00093                 std::cout << "          Type: uint\n";
00094                 break;
00095
00096             case Json::ValueType::nullValue:
00097                 std::cout << "          Type: null\n";
00098                 break;
00099
```

```
00100             default:
00101                 std::cout « "          Type: unknown\n";
00102                 break;
00103         }
00104     }
00105
00106     // Not error proof
00107     std::cout « "Outputfile: " « root["outputfile"].asString() « "\n";
00108     std::string outputfile = "output/" + root["outputfile"].asString();
00109     std::fstream batchFile;
00110     batchFile.open(outputfile, std::ios::out);
00111     batchFile « "#This is a test\n";
00112     // Very not error proof
00113     std::stringstream additionalPath;
00114     int counter = 0;
00115     std::cout « "Entries:\n";
00116
00117     batchFile « "@ECHO OFF\nC:\\Windows\\System32\\cmd.exe /k\n\"";
00118
00119     for (const auto entry : root["entries"]) {
00120         std::cout « "Entry " « counter « ":\n";
00121
00122         for (const auto key : entry.getMemberNames()) {
00123             std::cout « "    " « key « ": " « entry[key].asString() « "\n";
00124         }
00125
00126         if (entry["type"].asString() == "EXE") {
00127             batchFile « entry["command"].asString() « "&&\\\n";
00128         }
00129         else if (entry["type"].asString() == "ENV") {
00130             batchFile « "set " « entry["key"].asString() « "=" « entry["value"].asString() « "&&\\\n";
00131         }
00132         else if (entry["type"].asString() == "PATH") {
00133             additionalPath « entry["path"].asString() « ";\\\n";
00134         }
00135         else {
00136             batchFile « "\nCommand doesnt exist yet\n";
00137         }
00138
00139         ++counter;
00140     }
00141
00142     if (additionalPath.str() != "") {
00143         batchFile « "set path=%path%" « additionalPath.str();
00144     }
00145
00146     batchFile « "\"\n@ECHO ON";
00147     batchFile.close();
00148     LOG(INFO) « "Application exiting!";
00149     return 0;
00150 }
```

8.5 src/sources/StartupHandler.cpp File Reference

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

Include dependency graph for StartupHandler.cpp:



Namespaces

- namespace utils

  Namespace for utility functions.

Variables

- static int utils::verbose = 0

8.6 StartupHandler.cpp

Go to the documentation of this file.

```
00001 #include "StartupHandler.hpp"
00002 #include "easylogging++.h"
00003
00004 #include <getopt.h>
00005 #include <memory>
00006 #include <optional>
00007 #include <stdexcept>
00008
00009 namespace utils {
00010
00011 static int verbose = 0;
00012
00013 void StartupHandler::initEasyLogging()
00014 {
00015     el::Configurations conf("conf/easylogging.conf");
00016     el::Loggers::reconfigureLogger("default", conf);
00017     el::Loggers::reconfigureAllLoggers(conf);
00018     LOG(INFO) « "Easylogging initialized!";
00019 }
00020
00021 std::optional<std::string> StartupHandler::getOptions(int argc, char* argv[])
00022 {
00023     LOG(INFO) « "Parsing options...";
00024     static const struct option long_options[] = {
00025         /* These options set a flag. */
00026         {"verbose", no_argument, &verbose, 1},
00027         {"brief", no_argument, &verbose, 0},
00028         {"help", no_argument, nullptr, 'h'},
00029         {"version", no_argument, nullptr, 'V'},
00030         {"test", required_argument, nullptr, 0},
00031         nullptr
00032     };
00033
00034     do {
00035         int optIndex = -1;
00036         std::unique_ptr<struct option> opt = nullptr;
00037         auto result = getopt_long(argc, argv, "hV", long_options, &optIndex);
00038
```

```
00039         if (result == -1) {
00040             break;
00041         }
00042 
00043         switch (result) {
00044             case '?':
00045                 LOG(INFO) << "Unknown option given";
00046                 std::cout << "Not know\n";
00047                 break;
00048 
00049             case 'h':
00050                 LOG(INFO) << "Help option given";
00051                 std::cout << "long h\n";
00052                 break;
00053 
00054             case 'V':
00055                 LOG(INFO) << "Version option given";
00056                 std::cout << "long V\n";
00057 
00058             case '0':
00059                 opt = std::make_unique<struct option>(long_options[optIndex]);
00060                 LOG(INFO) << "Option " << opt->name << " given";
00061 
00062                 if (opt->has_arg == required_argument) {
00063                     LOG(INFO) << "Argument: " << optarg;
00064                 }
00065 
00066                 break;
00067 
00068             default:
00069                 std::cout << "I shouldnt have been here!\n";
00070                 break;
00071         }
00072     } while (true);
00073 
00074     LOG(INFO) << "Parsing options done";
00075     std::optional<std::string> filename = {};
00076     LOG(INFO) << "Parsing other arguments...";
00077 
00078     while (optind < argc) {
00079         if (filename.has_value()) {
00080             LOG(ERROR) << "Only one filename can be given!";
00081             throw std::invalid_argument("Only one filename can be given!\n");
00082         }
00083 
00084         LOG(INFO) << "Filename set to: " << argv[optind];
00085         filename = std::string(argv[optind++]);
00086     }
00087 
00088     return filename;
00089 }
00090 } // namespace utils
```

Index