jsonToBatProject

0.2.0

Generated on Wed Feb 28 2024 19:16:22 for jsonToBatProject by Doxygen 1.9.8

Wed Feb 28 2024 19:16:22

Chapter 1

Bug List

Namespace json

Name to similiar to "Json" namespace from the json library.

Member main (int argc, char ∗argv[])

Initielizes to early for config file to be loaded

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 2

Todo List

Member json::JSONHandler::assignHideShell ()

: Error handling if not found

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 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 json Namespace Reference

json namespace

Classes

- class JSONData
- class JSONHandler

  JSONHandler class.

6.1.1 Detailed Description

json namespace

The json namespace contains all classes and functions related to the json parsing and handling.

Bug Name to similiar to "Json" namespace from the json library.

6.2 utils Namespace Reference

Namespace for utility functions.

Classes

- class StartupHandler

  Handles startup task for the application.

Variables

- static int verbose = 0

6.2.1 Detailed Description

Namespace for utility functions.

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

6.2.2 Variable Documentation

6.2.2.1 verbose

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

Definition at line 11 of file StartupHandler.cpp.

Chapter 7

Class Documentation

7.1 json::JSONData Class Reference

`#include <JSONData.hpp>`

Public Member Functions

- void setOutputFile (std::string &outputfile)

  Set's the output file.
- void setHideShell (bool hideShell)

  Set's the hide shell flag.
- void setApplication (const std::string &application)

  Set's the application.
- void addCommand (const std::string &command)

  Add a command to the commands vector.
- void addEnvironmentVariable (const std::string &name, const std::string &value)

  Add an environment variable to the environmentVariables vector.
- void addPathValue (const std::string &pathValue)

  Add a path value to the pathValues vector.
- const std::string & getOutputFile () const

  Get the output file.
- bool getHideShell () const

  Get the hide shell flag.
- const std::string & getApplication () const

  Get the application.
- const std::vector< std::string > & getCommands () const

  Get the commands.
- const std::vector< std::tuple< std::string, std::string > > & getEnvironmentVariables () const

  Get the environment variables.
- const std::vector< std::string > & getPathValues () const

  Get the path values.

Private Attributes

- std::string outputfile
- bool hideShell
- std::optional< std::string > application
- std::vector< std::string > commands
- std::vector< std::tuple< std::string, std::string > > environmentVariables
- std::vector< std::string > pathValues

Static Private Attributes

- static const int8_t suffixLength = 4

7.1.1 Detailed Description

Definition at line 9 of file JSONData.hpp.

7.1.2 Member Function Documentation

7.1.2.1 addCommand()

```
void json::JSONData::addCommand (
            const std::string & command )
```

Add a command to the commands vector.

Parameters

command	The command

Exceptions

std::invalid_argument	if the command is empty

Definition at line 36 of file JSONData.cpp.

References commands.

7.1.2.2 addEnvironmentVariable()

```
void json::JSONData::addEnvironmentVariable (
            const std::string & name,
            const std::string & value )
```

Add an environment variable to the environmentVariables vector.

The environment variable is added as a tuple with the name and value as it's elements.

Parameters

name	The name of the environment variable
value	The value of the environment variable

Exceptions

std::invalid_argument	if the name or the value is empty

Definition at line 46 of file JSONData.cpp.

References environmentVariables.

7.1.2.3 addPathValue()

```
void json::JSONData::addPathValue (
            const std::string & pathValue )
```

Add a path value to the pathValues vector.

Parameters

pathValue	The path value

Exceptions

std::invalid_argument	if the pathValue is empty

Definition at line 58 of file JSONData.cpp.

References pathValues.

7.1.2.4 getApplication()

```
const std::string & json::JSONData::getApplication ( ) const  [inline]
```

Get the application.

Returns

The application

Definition at line 89 of file JSONData.hpp.

References application.

7.1.2.5 getCommands()

```
const std::vector< std::string > & json::JSONData::getCommands ( ) const  [inline]
```

Get the commands.

Returns

The commands

Definition at line 97 of file JSONData.hpp.

References commands.

7.1.2.6 getEnvironmentVariables()

```
const std::vector< std::tuple< std::string, std::string > > & json::JSONData::getEnvironment↩
Variables ( ) const  [inline]
```

Get the environment variables.

Returns

The environment variables

Definition at line 106 of file JSONData.hpp.

References environmentVariables.

7.1.2.7 getHideShell()

```
bool json::JSONData::getHideShell ( ) const  [inline]
```

Get the hide shell flag.

Returns

The hide shell flag

Definition at line 81 of file JSONData.hpp.

References hideShell.

7.1.2.8 getOutputFile()

```
const std::string & json::JSONData::getOutputFile ( ) const  [inline]
```

Get the output file.

Returns

The output file

Definition at line 73 of file JSONData.hpp.

References outputfile.

7.1.2.9 getPathValues()

```
const std::vector< std::string > & json::JSONData::getPathValues ( ) const  [inline]
```

Get the path values.

Returns

The path values

Definition at line 114 of file JSONData.hpp.

References pathValues.

7.1.2.10 setApplication()

```
void json::JSONData::setApplication (
            const std::string & application )
```

Set's the application.

Parameters

application	The application

Definition at line 31 of file JSONData.cpp.

References application.

7.1.2.11 setHideShell()

```
void json::JSONData::setHideShell (
            bool hideShell )  [inline]
```

Set's the hide shell flag.

Parameters

hideShell	The hide shell flag

Definition at line 28 of file JSONData.hpp.

References hideShell.

7.1.2.12 setOutputFile()

```
void json::JSONData::setOutputFile (
            std::string & outputfile )
```

Set's the output file.

Note

If the output file does not end with .bat, the function will append .bat to the output file.

Parameters

outputfile	The output file

Exceptions

std::invalid_argument	if the outputfile is empty
std::invalid_argument	if the outputfile is already set

Definition at line 7 of file JSONData.cpp.

References outputfile.

7.1.3 Member Data Documentation

7.1.3.1 application

```
std::optional<std::string> json::JSONData::application  [private]
```

Definition at line 121 of file JSONData.hpp.

7.1.3.2 commands

```
std::vector<std::string> json::JSONData::commands  [private]
```

Definition at line 122 of file JSONData.hpp.

7.1.3.3 environmentVariables

```
std::vector<std::tuple<std::string, std::string> > json::JSONData::environmentVariables  [private]
```

Definition at line 123 of file JSONData.hpp.

7.1.3.4 hideShell

```
bool json::JSONData::hideShell  [private]
```

Definition at line 120 of file JSONData.hpp.

7.1.3.5 outputfile

```
std::string json::JSONData::outputfile  [private]
```

Definition at line 119 of file JSONData.hpp.

7.1.3.6 pathValues

```
std::vector<std::string> json::JSONData::pathValues  [private]
```

Definition at line 124 of file JSONData.hpp.

7.1.3.7 suffixLength

```
const int8_t json::JSONData::suffixLength = 4  [static], [private]
```

Definition at line 125 of file JSONData.hpp.

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

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

7.2 json::JSONHandler Class Reference

JSONHandler class.

```
#include <JSONHandler.hpp>
```

Public Member Functions

- JSONHandler (const std::string &filename)
  Constructor.
- std::shared_ptr< JSONData > getJSONData ()
  Retrieve the JSONData object.

Private Member Functions

- std::shared_ptr< Json::Value > parseFile (const std::string &filename)
  Parse a file.
- void assignOutputFile ()
  Assigns the output file to the JSONData object.
- void assignHideShell ()
  Assigns the hide shell value to the JSONData object.
- void assignApplication ()
  Assigns the application to the JSONData object.
- void assignEntries ()
  Assigns the entries to the JSONData object.
- void assignCommand (const Json::Value &entry)
  Assigns a command to the JSONData object.
- void assignEnvironmentVariable (const Json::Value &entry)
  Assigns an environment variable to the JSONData object.
- void assignPathValue (const Json::Value &entry)
  Assigns a path value to the JSONData object.
- std::shared_ptr< JSONData > createJSONData ()
  Creates a JSONData object.

Private Attributes

- std::shared_ptr< Json::Value > root
- std::shared_ptr< JSONData > data

7.2.1 Detailed Description

JSONHandler class.

The JSONHandler class is responsible for parsing a json file and creating a JSONData object from it when requested. It assigns all necessary values to the JSONData object. Most of the error handling is done in the JSONData object.

Definition at line 29 of file JSONHandler.hpp.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 JSONHandler()

```
json::JSONHandler::JSONHandler (
            const std::string & filename )
```

Constructor.

The constructor calls the parseFile function to parse the file and adds it to the corresponding member variable.

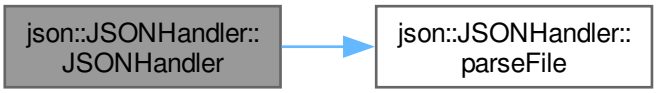
Parameters

filename	The filename to parse

Definition at line 8 of file JSONHandler.cpp.

References parseFile(), and root.

Here is the call graph for this function:



7.2.3 Member Function Documentation

7.2.3.1 assignApplication()

```
void json::JSONHandler::assignApplication ( )  [private]
```

Assigns the application to the JSONData object.

Note

How should error handling be done? Value can be empty, but what about null vs ""?

Definition at line 52 of file JSONHandler.cpp.

References data, and root.

Here is the caller graph for this function:

7.2.3.2 assignCommand()

```
void json::JSONHandler::assignCommand (
            const Json::Value & entry )  [private]
```

Assigns a command to the JSONData object.

The function takes a Json::Value object and assigns the command to the JSONData object

Parameters

entry	The entry to assign

Note

Error handling is done in the JSONData object

Definition at line 79 of file JSONHandler.cpp.

References data.

Here is the caller graph for this function:

7.2.3.3 assignEntries()

```
void json::JSONHandler::assignEntries ( )  [private]
```

Assigns the entries to the JSONData object.

The function loops through the entries and calls the corresponding function to assign the entry to the JSONData object

Exceptions

std::runtime_error	If the entry type is unknown

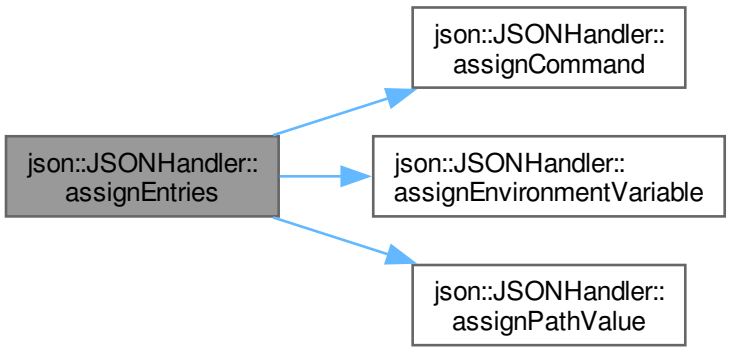
Note

Other error handling is done in the JSONData object

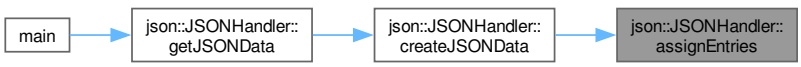
Definition at line 58 of file JSONHandler.cpp.

References assignCommand(), assignEnvironmentVariable(), assignPathValue(), and root.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.4 assignEnvironmentVariable()

```
void json::JSONHandler::assignEnvironmentVariable (
            const Json::Value & entry )  [private]
```

Assigns an environment variable to the JSONData object.

The function takes a Json::Value object and assigns a tuple of the environment variable to the JSONData object

Parameters

entry	The entry to assign

Note

Error handling is done in the JSONData object

Definition at line 85 of file JSONHandler.cpp.

References data.

Here is the caller graph for this function:

7.2.3.5 assignHideShell()

```
void json::JSONHandler::assignHideShell ( )  [private]
```

Assigns the hide shell value to the JSONData object.

Note

There is no real error handling for this value, it defaults to false

Todo : Error handling if not found

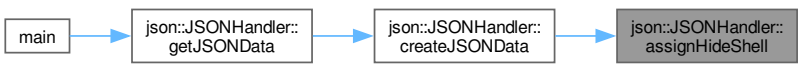
Note

: default to false

Definition at line 44 of file JSONHandler.cpp.

References data, and root.

Here is the caller graph for this function:



7.2.3.6 assignOutputFile()

```
void json::JSONHandler::assignOutputFile ( )  [private]
```

Assigns the output file to the JSONData object.

Note

Error handling is done in the JSONData object

Definition at line 38 of file JSONHandler.cpp.

References data, and root.

Here is the caller graph for this function:

7.2.3.7 assignPathValue()

```
void json::JSONHandler::assignPathValue (
            const Json::Value & entry )  [private]
```

Assigns a path value to the JSONData object.

The function takes a Json::Value object and assigns the path value to the JSONData object

Parameters

entry	The entry to assign

Note

Error handling is done in the JSONData object

Definition at line 92 of file JSONHandler.cpp.

References data.

Here is the caller graph for this function:

7.2.3.8 createJSONData()

```
std::shared_ptr< JSONData > json::JSONHandler::createJSONData ( )  [private]
```

Creates a JSONData object.

The function creates the JSONData object and calls all the necessary methods to assign the values to the object.

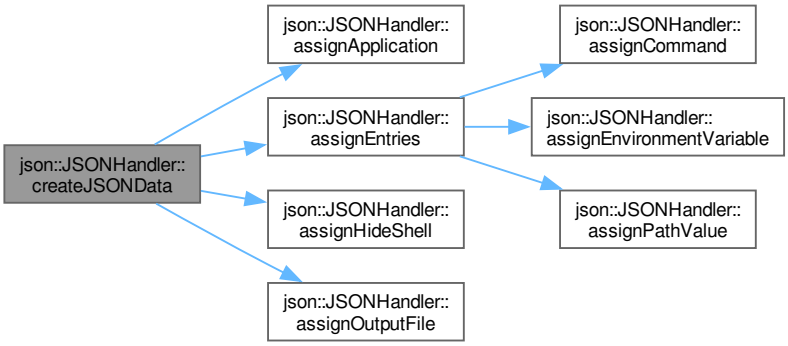
Returns

std::shared_ptr<JSONData> The JSONData object

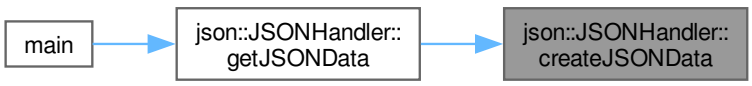
Definition at line 28 of file JSONHandler.cpp.

References assignApplication(), assignEntries(), assignHideShell(), assignOutputFile(), and data.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.9 getJSONData()

```
std::shared_ptr< JSONData > json::JSONHandler::getJSONData ( )
```

Retrieve the JSONData object.

The function takes the necesarry steps to create a JSONData object and then returns it

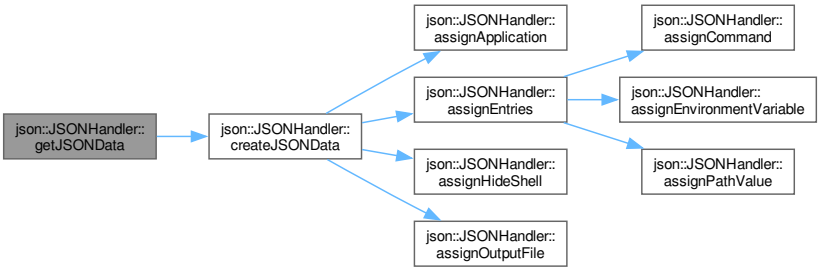
Returns

std::shared_ptr<JSONData> The JSONData object

Definition at line 23 of file JSONHandler.cpp.

References createJSONData().

Here is the call graph for this function:



Here is the caller graph for this function:

7.2.3.10 parseFile()

```
std::shared_ptr< Json::Value > json::JSONHandler::parseFile (
            const std::string & filename )  [private]
```

Parse a file.

The function takes a filename and parses the file into a Json::Value object.

Parameters

filename	The filename to parse

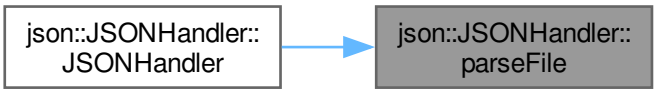
Returns

std::shared_ptr<Json::Value> The parsed file

Definition at line 13 of file JSONHandler.cpp.

References root.

Here is the caller graph for this function:



7.2.4 Member Data Documentation

7.2.4.1 data

```
std::shared_ptr<JSONData> json::JSONHandler::data  [private]
```

Definition at line 158 of file JSONHandler.hpp.

7.2.4.2 root

```
std::shared_ptr<Json::Value> json::JSONHandler::root  [private]
```

Definition at line 157 of file JSONHandler.hpp.

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

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

7.3 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)

7.3.1 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 26 of file StartupHandler.hpp.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 StartupHandler() [1/2]

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

Constructor (private)

This class should not be instantiated.

7.3.2.2 StartupHandler() [2/2]

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

Copy constructor (deleted)

This class should not be instantiated.

7.3.3 Member Function Documentation

7.3.3.1 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:

7.3.3.2 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:

7.3.3.3 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

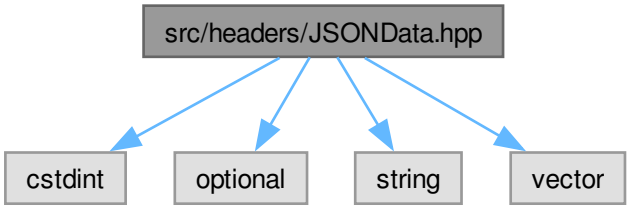
Chapter 8

File Documentation

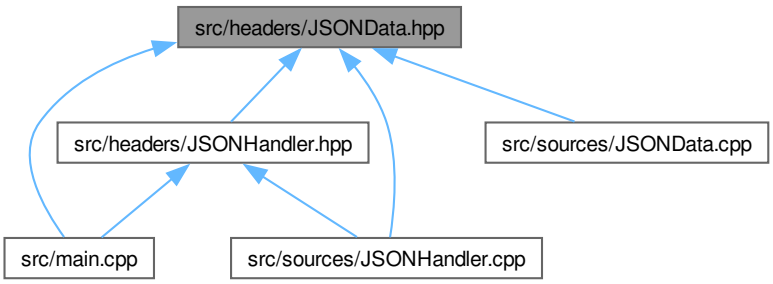
8.1 src/headers/JSONData.hpp File Reference

```
#include <cstdint>
#include <optional>
#include <string>
#include <vector>
```

Include dependency graph for JSONData.hpp:



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



Classes

- class json::JSONData

Namespaces

- namespace json

  json namespace

8.2 JSONData.hpp

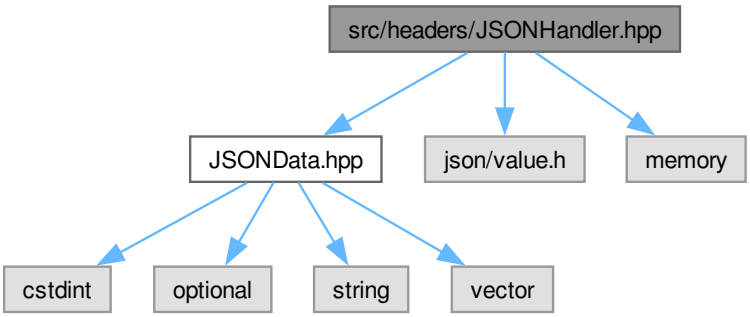
Go to the documentation of this file.

```
00001 #ifndef JSONDATA_HPP
00002 #define JSONDATA_HPP
00003
00004 #include <cstdint>
00005 #include <optional>
00006 #include <string>
00007 #include <vector>
00008 namespace json {
00009 class JSONData {
00010   public:
00022     void setOutputFile(std::string &outputfile);
00023
00028     void setHideShell(bool hideShell) {
00029         this->hideShell = hideShell;
00030     }
00031
00036     void setApplication(const std::string &application);
00037
00044     void addCommand(const std::string &command);
00045
00058     void addEnvironmentVariable(const std::string &name,
00059                                 const std::string &value);
00060
00067     void addPathValue(const std::string &pathValue);
00068
00073     const std::string &getOutputFile() const {
00074         return outputfile;
00075     }
00076
00081     bool getHideShell() const {
00082         return hideShell;
00083     }
00084
00089     const std::string &getApplication() const {
00090         return application.value();
00091     }
00092
00097     const std::vector<std::string> &getCommands() const {
00098         return commands;
00099     }
00100
00105     const std::vector<std::tuple<std::string, std::string»
00106     &getEnvironmentVariables() const {
00107         return environmentVariables;
00108     }
00109
00114     const std::vector<std::string> &getPathValues() const {
00115         return pathValues;
00116     }
00117
00118   private:
00119     std::string outputfile;
00120     bool hideShell;
00121     std::optional<std::string> application;
00122     std::vector<std::string> commands;
00123     std::vector<std::tuple<std::string, std::string» environmentVariables;
00124     std::vector<std::string> pathValues;
00125     const static int8_t suffixLength = 4;
00126 };
00127 } // namespace json
00128
00129 #endif // JSONDATA_HPP
```

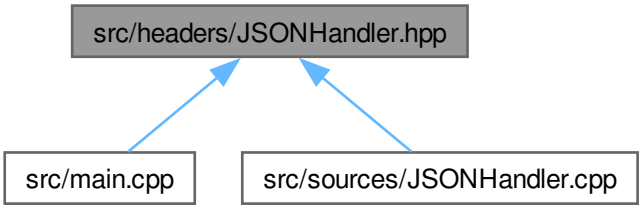
8.3 src/headers/JSONHandler.hpp File Reference

```
#include "JSONData.hpp"
#include "json/value.h"
#include <memory>
```

Include dependency graph for JSONHandler.hpp:



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



Classes

- class json::JSONHandler

  JSONHandler class.

Namespaces

- namespace json

  json namespace

8.4 JSONHandler.hpp

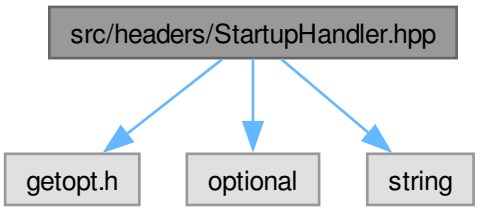
Go to the documentation of this file.

```
00001 #ifndef JSONHANDLER_HPP
00002 #define JSONHANDLER_HPP
00003
00004 #include "JSONData.hpp"
00005 #include "json/value.h"
00006 #include <memory>
00007
00017 namespace json {
00018
00029 class JSONHandler {
00030   public:
00040     JSONHandler(const std::string &filename);
00041
00051     std::shared_ptr<JSONData> getJSONData();
00052
00053   private:
00064     std::shared_ptr<Json::Value> parseFile(const std::string &filename);
00065
00072     void assignOutputFile();
00073
00080     void assignHideShell();
00081
00089     void assignApplication();
00090
00102     void assignEntries();
00103
00116     void assignCommand(const Json::Value &entry);
00117
00130     void assignEnvironmentVariable(const Json::Value &entry);
00131
00144     void assignPathValue(const Json::Value &entry);
00145
00155     std::shared_ptr<JSONData> createJSONData();
00156
00157     std::shared_ptr<Json::Value> root;
00158     std::shared_ptr<JSONData> data;
00159 };
00160 } // namespace json
00161
00162 #endif // JSONHANDLER_HPP
```

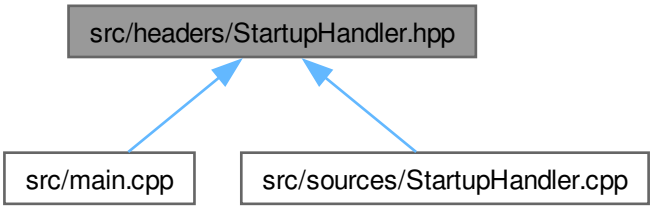
8.5 src/headers/StartupHandler.hpp File Reference

```
#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.

8.6 StartupHandler.hpp

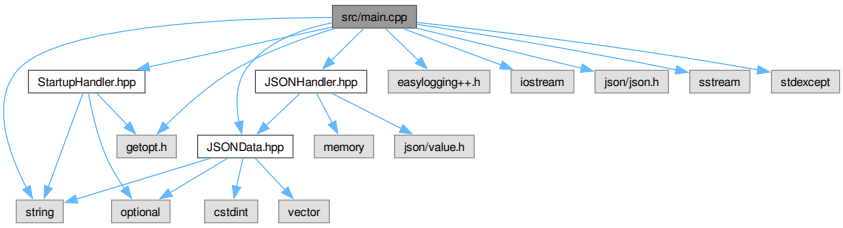
Go to the documentation of this file.

```
00001 #ifndef STARTUPHANDLER_HPP
00002 #define STARTUPHANDLER_HPP
00003
00004 #include <getopt.h>
00005 #include <optional>
00006 #include <string>
00007
00015 namespace utils {
00026 class StartupHandler {
00027   public:
00042     static void initEasyLogging();
00043
00072     static std::optional<std::string> getOptions(int argc, char* argv[]);
00073
00074   private:
00081     StartupHandler() = default;
00082
00089     StartupHandler(const StartupHandler &) = delete;
00090
00097     StartupHandler &operator=(const StartupHandler &) = delete;
00098
00099 };
00100 } // namespace utils
00101 #endif // STARTUPHANDLER_HPP
```

8.7 src/main.cpp File Reference

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

Include dependency graph for main.cpp:



Functions

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

  Main function.

8.7.1 Function Documentation

8.7.1.1 main()

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

Main function.

Bug Initielizes to early for config file to be loaded

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.

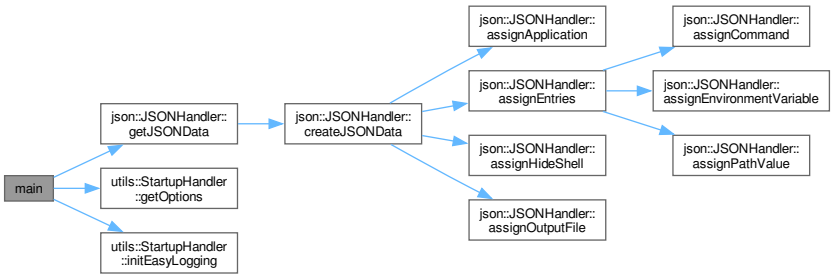
Note

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

Definition at line 28 of file main.cpp.

References json::JSONHandler::getJSONData(), utils::StartupHandler::getOptions(), and utils::StartupHandler::initEasyLogging().

Here is the call graph for this function:



8.8 main.cpp

Go to the documentation of this file.

```
00001 #include "JSONData.hpp"
00002 #include "StartupHandler.hpp"
00003 #include "JSONHandler.hpp"
00004
00005 #include <easylogging++.h>
00006 #include <getopt.h>
00007 #include <iostream>
00008 #include <json/json.h>
00009 #include <sstream>
00010 #include <stdexcept>
00011 #include <string>
00012
00014 INITIALIZE_EASYLOGGINGPP
00015
00028 int main(int argc, char* argv[])
00029 {
00030     std::cout << "Starting Application..." << std::endl;
00031     utils::StartupHandler::initEasyLogging();
00032
00033     if (argc <= 1) {
00034         LOG(WARNING) << "No arguments provided, exiting!";
00035         std::cout << "No arguments provided, exiting!\n";
00036         return 1;
00037     }
00038
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...";
```

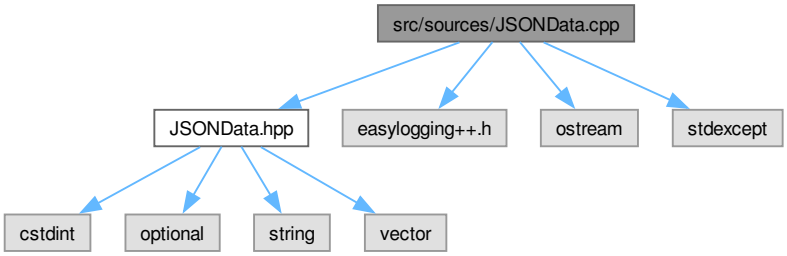
```
00058     std::cout « "Further processing..." « std::endl;
00064     // Initialize the JSONHandler with the file(name)
00065     json::JSONHandler jsonHandler(filename.value());
00066     // Get a JSONData object from the JSONHandler
00067     std::shared_ptr<json::JSONData> jsonData = jsonHandler.getJSONData();
00068     // Print the outputfile as a test
00069     std::cout « "Outputfile: " « jsonData->getOutputFile() « std::endl;
00070     // \note Code below only for me to see how I did it this morning
00071     /*
00072     for (auto name : memberNames) {
00073       std::cout « "    \"" « name « "\" :"
00074                 « "\n";
00075
00076       switch (root[name].type()) {
00077       case Json::ValueType::arrayValue:
00078         std::cout « "          Type: array\n";
00079         break;
00080
00081       case Json::ValueType::booleanValue:
00082         std::cout « "          Type: boolean\n";
00083         break;
00084
00085       case Json::ValueType::intValue:
00086         std::cout « "          Type: int\n";
00087         break;
00088
00089       case Json::ValueType::realValue:
00090         std::cout « "          Type: real\n";
00091         break;
00092
00093       case Json::ValueType::stringValue:
00094         std::cout « "          Type: string\n";
00095         break;
00096
00097       case Json::ValueType::uintValue:
00098         std::cout « "          Type: uint\n";
00099         break;
00100
00101       case Json::ValueType::nullValue:
00102         std::cout « "          Type: null\n";
00103         break;
00104
00105       default:
00106         std::cout « "          Type: unknown\n";
00107         break;
00108       }
00109     }
00110
00111     // Not error proof
00112     std::cout « "Outputfile: " « root["outputfile"].asString() « "\n";
00113     std::string outputfile = "output/" + root["outputfile"].asString();
00114     std::fstream batchFile;
00115     batchFile.open(outputfile, std::ios::out);
00116     batchFile « "#This is a test\n";
00117     // Very not error proof
00118     std::stringstream additionalPath;
00119     int counter = 0;
00120     std::cout « "Entries:\n";
00121
00122     batchFile « "@ECHO OFF\nC:\\Windows\\System32\\cmd.exe /k\n\"";
00123
00124     for (const auto entry : root["entries"]) {
00125       std::cout « "Entry " « counter « ":\n";
00126
00127       for (const auto key : entry.getMemberNames()) {
00128         std::cout « "    " « key « ": " « entry[key].asString() « "\n";
00129       }
00130
00131       if (entry["type"].asString() == "EXE") {
00132         batchFile « entry["command"].asString() « "&&\\\n";
00133       } else if (entry["type"].asString() == "ENV") {
00134         batchFile « "set " « entry["key"].asString() « "="
00135                   « entry["value"].asString() « "&&\\\n";
00136       } else if (entry["type"].asString() == "PATH") {
00137         additionalPath « entry["path"].asString() « ";\\\n";
00138       } else {
00139         batchFile « "\nCommand doesnt exist yet\n";
00140       }
00141
00142       ++counter;
00143     }
00144
00145     if (additionalPath.str() != "") {
00146       batchFile « "set path=%path%" « additionalPath.str();
00147     }
00148
00149     batchFile « "\"\n@ECHO ON";
```

```
00150     batchFile.close();
00151     */
00152     LOG(INFO) « "Application exiting!";
00153     return 0;
00154 }
```

8.9 src/sources/JSONData.cpp File Reference

```
#include "JSONData.hpp"
#include <easylogging++.h>
#include <ostream>
#include <stdexcept>
```

Include dependency graph for JSONData.cpp:



Namespaces

- namespace json

  json namespace

8.10 JSONData.cpp

Go to the documentation of this file.

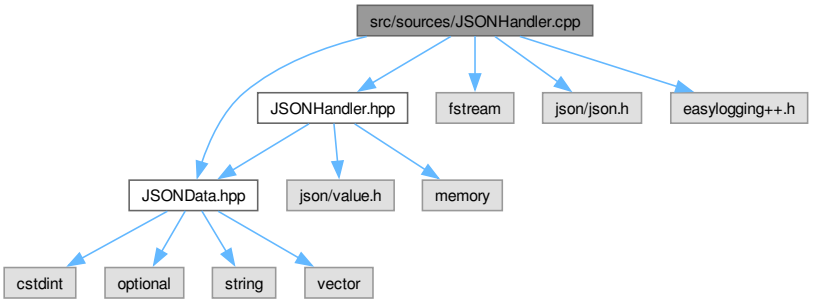
```
00001 #include "JSONData.hpp"
00002 #include <easylogging++.h>
00003 #include <ostream>
00004 #include <stdexcept>
00005
00006 namespace json {
00007 void JSONData::setOutputFile(std::string &outputfile)
00008 {
00009     if (outputfile.empty()) {
00010         LOG(ERROR) « "Tried to set empty outputfile!";
00011         throw std::invalid_argument("Outputfile cannot be empty");
00012     }
00013
00014     if (!this->outputfile.empty()) {
00015         LOG(ERROR) « "Outputfile already set!";
00016         throw std::invalid_argument("Outputfile already set");
00017     }
00018
00019     if (outputfile.find(".bat") == std::string::npos ||
00020         outputfile.find(".bat") != outputfile.size() - this->suffixLength) {
00021         outputfile += ".bat";
00022         std::cout « "Outputfile does not have .bat suffix, adding it now: "
00023                   « outputfile « std::endl;
00024         LOG(WARNING) « "Outputfile does not have .bat suffix, adding it now: "
00025                      « outputfile;
```

```
00026     }
00027
00028     this->outputfile = outputfile;
00029 }
00030
00031 void JSONData::setApplication(const std::string &application)
00032 {
00033     this->application.emplace(application);
00034 }
00035
00036 void JSONData::addCommand(const std::string &command)
00037 {
00038     if (command.empty()) {
00039         LOG(ERROR) « "Tried to add emoty command to data object!";
00040         throw std::invalid_argument("Command cannot be empty");
00041     }
00042
00043     this->commands.push_back(command);
00044 }
00045
00046 void JSONData::addEnvironmentVariable(const std::string &name,
00047                                       const std::string &value)
00048 {
00049     if (name.empty() || value.empty()) {
00050         LOG(ERROR) « "Tried to add invalid environment variable to data object!";
00051         LOG(INFO) « "Envirement variables have to have a name and a value!";
00052         throw std::invalid_argument("Name and value cannot be empty");
00053     }
00054
00055     this->environmentVariables.push_back(std::make_tuple(name, value));
00056 }
00057
00058 void JSONData::addPathValue(const std::string &pathValue)
00059 {
00060     if (pathValue.empty()) {
00061         LOG(ERROR) « "Tried to add empty path value to data object!";
00062         throw std::invalid_argument("Path value cannot be empty");
00063     }
00064
00065     this->pathValues.push_back(pathValue);
00066 }
00067 } // namespace json
```

8.11 src/sources/JSONHandler.cpp File Reference

```
#include "JSONHandler.hpp"
#include "JSONData.hpp"
#include <fstream>
#include <json/json.h>
#include <easylogging++.h>
```

Include dependency graph for JSONHandler.cpp:



Namespaces

- namespace json

json namespace

8.12 JSONHandler.cpp

Go to the documentation of this file.

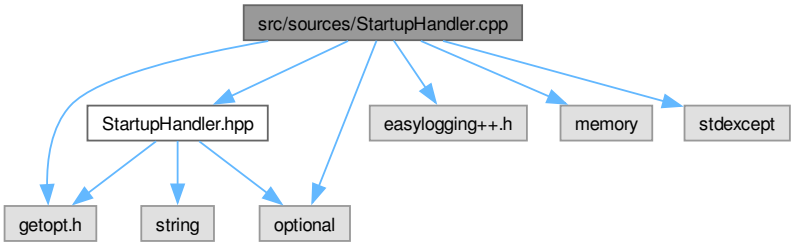
```
00001 #include "JSONHandler.hpp"
00002 #include "JSONData.hpp"
00003 #include <fstream>
00004 #include <json/json.h>
00005 #include <easylogging++.h>
00006
00007 namespace json {
00008 JSONHandler::JSONHandler(const std::string &filename)
00009 {
00010     this->root = parseFile(filename);
00011 }
00012
00013 std::shared_ptr<Json::Value> JSONHandler::parseFile(const std::string
00014                                                     &filename)
00015 {
00016     std::ifstream file(filename);
00017     Json::Value root;
00018     Json::Reader reader;
00019     reader.parse(file, root);
00020     return std::make_shared<Json::Value>(root);
00021 }
00022
00023 std::shared_ptr<JSONData> JSONHandler::getJSONData()
00024 {
00025     return this->createJSONData();
00026 }
00027
00028 std::shared_ptr<JSONData> JSONHandler::createJSONData()
00029 {
00030     this->data = std::make_shared<JSONData>();
00031     this->assignOutputFile();
00032     this->assignHideShell();
00033     this->assignApplication();
00034     this->assignEntries();
00035     this->assignEntries();
00036     return this->data;
00037 }
00038 void JSONHandler::assignOutputFile()
00039 {
00040     std::string outputFile = this->root->get("outputfile", "").asString();
00041     this->data->setOutputFile(outputFile);
00042 }
00043
00044 void JSONHandler::assignHideShell()
00045 {
00048     bool hideShell = this->root->get("hideShell", false).asBool();
00049     this->data->setHideShell(hideShell);
00050 }
00051
00052 void JSONHandler::assignApplication()
00053 {
00054     std::string application = this->root->get("application", "").asString();
00055     this->data->setApplication(application);
00056 }
00057
00058 void JSONHandler::assignEntries()
00059 {
00060     for (auto entry : this->root->get("entries", "")) {
00061         std::string entryType = entry.get("type", "").asString();
00062
00063         if (entryType == "EXE") {
00064             this->assignCommand(entry);
00065         }
00066         else if (entryType == "ENV") {
00067             this->assignEnvironmentVariable(entry);
00068         }
00069         else if (entryType == "PATH") {
00070             this->assignPathValue(entry);
00071         }
00072         else {
00073             LOG(ERROR) « "Unknown entry type";
00074             throw std::runtime_error("Unknown entry type");
00075         }
00076     }
00077 }
00078
```

```
00079 void JSONHandler::assignCommand(const Json::Value &entry)
00080 {
00081     std::string command = entry.get("command", "").asString();
00082     this->data->addCommand(command);
00083 }
00084
00085 void JSONHandler::assignEnvironmentVariable(const Json::Value &entry)
00086 {
00087     std::string key = entry.get("key", "").asString();
00088     std::string value = entry.get("value", "").asString();
00089     this->data->addEnvironmentVariable(key, value);
00090 }
00091
00092 void JSONHandler::assignPathValue(const Json::Value &entry)
00093 {
00094     std::string pathValue = entry.get("path", "").asString();
00095     this->data->addPathValue(pathValue);
00096 }
00097 } // namespace json
```

8.13 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

8.14 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