

jsonToBatProject

0.2.0

Generated on Wed Feb 28 2024 22:07:52 for jsonToBatProject by Doxygen 1.9.8

Wed Feb 28 2024 22:07:52

1 Bug List	1
2 Todo List	3
3 Namespace Index	5
3.1 Namespace List	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Namespace Documentation	11
6.1 batch Namespace Reference	11
6.2 clInterface Namespace Reference	11
6.3 json Namespace Reference	11
6.3.1 Detailed Description	11
6.4 utils Namespace Reference	12
6.4.1 Detailed Description	12
6.4.2 Variable Documentation	12
6.4.2.1 verbose	12
7 Class Documentation	13
7.1 batch::BatchCreator Class Reference	13
7.1.1 Detailed Description	13
7.1.2 Constructor & Destructor Documentation	14
7.1.2.1 BatchCreator()	14
7.1.3 Member Function Documentation	14
7.1.3.1 createBatchFile()	14
7.1.3.2 writeApplication()	14
7.1.3.3 writeCommands()	14
7.1.3.4 writeEnvironmentVariables()	14
7.1.3.5 writeHideShellEnd()	14
7.1.3.6 writeHideShellStart()	14
7.1.3.7 writePathValue()	15
7.1.3.8 writeShell()	15
7.1.4 Member Data Documentation	15
7.1.4.1 batchFile	15
7.1.4.2 jsonData	15
7.2 clInterface::CliHandler Class Reference	15
7.2.1 Detailed Description	15
7.3 json::JSONData Class Reference	15
7.3.1 Detailed Description	16

7.3.2 Member Function Documentation	16
7.3.2.1 addCommand()	16
7.3.2.2 addEnvironmentVariable()	17
7.3.2.3 addPathValue()	17
7.3.2.4 getApplication()	18
7.3.2.5 getCommands()	18
7.3.2.6 getEnvironmentVariables()	18
7.3.2.7 getHideShell()	19
7.3.2.8 getOutputFile()	19
7.3.2.9 getPathValues()	19
7.3.2.10 setApplication()	19
7.3.2.11 setHideShell()	20
7.3.2.12 setOutputFile()	20
7.3.3 Member Data Documentation	21
7.3.3.1 application	21
7.3.3.2 commands	21
7.3.3.3 environmentVariables	21
7.3.3.4 hideShell	21
7.3.3.5 outputfile	21
7.3.3.6 pathValues	21
7.3.3.7 suffixLength	21
7.4 json::JSONHandler Class Reference	22
7.4.1 Detailed Description	22
7.4.2 Constructor & Destructor Documentation	23
7.4.2.1 JSONHandler()	23
7.4.3 Member Function Documentation	23
7.4.3.1 assignApplication()	23
7.4.3.2 assignCommand()	24
7.4.3.3 assignEntries()	24
7.4.3.4 assignEnvironmentVariable()	25
7.4.3.5 assignHideShell()	26
7.4.3.6 assignOutputFile()	26
7.4.3.7 assignPathValue()	27
7.4.3.8 createJSONData()	27
7.4.3.9 getJSONData()	28
7.4.3.10 parseFile()	29
7.4.4 Member Data Documentation	30
7.4.4.1 data	30
7.4.4.2 root	30
7.5 utils::StartupHandler Class Reference	30
7.5.1 Detailed Description	31
7.5.2 Constructor & Destructor Documentation	31

7.5.2.1 StartupHandler() [1/2]	31
7.5.2.2 StartupHandler() [2/2]	31
7.5.3 Member Function Documentation	32
7.5.3.1 getOptions()	32
7.5.3.2 initEasyLogging()	33
7.5.3.3 operator=()	33
8 File Documentation	35
8.1 src/headers/BatchCreator.hpp File Reference	35
8.2 BatchCreator.hpp	36
8.3 src/headers/CliHandler.hpp File Reference	36
8.4 CliHandler.hpp	37
8.5 src/headers/JSONData.hpp File Reference	37
8.6 JSONData.hpp	38
8.7 src/headers/JSONHandler.hpp File Reference	39
8.8 JSONHandler.hpp	40
8.9 src/headers/StartupHandler.hpp File Reference	40
8.10 StartupHandler.hpp	41
8.11 src/main.cpp File Reference	42
8.11.1 Function Documentation	42
8.11.1.1 main()	42
8.12 main.cpp	43
8.13 src/sources/BatchCreator.cpp File Reference	44
8.14 BatchCreator.cpp	45
8.15 src/sources/CliHandler.cpp File Reference	46
8.16 CliHandler.cpp	46
8.17 src/sources/JSONData.cpp File Reference	46
8.18 JSONData.cpp	47
8.19 src/sources/JSONHandler.cpp File Reference	48
8.20 JSONHandler.cpp	48
8.21 src/sources/StartupHandler.cpp File Reference	49
8.22 StartupHandler.cpp	50
Index	53

Chapter 1

Bug List

Class `batch::BatchCreator`

HideShell is not implemented correctly

Namespace `json`

Name too similar to "Json" namespace from the json library.

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

Initializes too 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 config only recognized when running application from source dir

Chapter 2

Todo List

Member `json::JSONHandler::assignHideShell () const`

: 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:

batch	11
cliInterface	11
json		
	Json namespace	11
utils		
	Namespace for utility functions	12

Chapter 4

Class Index

4.1 Class List

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

batch::BatchCreator	13
cliInterface::CliHandler	15
json::JSONData	15
json::JSONHandler	
JSONHandler class	22
utils::StartupHandler	
Handles startup task for the application	30

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	42
src/headers/BatchCreator.hpp	35
src/headers/CliHandler.hpp	36
src/headers/JSONData.hpp	37
src/headers/JSONHandler.hpp	39
src/headers/StartupHandler.hpp	40
src/sources/BatchCreator.cpp	44
src/sources/CliHandler.cpp	46
src/sources/JSONData.cpp	46
src/sources/JSONHandler.cpp	48
src/sources/StartupHandler.cpp	49

Chapter 6

Namespace Documentation

6.1 batch Namespace Reference

Classes

- class [BatchCreator](#)

6.2 cliInterface Namespace Reference

Classes

- class [CliHandler](#)

6.3 json Namespace Reference

json namespace

Classes

- class [JSONData](#)
- class [JSONHandler](#)
[JSONHandler](#) class.

6.3.1 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.

6.4 utils Namespace Reference

Namespace for utility functions.

Classes

- class [StartupHandler](#)
Handles startup task for the application.

Variables

- static int [verbose](#) = 0

6.4.1 Detailed Description

Namespace for utility functions.

This namespace contains utility functions for the application. Currently, it contains the [StartupHandler](#) class.

6.4.2 Variable Documentation

6.4.2.1 verbose

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

Definition at line 11 of file [StartupHandler.cpp](#).

Chapter 7

Class Documentation

7.1 batch::BatchCreator Class Reference

```
#include <BatchCreator.hpp>
```

Public Member Functions

- [BatchCreator](#) (std::shared_ptr< [json::JSONData](#) > [jsonData](#))
- std::shared_ptr< std::ofstream > [createBatchFile](#) ()

Private Member Functions

- void [writeHideShellStart](#) ()
- void [writeHideShellEnd](#) ()
- void [writeShell](#) ()
- void [writeCommands](#) ()
- void [writeEnvironmentVariables](#) ()
- void [writePathValue](#) ()
- void [writeApplication](#) ()

Private Attributes

- std::shared_ptr< [json::JSONData](#) > [jsonData](#)
- std::shared_ptr< std::ofstream > [batchFile](#) = nullptr

7.1.1 Detailed Description

Bug HideShell is not implemented correctly

Definition at line 10 of file [BatchCreator.hpp](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BatchCreator()

```
batch::BatchCreator::BatchCreator (
    std::shared_ptr< json::JSONData > jsonData ) [explicit]
```

7.1.3 Member Function Documentation

7.1.3.1 createBatchFile()

```
std::shared_ptr< std::ofstream > batch::BatchCreator::createBatchFile ( )
```

Here is the caller graph for this function:



7.1.3.2 writeApplication()

```
void batch::BatchCreator::writeApplication ( ) [private]
```

7.1.3.3 writeCommands()

```
void batch::BatchCreator::writeCommands ( ) [private]
```

7.1.3.4 writeEnvironmentVariables()

```
void batch::BatchCreator::writeEnvironmentVariables ( ) [private]
```

7.1.3.5 writeHideShellEnd()

```
void batch::BatchCreator::writeHideShellEnd ( ) [private]
```

7.1.3.6 writeHideShellStart()

```
void batch::BatchCreator::writeHideShellStart ( ) [private]
```

7.1.3.7 writePathValue()

```
void batch::BatchCreator::writePathValue ( ) [private]
```

7.1.3.8 writeShell()

```
void batch::BatchCreator::writeShell ( ) [private]
```

7.1.4 Member Data Documentation

7.1.4.1 batchFile

```
std::shared_ptr<std::ofstream> batch::BatchCreator::batchFile = nullptr [private]
```

Definition at line 18 of file [BatchCreator.hpp](#).

7.1.4.2 jsonData

```
std::shared_ptr<json::JSONData> batch::BatchCreator::jsonData [private]
```

Definition at line 17 of file [BatchCreator.hpp](#).

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

- src/headers/[BatchCreator.hpp](#)

7.2 cllInterface::CliHandler Class Reference

```
#include <CliHandler.hpp>
```

7.2.1 Detailed Description

Definition at line 10 of file [CliHandler.hpp](#).

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

- src/headers/[CliHandler.hpp](#)

7.3 json::JSONData Class Reference

```
#include <JSONData.hpp>
```

Public Member Functions

- void [setOutputFile](#) (std::string &newOutputfile)
Set's the output file.
- void [setHideShell](#) (bool newHideShell)
Set's the hide shell flag.
- void [setApplication](#) (const std::string &newApplication)
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::optional< 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

7.3.1 Detailed Description

Definition at line 10 of file [JSONData.hpp](#).

7.3.2 Member Function Documentation

7.3.2.1 addCommand()

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

Add a command to the commands vector.

Parameters

<i>command</i>	The command
----------------	-------------

Exceptions

<i>std::invalid_argument</i>	if the command is empty
------------------------------	-------------------------

Definition at line 37 of file [JSONData.cpp](#).

References [commands](#).

7.3.2.2 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

<i>name</i>	The name of the environment variable
<i>value</i>	The value of the environment variable

Exceptions

<i>std::invalid_argument</i>	if the name or the value is empty
------------------------------	-----------------------------------

Definition at line 46 of file [JSONData.cpp](#).

References [environmentVariables](#).

7.3.2.3 addPathValue()

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

Add a path value to the pathValues vector.

Parameters

<i>pathValue</i>	The path value
------------------	----------------

Exceptions

<code>std::invalid_argument</code>	if the pathValue is empty
------------------------------------	---------------------------

Definition at line 57 of file [JSONData.cpp](#).

References [pathValues](#).

7.3.2.4 `getApplication()`

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

Get the application.

Returns

The application

Definition at line 90 of file [JSONData.hpp](#).

References [application](#).

7.3.2.5 `getCommands()`

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

Get the commands.

Returns

The commands

Definition at line 98 of file [JSONData.hpp](#).

References [commands](#).

7.3.2.6 `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 107 of file [JSONData.hpp](#).

References [environmentVariables](#).

7.3.2.7 getHideShell()

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

Get the hide shell flag.

Returns

The hide shell flag

Definition at line 82 of file [JSONData.hpp](#).

References [hideShell](#).

7.3.2.8 getOutputFile()

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

Get the output file.

Returns

The output file

Definition at line 74 of file [JSONData.hpp](#).

References [outputfile](#).

7.3.2.9 getPathValues()

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

Get the path values.

Returns

The path values

Definition at line 115 of file [JSONData.hpp](#).

References [pathValues](#).

7.3.2.10 setApplication()

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

Set's the application.

Parameters

<i>application</i>	The application
--------------------	-----------------

Definition at line 30 of file [JSONData.cpp](#).

References [application](#).

7.3.2.11 setHideShell()

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

Set's the hide shell flag.

Parameters

<i>hideShell</i>	The hide shell flag
------------------	---------------------

Definition at line 29 of file [JSONData.hpp](#).

References [hideShell](#).

7.3.2.12 setOutputFile()

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

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

<i>outputfile</i>	The output file
-------------------	-----------------

Exceptions

<i>std::invalid_argument</i>	if the outputfile is empty
<i>std::invalid_argument</i>	if the outputfile is already set

Definition at line 7 of file [JSONData.cpp](#).

References [outputfile](#), and [suffixLength](#).

7.3.3 Member Data Documentation

7.3.3.1 application

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

Definition at line 122 of file [JSONData.hpp](#).

7.3.3.2 commands

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

Definition at line 123 of file [JSONData.hpp](#).

7.3.3.3 environmentVariables

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

Definition at line 124 of file [JSONData.hpp](#).

7.3.3.4 hideShell

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

Definition at line 121 of file [JSONData.hpp](#).

7.3.3.5 outputfile

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

Definition at line 120 of file [JSONData.hpp](#).

7.3.3.6 pathValues

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

Definition at line 125 of file [JSONData.hpp](#).

7.3.3.7 suffixLength

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

Definition at line 126 of file [JSONData.hpp](#).

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

- [src/headers/JSONData.hpp](#)
- [src/sources/JSONData.cpp](#)

7.4 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) const
Parse a file.
- void [assignOutputFile](#) () const
Assigns the output file to the [JSONData](#) object.
- void [assignHideShell](#) () const
Assigns the hide shell value to the [JSONData](#) object.
- void [assignApplication](#) () const
Assigns the application to the [JSONData](#) object.
- void [assignEntries](#) () const
Assigns the entries to the [JSONData](#) object.
- void [assignCommand](#) (const Json::Value &entry) const
Assigns a command to the [JSONData](#) object.
- void [assignEnvironmentVariable](#) (const Json::Value &entry) const
Assigns an environment variable to the [JSONData](#) object.
- void [assignPathValue](#) (const Json::Value &entry) const
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](#)

7.4.1 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](#).

7.4.2 Constructor & Destructor Documentation

7.4.2.1 JSONHandler()

```
json::JSONHandler::JSONHandler (
    const std::string & filename ) [explicit]
```

Constructor.

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

Parameters

<i>filename</i>	The filename to parse
-----------------	-----------------------

Definition at line 9 of file [JSONHandler.cpp](#).

References [parseFile\(\)](#), and [root](#).

Here is the call graph for this function:



7.4.3 Member Function Documentation

7.4.3.1 assignApplication()

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

Assigns the application to the [JSONData](#) object.

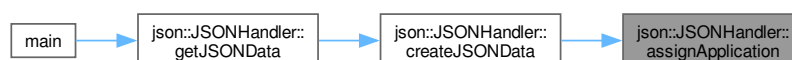
Note

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

Definition at line 47 of file [JSONHandler.cpp](#).

References [data](#), and [root](#).

Here is the caller graph for this function:



7.4.3.2 assignCommand()

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

Assigns a command to the [JSONData](#) object.

The function takes a `Json::Value` object and assigns the command to the [JSONData](#) object

Parameters

<code>entry</code>	The entry to assign
--------------------	---------------------

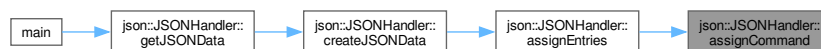
Note

Error handling is done in the [JSONData](#) object

Definition at line 69 of file [JSONHandler.cpp](#).

References [data](#).

Here is the caller graph for this function:



7.4.3.3 assignEntries()

```
void json::JSONHandler::assignEntries ( ) const [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

<code>std::runtime_error</code>	If the entry type is unknown
---------------------------------	------------------------------

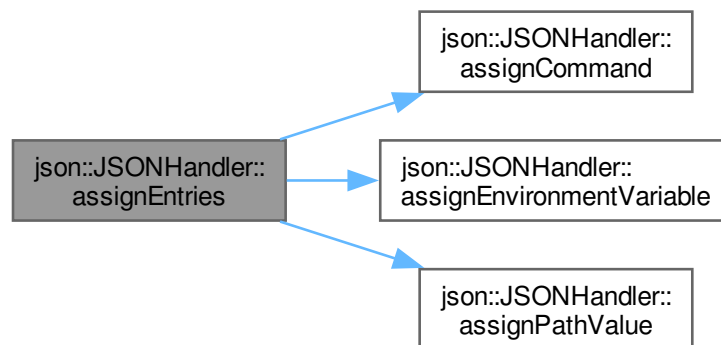
Note

Other error handling is done in the [JSONData](#) object

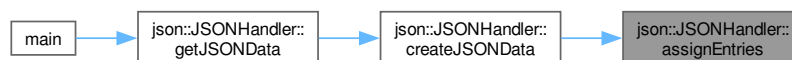
Definition at line 52 of file [JSONHandler.cpp](#).

References [assignCommand\(\)](#), [assignEnvironmentVariable\(\)](#), [assignPathValue\(\)](#), and [root](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.4 assignEnvironmentVariable()

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

<i>entry</i>	The entry to assign
--------------	---------------------

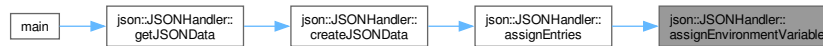
Note

Error handling is done in the [JSONData](#) object

Definition at line [74](#) of file [JSONHandler.cpp](#).

References [data](#).

Here is the caller graph for this function:



7.4.3.5 assignHideShell()

```
void json::JSONHandler::assignHideShell ( ) const [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 40 of file [JSONHandler.cpp](#).

References [data](#), and [root](#).

Here is the caller graph for this function:



7.4.3.6 assignOutputFile()

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

Assigns the output file to the [JSONData](#) object.

Note

Error handling is done in the [JSONData](#) object

Definition at line 35 of file [JSONHandler.cpp](#).

References [data](#), and [root](#).

Here is the caller graph for this function:



7.4.3.7 assignPathValue()

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

<i>entry</i>	The entry to assign
--------------	---------------------

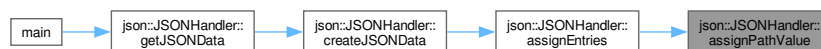
Note

Error handling is done in the [JSONData](#) object

Definition at line 80 of file [JSONHandler.cpp](#).

References [data](#).

Here is the caller graph for this function:



7.4.3.8 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.

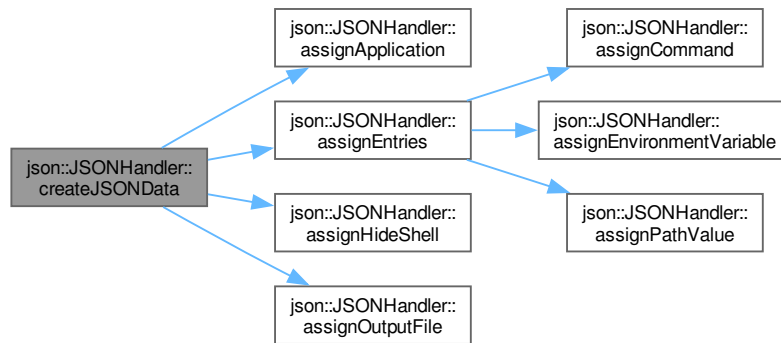
Returns

`std::shared_ptr<JSONData>` The [JSONData](#) object

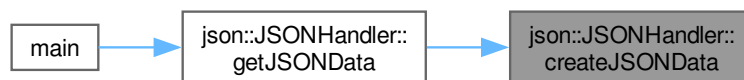
Definition at line 26 of file [JSONHandler.cpp](#).

References [assignApplication\(\)](#), [assignEntries\(\)](#), [assignHideShell\(\)](#), [assignOutputFile\(\)](#), and [data](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.9 getJSONData()

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

Retrieve the [JSONData](#) object.

The function takes the necessary steps to create a [JSONData](#) object and then returns it

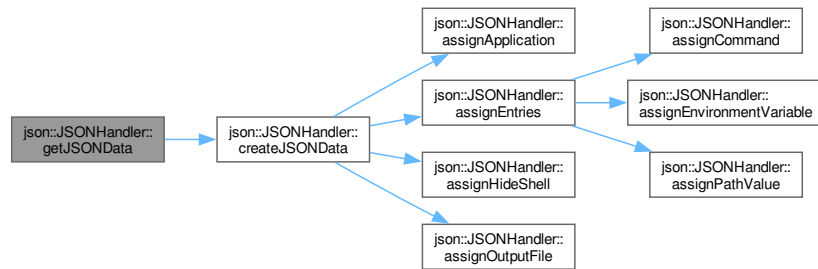
Returns

`std::shared_ptr<JSONData>` The [JSONData](#) object

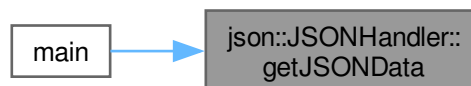
Definition at line 22 of file [JSONHandler.cpp](#).

References [createJSONData\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.10 parseFile()

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

Parse a file.

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

Parameters

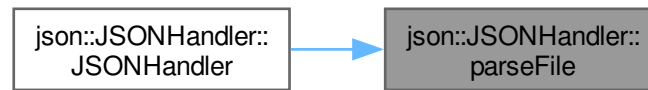
<i>filename</i>	The filename to parse
-----------------	-----------------------

Returns

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

Definition at line 13 of file [JSONHandler.cpp](#).

Here is the caller graph for this function:



7.4.4 Member Data Documentation

7.4.4.1 data

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

Definition at line 158 of file [JSONHandler.hpp](#).

7.4.4.2 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](#)

7.5 utils::StartupHandler Class Reference

Handles startup task for the application.

```
#include <StartupHandler.hpp>
```

Public Member Functions

- [StartupHandler](#) (const [StartupHandler](#) &)=delete
Copy constructor (deleted)
- [StartupHandler](#) & [operator=](#) (const [StartupHandler](#) &)=delete
Assignment operator (deleted)

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)

7.5.1 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](#).

7.5.2 Constructor & Destructor Documentation

7.5.2.1 StartupHandler() [1/2]

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

Copy constructor (deleted)

This class should not be instantiated.

7.5.2.2 StartupHandler() [2/2]

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

Constructor (private)

This class should not be instantiated.

7.5.3 Member Function Documentation

7.5.3.1 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 handle 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

<i>argc</i>	Number of arguments
<i>argv</i>	Arguments

Returns

Returns either the filename or nothing.

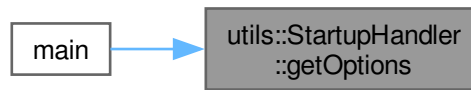
Exceptions

<i>std::invalid_argument</i>	If more than one filename is given.
------------------------------	-------------------------------------

Definition at line 20 of file [StartupHandler.cpp](#).

References [utils::verbose](#).

Here is the caller graph for this function:



7.5.3.2 initEasyLogging()

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

Initialize easylogging.

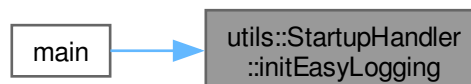
This function initializes easylogging with the configuration file "\$SOURCE/conf/easylogging.conf".

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:



7.5.3.3 operator=()

```
StartupHandler & utils::StartupHandler::operator= (
    const StartupHandler & ) [deleted]
```

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

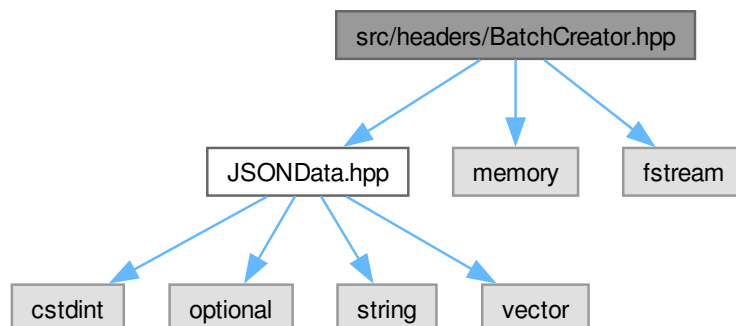
8.1 src/headers/BatchCreator.hpp File Reference

```
#include "JSONData.hpp"
```

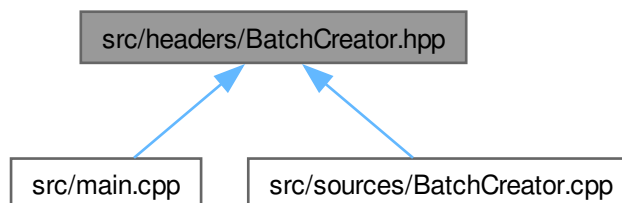
```
#include <memory>
```

```
#include <fstream>
```

Include dependency graph for BatchCreator.hpp:



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



Classes

- class [batch::BatchCreator](#)

Namespaces

- namespace [batch](#)

8.2 BatchCreator.hpp

[Go to the documentation of this file.](#)

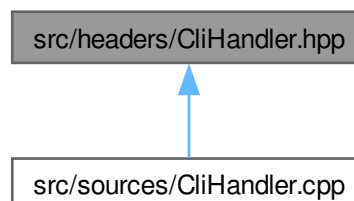
```

00001 #ifndef BATCHCREATOR_HPP
00002 #define BATCHCREATOR_HPP
00003
00004 #include "JSONData.hpp"
00005 #include <memory>
00006 #include <fstream>
00007
00008 namespace batch {
00009     class BatchCreator {
00010     public:
00011         explicit BatchCreator(std::shared_ptr<json::JSONData> jsonData);
00012
00013         std::shared_ptr<std::ofstream> createBatchFile();
00014
00015     private:
00016         std::shared_ptr<json::JSONData> jsonData;
00017         std::shared_ptr<std::ofstream> batchFile = nullptr;
00018
00019         void writeHideShellStart();
00020
00021         void writeHideShellEnd();
00022
00023         void writeShell();
00024
00025         void writeCommands();
00026
00027         void writeEnvironmentVariables();
00028
00029         void writePathValue();
00030
00031         void writeApplication();
00032
00033     };
00034 } // namespace batch
00035
00036 #endif // BATCHCREATOR_HPP

```

8.3 src/headers/CliHandler.hpp File Reference

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



Classes

- class `clInterface::CliHandler`

Namespaces

- namespace `clInterface`

8.4 CliHandler.hpp

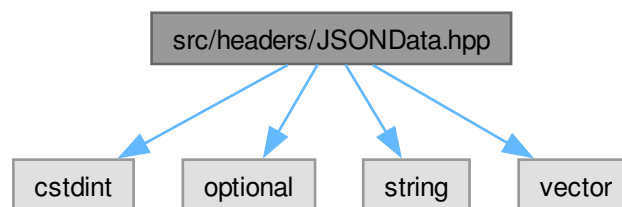
[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by simon on 28.02.24.
00003 //
00004
00005 #ifndef JSONTOBATCH_CLIHANDLER_HPP
00006 #define JSONTOBATCH_CLIHANDLER_HPP
00007
00008 namespace clInterface {
00009
00010     class CliHandler {
00011     };
00012
00013 }
00014 } // clInterface
00015
00016 #endif //JSONTOBATCH_CLIHANDLER_HPP
```

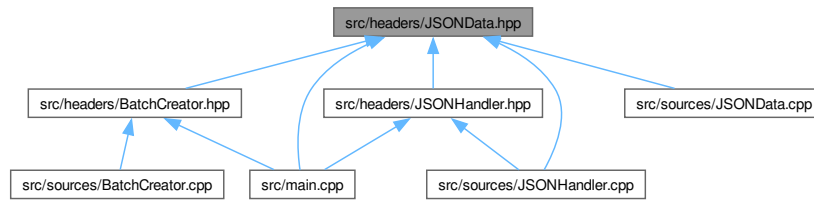
8.5 src/headers/JSONData.hpp File Reference

```
#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

8.6 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
00009 namespace json {
00010     class JSONData {
00011     public:
00023         void setOutputFile(std::string &newOutputfile);
00024
00029         void setHiddenShell(bool newHideShell) {
00030             this->hideShell = newHideShell;
00031         }
00032
00037         void setApplication(const std::string &newApplication);
00038
00045         void addCommand(const std::string &command);
00046
00059         void addEnvironmentVariable(const std::string &name,
00060                                     const std::string &value);
00061
00068         void addPathValue(const std::string &pathValue);
00069
00074         [[nodiscard]] const std::string &getOutputFile() const {
00075             return outputfile;
00076         }
00077
00082         [[nodiscard]] bool getHideShell() const {
00083             return hideShell;
00084         }
00085
00090         [[nodiscard]] const std::optional<std::string> &getApplication() const {
00091             return application;
00092         }
00093
00098         [[nodiscard]] const std::vector<std::string> &getCommands() const {
00099             return commands;
00100         }
  
```

```

00101
00106     [[nodiscard]] const std::vector<std::tuple<std::string, std::string>>
00107     &getEnvironmentVariables() const {
00108         return environmentVariables;
00109     }
00110
00115     [[nodiscard]] const std::vector<std::string> &getPathValues() const {
00116         return pathValues;
00117     }
00118
00119     private:
00120         std::string outputfile;
00121         bool hideShell;
00122         std::optional<std::string> application;
00123         std::vector<std::string> commands;
00124         std::vector<std::tuple<std::string, std::string>> environmentVariables;
00125         std::vector<std::string> pathValues;
00126         const static int8_t suffixLength = 4;
00127     };
00128 } // namespace json
00129
00130 #endif // JSONDATA_HPP

```

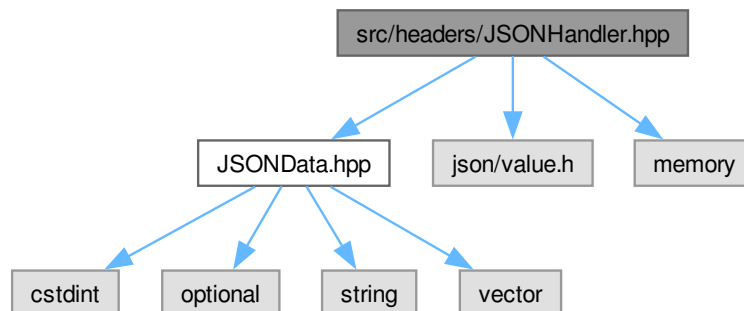
8.7 src/headers/JSONHandler.hpp File Reference

```

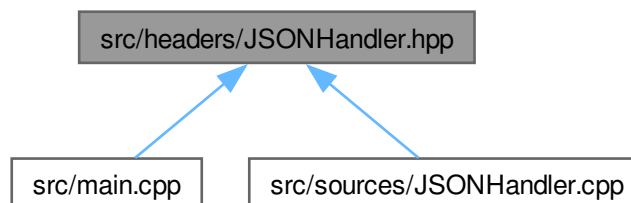
#include "JSONData.hpp"
#include "json/value.h"
#include <memory>

```

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

8.8 JSONHandler.hpp

[Go to the documentation of this file.](#)

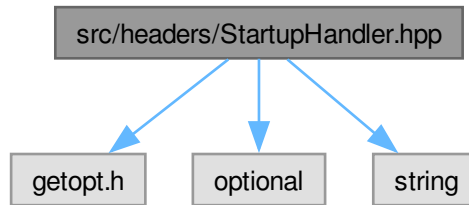
```
00001 #ifndef JSONHANDLER_HPP
00002 #define JSONHANDLER_HPP
00003
00004 #include "JSONData.hpp"
00005 #include "json/value.h"
00006 #include <memory>
00007
00017 namespace json {
00018
00029     class JSONHandler {
00030     public:
00040         explicit JSONHandler(const std::string &filename);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

8.9 src/headers/StartupHandler.hpp File Reference

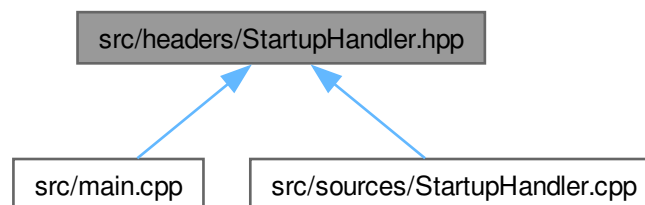
```
#include <getopt.h>
#include <optional>
```

```
#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.

8.10 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
00080         StartupHandler(const StartupHandler &) = delete;
00081
00088         StartupHandler &operator=(const StartupHandler &) = delete;
00089
00090     private:
00097         StartupHandler() = default;
00098
00099
00100     };
00101 } // namespace utils
00102 #endif // STARTUPHANDLER_HPP

```

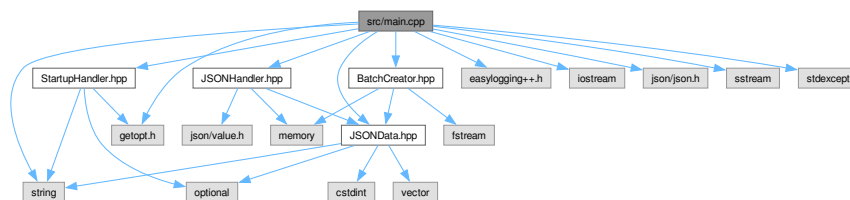
8.11 src/main.cpp File Reference

```

#include "JSONData.hpp"
#include "StartupHandler.hpp"
#include "JSONHandler.hpp"
#include "BatchCreator.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.

8.11.1 Function Documentation

8.11.1.1 main()

```

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

```

Main function.

Bug Initializes too 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. Furthermore it provides a CLI to help the user to interact with the application.

Bug Getopt is not working on Windows.

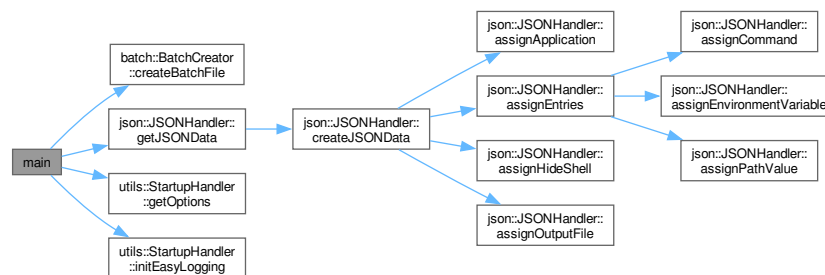
Note

maybe close in creator? But this leaves possibility to add more stuff - why?

Definition at line 29 of file [main.cpp](#).

References [batch::BatchCreator::createBatchFile\(\)](#), [json::JSONHandler::getJSONData\(\)](#), [utils::StartupHandler::getOptions\(\)](#), and [utils::StartupHandler::initEasyLogging\(\)](#).

Here is the call graph for this function:



8.12 main.cpp

[Go to the documentation of this file.](#)

```

00001 #include "JSONData.hpp"
00002 #include "StartupHandler.hpp"
00003 #include "JSONHandler.hpp"
00004 #include "BatchCreator.hpp"
00005
00006 #include <easylogging++.h>
00007 #include <getopt.h>
00008 #include <iostream>
00009 #include <json/json.h>
00010 #include <sstream>
00011 #include <stdexcept>
00012 #include <string>
00013
00015 INITIALIZE_EASYLOGGINGPP
00016
00029 int main(int argc, char *argv[]) {
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;
00059
00060     // Initialize the JSONHandler with the file(name)
00061     json::JSONHandler jsonHandler(filename.value());
00062     // Get a JSONData object from the JSONHandler
00063     std::shared_ptr<json::JSONData> jsonData = jsonHandler.getJSONData();
00064     // Print the outputfile as a test
00065     std::cout << "Outputfile: " << jsonData->getOutputFile() << std::endl;
00066
00067     batch::BatchCreator batchCreator(jsonData);
00070     std::shared_ptr<std::ofstream> batchFile = batchCreator.createBatchFile();
00071     batchFile->close();
00072
00073     LOG(INFO) << "Application exiting!";
00074     return 0;
00075 }

```

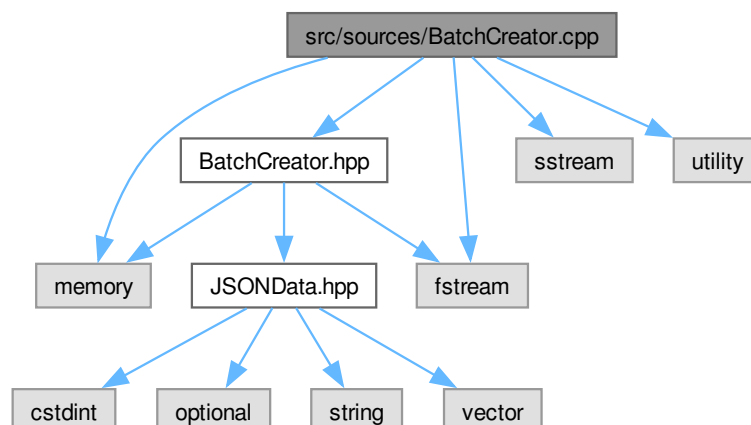
8.13 src/sources/BatchCreator.cpp File Reference

```

#include "BatchCreator.hpp"
#include <fstream>
#include <memory>
#include <sstream>
#include <utility>

```

Include dependency graph for BatchCreator.cpp:



8.14 BatchCreator.cpp

[Go to the documentation of this file.](#)

```

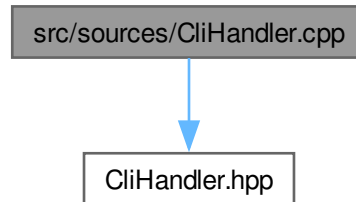
00001 #include "BatchCreator.hpp"
00002 #include <fstream>
00003 #include <memory>
00004 #include <sstream>
00005 #include <utility>
00006
00007 namespace batch {
00008
00009     BatchCreator::BatchCreator(std::shared_ptr<json::JSONData> jsonData)
00010         : jsonData(std::move(jsonData)) {}
00011
00012     std::shared_ptr<std::ofstream> BatchCreator::createBatchFile() {
00013         this->batchFile = std::make_shared<std::ofstream>();
00014         this->batchFile->open(this->jsonData->getOutputFile());
00015         if (this->jsonData->getHideShell()) {
00016             this->writeHideShellStart();
00017         }
00018         this->writeShell();
00019         this->writeCommands();
00020         this->writeEnvironmentVariables();
00021         this->writePathValue();
00022         this->writeApplication();
00023         if (this->jsonData->getHideShell()) {
00024             this->writeHideShellEnd();
00025         }
00026         return this->batchFile;
00027     }
00028
00029     void BatchCreator::writeHideShellStart() { *this->batchFile << "@ECHO OFF\n"; }
00030
00031     void BatchCreator::writeHideShellEnd() { *this->batchFile << "@ECHO ON\n"; }
00032
00033     void BatchCreator::writeShell() {
00034         if (this->jsonData->getHideShell()) {
00035             *this->batchFile << R"(START C:\Windows\System32\cmd.exe /C ")";
00036             return;
00037         }
00038         *this->batchFile << R"(START C:\Windows\System32\cmd.exe /K ")";
00039     }
00040
00041     void BatchCreator::writeCommands() {
00042         for (const auto &command: this->jsonData->getCommands()) {
00043             *this->batchFile << "CALL " << command << " && ^\n";
00044         }
00045     }
00046
00047     void BatchCreator::writeEnvironmentVariables() {
00048         for (const auto &envVar: this->jsonData->getEnvironmentVariables()) {
00049             *this->batchFile << "SET " << std::get<0>(envVar) << "="
00050                 << std::get<1>(envVar) << " && ^\n";
00051         }
00052     }
00053
00054     void BatchCreator::writePathValue() {
00055         std::stringstream additionalPaths;
00056         for (const auto &pathValue: this->jsonData->getPathValues()) {
00057             additionalPaths << pathValue << ";";
00058         }
00059         *this->batchFile << "SET PATH=%PATH%; " << additionalPaths.str();
00060     }
00061
00062     void BatchCreator::writeApplication() {
00063         if (!this->jsonData->getApplication().has_value()) {
00064             *this->batchFile << "\"\n";
00065             return;
00066         }
00067         *this->batchFile << " && ^\n"
00068             << this->jsonData->getApplication().value()
00069             << "\"\n";
00070     }
00071 } // namespace batch
00072

```

8.15 src/sources/CliHandler.cpp File Reference

```
#include "CliHandler.hpp"
```

Include dependency graph for CliHandler.cpp:



Namespaces

- namespace [cliInterface](#)

8.16 CliHandler.cpp

[Go to the documentation of this file.](#)

```

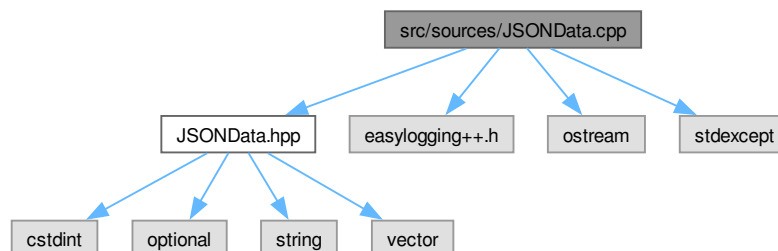
00001 //
00002 // Created by simon on 28.02.24.
00003 //
00004
00005 #include "CliHandler.hpp"
00006
00007 namespace cliInterface {
00008 } // cliInterface
  
```

8.17 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

8.18 JSONData.cpp

[Go to the documentation of this file.](#)

```

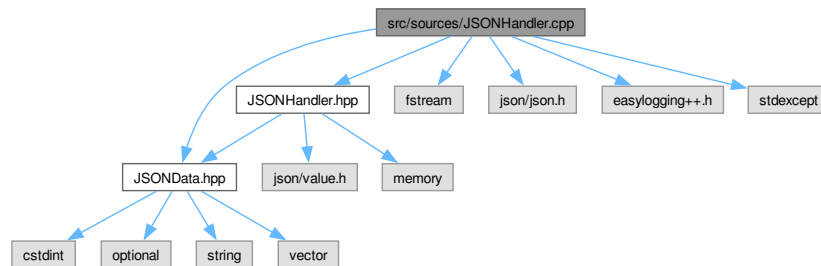
00001 #include "JSONData.hpp"
00002 #include <easylogging++.h>
00003 #include <ostream>
00004 #include <stdexcept>
00005
00006 namespace json {
00007     void JSONData::setOutputFile(std::string &newOutputfile) {
00008         if (newOutputfile.empty()) {
00009             LOG(ERROR) << "Tried to set empty outputfile!";
00010             throw std::invalid_argument("Outputfile cannot be empty");
00011         }
00012
00013         if (!this->outputfile.empty()) {
00014             LOG(ERROR) << "Outputfile already set!";
00015             throw std::invalid_argument("Outputfile already set");
00016         }
00017
00018         if (newOutputfile.find(".bat") == std::string::npos ||
00019             newOutputfile.find(".bat") != newOutputfile.size() - JSONData::suffixLength) {
00020             newOutputfile += ".bat";
00021             std::cerr << "Outputfile does not have .bat suffix, adding it now: "
00022                 << newOutputfile << std::endl;
00023             LOG(WARNING) << "Outputfile does not have .bat suffix, adding it now: "
00024                 << newOutputfile;
00025         }
00026
00027         this->outputfile = newOutputfile;
00028     }
00029
00030     void JSONData::setApplication(const std::string &newApplication) {
00031         if (newApplication.empty()) {
00032             return;
00033         }
00034         this->application.emplace(newApplication);
00035     }
00036
00037     void JSONData::addCommand(const std::string &command) {
00038         if (command.empty()) {
00039             LOG(ERROR) << "Tried to add empty 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         if (name.empty() || value.empty()) {
00049             LOG(ERROR) << "Tried to add invalid environment variable to data object!";
00050             LOG(INFO) << "Envirement variables have to have a name and a value!";
00051             throw std::invalid_argument("Name and value cannot be empty");
00052         }
00053
00054         this->environmentVariables.emplace_back(name, value);
00055     }
00056
00057     void JSONData::addPathValue(const std::string &pathValue) {
00058         if (pathValue.empty()) {
00059             LOG(ERROR) << "Tried to add empty path value to data object!";
00060             throw std::invalid_argument("Path value cannot be empty");
00061         }
00062
00063         this->pathValues.push_back(pathValue);
00064     }
00065 } // namespace json

```

8.19 src/sources/JSONHandler.cpp File Reference

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

Include dependency graph for JSONHandler.cpp:



Namespaces

- namespace [json](#)
json namespace

8.20 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 #include <stdexcept>
00007
00008 namespace json {
00009     JSONHandler::JSONHandler(const std::string &filename) {
00010         this->root = parseFile(filename);
00011     }
00012
00013     std::shared_ptr<Json::Value> JSONHandler::parseFile(const std::string
00014                                                         &filename) const {
00015         std::ifstream file(filename);
00016         Json::Value newRoot;
00017         Json::Reader reader;
00018         reader.parse(file, newRoot);
00019         return std::make_shared<Json::Value>(newRoot);
00020     }
00021
00022     std::shared_ptr<JSONData> JSONHandler::getJSONData() {
00023         return this->createJSONData();
00024     }
00025
00026     std::shared_ptr<JSONData> JSONHandler::createJSONData() {
00027         this->data = std::make_shared<JSONData>();
00028         this->assignOutputFile();
00029         this->assignHideShell();
00030         this->assignApplication();
00031         this->assignEntries();
00032         return this->data;
00033     }
```

```

00034
00035 void JSONHandler::assignOutputFile() const {
00036     std::string outputFile = this->root->get("outputfile", "").asString();
00037     this->data->setOutputFile(outputFile);
00038 }
00039
00040 void JSONHandler::assignHideShell() const {
00043     bool hideShell = this->root->get("hideshell", false).asBool();
00044     this->data->setHideShell(hideShell);
00045 }
00046
00047 void JSONHandler::assignApplication() const {
00048     std::string application = this->root->get("application", "").asString();
00049     this->data->setApplication(application);
00050 }
00051
00052 void JSONHandler::assignEntries() const {
00053     for (const auto &entry: this->root->get("entries", "")) {
00054         std::string entryType = entry.get("type", "").asString();
00055
00056         if (entryType == "EXE") {
00057             this->assignCommand(entry);
00058         } else if (entryType == "ENV") {
00059             this->assignEnvironmentVariable(entry);
00060         } else if (entryType == "PATH") {
00061             this->assignPathValue(entry);
00062         } else {
00063             LOG(ERROR) << "Unknown entry type";
00064             throw std::invalid_argument("Unknown entry type");
00065         }
00066     }
00067 }
00068
00069 void JSONHandler::assignCommand(const Json::Value &entry) const {
00070     std::string command = entry.get("command", "").asString();
00071     this->data->addCommand(command);
00072 }
00073
00074 void JSONHandler::assignEnvironmentVariable(const Json::Value &entry) const {
00075     std::string key = entry.get("key", "").asString();
00076     std::string value = entry.get("value", "").asString();
00077     this->data->addEnvironmentVariable(key, value);
00078 }
00079
00080 void JSONHandler::assignPathValue(const Json::Value &entry) const {
00081     std::string pathValue = entry.get("path", "").asString();
00082     this->data->addPathValue(pathValue);
00083 }
00084 } // namespace json

```

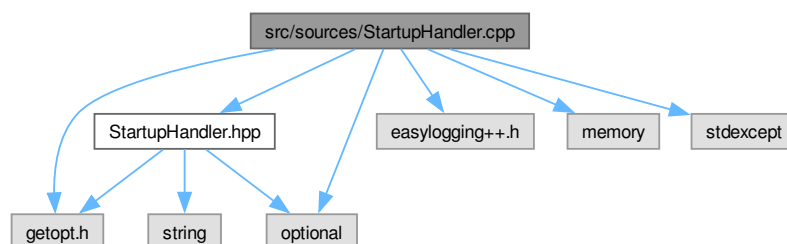
8.21 src/sources/StartupHandler.cpp File Reference

```

#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

8.22 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         el::Configurations conf("conf/easylogging.conf");
00015         el::Loggers::reconfigureLogger("default", conf);
00016         el::Loggers::reconfigureAllLoggers(conf);
00017         LOG(INFO) << "Easylogging initialized!";
00018     }
00019
00020     std::optional<std::string> StartupHandler::getOptions(int argc, char *argv[]) {
00021         LOG(INFO) << "Parsing options...";
00022         static const struct option long_options[] = {
00023             /* These options set a flag. */
00024             {"verbose", no_argument, &verbose, 1},
00025             {"brief", no_argument, &verbose, 0},
00026             {"help", no_argument, nullptr, 'h'},
00027             {"version", no_argument, nullptr, 'V'},
00028             {"test", required_argument, nullptr, 0},
00029             nullptr
00030         };
00031
00032         do {
00033             int optIndex = -1;
00034             std::unique_ptr<struct option> opt = nullptr;
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                     break;
00064                 default:
```



```
00065         std::cout << "I shouldnt have been here!\n";
00066         break;
00067     }
00068     } while (true);
00069
00070     LOG(INFO) << "Parsing options done";
00071     std::optional<std::string> filename = {};
00072     LOG(INFO) << "Parsing other arguments...";
00073
00074     while (optind < argc) {
00075         if (filename.has_value()) {
00076             LOG(ERROR) << "Only one filename can be given!";
00077             throw std::invalid_argument("Only one filename can be given!\n");
00078         }
00079
00080         LOG(INFO) << "Filename set to: " << argv[optind];
00081         filename = std::string(argv[optind++]);
00082     }
00083
00084     return filename;
00085 }
00086 } // namespace utils
```


Index

- addCommand
 - json::JSONData, [16](#)
- addEnvironmentVariable
 - json::JSONData, [17](#)
- addPathValue
 - json::JSONData, [17](#)
- application
 - json::JSONData, [21](#)
- assignApplication
 - json::JSONHandler, [23](#)
- assignCommand
 - json::JSONHandler, [23](#)
- assignEntries
 - json::JSONHandler, [24](#)
- assignEnvironmentVariable
 - json::JSONHandler, [25](#)
- assignHideShell
 - json::JSONHandler, [26](#)
- assignOutputFile
 - json::JSONHandler, [26](#)
- assignPathValue
 - json::JSONHandler, [26](#)
- batch, [11](#)
- batch::BatchCreator, [13](#)
 - BatchCreator, [14](#)
 - batchFile, [15](#)
 - createBatchFile, [14](#)
 - jsonData, [15](#)
 - writeApplication, [14](#)
 - writeCommands, [14](#)
 - writeEnvironmentVariables, [14](#)
 - writeHideShellEnd, [14](#)
 - writeHideShellStart, [14](#)
 - writePathValue, [14](#)
 - writeShell, [15](#)
- BatchCreator
 - batch::BatchCreator, [14](#)
- batchFile
 - batch::BatchCreator, [15](#)
- Bug List, [1](#)
- cliInterface, [11](#)
- cliInterface::CliHandler, [15](#)
- commands
 - json::JSONData, [21](#)
- createBatchFile
 - batch::BatchCreator, [14](#)
- createJSONData
 - json::JSONHandler, [27](#)
- data
 - json::JSONHandler, [30](#)
- environmentVariables
 - json::JSONData, [21](#)
- getApplication
 - json::JSONData, [18](#)
- getCommands
 - json::JSONData, [18](#)
- getEnvironmentVariables
 - json::JSONData, [18](#)
- getHideShell
 - json::JSONData, [18](#)
- getJSONData
 - json::JSONHandler, [28](#)
- getOptions
 - utils::StartupHandler, [32](#)
- getOutputFile
 - json::JSONData, [19](#)
- getPathValues
 - json::JSONData, [19](#)
- hideShell
 - json::JSONData, [21](#)
- initEasyLogging
 - utils::StartupHandler, [33](#)
- json, [11](#)
- json::JSONData, [15](#)
 - addCommand, [16](#)
 - addEnvironmentVariable, [17](#)
 - addPathValue, [17](#)
 - application, [21](#)
 - commands, [21](#)
 - environmentVariables, [21](#)
 - getApplication, [18](#)
 - getCommands, [18](#)
 - getEnvironmentVariables, [18](#)
 - getHideShell, [18](#)
 - getOutputFile, [19](#)
 - getPathValues, [19](#)
 - hideShell, [21](#)
 - outputfile, [21](#)
 - pathValues, [21](#)
 - setApplication, [19](#)
 - setHideShell, [20](#)
 - setOutputFile, [20](#)
 - suffixLength, [21](#)
- json::JSONHandler, [22](#)

- assignApplication, [23](#)
- assignCommand, [23](#)
- assignEntries, [24](#)
- assignEnvironmentVariable, [25](#)
- assignHideShell, [26](#)
- assignOutputFile, [26](#)
- assignPathValue, [26](#)
- createJSONData, [27](#)
- data, [30](#)
- getJSONData, [28](#)
- JSONHandler, [23](#)
- parseFile, [29](#)
- root, [30](#)
- jsonData
 - batch::BatchCreator, [15](#)
- JSONHandler
 - json::JSONHandler, [23](#)
- main
 - main.cpp, [42](#)
- main.cpp
 - main, [42](#)
- operator=
 - utils::StartupHandler, [33](#)
- outputfile
 - json::JSONData, [21](#)
- parseFile
 - json::JSONHandler, [29](#)
- pathValues
 - json::JSONData, [21](#)
- root
 - json::JSONHandler, [30](#)
- setApplication
 - json::JSONData, [19](#)
- setHideShell
 - json::JSONData, [20](#)
- setOutputFile
 - json::JSONData, [20](#)
- src/headers/BatchCreator.hpp, [35](#), [36](#)
- src/headers/CliHandler.hpp, [36](#), [37](#)
- src/headers/JSONData.hpp, [37](#), [38](#)
- src/headers/JSONHandler.hpp, [39](#), [40](#)
- src/headers/StartupHandler.hpp, [40](#), [41](#)
- src/main.cpp, [42](#), [43](#)
- src/sources/BatchCreator.cpp, [44](#), [45](#)
- src/sources/CliHandler.cpp, [46](#)
- src/sources/JSONData.cpp, [46](#), [47](#)
- src/sources/JSONHandler.cpp, [48](#)
- src/sources/StartupHandler.cpp, [49](#), [50](#)
- StartupHandler
 - utils::StartupHandler, [31](#)
- suffixLength
 - json::JSONData, [21](#)
- Todo List, [3](#)
- utils, [12](#)
 - verbose, [12](#)
- utils::StartupHandler, [30](#)
 - getOptions, [32](#)
 - initEasyLogging, [33](#)
 - operator=, [33](#)
 - StartupHandler, [31](#)
- verbose
 - utils, [12](#)
- writeApplication
 - batch::BatchCreator, [14](#)
- writeCommands
 - batch::BatchCreator, [14](#)
- writeEnvironmentVariables
 - batch::BatchCreator, [14](#)
- writeHideShellEnd
 - batch::BatchCreator, [14](#)
- writeHideShellStart
 - batch::BatchCreator, [14](#)
- writePathValue
 - batch::BatchCreator, [14](#)
- writeShell
 - batch::BatchCreator, [15](#)