JSON2Batch

0.2.2

Generated on Fri Apr 26 2024 14:13:56 for JSON2Batch by Doxygen 1.9.8

Fri Apr 26 2024 14:13:56

Chapter 1

JSON2Batch

1.1 JSON2Batch

JSON2Batch was developed during a project during our first and second semester of university. It generates batch files from JSON files, which can spawn terminals or applications, that run under certain parameters specified within the JSON file.

The project was carried out by **Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci**.

1.1.1 Table of Contents

1.1.2 Build Instructions

1.1.2.1 Linux

```
git clone https://github.com/DHBWProjectsIT23/JSON2Bat/!TODO
cmake -S . -B build --config Release
cmake --build build
```

UNIX Compiler Compatibility

The project has been tested with GCC version >= 10.5 and Clang version > 14.

1.1.2.2 Windows

@TODO Fix Windows

Windows Compiler Compatibility

1.1.2.3 Generating Documentation

If the *doxygen* executable is installed local documentation can be generated using:

```
git clone https://github.com/DHBWProjectsIT23/JSON2Bat/!TODO
cmake -S . -B build --config Release
cmake --build build --target doxygen_build
```

1.1.3 Documentation

The documentation for this project can be found here. A PDF version can be found [here]() and a short man page can be found [here]().

1.1.3.1 Project Structure

The project directory is structured as follows:

- assets > *Includes files, not directly related to the code*
- man > *Includes the man page*
- conf > *Includes files which will be configured by CMake*
- include > *Includes header files for external libraries*
- lib > *Includes source/binary files for external libraries*
- src > *Includes the source code for the project*
  - sources > *Includes all ".cpp" files*
  - include > *Includes all ".hpp" files*
  - main.cpp

1.1.4 External Libraries

1.1.4.1 easylogging++

The easylogging++ library is used for logging within the application. The configuration for the library is done via a logging file which can be found in conf/easylogging.in.conf. Cmake configures this file into the binary directory upon building. If the configuration file is removed, the application will no longer run.

1.1.4.2 LoggingWrapper

While easylogging++ is used for the logging back–end within the code there are little remains apart from the configuration. The logging and output of the application is done over a self written wrapper. Altough it is self written, due to it beeing not part of the project we consider it an external libraries. The wrapper is used to simplify parallel output to stdout and the logfile and also enables increased output to stdout for the verbose mode. A few macros are defined for use within the application:

- OUTPUT > *Outputs to stdout and the logfile*
- LOG_INFO > *By default only outputs to the logfile*
- LOG_WARNING > *Formats text and outputs to stdout and the logfile*
- LOG_ERROR > *Same as LOG_WARNING but in red and bold*

The macros can be used with streaming in the same way as std::cout would be used. Furthermore, some rudimentary performance tests showed, that the use of the wrapper, does not affect performance in comparison to using both std::cout and easylogging itself.

1.1.4.3 jsoncpp

For parsing the JSON files, the `jsoncpp` library is used. On UNIX system this library can simply be installed using the systems package manager (tested with WSL/Ubuntu and Arch). For Windows system a prebuild version is included - See Windows for more information.

1.1.5 License

The project is published under the Apache License V2.0. Check the [license file](LICENSE) for more information!

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Topic Documentation

7.1 StyleHelpers

Static variables to help with CLI styling.

Static variables to help with CLI styling.

A group of strings, that use escape sequences to easily style the command line interface on Unix systems. When compiling for Windows all of these strings will be empty, as escape sequences can't be used the same way.

Chapter 8

Namespace Documentation

8.1 cli Namespace Reference

Includes everything regarding the CLI.

Classes

- class CommandLineHandler

  Responsible for the Command Line Interface.

Variables

- static const struct option options [ ]

8.1.1 Detailed Description

Includes everything regarding the CLI.

This namespace includes all the code regarding the Command Line Interface. This includes the CommandLineHandler Class, the struct for the options and helpers for Styling.

See also

CommandLineHandler

options

StyleHelpers

8.1.2 Variable Documentation

8.1.2.1 options

```
const struct option cli::options[]  [static]
```

Initial value:

```
= {
    {"help", no_argument, nullptr, 'h'},
    {"version", no_argument, nullptr, 'v'},
    {"credits", no_argument, nullptr, 'c'},
    {"verbose", no_argument, nullptr, 0},
    {"outdir", required_argument, nullptr, 'o'},
    nullptr
}
```

Definition at line 111 of file CommandLineHandler.hpp.

8.2 config Namespace Reference

Namespace used for general project information.

Variables

- constexpr auto LOG_CONFIG
- constexpr auto EXECUTABLE_NAME = "json2batch"
- constexpr auto MAJOR_VERSION = "0"
- constexpr auto MINOR_VERSION = "2"
- constexpr auto PATCH_VERSION = "2"
- constexpr auto DESCRIPTION = "A simple tool to convert json to batch."
- constexpr auto PROJECT_NAME = "JSON2Batch"
- constexpr auto AUTHORS = "@AUTHORS"
- constexpr auto HOMEPAGE_URL

8.2.1 Detailed Description

Namespace used for general project information.

8.2.2 Variable Documentation

8.2.2.1 AUTHORS

```
constexpr auto config::AUTHORS = "@AUTHORS"  [inline], [constexpr]
```

Definition at line 34 of file config.hpp.

8.2.2.2 DESCRIPTION

```
constexpr auto config::DESCRIPTION = "A simple tool to convert json to batch."  [inline],
[constexpr]
```

Definition at line 32 of file config.hpp.

8.2.2.3 EXECUTABLE_NAME

```
constexpr auto config::EXECUTABLE_NAME = "json2batch"  [inline], [constexpr]
```

Definition at line 28 of file config.hpp.

8.2.2.4 HOMEPAGE_URL

```
constexpr auto config::HOMEPAGE_URL  [inline], [constexpr]
```

Initial value:

```
=
            "https://dhbwprojectsit23.github.io/JSON2Bat"
```

Definition at line 35 of file config.hpp.

8.2.2.5 LOG_CONFIG

```
constexpr auto config::LOG_CONFIG  [inline], [constexpr]
```

Initial value:

```
=
            "/home/simon/1_Coding/cpp/JsonToBat/build/Debug/config/easylogging.conf"
```

Definition at line 26 of file config.hpp.

8.2.2.6 MAJOR_VERSION

```
constexpr auto config::MAJOR_VERSION = "0"  [inline], [constexpr]
```

Definition at line 29 of file config.hpp.

8.2.2.7 MINOR_VERSION

```
constexpr auto config::MINOR_VERSION = "2"  [inline], [constexpr]
```

Definition at line 30 of file config.hpp.

8.2.2.8 PATCH_VERSION

```
constexpr auto config::PATCH_VERSION = "2"  [inline], [constexpr]
```

Definition at line 31 of file config.hpp.

8.2.2.9 PROJECT_NAME

```
constexpr auto config::PROJECT_NAME = "JSON2Batch"  [inline], [constexpr]
```

Definition at line 33 of file config.hpp.

8.3 exceptions Namespace Reference

Namespace used for customized exceptions.

Classes

- class CustomException

  Base class for all custom exceptions.
- class FailedToOpenFileException

  Exception for when a file can't be opened.
- class FileExistsException

  Exception for an already exisiting outputfile.
- class InvalidKeyException

  Exception for invalid keys.
- class InvalidTypeException

  Exception for invalid types.
- class InvalidValueException

  Exception for an ivalid (usually empty) value field.
- class MissingKeyException

  Exception for missing keys within entries.
- class MissingTypeException

  Exception for missing types of entries.
- class NoSuchDirException

  Exception for when a directory does not exist.
- class ParsingException

  Exception for syntax errors within the json file.
- class UnreachableCodeException

  Exception for when the application reaches code it shouldn't reach.

8.3.1 Detailed Description

Namespace used for customized exceptions.

8.4 parsing Namespace Reference

The namespace containing everything relevant to parsing.

Classes

- class FileData

  This class contains all data from the json file.
- class JsonHandler

  This file reads all data from the json file.
- class KeyValidator

  Validates keys of a Json::Value object.

8.4.1 Detailed Description

The namespace containing everything relevant to parsing.

This namespace contains all relevant classes to parsing the json file and creating the batch output.

See also

JsonHandler

FileData

KeyValidator

BatchCreator

8.5 utilities Namespace Reference

Includes all utilities.

Classes

- class Utils

  Responsible for utility function.

8.5.1 Detailed Description

Includes all utilities.

This namespace includes the Utils class with utility functions which can be used throughout the project.

See also

Utils

Chapter 9

Class Documentation

9.1 BatchCreator Class Reference

Creates a batch file from a FileData obeject.

`#include <BatchCreator.hpp>`

Public Member Functions

- BatchCreator (std::shared_ptr< parsing::FileData > fileData)
  Initializes the BatchCreator.
- std::shared_ptr< std::stringstream > getDataStream () const
  Returns the stringstream.

Private Member Functions

- void createBatch () const
  Creates the batch stream.
- void writeStart () const
  Wirtes the start of the batch file.
- void writeHideShell () const
  Writes the visibility of the shell.
- void writeCommands () const
  Writes the commands to be executed.
- void writeEnvVariables () const
  Set's environment variables.
- void writePathVariables () const
  Set's the path variables.
- void writeApp () const
  If an application is given, it is started at the end.
- void writeEnd () const
  Writes the end of the batch file.

Private Attributes

- std::shared_ptr< std::stringstream > dataStream
- std::shared_ptr< parsing::FileData > fileData

9.1.1 Detailed Description

Creates a batch file from a FileData obeject.

Uses a FileData object to create a string stream, which can then be streamed into a batch file.

See also

FileData

Definition at line 29 of file BatchCreator.hpp.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 BatchCreator()

```
BatchCreator::BatchCreator (
            std::shared_ptr< parsing::FileData > fileData )  [explicit]
```

Initializes the BatchCreator.

Creates a stringstream and calls the createBatch() function

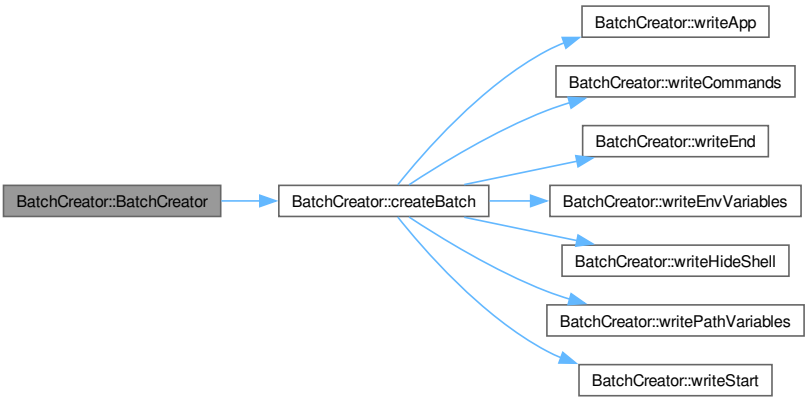
Parameters

filenData	A shared pointer to the FileData object

Definition at line 18 of file BatchCreator.cpp.

References createBatch(), and dataStream.

Here is the call graph for this function:



9.1.3 Member Function Documentation

9.1.3.1 createBatch()

```
void BatchCreator::createBatch ( ) const  [private]
```

Creates the batch stream.

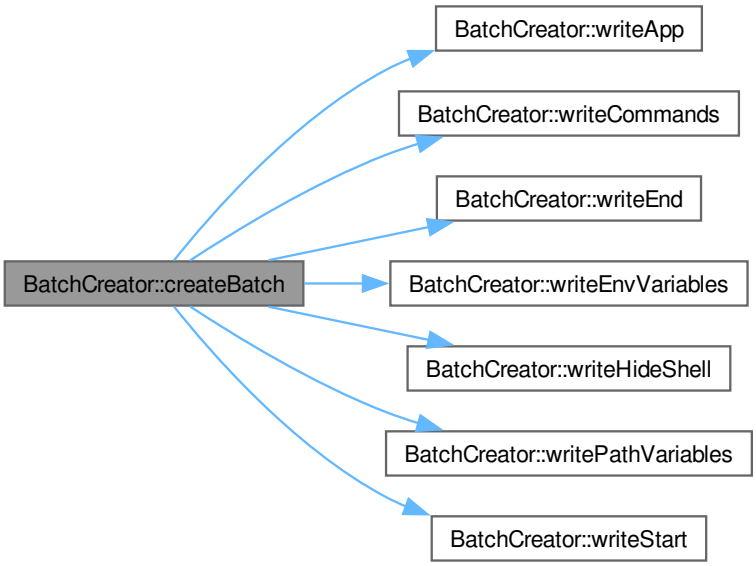
< FileData object

The method calls all necessary functions to create the stream for the batch file.

Definition at line 25 of file BatchCreator.cpp.

References writeApp(), writeCommands(), writeEnd(), writeEnvVariables(), writeHideShell(), writePathVariables(), and writeStart().

Here is the call graph for this function:



Here is the caller graph for this function:

9.1.3.2 getDataStream()

```
std::shared_ptr< std::stringstream > BatchCreator::getDataStream ( ) const  [inline]
```

Returns the stringstream.

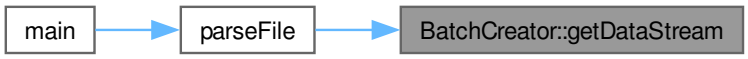
Returns

A shared pointer to the stringstream

Definition at line 46 of file BatchCreator.hpp.

References dataStream.

Here is the caller graph for this function:



9.1.3.3 writeApp()

```
void BatchCreator::writeApp ( ) const  [private]
```

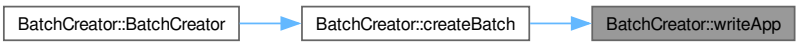
If an application is given, it is started at the end.

If the key "application" is given in the json file, the application is started at the end of the batch file.

Definition at line 80 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:



9.1.3.4 writeCommands()

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

Writes the commands to be executed.

Writes the commands to be executed from the FileData object. Those originiate from the "commands" entry in the json file

Definition at line 52 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:

9.1.3.5 writeEnd()

```
void BatchCreator::writeEnd ( ) const  [private]
```

Writes the end of the batch file.

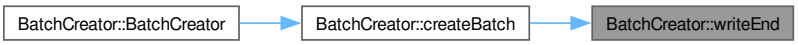
Writes the end of the batch file, which is always the same:

- @ECHO ON

Definition at line 95 of file BatchCreator.cpp.

References dataStream.

Here is the caller graph for this function:



9.1.3.6 writeEnvVariables()

```
void BatchCreator::writeEnvVariables ( ) const  [private]
```

Set's environment variables.

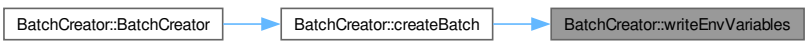
Set's the envirment variables for the batch. Those originiate from the "ENV" entry in the json file with the following syntax:

- Entry under "key" = Entry under "value"

Definition at line 61 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:



9.1.3.7 writeHideShell()

```
void BatchCreator::writeHideShell ( ) const  [private]
```

Writes the visibility of the shell.

This hides/shows the shell after the batch file has been executed

Definition at line 41 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:

9.1.3.8 writePathVariables()

```
void BatchCreator::writePathVariables ( ) const  [private]
```

Set's the path variables.

Set's the path variables for the batch. Those originiate from the "PATH" entry in the json file

Definition at line 69 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:

9.1.3.9 writeStart()

```
void BatchCreator::writeStart ( ) const  [private]
```

Wirtes the start of the batch file.

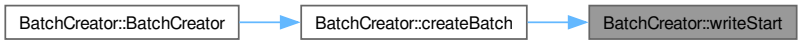
Writes the start of the batch file, which is always the same:

- setzt ECHO off
- startet cmd.exe

Definition at line 36 of file BatchCreator.cpp.

References dataStream.

Here is the caller graph for this function:



9.1.4 Member Data Documentation

9.1.4.1 dataStream

```
std::shared_ptr<std::stringstream> BatchCreator::dataStream  [private]
```

Definition at line 52 of file BatchCreator.hpp.

9.1.4.2 fileData

```
std::shared_ptr<parsing::FileData> BatchCreator::fileData  [private]
```

< stringstream for the batch file

Definition at line 54 of file BatchCreator.hpp.

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

- src/include/BatchCreator.hpp
- src/sources/BatchCreator.cpp

9.2 cli::CommandLineHandler Class Reference

Responsible for the Command Line Interface.

```
#include <CommandLineHandler.hpp>
```

Public Member Functions

- CommandLineHandler ()=delete

  The Constructor of the CommandLineHandler Class.
- ~CommandLineHandler ()=delete

  The Destructor of the CommandLineHandler Class.

Static Public Member Functions

- static void printHelp ()

  Prints the help message.
- static void printVersion ()

  Prints the version message.
- static void printCredits ()

  Prints the credits message.
- static std::tuple< std::optional< std::string >, std::vector< std::string > > parseArguments (int argc, char *argv[])

  Parses the Command Line Arguments.

9.2.1 Detailed Description

Responsible for the Command Line Interface.

This class is responsible for parsing the command line arguments, printing Help/Version/Credits messages and returning inputted files.

Author

Simon Blum

Date

2024-04-18

Version

0.1.5

See also

options

Definition at line 55 of file CommandLineHandler.hpp.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 CommandLineHandler()

```
cli::CommandLineHandler::CommandLineHandler ( )  [delete]
```

The Constructor of the CommandLineHandler Class.

Note

As all functions are static it should not be used and as such is deleted.

9.2.2.2 ~CommandLineHandler()

```
cli::CommandLineHandler::~CommandLineHandler ( )  [delete]
```

The Destructor of the CommandLineHandler Class.

Note

As all functions are static it should not be used and as such is deleted.

9.2.3 Member Function Documentation

9.2.3.1 parseArguments()

```
std::tuple< std::optional< std::string >, std::vector< std::string > > cli::CommandLine↩
Handler::parseArguments (
            int argc,
            char * argv[] )  [static]
```

Parses the Command Line Arguments.

This function uses the "getopt.h" library to parse all options given and then returns all files which are given as arguments.

Parameters

argc	The number of arguments given
argv	The arguments given

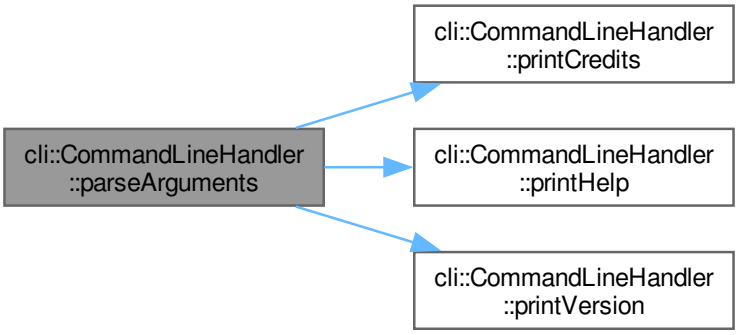
Returns

Returns a tuple containing the output directory and the files

Definition at line 68 of file CommandLineHandler.cpp.

References printCredits(), printHelp(), and printVersion().

Here is the call graph for this function:



Here is the caller graph for this function:

9.2.3.2 printCredits()

```
void cli::CommandLineHandler::printCredits ( )  [static]
```

Prints the credits message.

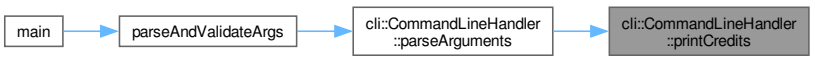
Note

This function ends the application.

Definition at line 50 of file CommandLineHandler.cpp.

References config::AUTHORS, config::DESCRIPTION, config::HOMEPAGE_URL, config::MAJOR_VERSION, config::MINOR_VERSION, config::PATCH_VERSION, and config::PROJECT_NAME.

Here is the caller graph for this function:



9.2.3.3 printHelp()

```
void cli::CommandLineHandler::printHelp ( )  [static]
```

Prints the help message.

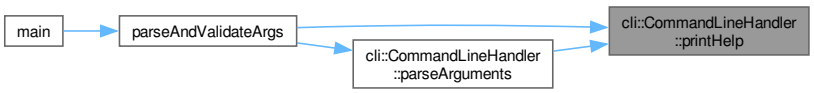
Note

This function ends the application.

Definition at line 22 of file CommandLineHandler.cpp.

References config::EXECUTABLE_NAME.

Here is the caller graph for this function:



9.2.3.4 printVersion()

```
void cli::CommandLineHandler::printVersion ( )  [static]
```

Prints the version message.

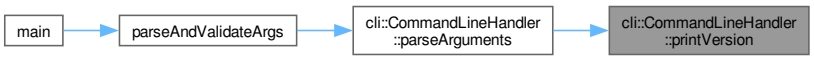
Note

This function ends the application.

Definition at line 44 of file CommandLineHandler.cpp.

References config::MAJOR_VERSION, config::MINOR_VERSION, config::PATCH_VERSION, and config::PROJECT_NAME.

Here is the caller graph for this function:



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

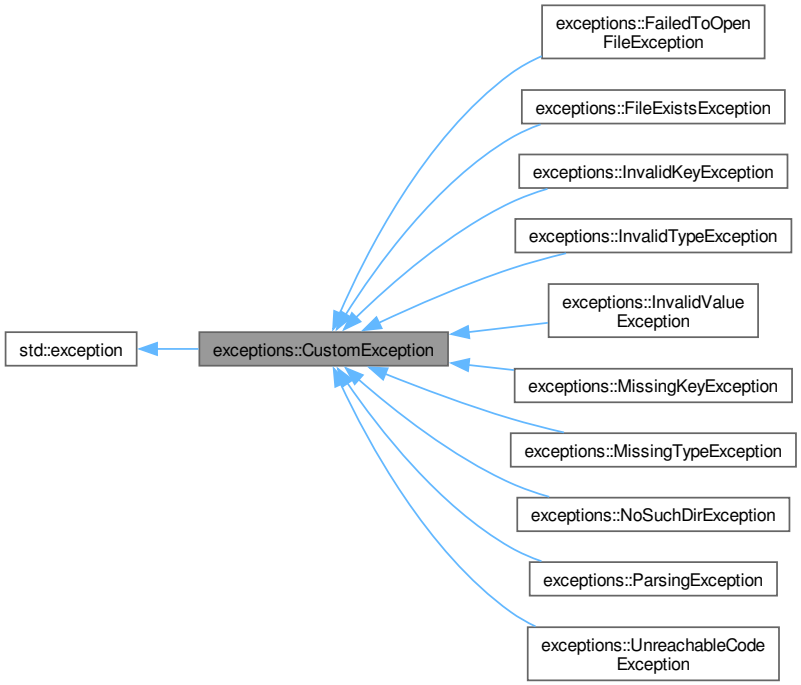
- src/include/CommandLineHandler.hpp
- src/sources/CommandLineHandler.cpp

9.3 exceptions::CustomException Class Reference

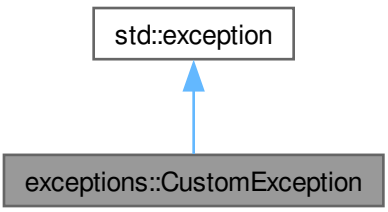
Base class for all custom exceptions.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::CustomException:



Collaboration diagram for exceptions::CustomException:



Public Member Functions

- const char ∗ what () const noexcept override

9.3.1 Detailed Description

Base class for all custom exceptions.

This class is the base class which is inherited by all custom exceptions. It can be used to catch all exceptions that are thrown by us.

See also

std::exception

Definition at line 35 of file Exceptions.hpp.

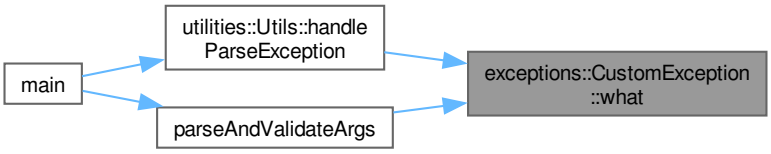
9.3.2 Member Function Documentation

9.3.2.1 what()

```
const char * exceptions::CustomException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 37 of file Exceptions.hpp.

Here is the caller graph for this function:



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

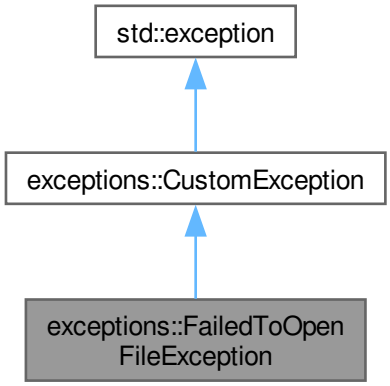
- src/include/Exceptions.hpp

9.4 exceptions::FailedToOpenFileException Class Reference

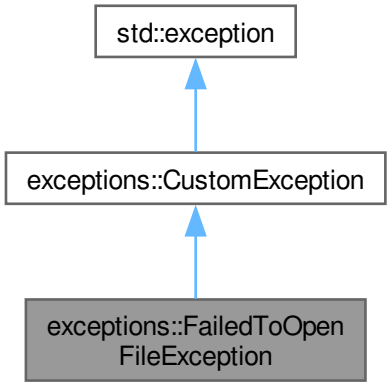
Exception for when a file can't be opened.

```
#include <Exceptions.hpp>
```

Inheritance diagram for exceptions::FailedToOpenFileException:



Collaboration diagram for exceptions::FailedToOpenFileException:



Public Member Functions

- FailedToOpenFileException (const std::string &file)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- std::string message

9.4.1 Detailed Description

Exception for when a file can't be opened.

Definition at line 259 of file Exceptions.hpp.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 FailedToOpenFileException()

```
exceptions::FailedToOpenFileException::FailedToOpenFileException (
            const std::string & file )  [inline], [explicit]
```

Definition at line 264 of file Exceptions.hpp.

References message.

9.4.3 Member Function Documentation

9.4.3.1 what()

```
const char * exceptions::FailedToOpenFileException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 268 of file Exceptions.hpp.

References message.

9.4.4 Member Data Documentation

9.4.4.1 message

```
std::string exceptions::FailedToOpenFileException::message  [private]
```

Definition at line 261 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

9.5 parsing::FileData Class Reference

This class contains all data from the json file.

```
#include <FileData.hpp>
```

Public Member Functions

- void setOutputFile (std::string &newOutputfile)

  Setter for this->outputfile.
- void setHideShell (bool newHideShell)

  Setter for this->hideshell.
- void setApplication (const std::string &newApplication)

  Setter for this->application.
- void addCommand (const std::string &command)

  Adds a given command to this->commands.
- void addEnvironmentVariable (const std::string &name, const std::string &value)

  Adds a given tuple to this->environmentVariables.
- void addPathValue (const std::string &pathValue)

  Add's a given value to this->pathValues.
- const std::string & getOutputFile () const

  Getter for this->outputfile.
- bool getHideShell () const

  Getter for this->hideShell.
- const std::optional< std::string > & getApplication () const

  Getter for this->application.
- const std::vector< std::string > & getCommands () const

  Getter for this->commands.
- const std::vector< std::tuple< std::string, std::string > > & getEnvironmentVariables () const

  Getter for this->environmentVariables.
- const std::vector< std::string > & getPathValues () const

  Getter for this->pathValues.

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

9.5.1 Detailed Description

This class contains all data from the json file.

The data from the json file is parsed by the JsonHandler and then assigned to the attributes of an instance of this class. This class also handles a part of the error handling.

Definition at line 31 of file FileData.hpp.

9.5.2 Member Function Documentation

9.5.2.1 addCommand()

```
void parsing::FileData::addCommand (
            const std::string & command )
```

Adds a given command to this->commands.

Makes sure, that the given command value is not empty and then add's it to the commands attribute.

Parameters

command	The command to be added

Exceptions

exceptions::InvalidValueException	

Definition at line 56 of file FileData.cpp.

References commands.

9.5.2.2 addEnvironmentVariable()

```
void parsing::FileData::addEnvironmentVariable (
            const std::string & name,
            const std::string & value )
```

Adds a given tuple to this->environmentVariables.

Makes sure that neither the key nor the value is empty and then adds a tuple with both values to the environment↩ Variables attribute

Parameters

name	The name of the env variable
value	The value of the env variable

Exceptions

exceptions::InvalidValueException	

Definition at line 67 of file FileData.cpp.

References environmentVariables.

9.5.2.3 addPathValue()

```
void parsing::FileData::addPathValue (
            const std::string & pathValue )
```

Add's a given value to this->pathValues.

Makes sure that the given value is not empty and then assigns it to the given pathValues attribute

Parameters

pathValue	The value to be added

Exceptions

exceptions::InvalidValueException	

Definition at line 83 of file FileData.cpp.

References pathValues.

9.5.2.4 getApplication()

```
const std::optional< std::string > & parsing::FileData::getApplication ( ) const  [inline]
```

Getter for this->application.

Returns

The assigned application

Definition at line 121 of file FileData.hpp.

References application.

9.5.2.5 getCommands()

```
const std::vector< std::string > & parsing::FileData::getCommands ( ) const  [inline]
```

Getter for this->commands.

Returns

The vector of assigned commands

Definition at line 129 of file FileData.hpp.

References commands.

9.5.2.6 getEnvironmentVariables()

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

Getter for this->environmentVariables.

Returns

The vector of assigned env variables

Definition at line 138 of file FileData.hpp.

References environmentVariables.

9.5.2.7 getHideShell()

```
bool parsing::FileData::getHideShell ( ) const  [inline]
```

Getter for this->hideShell.

Returns

The assigned value for hideshell

Definition at line 113 of file FileData.hpp.

References hideShell.

9.5.2.8 getOutputFile()

```
const std::string & parsing::FileData::getOutputFile ( ) const  [inline]
```

Getter for this->outputfile.

Returns

The assigned outputfile

Definition at line 105 of file FileData.hpp.

References outputfile.

9.5.2.9 getPathValues()

```
const std::vector< std::string > & parsing::FileData::getPathValues ( ) const  [inline]
```

Getter for this->pathValues.

Returns

The vector of assigned pathValues

Definition at line 146 of file FileData.hpp.

References pathValues.

9.5.2.10 setApplication()

```
void parsing::FileData::setApplication (
            const std::string & newApplication )
```

Setter for this->application.

Set's the application attribute. Return's if the given string is empty.

Parameters

newApplication	THe application to be set

Definition at line 46 of file FileData.cpp.

References application.

9.5.2.11 setHideShell()

```
void parsing::FileData::setHideShell (
            bool newHideShell )  [inline]
```

Setter for this->hideshell.

Parameters

newHideShell	The hideshell value to be set

Definition at line 49 of file FileData.hpp.

References hideShell.

9.5.2.12 setOutputFile()

```
void parsing::FileData::setOutputFile (
            std::string & newOutputfile )
```

Setter for this->outputfile.

Checks that neither the given string is empty, nor that the outputfile is already set and then assigns the newOutputfile to the instance.

Parameters

newOutputfile	The outputfile to be set

Exceptions

exceptions::InvalidValueException	

Definition at line 18 of file FileData.cpp.

References outputfile.

9.5.3 Member Data Documentation

9.5.3.1 application

```
std::optional<std::string> parsing::FileData::application  [private]
```

Definition at line 153 of file FileData.hpp.

9.5.3.2 commands

```
std::vector<std::string> parsing::FileData::commands  [private]
```

Definition at line 154 of file FileData.hpp.

9.5.3.3 environmentVariables

```
std::vector<std::tuple<std::string, std::string> > parsing::FileData::environmentVariables
[private]
```

Definition at line 156 of file FileData.hpp.

9.5.3.4 hideShell

```
bool parsing::FileData::hideShell  [private]
```

Definition at line 152 of file FileData.hpp.

9.5.3.5 outputfile

```
std::string parsing::FileData::outputfile  [private]
```

Definition at line 151 of file FileData.hpp.

9.5.3.6 pathValues

```
std::vector<std::string> parsing::FileData::pathValues  [private]
```

Definition at line 157 of file FileData.hpp.

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

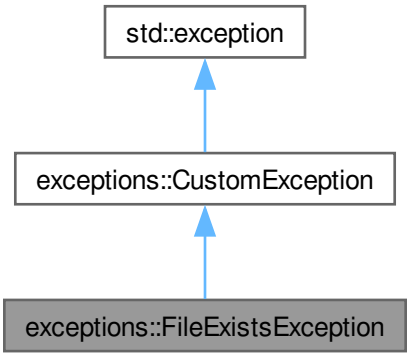
- src/include/FileData.hpp
- src/sources/FileData.cpp

9.6 exceptions::FileExistsException Class Reference

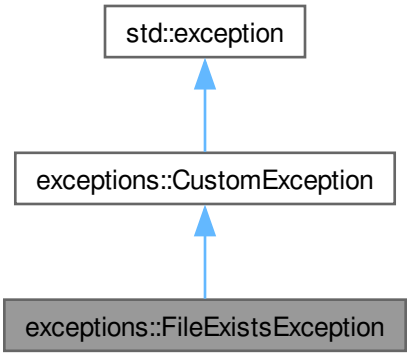
Exception for an already exisiting outputfile.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::FileExistsException:



Collaboration diagram for exceptions::FileExistsException:



Public Member Functions

- FileExistsException (const std::string &file)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- const std::string file
- std::string message

9.6.1 Detailed Description

Exception for an already exisiting outputfile.

Definition at line 74 of file Exceptions.hpp.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 FileExistsException()

```
exceptions::FileExistsException::FileExistsException (
            const std::string & file )  [inline], [explicit]
```

Note

I planned to use std::format, however it seems that the required Compiler Version is not yet available in the stable Ubuntu Repo!

Definition at line 80 of file Exceptions.hpp.

References file, and message.

9.6.3 Member Function Documentation

9.6.3.1 what()

```
const char * exceptions::FileExistsException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 92 of file Exceptions.hpp.

References message.

9.6.4 Member Data Documentation

9.6.4.1 file

```
const std::string exceptions::FileExistsException::file  [private]
```

Definition at line 76 of file Exceptions.hpp.

9.6.4.2 message

```
std::string exceptions::FileExistsException::message  [private]
```

Definition at line 77 of file Exceptions.hpp.

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

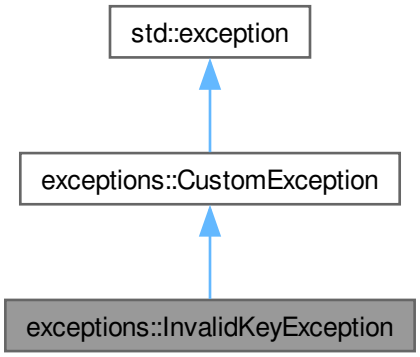
- src/include/Exceptions.hpp

9.7 exceptions::InvalidKeyException Class Reference

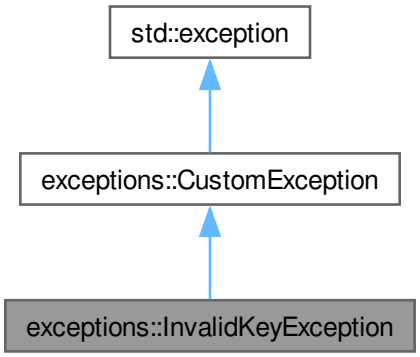
Exception for invalid keys.

```
#include <Exceptions.hpp>
```

Inheritance diagram for exceptions::InvalidKeyException:



Collaboration diagram for exceptions::InvalidKeyException:



Public Member Functions

- InvalidKeyException (const std::vector< std::tuple< int, std::string > > &keys)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- std::string message = "Invalid key found!"

9.7.1 Detailed Description

Exception for invalid keys.

This exception is thrown when a key is found within the json file, that is not part of the valid keys. It will also display the name and the line of the invalid key.

See also

parsing::KeyValidator::validKeys

parsing::KeyValidator::validEntryKeys

Definition at line 135 of file Exceptions.hpp.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 InvalidKeyException()

```
exceptions::InvalidKeyException::InvalidKeyException (
            const std::vector< std::tuple< int, std::string > > & keys )  [inline], [explicit]
```

Definition at line 140 of file Exceptions.hpp.

References message.

9.7.3 Member Function Documentation

9.7.3.1 what()

```
const char * exceptions::InvalidKeyException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 149 of file Exceptions.hpp.

References message.

9.7.4 Member Data Documentation

9.7.4.1 message

```
std::string exceptions::InvalidKeyException::message = "Invalid key found!"  [private]
```

Definition at line 137 of file Exceptions.hpp.

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

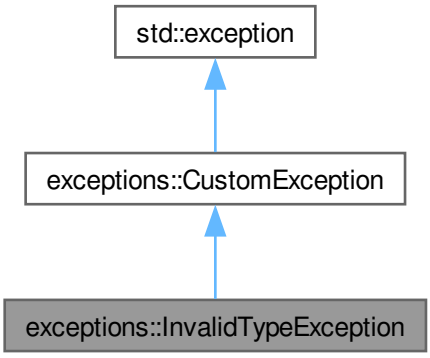
- src/include/Exceptions.hpp

9.8 exceptions::InvalidTypeException Class Reference

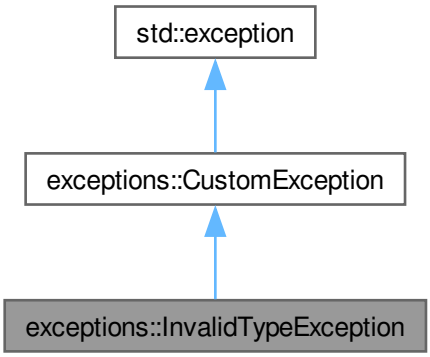
Exception for invalid types.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::InvalidTypeException:



Collaboration diagram for exceptions::InvalidTypeException:



Public Member Functions

- InvalidTypeException (const std::string &type, int line)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- const std::string type
- std::string message

9.8.1 Detailed Description

Exception for invalid types.

This exception is thrown when the value of the "type" field within the entries is invalid (not "EXE", "PATH", "ENV"). It also prints the type and the line of the invalid type.

Definition at line 162 of file Exceptions.hpp.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 InvalidTypeException()

```
exceptions::InvalidTypeException::InvalidTypeException (
            const std::string & type,
            int line )  [inline]
```

Note

I planned to use std::format, however it seems that the required Compiler Version is not yet available in the stable Ubuntu Repo!

Definition at line 168 of file Exceptions.hpp.

References message, and type.

9.8.3 Member Function Documentation

9.8.3.1 what()

```
const char * exceptions::InvalidTypeException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 179 of file Exceptions.hpp.

References message.

9.8.4 Member Data Documentation

9.8.4.1 message

```
std::string exceptions::InvalidTypeException::message  [private]
```

Definition at line 165 of file Exceptions.hpp.

9.8.4.2 type

```
const std::string exceptions::InvalidTypeException::type  [private]
```

Definition at line 164 of file Exceptions.hpp.

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

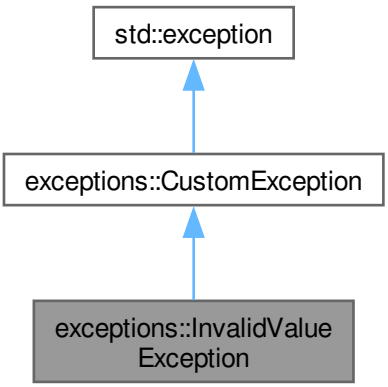
- src/include/Exceptions.hpp

9.9 exceptions::InvalidValueException Class Reference

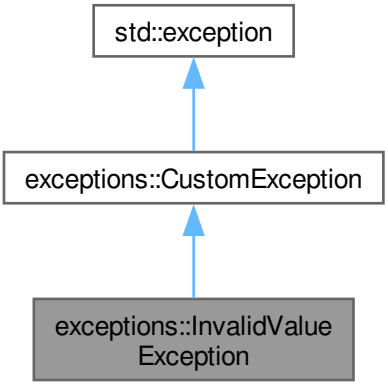
Exception for an ivalid (usually empty) value field.

```
#include <Exceptions.hpp>
```

Inheritance diagram for exceptions::InvalidValueException:



Collaboration diagram for exceptions::InvalidValueException:



Public Member Functions

- InvalidValueException (const std::string &key, const std::string &issue)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- const std::string key
- std::string message

9.9.1 Detailed Description

Exception for an ivalid (usually empty) value field.

Definition at line 101 of file Exceptions.hpp.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 InvalidValueException()

```
exceptions::InvalidValueException::InvalidValueException (
            const std::string & key,
            const std::string & issue )  [inline]
```

Note

I planned to use std::format, however it seems that the required Compiler Version is not yet available in the stable Ubuntu Repo!

Definition at line 107 of file Exceptions.hpp.

References key, and message.

9.9.3 Member Function Documentation

9.9.3.1 what()

```
const char * exceptions::InvalidValueException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 119 of file Exceptions.hpp.

References message.

9.9.4 Member Data Documentation

9.9.4.1 key

```
const std::string exceptions::InvalidValueException::key  [private]
```

Definition at line 103 of file Exceptions.hpp.

9.9.4.2 message

```
std::string exceptions::InvalidValueException::message  [private]
```

Definition at line 104 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

9.10 parsing::JsonHandler Class Reference

This file reads all data from the json file.

```
#include <JsonHandler.hpp>
```

Public Member Functions

- JsonHandler ()
  Constructor without arguments.
- JsonHandler (const std::string &filename)
  The constructor.
- std::shared_ptr< FileData > getFileData ()
  Retrieve the data from the json file.

Private Member Functions

- void assignOutputFile () const
  Assigns the outputfile to this->data.
- void assignHideShell () const
  Assigns the hideshell value to this->data.
- void assignApplication () const
  Assigns application to this->data.
- void assignEntries () const
  Assigns entries to this->data.
- void assignCommand (const Json::Value &entry) const
  Assigns an command to this->data.
- void assignEnvironmentVariable (const Json::Value &entry) const
  Assigns an environmentVariable to this->data.
- void assignPathValue (const Json::Value &entry) const
  Assigns a path value to this->data.
- std::shared_ptr< FileData > createFileData ()
  Creates the FileData instance.

Static Private Member Functions

- static std::shared_ptr< Json::Value > parseFile (const std::string &filename)
  Parses the given json file.

Private Attributes

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

9.10.1 Detailed Description

This file reads all data from the json file.

This file uses the jsoncpp library to parse all data from a json file, validate it to some degree.

See also

https://github.com/open-source-parsers/jsoncpp

Definition at line 47 of file JsonHandler.hpp.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 JsonHandler() [1/2]

```
parsing::JsonHandler::JsonHandler ( )  [inline]
```

Constructor without arguments.

This constructor can be used to initialise an instance in an outer scope and then assign it values from an inner scope.

Definition at line 55 of file JsonHandler.hpp.

9.10.2.2 JsonHandler() [2/2]

```
parsing::JsonHandler::JsonHandler (
            const std::string & filename )  [explicit]
```

The constructor.

This constructor calls this->parseFile() when called.

Parameters

filename	Name of the json file

Definition at line 20 of file JsonHandler.cpp.

References parseFile(), and root.

Here is the call graph for this function:

9.10.3 Member Function Documentation

9.10.3.1 assignApplication()

```
void parsing::JsonHandler::assignApplication ( ) const  [private]
```

Assigns application to this->data.

Retrieves the value of the application key from Json::Value this->root and defaults to an empty string.

Definition at line 75 of file JsonHandler.cpp.

References data, and root.

Here is the caller graph for this function:

9.10.3.2 assignCommand()

```
void parsing::JsonHandler::assignCommand (
            const Json::Value & entry ) const  [private]
```

Assigns an command to this->data.

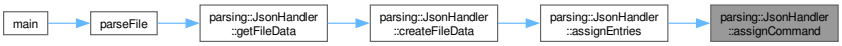
Parameters

entry	The entry with the command

Definition at line 107 of file JsonHandler.cpp.

References data.

Here is the caller graph for this function:



9.10.3.3 assignEntries()

```
void parsing::JsonHandler::assignEntries ( ) const  [private]
```

Assigns entries to this->data.

Goes through each of the entries from Json::Value this->root and calls the relevant method depending on it's type. All "type" keys should be valid by this point.

Parameters

entry	Json::Value containing an array with entries

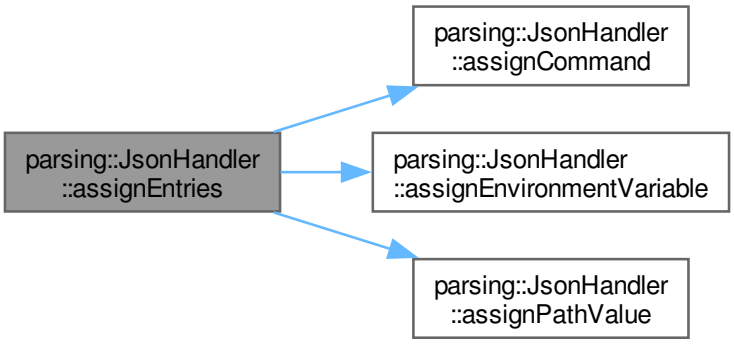
Exceptions

exceptions::UnreachableCodeException	

Definition at line 80 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:

9.10.3.4 assignEnvironmentVariable()

```
void parsing::JsonHandler::assignEnvironmentVariable (
            const Json::Value & entry ) const  [private]
```

Assigns an environmentVariable to this->data.

Parameters

entry	The entry with the environmentVariable

Definition at line 112 of file JsonHandler.cpp.

References data.

Here is the caller graph for this function:

9.10.3.5 assignHideShell()

```
void parsing::JsonHandler::assignHideShell ( ) const  [private]
```

Assigns the hideshell value to this->data.

Retrieves the value of the hideshell key from Json::Value this->root and defaults to negative.

Definition at line 69 of file JsonHandler.cpp.

References data, and root.

Here is the caller graph for this function:

9.10.3.6 assignOutputFile()

```
void parsing::JsonHandler::assignOutputFile ( ) const  [private]
```

Assigns the outputfile to this->data.

Retrieves the outputfile from Json::Value this->root and makes sure, that the file doesn't already exist.

Exceptions

exceptions::FileExistsException	

Definition at line 63 of file JsonHandler.cpp.

References data, and root.

Here is the caller graph for this function:

9.10.3.7 assignPathValue()

```
void parsing::JsonHandler::assignPathValue (
            const Json::Value & entry ) const  [private]
```

Assigns a path value to this->data.

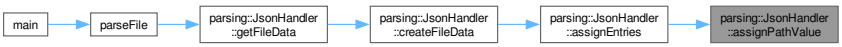
Parameters

entry	The entry with the path value

Definition at line 119 of file JsonHandler.cpp.

References data.

Here is the caller graph for this function:



9.10.3.8 createFileData()

```
std::shared_ptr< FileData > parsing::JsonHandler::createFileData ( )  [private]
```

Creates the FileData instance.

Instantiates the FileData instance, calls all nessecary functions and returns a shared pointer to it.

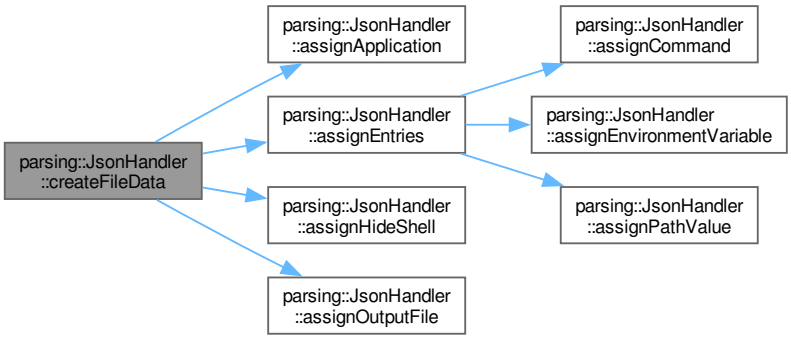
Returns

Pointer to the created instance of FileData

Definition at line 53 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:

9.10.3.9 getFileData()

```
std::shared_ptr< FileData > parsing::JsonHandler::getFileData ( )
```

Retrieve the data from the json file.

This method calls this->createFileData() needed to retrieve the values from the Json::Value this->root and then returns a shared pointer to the created FileData object.

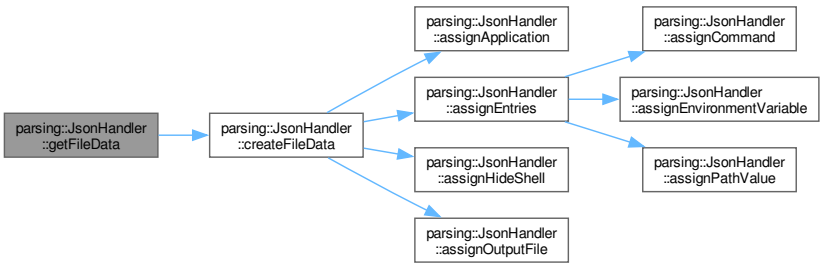
Returns

Pointer to the FileData Object with the parsed data from json

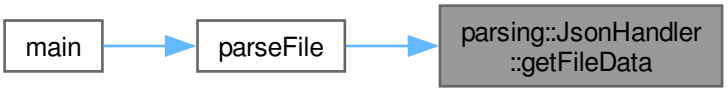
Definition at line 48 of file JsonHandler.cpp.

References createFileData().

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.3.10 parseFile()

```
std::shared_ptr< Json::Value > parsing::JsonHandler::parseFile (
            const std::string & filename )  [static], [private]
```

Parses the given json file.

This method first creates a new Json::Value instance and then tries to parse the given json file. It then validates the keys of the instance using the KeyValidator class.

Parameters

filename	The name of the file wich should be parsed

Returns

A shared pointer to the Json::Value instance

See also

KeyValidator::validateKeys()

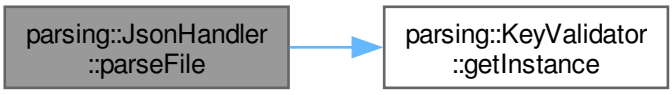
Exceptions

exceptions::ParsingException	
exceptions::InvalidKeyException	

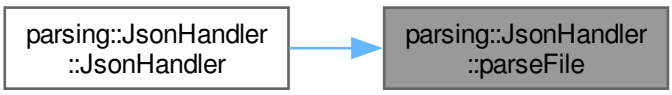
Definition at line 25 of file JsonHandler.cpp.

References parsing::KeyValidator::getInstance().

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.4 Member Data Documentation

9.10.4.1 data

```
std::shared_ptr<FileData> parsing::JsonHandler::data  [private]
```

Definition at line 155 of file JsonHandler.hpp.

9.10.4.2 root

```
std::shared_ptr<Json::Value> parsing::JsonHandler::root  [private]
```

Definition at line 154 of file JsonHandler.hpp.

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

- src/include/JsonHandler.hpp
- src/sources/JsonHandler.cpp

9.11 parsing::KeyValidator Class Reference

Validates keys of a Json::Value object.

```
#include <KeyValidator.hpp>
```

Public Member Functions

- std::vector< std::tuple< int, std::string > > validateKeys (const Json::Value &root, const std::string &filename)

  Validate keys off a Json::Value object.

Static Public Member Functions

- static KeyValidator & getInstance ()

  Get the instance of this class.

Private Member Functions

- std::vector< std::tuple< int, std::string > > getWrongKeys (const Json::Value &root, const std::string &filename) const

  Retrieve the wrong keys from a Json::Value object.
- void validateTypes (const std::string &filename, const Json::Value &entry, const std::unordered_set< std::string > &entryKeys)

  Validates types from the entries array.
- std::vector< std::tuple< int, std::string > > validateEntries (const std::string &filename, const std::unordered_set< std::string > &entryKeys) const

  Validates that keys within the entries array are valid.

Static Private Member Functions

- static std::optional< int > getUnknownKeyLine (const std::string &filename, const std::string &wrongKey)

  Get the line of an unknown key.

Private Attributes

- std::unordered_set< std::string > validKeys
- std::unordered_set< std::string > validEntryKeys
- std::unordered_map< std::string_view, std::vector< std::string > > typeToKeys

9.11.1 Detailed Description

Validates keys of a Json::Value object.

This class is singleton. That way when multiple files are parsed with the application, the maps for valid keys and the set for the type entries field only have to be allocated once when parsing multiple files.

Definition at line 30 of file KeyValidator.hpp.

9.11.2 Member Function Documentation

9.11.2.1 getInstance()

```
KeyValidator & parsing::KeyValidator::getInstance ( )  [static]
```

Get the instance of this class.

Returns

Reference to the instance of this class

Definition at line 20 of file KeyValidator.cpp.

Here is the caller graph for this function:

9.11.2.2 getUnknownKeyLine()

```
std::optional< int > parsing::KeyValidator::getUnknownKeyLine (
            const std::string & filename,
            const std::string & wrongKey )  [static], [private]
```

Get the line of an unknown key.

This method goes through each line of the given file and checks if the line contains the given key. Returns std::nullopt if the file can't be opened or the key was not found.

Parameters

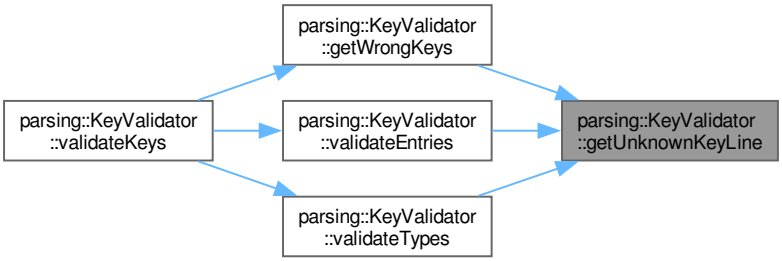
filename	The filename which should contain the key
wrongKey	The key to be searched for

Returns

The line of the key, if it was found

Definition at line 124 of file KeyValidator.cpp.

Here is the caller graph for this function:



9.11.2.3 getWrongKeys()

```
std::vector< std::tuple< int, std::string > > parsing::KeyValidator::getWrongKeys (
            const Json::Value & root,
            const std::string & filename ) const  [private]
```

Retrieve the wrong keys from a Json::Value object.

This method goes through each key of the Json::Value object and makes sure it's valid.

Parameters

root	The Json::Value object to be validated.
filename	The filename from which 'root' is from.

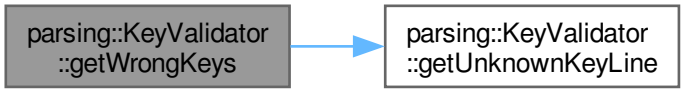
Returns

A vector with tuples, containing the line and name of invalid types.

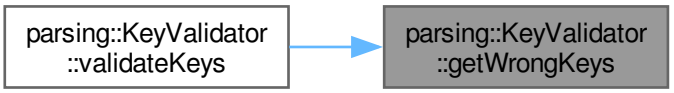
Definition at line 49 of file KeyValidator.cpp.

References getUnknownKeyLine(), and validKeys.

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.2.4 validateEntries()

```
std::vector< std::tuple< int, std::string > > parsing::KeyValidator::validateEntries (
            const std::string & filename,
            const std::unordered_set< std::string > & entryKeys ) const  [private]
```

Validates that keys within the entries array are valid.

This mehthod goes through each of the entries, and validates, that the keys are part of the validEntryKeys attribute.

Parameters

filename	The filename from which the entries are from
entryKeys	The keys of the entries

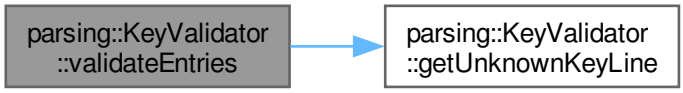
Returns

A vector with tuples, containing the line and name of invalid entrie keys

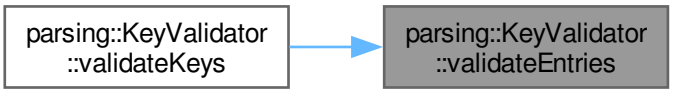
Definition at line 71 of file KeyValidator.cpp.

References getUnknownKeyLine(), and validEntryKeys.

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.2.5 validateKeys()

```
std::vector< std::tuple< int, std::string > > parsing::KeyValidator::validateKeys (
            const Json::Value & root,
            const std::string & filename )
```

Validate keys off a Json::Value object.

This method goes through the MemberNames of a Json::Value object and validates, that they are part of the valid↩ Key attribute. It calls the nessecary methods to validate the keys within the entries array.

Parameters

root	The Json::Value object to be validated.
filename	The filename from which 'root' is from.

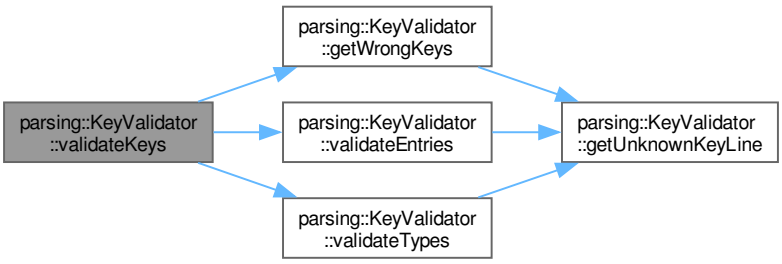
Returns

A vector with tuples, containing the line and name of invalid types.

Definition at line 26 of file KeyValidator.cpp.

References getWrongKeys(), validateEntries(), and validateTypes().

Here is the call graph for this function:



9.11.2.6 validateTypes()

```
void parsing::KeyValidator::validateTypes (
            const std::string & filename,
            const Json::Value & entry,
            const std::unordered_set< std::string > & entryKeys )  [private]
```

Validates types from the entries array.

This method goes makes sure, that the type of the given entry is valid and that it contains it's necessary keys. It will throw an exception if the type is missing, if the type is invalid or if the type is missing a key.

Note

Unnecessary keys within a type entry, don't cause an exception and are ignored.

Parameters

filename	The filename from which 'entry' is from
entry	The entry to be validated
entryKeys	The keys of the entry

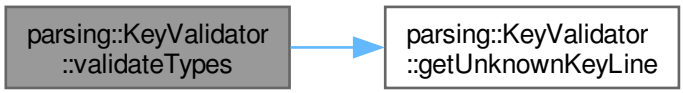
Exceptions

exceptions::MissingTypeException	
exceptions::InvalidTypeException	
exceptions::MissingKeyException	

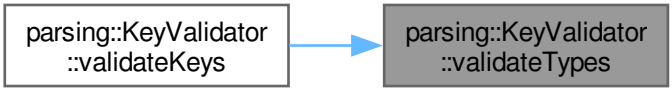
Definition at line 92 of file KeyValidator.cpp.

References getUnknownKeyLine(), and typeToKeys.

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3 Member Data Documentation

9.11.3.1 typeToKeys

```
std::unordered_map<std::string_view, std::vector<std::string> > parsing::KeyValidator::type↩
ToKeys  [private]
```

Initial value:

```
= {
        {"EXE", {"command"}}, {"PATH", {"path"}}, {"ENV", {"key", "value"}}
    }
```

Note

Changed from if/else clause within function to map in 0.2.1

Definition at line 144 of file KeyValidator.hpp.

9.11.3.2 validEntryKeys

```
std::unordered_set<std::string> parsing::KeyValidator::validEntryKeys  [private]
```

Initial value:

```
= {"type", "key", "value",
        "path", "command"
    }
```

Note

Changed from vector to unordered_set in 0.2.1 - as this shoud improve lookup performance from O(n) to O(1)

Definition at line 137 of file KeyValidator.hpp.

9.11.3.3 validKeys

```
std::unordered_set<std::string> parsing::KeyValidator::validKeys  [private]
```

Initial value:

```
= {"outputfile", "hideshell",
        "entries", "application"
    }
```

Note

Changed from vector to unordered_set in 0.2.1 - as this shoud improve lookup performance from O(n) to O(1)

Definition at line 130 of file KeyValidator.hpp.

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

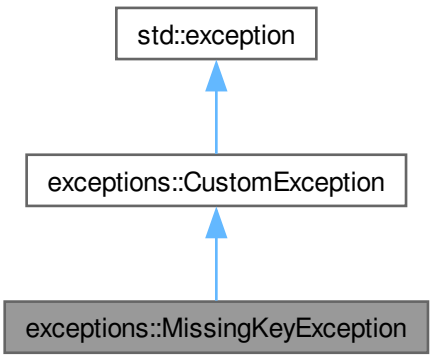
- src/include/KeyValidator.hpp
- src/sources/KeyValidator.cpp

9.12 exceptions::MissingKeyException Class Reference

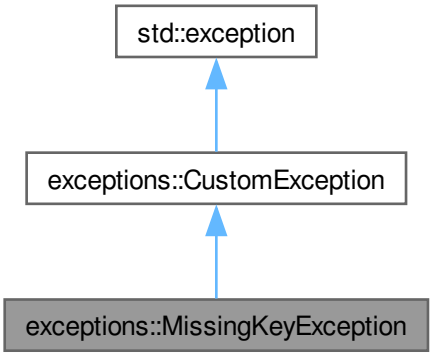
Exception for missing keys within entries.

```
#include <Exceptions.hpp>
```

Inheritance diagram for exceptions::MissingKeyException:



Collaboration diagram for exceptions::MissingKeyException:



Public Member Functions

- MissingKeyException (const std::string &key, const std::string &type)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- std::string message
- std::string type
- std::string key

9.12.1 Detailed Description

Exception for missing keys within entries.

This exception is thrown when a key (such as "path" or "command") is missing from an entry. It also prints the type and which key it is missing.

Definition at line 191 of file Exceptions.hpp.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 MissingKeyException()

```
exceptions::MissingKeyException::MissingKeyException (
            const std::string & key,
            const std::string & type )  [inline]
```

Note

I planned to use std::format, however it seems that the required Compiler Version is not yet available in the stable Ubuntu Repo!

Definition at line 198 of file Exceptions.hpp.

References key, message, and type.

9.12.3 Member Function Documentation

9.12.3.1 what()

```
const char * exceptions::MissingKeyException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 210 of file Exceptions.hpp.

References message.

9.12.4 Member Data Documentation

9.12.4.1 key

```
std::string exceptions::MissingKeyException::key  [private]
```

Definition at line 195 of file Exceptions.hpp.

9.12.4.2 message

```
std::string exceptions::MissingKeyException::message  [private]
```

Definition at line 193 of file Exceptions.hpp.

9.12.4.3 type

```
std::string exceptions::MissingKeyException::type  [private]
```

Definition at line 194 of file Exceptions.hpp.

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

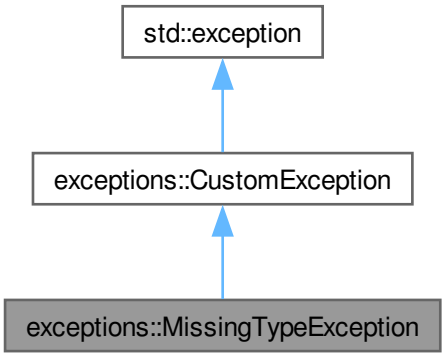
- src/include/Exceptions.hpp

9.13 exceptions::MissingTypeException Class Reference

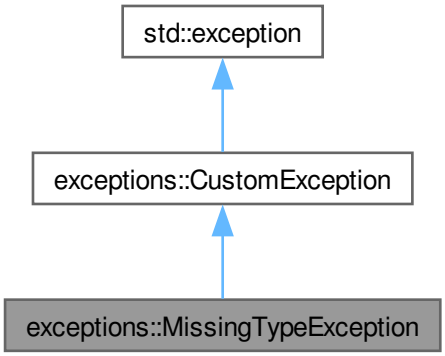
Exception for missing types of entries.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::MissingTypeException:



Collaboration diagram for exceptions::MissingTypeException:



Public Member Functions

- MissingTypeException ()
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- std::string message = "Missing \"type\" key for at least one entry!"

9.13.1 Detailed Description

Exception for missing types of entries.

This exception is thrown, when an entry is missing it's "type" key.

Definition at line 221 of file Exceptions.hpp.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 MissingTypeException()

```
exceptions::MissingTypeException::MissingTypeException ( )  [inline]
```

Definition at line 226 of file Exceptions.hpp.

References message.

9.13.3 Member Function Documentation

9.13.3.1 what()

```
const char * exceptions::MissingTypeException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 229 of file Exceptions.hpp.

References message.

9.13.4 Member Data Documentation

9.13.4.1 message

```
std::string exceptions::MissingTypeException::message = "Missing \"type\" key for at least one
entry!"  [private]
```

Definition at line 223 of file Exceptions.hpp.

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

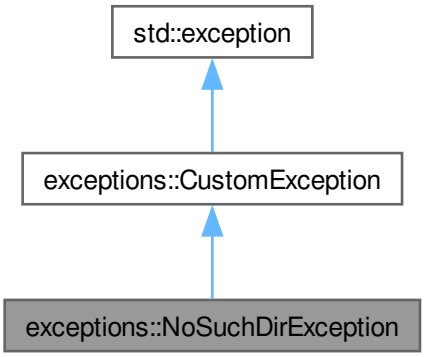
- src/include/Exceptions.hpp

9.14 exceptions::NoSuchDirException Class Reference

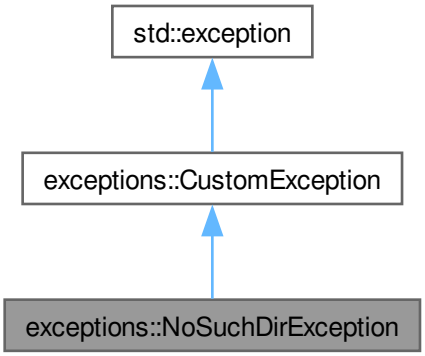
Exception for when a directory does not exist.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::NoSuchDirException:



Collaboration diagram for exceptions::NoSuchDirException:



Public Member Functions

- NoSuchDirException (const std::string &dir)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- std::string message

9.14.1 Detailed Description

Exception for when a directory does not exist.

Definition at line 277 of file Exceptions.hpp.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 NoSuchDirException()

```
exceptions::NoSuchDirException::NoSuchDirException (
            const std::string & dir )  [inline], [explicit]
```

Definition at line 282 of file Exceptions.hpp.

References message.

9.14.3 Member Function Documentation

9.14.3.1 what()

```
const char * exceptions::NoSuchDirException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 286 of file Exceptions.hpp.

References message.

9.14.4 Member Data Documentation

9.14.4.1 message

```
std::string exceptions::NoSuchDirException::message  [private]
```

Definition at line 279 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

9.15 options Struct Reference

The struct containing all possible options.

`#include <CommandLineHandler.hpp>`

9.15.1 Detailed Description

The struct containing all possible options.

This struct contains all long and short options which can be used and will be parsed using "getopt.h"

See also

CommandLineHandler

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

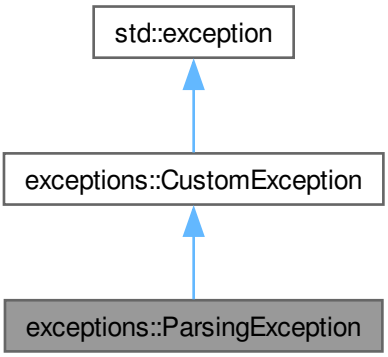
- src/include/CommandLineHandler.hpp

9.16 exceptions::ParsingException Class Reference

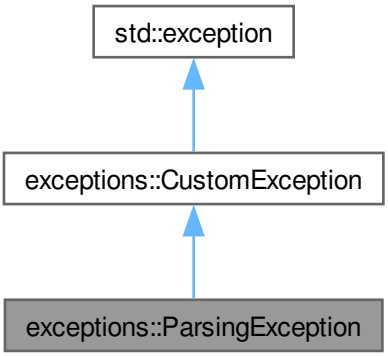
Exception for syntax errors within the json file.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::ParsingException:



Collaboration diagram for exceptions::ParsingException:



Public Member Functions

- ParsingException (const std::string &file)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- const std::string file
- std::string message

9.16.1 Detailed Description

Exception for syntax errors within the json file.

Definition at line 46 of file Exceptions.hpp.

9.16.2 Constructor & Destructor Documentation

9.16.2.1 ParsingException()

```
exceptions::ParsingException::ParsingException (
            const std::string & file )  [inline], [explicit]
```

Note

I planned to use std::format, however it seems that the required Compiler Version is not yet available in the stable Ubuntu Repo!

Definition at line 52 of file Exceptions.hpp.

References file, and message.

9.16.3 Member Function Documentation

9.16.3.1 what()

```
const char * exceptions::ParsingException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 65 of file Exceptions.hpp.

References message.

9.16.4 Member Data Documentation

9.16.4.1 file

```
const std::string exceptions::ParsingException::file  [private]
```

Definition at line 48 of file Exceptions.hpp.

9.16.4.2 message

```
std::string exceptions::ParsingException::message  [private]
```

Definition at line 49 of file Exceptions.hpp.

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

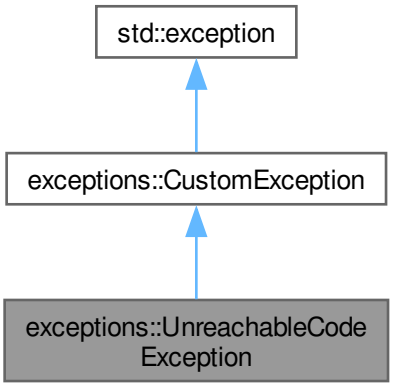
- src/include/Exceptions.hpp

9.17 exceptions::UnreachableCodeException Class Reference

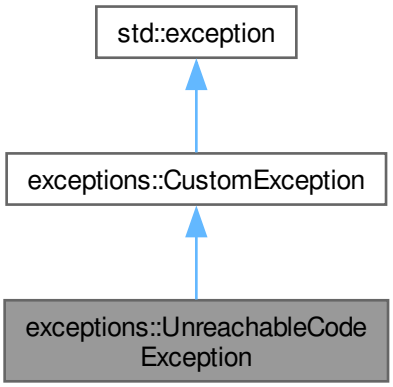
Exception for when the application reaches code it shouldn't reach.

```
#include <Exceptions.hpp>
```

Inheritance diagram for exceptions::UnreachableCodeException:



Collaboration diagram for exceptions::UnreachableCodeException:



Public Member Functions

- UnreachableCodeException (const std::string &message)
- const char * what () const noexcept override

Public Member Functions inherited from exceptions::CustomException

- const char * what () const noexcept override

Private Attributes

- std::string message

9.17.1 Detailed Description

Exception for when the application reaches code it shouldn't reach.

Definition at line 238 of file Exceptions.hpp.

9.17.2 Constructor & Destructor Documentation

9.17.2.1 UnreachableCodeException()

```
exceptions::UnreachableCodeException::UnreachableCodeException (
            const std::string & message )  [inline], [explicit]
```

Definition at line 243 of file Exceptions.hpp.

References config::EXECUTABLE_NAME, and message.

9.17.3 Member Function Documentation

9.17.3.1 what()

```
const char * exceptions::UnreachableCodeException::what ( ) const  [inline], [override], [noexcept]
```

Definition at line 250 of file Exceptions.hpp.

References message.

9.17.4 Member Data Documentation

9.17.4.1 message

```
std::string exceptions::UnreachableCodeException::message  [private]
```

Definition at line 240 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

9.18 utilities::Utils Class Reference

Responsible for utility function.

`#include <Utils.hpp>`

Static Public Member Functions

- static void setupEasyLogging (const std::string &configFile)
  Set up easylogging.
- static bool handleParseException (const exceptions::CustomException &e, const std::vector< std::string >↩ ::iterator &file, const std::vector< std::string > &files)
  Handle an exception within the main parsing loop.
- static bool askToContinue (const std::string &prompt="Do you want to continue? (Y/N)\n")
  Asks if the user wants to continue.
- static void checkConfigFile (const std::string &configFile)
  Checks if the easylogging-config file exists.
- static const std::string & checkDirectory (std::string &directory)
  Checks if the given directory exists and is valid.

9.18.1 Detailed Description

Responsible for utility function.

This class is responsible for handling miscellaneous utility functions which be used throughout the whole project.

Definition at line 42 of file Utils.hpp.

9.18.2 Member Function Documentation

9.18.2.1 askToContinue()

```
bool utilities::Utils::askToContinue (
            const std::string & prompt = "Do you want to continue? (Y/N)\n" )  [static]
```

Asks if the user wants to continue.

Asks the user if they want to continue and prompts them for a response.

Parameters

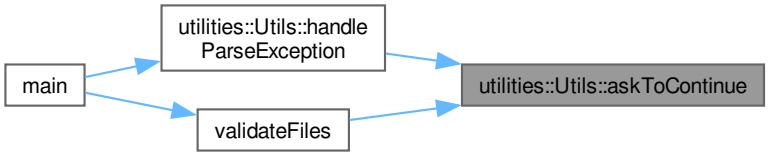
prompt	(Optional) A custom prompt to be used.

Returns

Returns true if the user wants to continue and false otherwise.

Definition at line 34 of file Utils.cpp.

Here is the caller graph for this function:



9.18.2.2 checkConfigFile()

```
void utilities::Utils::checkConfigFile (
            const std::string & configFile )  [static]
```

Checks if the easylogging-config file exists.

Parameters

configFile	The config file to be checked

Definition at line 55 of file Utils.cpp.

Here is the caller graph for this function:

9.18.2.3 checkDirectory()

```
const std::string & utilities::Utils::checkDirectory (
            std::string & directory )  [static]
```

Checks if the given directory exists and is valid.

This function checks if the given directory exists and is valid. If the directory does not end with a '/' or a '\', it will be added.

Parameters

directory	The directory to be checked

Returns

The checked directory

Definition at line 65 of file Utils.cpp.

Here is the caller graph for this function:

9.18.2.4 handleParseException()

```
bool utilities::Utils::handleParseException (
            const exceptions::CustomException & e,
            const std::vector< std::string >::iterator & file,
            const std::vector< std::string > & files )  [static]
```

Handle an exception within the main parsing loop.

This function handles an exception within the main parsing loop. It displays the error message and asks the user if they want to continue.

- Moved to Utils in 0.2.2 to improve readibility in main.cpp

Parameters

e	The exception to be handled
file	The file which caused the exception
files	The list of files

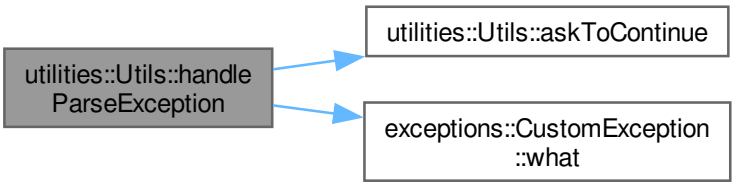
Returns

Returns true if the user wants to continue and false otherwise

Definition at line 77 of file Utils.cpp.

References askToContinue(), and exceptions::CustomException::what().

Here is the call graph for this function:



Here is the caller graph for this function:

9.18.2.5 setupEasyLogging()

```
void utilities::Utils::setupEasyLogging (
            const std::string & configFile )  [static]
```

Set up easylogging.

This function sets up the easylogging library based on the given config file.

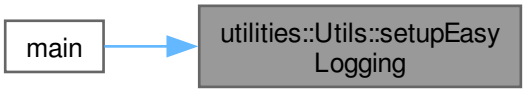
Parameters

configFile	The config file which is used

Definition at line 25 of file Utils.cpp.

References config::HOMEPAGE_URL, config::MAJOR_VERSION, config::MINOR_VERSION, config::PATCH_VERSION, and config::PROJECT_NAME.

Here is the caller graph for this function:



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

- src/include/Utils.hpp
- src/sources/Utils.cpp

Chapter 10

File Documentation

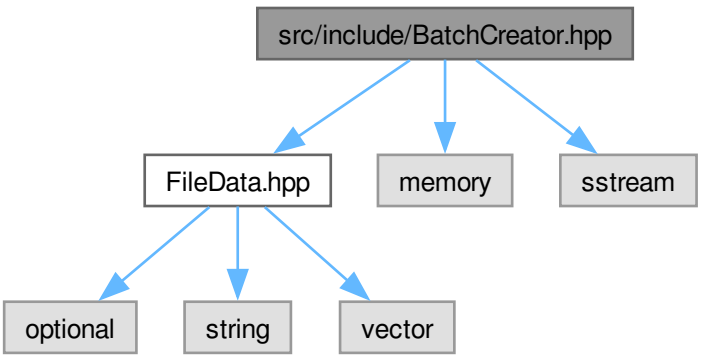
10.1 README.md File Reference

10.2 src/include/BatchCreator.hpp File Reference

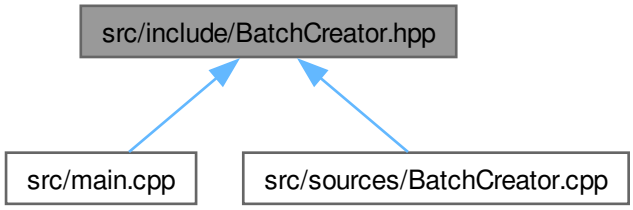
Contains the BatchCreator class.

```
#include "FileData.hpp"
#include <memory>
#include <sstream>
```

Include dependency graph for BatchCreator.hpp:



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



Classes

- class BatchCreator

  Creates a batch file from a FileData obeject.

10.2.1 Detailed Description

Contains the BatchCreator class.

Author

Maximilian Rodler

Date

2024-04-22

Version

0.2.1

See also

BatchCreator

src/sources/BatchCreator.cpp

Copyright

See LICENSE file

Definition in file BatchCreator.hpp.

10.3 BatchCreator.hpp

Go to the documentation of this file.

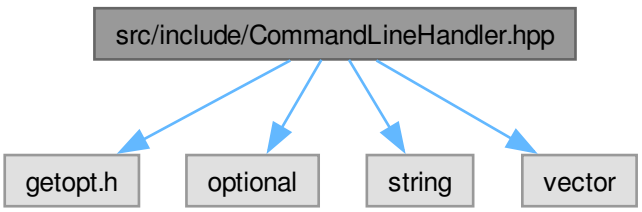
```
00001
00016 #include "FileData.hpp"
00017 #include <memory>
00018 #include <sstream>
00019
00029 class BatchCreator {
00030   public:
00039     explicit BatchCreator(std::shared_ptr<parsing::FileData> fileData);
00040
00046     [[nodiscard]] std::shared_ptr<std::stringstream> getDataStream() const {
00047         return dataStream;
00048     }
00049
00050   private:
00051     std::shared_ptr<std::stringstream>
00052     dataStream;
00054     std::shared_ptr<parsing::FileData> fileData;
00063     void createBatch() const;
00064
00073     void writeStart() const;
00074
00081     void writeHideShell() const;
00082
00090     void writeCommands() const;
00091
00101     void writeEnvVariables() const;
00102
00109     void writePathVariables() const;
00110
00118     void writeApp() const;
00119
00127     void writeEnd() const;
00128 };
```

10.4 src/include/CommandLineHandler.hpp File Reference

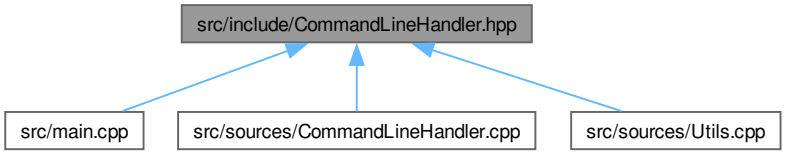
Responsible for the Command Line Interface.

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

Include dependency graph for CommandLineHandler.hpp:



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



Classes

- class cli::CommandLineHandler

  Responsible for the Command Line Interface.

Namespaces

- namespace cli

  Includes everything regarding the CLI.

Variables

- static const struct option cli::options [ ]

10.4.1 Detailed Description

Responsible for the Command Line Interface.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

This file is responsible for the Command Line Interface. As such it includes things such as the CommandLine↩ Handler class, possible options and style helpers.

See also

cli

CommandLineHandler

options

StyleHelpers

src/sources/CommandLineHandler.cpp

Copyright

See LICENSE file

Definition in file CommandLineHandler.hpp.

10.5 CommandLineHandler.hpp

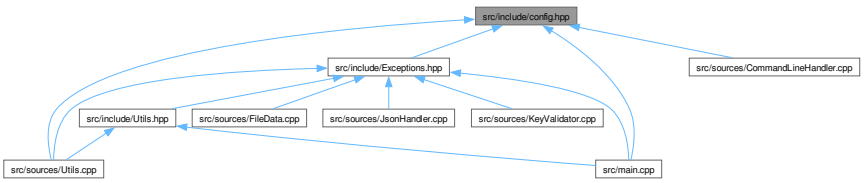
Go to the documentation of this file.

```
00001
00021 #ifndef COMMANDLINEHANDLER_HPP
00022 #define COMMANDLINEHANDLER_HPP
00023
00024 #include <getopt.h>
00025 #include <optional>
00026 #include <string>
00027 #include <vector>
00028
00041 namespace cli {
00042
00055 class CommandLineHandler {
00056   public:
00062     [[noreturn]] static void printHelp();
00068     [[noreturn]] static void printVersion();
00074     [[noreturn]] static void printCredits();
00086     static std::tuple<std::optional<std::string>, std::vector<std::string»
00087     parseArguments(int argc, char* argv[]);
00093     CommandLineHandler() = delete;
00099     ~CommandLineHandler() = delete;
00100 };
00101
00111 static const struct option options[] = {
00112     {"help", no_argument, nullptr, 'h'},
00113     {"version", no_argument, nullptr, 'v'},
00114     {"credits", no_argument, nullptr, 'c'},
00115     {"verbose", no_argument, nullptr, 0},
00116     {"outdir", required_argument, nullptr, 'o'},
00117     nullptr
00118 };
00119
00131 #ifdef IS_UNIX // CLI Formatting for Linux
00132 static const std::string CLEAR_TERMINAL = "\033[2J\033[1;1H";
00133 static const std::string RESET = "\033[0m";
00134 static const std::string RED = "\033[0;31m";
00135 static const std::string GREEN = "\033[0;32m";
00136 static const std::string YELLOW = "\033[0;33m";
00137 static const std::string BLUE = "\033[0;34m";
00138 static const std::string MAGENTA = "\033[0;35m";
00139 static const std::string CYAN = "\033[0;36m";
00140 static const std::string WHITE = "\033[0;37m";
00141 static const std::string BOLD = "\033[1m";
00142 static const std::string UNDERLINE = "\033[4m";
00143 static const std::string ITALIC = "\033[3m";
00144 //@note Windows doesn't support ANSI escape codes the same way
00145 #elif defined(IS_WINDOWS)
00146 static const std::string CLEAR_TERMINAL = "";
00147 static const std::string RESET = "";
00148 static const std::string RED = "";
00149 static const std::string GREEN = "";
00150 static const std::string YELLOW = "";
00151 static const std::string BLUE = "";
00152 static const std::string MAGENTA = "";
00153 static const std::string CYAN = "";
00154 static const std::string WHITE = "";
00155 static const std::string BOLD = "";
00156 static const std::string UNDERLINE = "";
00157 static const std::string ITALIC = "";
00158 #endif
// end of group StyleHelpers  00160
00161 } // namespace cli
00162
00163 #endif // COMMANDLINEHANDLER_HPP
```

10.6 src/include/config.hpp File Reference

Configures general project information.

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



Namespaces

- namespace config

  Namespace used for general project information.

Variables

- constexpr auto config::LOG_CONFIG
- constexpr auto config::EXECUTABLE_NAME = "json2batch"
- constexpr auto config::MAJOR_VERSION = "0"
- constexpr auto config::MINOR_VERSION = "2"
- constexpr auto config::PATCH_VERSION = "2"
- constexpr auto config::DESCRIPTION = "A simple tool to convert json to batch."
- constexpr auto config::PROJECT_NAME = "JSON2Batch"
- constexpr auto config::AUTHORS = "@AUTHORS"
- constexpr auto config::HOMEPAGE_URL

10.6.1 Detailed Description

Configures general project information.

Author

Simon Blum

Date

2024-04-18

Version

0.1.5

This file is used by CMake to configure general information which can be used throughout the project.

Note

This file is automatically configured by CMake. The original file can be found in conf/config.hpp.in @license GNU GPLv3

Copyright

See LICENSE file

Definition in file config.hpp.

10.7 config.hpp

Go to the documentation of this file.

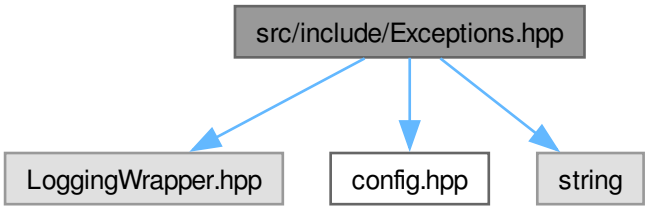
```
00001
00016 // This file is autogenerated. Changes will be overwritten
00017
00018 #ifndef CONFIG_HPP
00019 #define CONFIG_HPP
00020
00025 namespace config {
00026 inline constexpr auto LOG_CONFIG =
00027             "/home/simon/1_Coding/cpp/JsonToBat/build/Debug/config/easylogging.conf";
00028 inline constexpr auto EXECUTABLE_NAME = "json2batch";
00029 inline constexpr auto MAJOR_VERSION = "0";
00030 inline constexpr auto MINOR_VERSION = "2";
00031 inline constexpr auto PATCH_VERSION = "2";
00032 inline constexpr auto DESCRIPTION = "A simple tool to convert json to batch.";
00033 inline constexpr auto PROJECT_NAME = "JSON2Batch";
00034 inline constexpr auto AUTHORS = "@AUTHORS";
00035 inline constexpr auto HOMEPAGE_URL =
00036             "https://dhbwprojectsit23.github.io/JSON2Bat";
00037 } // namespace config
00038
00039 #endif
```

10.8 src/include/Exceptions.hpp File Reference

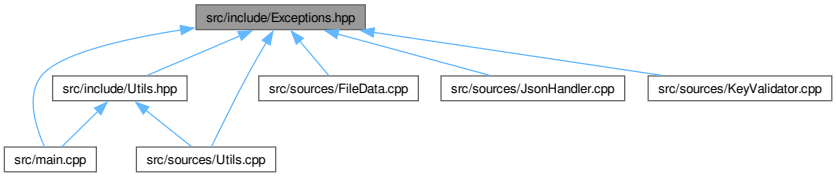
Contains all the custom exceptions used in the project.

```
#include "LoggingWrapper.hpp"
#include "config.hpp"
#include <string>
```

Include dependency graph for Exceptions.hpp:



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



Classes

- class exceptions::CustomException
  Base class for all custom exceptions.
- class exceptions::ParsingException
  Exception for syntax errors within the json file.
- class exceptions::FileExistsException
  Exception for an already exisiting outputfile.
- class exceptions::InvalidValueException
  Exception for an ivalid (usually empty) value field.
- class exceptions::InvalidKeyException
  Exception for invalid keys.
- class exceptions::InvalidTypeException
  Exception for invalid types.
- class exceptions::MissingKeyException
  Exception for missing keys within entries.
- class exceptions::MissingTypeException
  Exception for missing types of entries.
- class exceptions::UnreachableCodeException
  Exception for when the application reaches code it shouldn't reach.
- class exceptions::FailedToOpenFileException
  Exception for when a file can't be opened.
- class exceptions::NoSuchDirException
  Exception for when a directory does not exist.

Namespaces

- namespace exceptions
  Namespace used for customized exceptions.

10.8.1 Detailed Description

Contains all the custom exceptions used in the project.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

The error handling within this project is exception based. This allows us to throw custom exceptions throughout any part of the process and allow us to deal with them when necessary.

Copyright

See LICENSE file

Definition in file Exceptions.hpp.

10.9 Exceptions.hpp

Go to the documentation of this file.

```
00001
00014 #ifndef EXCEPTIONS_HPP
00015 #define EXCEPTIONS_HPP
00016
00017 #include "LoggingWrapper.hpp"
00018 #include "config.hpp"
00019 #include <string>
00020
00025 namespace exceptions {
00035 class CustomException : public std::exception {
00036   public:
00037     [[nodiscard]] const char* what() const noexcept override {
00038         return "Base Exception";
00039     }
00040 };
00041
00046 class ParsingException : public CustomException {
00047   private:
00048     const std::string file;
00049     std::string message;
00050
00051   public:
00052     explicit ParsingException(const std::string &file) : file(file) {
00058         std::stringstream ss;
00059         ss « "Error while trying to parse \"" « file « "\"!\n"
00060            « "There most likely is a syntax error within the \".json\" file.";
00061         this->message = ss.str();
00062         LOG_INFO « "ParsingException: " « message;
00063     }
00064
00065     [[nodiscard]] const char* what() const noexcept override {
00066         return message.c_str();
00067     }
00068 };
00069
00074 class FileExistsException : public CustomException {
00075   private:
00076     const std::string file;
00077     std::string message;
00078
00079   public:
00080     explicit FileExistsException(const std::string &file) : file(file) {
00086         std::stringstream ss;
00087         ss « "The outputfile \"" « file « "\" already exists!";
00088         this->message = ss.str();
00089         LOG_INFO « "BatchExistsException: " « message;
00090     }
00091
00092     [[nodiscard]] const char* what() const noexcept override {
00093         return message.c_str();
00094     }
00095 };
00096
00101 class InvalidValueException : public CustomException {
00102   private:
00103     const std::string key;
00104     std::string message;
00105
00106   public:
00107     InvalidValueException(const std::string &key, const std::string &issue)
00108         : key(key) {
00114         std::stringstream ss;
00115         ss « "Error at key \"" « key « "\"! " « issue;
00116         this->message = ss.str();
00117         LOG_INFO « "InvalidValueException: " « message;
00118     }
00119     [[nodiscard]] const char* what() const noexcept override {
00120         return message.c_str();
00121     }
00122 };
00123
00135 class InvalidKeyException : public CustomException {
00136   private:
00137     std::string message = "Invalid key found!";
00138
00139   public:
00140     explicit InvalidKeyException(
00141                 const std::vector<std::tuple<int, std::string» &keys) {
00142         LOG_INFO « "InvalidKeyException: " « message;
00143
00144         for (const auto &[line, key] : keys) {
00145             LOG_WARNING « "Invalid key found at line " « line « ": \"" « key
```

```
00146                         « "\"!";
00147         }
00148     }
00149     [[nodiscard]] const char* what() const noexcept override {
00150         return message.c_str();
00151     }
00152 };
00153
00162 class InvalidTypeException : public CustomException {
00163   private:
00164     const std::string type;
00165     std::string message;
00166
00167   public:
00168     InvalidTypeException(const std::string &type, int line) : type(type) {
00174         std::stringstream ss;
00175         ss « "Invalid type found at line " « line « ": \"" « type « "\"";
00176         this->message = ss.str();
00177         LOG_INFO « "InvalidTypeException: " « message;
00178     }
00179     [[nodiscard]] const char* what() const noexcept override {
00180         return message.c_str();
00181     }
00182 };
00183
00191 class MissingKeyException : public CustomException {
00192   private:
00193     std::string message;
00194     std::string type;
00195     std::string key;
00196
00197   public:
00198     MissingKeyException(const std::string &key, const std::string &type)
00199         : type(type), key(key) {
00205         std::stringstream ss;
00206         ss « "Missing key \"" « key « "\" for type \"" « type « "\"!";
00207         this->message = ss.str();
00208         LOG_INFO « "MissingKeyException: " « message;
00209     }
00210     [[nodiscard]] const char* what() const noexcept override {
00211         return message.c_str();
00212     }
00213 };
00214
00221 class MissingTypeException : public CustomException {
00222   private:
00223     std::string message = "Missing \"type\" key for at least one entry!";
00224
00225   public:
00226     MissingTypeException() {
00227         LOG_INFO « "MissingTypeException: " « message;
00228     }
00229     [[nodiscard]] const char* what() const noexcept override {
00230         return message.c_str();
00231     }
00232 };
00233
00238 class UnreachableCodeException : public CustomException {
00239   private:
00240     std::string message;
00241
00242   public:
00243     explicit UnreachableCodeException(const std::string &message)
00244         : message(message) {
00245         OUTPUT « "This exception happened due to a bug in the application!\n"
00246                « "Please report this bug! See " « config::EXECUTABLE_NAME
00247                « " -c for contact information.\n";
00248         LOG_INFO « "UnreachableCodeException: " « message;
00249     }
00250     [[nodiscard]] const char* what() const noexcept override {
00251         return message.c_str();
00252     }
00253 };
00254
00259 class FailedToOpenFileException : public CustomException {
00260   private:
00261     std::string message;
00262
00263   public:
00264     explicit FailedToOpenFileException(const std::string &file) {
00265         message = "Failed to open file: " + file;
00266         LOG_INFO « "FailedToOpenFileException: " « message;
00267     }
00268     [[nodiscard]] const char* what() const noexcept override {
00269         return message.c_str();
00270     }
00271 };
```

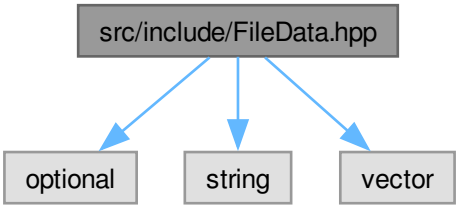
```
00272
00277 class NoSuchDirException : public CustomException {
00278   private:
00279     std::string message;
00280
00281   public:
00282     explicit NoSuchDirException(const std::string &dir) {
00283         message = "No such directory: " + dir;
00284         LOG_INFO « "NoSuchDirException: " « message;
00285     }
00286     [[nodiscard]] const char* what() const noexcept override {
00287         return message.c_str();
00288     }
00289 };
00290
00291 } // namespace exceptions
00292
00293 #endif
```

10.10 src/include/FileData.hpp File Reference

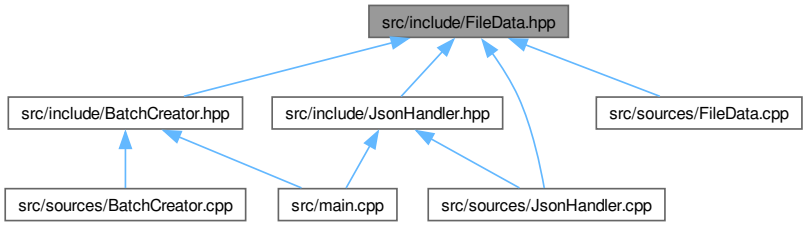
This file contains the FileData class.

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

Include dependency graph for FileData.hpp:



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



Classes

- class parsing::FileData

  This class contains all data from the json file.

Namespaces

- namespace parsing

  The namespace containing everything relevant to parsing.

10.10.1 Detailed Description

This file contains the FileData class.

Author

Sonia Sinacci, Elena Schwartzbach

Date

16.04.2024

Version

0.1.5

See also

parsing::FileData

src/sources/FileData.cpp

Copyright

See LICENSE file

Definition in file FileData.hpp.

10.11 FileData.hpp

Go to the documentation of this file.

```
00001
00015 #ifndef FILEDATA_HPP
00016 #define FILEDATA_HPP
00017
00018 #include <optional>
00019 #include <string>
00020 #include <vector>
00021
00022 namespace parsing {
00031 class FileData {
00032   public:
00043     void setOutputFile(std::string &newOutputfile);
00044
00049     void setHideShell(bool newHideShell) {
00050         this->hideShell = newHideShell;
00051     }
00052
00061     void setApplication(const std::string &newApplication);
00062
00073     void addCommand(const std::string &command);
00074
00086     void addEnvironmentVariable(const std::string &name,
```

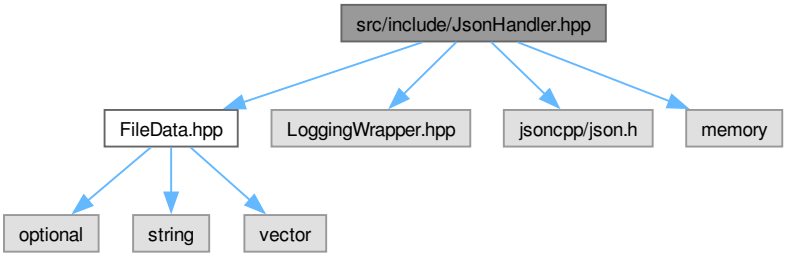
```
00087                                const std::string &value);
00088
00099     void addPathValue(const std::string &pathValue);
00100
00105     [[nodiscard]] const std::string &getOutputFile() const {
00106         return outputfile;
00107     }
00108
00113     [[nodiscard]] bool getHideShell() const {
00114         return hideShell;
00115     }
00116
00121     [[nodiscard]] const std::optional<std::string> &getApplication() const {
00122         return application;
00123     }
00124
00129     [[nodiscard]] const std::vector<std::string> &getCommands() const {
00130         return commands;
00131     }
00132
00137     [[nodiscard]] const std::vector<std::tuple<std::string, std::string>> &
00138     getEnvironmentVariables() const {
00139         return environmentVariables;
00140     }
00141
00146     [[nodiscard]] const std::vector<std::string> &getPathValues() const {
00147         return pathValues;
00148     }
00149
00150   private:
00151     std::string outputfile;
00152     bool hideShell;
00153     std::optional<std::string> application;
00154     std::vector<std::string> commands;
00155     // Tuple<Name, Value>
00156     std::vector<std::tuple<std::string, std::string>> environmentVariables;
00157     std::vector<std::string> pathValues;
00158 };
00159 } // namespace parsing
00160
00161 #endif // FILEDATA_HPP
```

10.12 src/include/JsonHandler.hpp File Reference

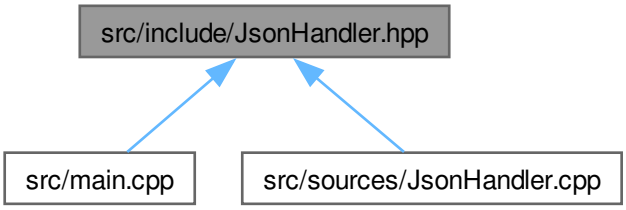
This file contains the JsonHandler class.

```
#include "FileData.hpp"
#include "LoggingWrapper.hpp"
#include <jsoncpp/json.h>
#include <memory>
```

Include dependency graph for JsonHandler.hpp:



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



Classes

- class parsing::JsonHandler

  This file reads all data from the json file.

Namespaces

- namespace parsing

  The namespace containing everything relevant to parsing.

10.12.1 Detailed Description

This file contains the JsonHandler class.

Author

Sonia Sinacci, Elena Schwartzbach

Date

23.04.2024

Version

0.1.5

See also

parsing::JsonHandler

src/sources/JsonHandler.cpp

Copyright

See LICENSE file

Definition in file JsonHandler.hpp.

10.13 JsonHandler.hpp

Go to the documentation of this file.

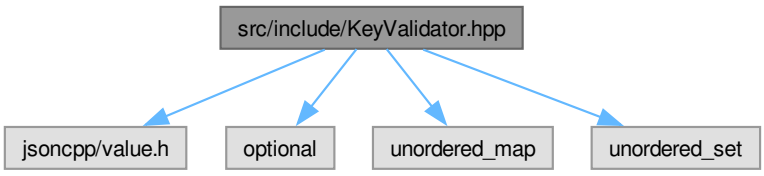
```
00001
00015 #ifndef JSONHANDLER_HPP
00016 #define JSONHANDLER_HPP
00017
00018 #include "FileData.hpp"
00019 #include "LoggingWrapper.hpp"
00020 #include <jsoncpp/json.h>
00021
00022 #include <memory>
00023
00036 namespace parsing {
00037
00047 class JsonHandler {
00048   public:
00055     JsonHandler() {
00056         LOG_INFO << "Initialising empty JsonHandler";
00057     }
00065     explicit JsonHandler(const std::string &filename);
00075     std::shared_ptr<FileData> getFileData();
00076
00077   private:
00093     [[nodiscard]] static std::shared_ptr<Json::Value>
00094     parseFile(const std::string &filename);
00103     void assignOutputFile() const;
00110     void assignHideShell() const;
00117     void assignApplication() const;
00129     void assignEntries() const;
00134     void assignCommand(const Json::Value &entry) const;
00139     void assignEnvironmentVariable(const Json::Value &entry) const;
00144     void assignPathValue(const Json::Value &entry) const;
00153     std::shared_ptr<FileData> createFileData();
00154     std::shared_ptr<Json::Value> root;
00155     std::shared_ptr<FileData> data;
00156 };
00157 } // namespace parsing
00158
00159 #endif // JSONHANDLER_HPP
```

10.14 src/include/KeyValidator.hpp File Reference

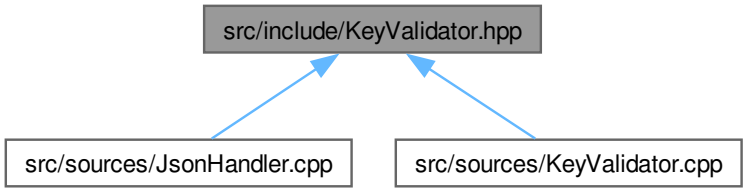
This file contains the KeyValidator class.

```
#include "jsoncpp/value.h"
#include <optional>
#include <unordered_map>
#include <unordered_set>
```

Include dependency graph for KeyValidator.hpp:



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



Classes

- class parsing::KeyValidator

  Validates keys of a Json::Value object.

Namespaces

- namespace parsing

  The namespace containing everything relevant to parsing.

10.14.1 Detailed Description

This file contains the KeyValidator class.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

See also

parsing::KeyValidator

src/sources/KeyValidator.cpp

Copyright

See LICENSE file

Definition in file KeyValidator.hpp.

10.15 KeyValidator.hpp

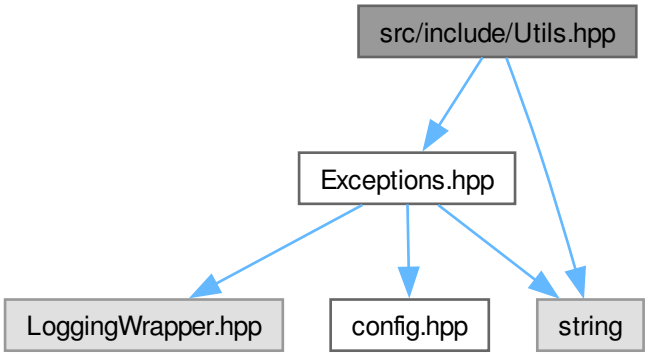
Go to the documentation of this file.

```
00001
00014 #ifndef KEYVALIDATOR_HPP
00015 #define KEYVALIDATOR_HPP
00016
00017 #include "jsoncpp/value.h"
00018 #include <optional>
00019 #include <unordered_map>
00020 #include <unordered_set>
00021 namespace parsing {
00030 class KeyValidator {
00031   public:
00037     static KeyValidator &getInstance();
00038
00053     std::vector<std::tuple<int, std::string>>
00054     validateKeys(const Json::Value &root, const std::string &filename);
00055
00056   private:
00069     std::vector<std::tuple<int, std::string>>
00070     getWrongKeys(const Json::Value &root, const std::string &filename) const;
00071
00091     void validateTypes(const std::string &filename, const Json::Value &entry,
00092                        const std::unordered_set<std::string> &entryKeys);
00093
00107     std::vector<std::tuple<int, std::string>>
00108     validateEntries(const std::string &filename,
00109                     const std::unordered_set<std::string> &entryKeys) const;
00110
00123     static std::optional<int> getUnknownKeyLine(const std::string &filename,
00124                                                 const std::string &wrongKey);
00125
00130     std::unordered_set<std::string> validKeys = {"outputfile", "hideshell",
00131         "entries", "application"
00132     };
00137     std::unordered_set<std::string> validEntryKeys = {"type", "key", "value",
00138         "path", "command"
00139     };
00140
00144     std::unordered_map<std::string_view, std::vector<std::string>> typeToKeys = {
00145         {"EXE", {"command"}}, {"PATH", {"path"}}, {"ENV", {"key", "value"}}
00146     };
00147 };
00148 } // namespace parsing
00149
00150 #endif
```

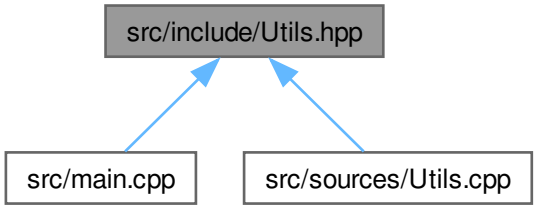
10.16 src/include/Utils.hpp File Reference

```
#include "Exceptions.hpp"
#include <string>
```

Include dependency graph for Utils.hpp:



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



Classes

- class utilities::Utils

  Responsible for utility function.

Namespaces

- namespace utilities

  Includes all utilities.

10.17 Utils.hpp

Go to the documentation of this file.

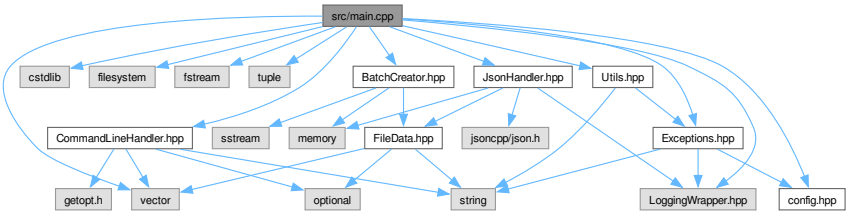
```
00001
00018 #ifndef UTILITIES_HPP
00019 #define UTILITIES_HPP
00020
00021 #include "Exceptions.hpp"
00022 #include <string>
00023
00033 namespace utilities {
00034
00042 class Utils {
00043   public:
00051     static void setupEasyLogging(const std::string &configFile);
00052
00066     static bool
00067     handleParseException(const exceptions::CustomException &e,
00068                          const std::vector<std::string>::iterator &file,
00069                          const std::vector<std::string> &files);
00070
00078     static bool
00079     askToContinue(const std::string &prompt = "Do you want to continue? (Y/N)\n");
00080
00085     static void checkConfigFile(const std::string &configFile);
00086
00098     static const std::string &checkDirectory(std::string &directory);
00099 };
00100 } // namespace utilities
00101
00102 #endif // UTILITIES_HPP
```

10.18 src/main.cpp File Reference

Contains the main function.

```
#include <LoggingWrapper.hpp>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <tuple>
#include <vector>
#include "BatchCreator.hpp"
#include "CommandLineHandler.hpp"
#include "Exceptions.hpp"
#include "JsonHandler.hpp"
#include "Utils.hpp"
#include "config.hpp"
```

Include dependency graph for main.cpp:



Functions

- std::tuple< std::vector< std::string >, std::string > parseAndValidateArgs (int argc, char *argv[])
  Validates and parses arguments.
- std::vector< std::string > validateFiles (const std::vector< std::string > &files)
  Checks if the files are valid.
- void parseFile (const std::string &file, const std::string &outputDirectory)
  Parses the given file and writes the output to the output directory.
- int main (int argc, char *argv[])
  Main function of the program.

10.18.1 Detailed Description

Contains the main function.

Author

Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci

Date

2024-04-26

Version

0.2.2

The main function is responsible for connection all parts of the programm. It calls all relevant classes and finishes when everything is done.

Copyright

See LICENSE file

Definition in file main.cpp.

10.18.2 Function Documentation

10.18.2.1 main()

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

Main function of the program.

The main function is responsible for connection all parts of the programm. It calls all relevant classes and finishes when everything is done.

Parameters

argc	The number of arguments given
argv	The command line arguments given

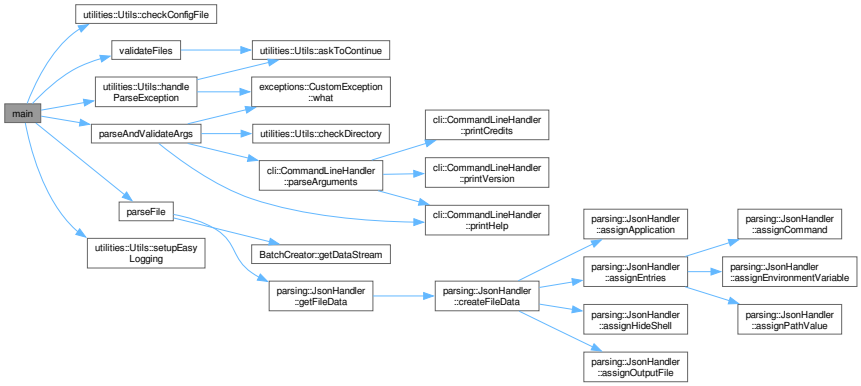
Returns

Returns 0 on success, 1 on failure

Definition at line 67 of file main.cpp.

References utilities::Utils::checkConfigFile(), utilities::Utils::handleParseException(), config::LOG_CONFIG, parseAndValidateArgs(), parseFile(), utilities::Utils::setupEasyLogging(), and validateFiles().

Here is the call graph for this function:



10.18.2.2 parseAndValidateArgs()

```
std::tuple< std::vector< std::string >, std::string > parseAndValidateArgs (
            int argc,
            char * argv[] )
```

Validates and parses arguments.

Parameters

argc	Number of arguments provided
argv	The arguments provided

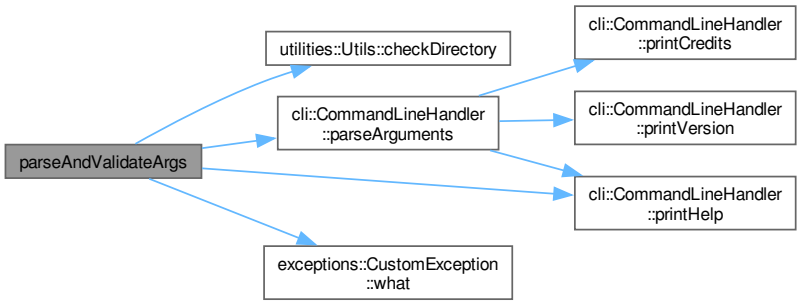
Returns

A tuple containing the files to be parsed and the output directory

Definition at line 103 of file main.cpp.

References utilities::Utils::checkDirectory(), cli::CommandLineHandler::parseArguments(), cli::CommandLineHandler::printHelp(), and exceptions::CustomException::what().

Here is the call graph for this function:



Here is the caller graph for this function:

10.18.2.3 parseFile()

```
void parseFile (
            const std::string & file,
            const std::string & outputDirectory )
```

Parses the given file and writes the output to the output directory.

Creates the Batch file from the given file

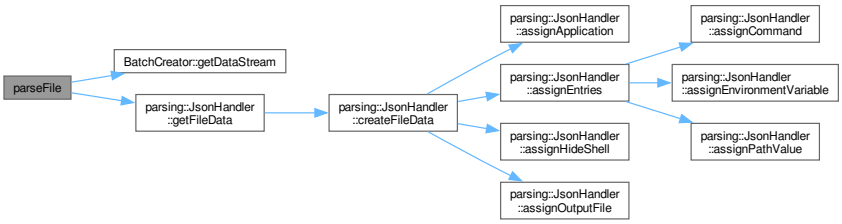
Parameters

file	The file to be parsed

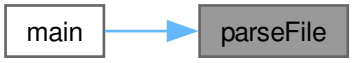
Definition at line 170 of file main.cpp.

References BatchCreator::getDataStream(), and parsing::JsonHandler::getFileData().

Here is the call graph for this function:



Here is the caller graph for this function:



10.18.2.4 validateFiles()

```
std::vector< std::string > validateFiles (
            const std::vector< std::string > & files )
```

Checks if the files are valid.

Makes sures, that provided files exists and checks their file ending

Parameters

files	The files to be checked

Returns

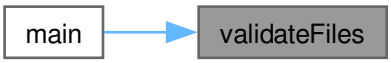
A vector containing the valid files

Definition at line 130 of file main.cpp.

References utilities::Utils::askToContinue().

Here is the call graph for this function:

Here is the caller graph for this function:



10.19 main.cpp

Go to the documentation of this file.

```
00001
00013 #include <LoggingWrapper.hpp>
00014 #include <cstdlib>
00015 #include <filesystem>
00016 #include <fstream>
00017 #include <tuple>
00018 #include <vector>
00019
00020 #include "BatchCreator.hpp"
00021 #include "CommandLineHandler.hpp"
00022 #include "Exceptions.hpp"
00023 #include "JsonHandler.hpp"
00024 #include "Utils.hpp"
00025 #include "config.hpp"
00026
00034 std::tuple<std::vector<std::string>, std::string>
00035 parseAndValidateArgs(int argc, char *argv[]);
00036
00044 std::vector<std::string> validateFiles(const std::vector<std::string> &files);
00045
00052 void parseFile(const std::string &file, const std::string &outputDirectory);
00053
00067 int main(int argc, char *argv[]) {
00068   // Setup logging
00069   utilities::Utils::checkConfigFile(config::LOG_CONFIG);
00070   utilities::Utils::setupEasyLogging(config::LOG_CONFIG);
00071   // Parse and validate arguments
00072   auto [files, outDir] = parseAndValidateArgs(argc, argv);
00073   OUTPUT « cli::BOLD « "Parsing the following files:\n" « cli::RESET;
00074
00075   for (const auto &file : files) {
00076     OUTPUT « "\t - " « file « "\n";
00077   }
00078
00079   files = validateFiles(files);
00080
00081   // Main parsing loop
00082   for (auto file = files.begin(); file != files.end(); ++file) {
00083     OUTPUT « cli::ITALIC « "\nParsing file: " « *file « "...\n"
00084            « cli::RESET;
00085
```

```
00086     try {
00087       parseFile(*file, outDir);
00088       // Only catch custom exceptions, other exceptions are fatal
00089     } catch (const exceptions::CustomException &e) {
00090       if (utilities::Utils::handleParseException(e, file, files)) {
00091         continue;
00092       }
00093
00094       exit(1);
00095     }
00096   }
00097
00098   LOG_INFO << "Exiting...";
00099   return 0;
00100 }
00101
00102 std::tuple<std::vector<std::string>, std::string>
00103 parseAndValidateArgs(int argc, char *argv[]) {
00104   if (argc < 2) {
00105     LOG_ERROR << "No options given!\n";
00106     cli::CommandLineHandler::printHelp();
00107   }
00108
00109   auto [outOption, files] = cli::CommandLineHandler::parseArguments(argc, argv);
00110   // Set the output directory if given
00111   std::string outDir = outOption.value_or("");
00112
00113   if (!outDir.empty()) {
00114     try {
00115       outDir = utilities::Utils::checkDirectory(outDir);
00116     } catch (const exceptions::CustomException &e) {
00117       LOG_ERROR << e.what();
00118       exit(1);
00119     }
00120   }
00121
00122   if (files.empty()) {
00123     LOG_ERROR << "No files were given as arguments!\n";
00124     exit(1);
00125   }
00126
00127   return {files, outDir};
00128 }
00129
00130 std::vector<std::string> validateFiles(const std::vector<std::string> &files) {
00131   std::vector<std::string> validFiles;
00132   // Reserve space, to avaid reallocating with each valid file
00133   validFiles.reserve(files.size());
00134
00135   for (const std::filesystem::path file : files) {
00136     // Check that the file exists
00137     if (!std::filesystem::is_regular_file(file)) {
00138       LOG_ERROR << "The file \"" << file << "\" does not exist!\n";
00139
00140       if (files.size() > 1 && !utilities::Utils::askToContinue()) {
00141         OUTPUT << "Aborting...\n";
00142         LOG_INFO << "Application ended by user Input";
00143         exit(1);
00144       }
00145
00146       continue;
00147     }
00148
00149     // Check if the file ends in .json
00150     if (file.extension() != ".json") {
00151       LOG_WARNING << "The file \"" << file << "\" does not end in \".json\"\n";
00152       OUTPUT << "If the file is not in JSON Format, continuing may "
00153                 "result in\nunexpected behaviour!\n";
00154
00155       if (!utilities::Utils::askToContinue()) {
00156         OUTPUT << "Aborting...\n";
00157         LOG_INFO << "Application ended by user Input";
00158         exit(1);
00159       }
00160     }
00161
00162     validFiles.push_back(file);
00163   }
00164
00165   // Shrinks the vector if invalid files were found
00166   validFiles.shrink_to_fit();
00167   return validFiles;
00168 }
00169
00170 void parseFile(const std::string &file, const std::string &outputDirectory) {
00171   parsing::JsonHandler jsonHandler(file);
00172   const auto fileData = jsonHandler.getFileData();
```

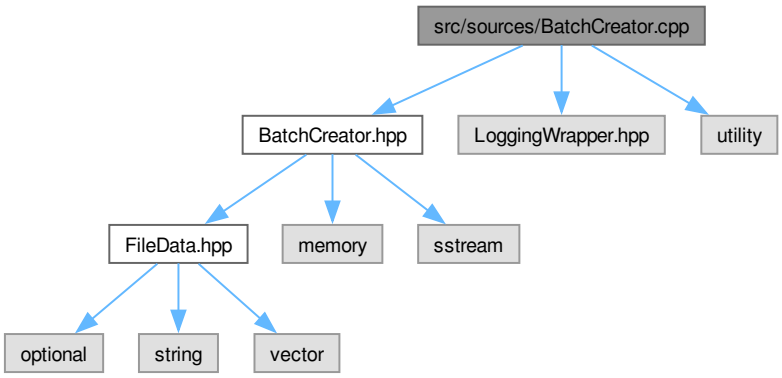
```
00173   BatchCreator batchCreator(fileData);
00174   const std::shared_ptr<std::stringstream> dataStream =
00175       batchCreator.getDataStream();
00176   // Full filename is output directory + output file
00177   const std::string outputFileName =
00178       outputDirectory + fileData->getOutputFile();
00179   std::ofstream outFile(outputFileName);
00180
00181   if (!outFile.good()) {
00182     throw exceptions::FailedToOpenFileException(outputFileName);
00183   }
00184
00185   outFile « dataStream->str();
00186   OUTPUT « "Done with files!\n";
00187 }
00188
00189 // Initialize easylogging++
00190 // Moved to bottom because it messed with doxygen
00191 INITIALIZE_EASYLOGGINGPP
```

10.20 src/sources/BatchCreator.cpp File Reference

Contains the implementation of the BatchCreator class.

```
#include "BatchCreator.hpp"
#include "LoggingWrapper.hpp"
#include <utility>
```

Include dependency graph for BatchCreator.cpp:



10.20.1 Detailed Description

Contains the implementation of the BatchCreator class.

Author

Maximilian Rodler

Date

22.04.2024

Version

0.2.2

See also

src/include/BatchCreator.hpp

Copyright

See LICENSE file

Definition in file BatchCreator.cpp.

10.21 BatchCreator.cpp

Go to the documentation of this file.

```
00001
00013 #include "BatchCreator.hpp"
00014
00015 #include "LoggingWrapper.hpp"
00016 #include <utility>
00017
00018 BatchCreator::BatchCreator(std::shared_ptr<parsing::FileData> fileData)
00019     : fileData(std::move(fileData)) {
00020     LOG_INFO << "Initializing BatchCreator";
00021     this->dataStream = std::make_shared<std::stringstream>();
00022     this->createBatch();
00023 }
00024
00025 void BatchCreator::createBatch() const {
00026     LOG_INFO << "Creating Batch file";
00027     this->writeStart();
00028     this->writeHideShell();
00029     this->writeCommands();
00030     this->writeEnvVariables();
00031     this->writePathVariables();
00032     this->writeApp();
00033     this->writeEnd();
00034 }
00035
00036 void BatchCreator::writeStart() const {
00037     LOG_INFO << "writing Start of Batch";
00038     *this->dataStream << "@ECHO OFF\r\nC:\\Windows\\System32\\cmd.exe ";
00039 }
00040
00041 void BatchCreator::writeHideShell() const {
00042     if (this->fileData->getHideShell()) {
00043         LOG_INFO << "writing hide Shell";
00044         *this->dataStream << "/c ";
00045     }
00046     else {
00047         LOG_INFO << "writing show Shell";
00048         *this->dataStream << "/k ";
00049     }
00050 }
00051
00052 void BatchCreator::writeCommands() const {
00053     LOG_INFO << "writing Commands";
00054     *this->dataStream << "\"";
00055
00056     for (const std::string &command : this->fileData->getCommands()) {
00057         *this->dataStream << command << " && ";
00058     }
00059 }
00060
00061 void BatchCreator::writeEnvVariables() const {
00062     LOG_INFO << "writing Environment Variables";
00063
00064     for (const auto &[key, value] : this->fileData->getEnvironmentVariables()) {
00065         *this->dataStream << "set " << key << "=" << value << " && ";
00066     }
00067 }
```

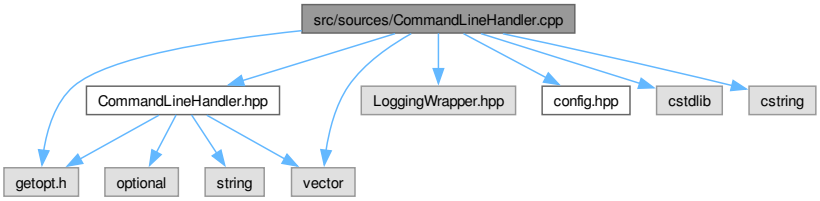
```
00068
00069 void BatchCreator::writePathVariables() const {
00070     LOG_INFO « "writing Path Variables";
00071     *this->dataStream « "set path=";
00072
00073     for (const std::string &path : this->fileData->getPathValues()) {
00074         *this->dataStream « path « ";";
00075     }
00076
00077     *this->dataStream « "%path%";
00078 }
00079
00080 void BatchCreator::writeApp() const {
00081     std::string appName = this->fileData->getOutputFile();
00082     appName = appName.substr(0, appName.find('.'));
00083
00084     if (this->fileData->getApplication().has_value()) {
00085         LOG_INFO « "writing start Application";
00086         *this->dataStream « " && start \"" « appName « "\" "
00087                           « this->fileData->getApplication().value() « "\"\r\n";
00088     }
00089     else {
00090         LOG_INFO « "writing not start Application";
00091         *this->dataStream « "\"\r\n";
00092     }
00093 }
00094
00095 void BatchCreator::writeEnd() const {
00096     *this->dataStream « "@ECHO ON";
00097 }
```

10.22 src/sources/CommandLineHandler.cpp File Reference

Implementation for the Command Line Interface.

```
#include "CommandLineHandler.hpp"
#include "LoggingWrapper.hpp"
#include "config.hpp"
#include <cstdlib>
#include <cstring>
#include <getopt.h>
#include <vector>
```

Include dependency graph for CommandLineHandler.cpp:



Namespaces

- namespace cli

  Includes everything regarding the CLI.

10.22.1 Detailed Description

Implementation for the Command Line Interface.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

See also

src/include/utility/CommandLineHandler.hpp

Copyright

See LICENSE file

Definition in file CommandLineHandler.cpp.

10.23 CommandLineHandler.cpp

Go to the documentation of this file.

```
00001
00013 #include "CommandLineHandler.hpp"
00014 #include "LoggingWrapper.hpp"
00015 #include "config.hpp"
00016 #include <cstdlib>
00017 #include <cstring>
00018 #include <getopt.h>
00019 #include <vector>
00020
00021 namespace cli {
00022 void CommandLineHandler::printHelp() {
00023     LOG_INFO << "Printing help message...";
00024     OUTPUT << BOLD << "Usage:\n"
00025            << RESET << "----------\n"
00026            << config::EXECUTABLE_NAME << " [options] [filenames]\n"
00027            << "\n"
00028            << BOLD << "Options:\n"
00029            << RESET << "----------\n"
00030            << "-o, --outdir\t [path]\t\tOutput the batch file to the given "
00031            "dir\n"
00032            << "-h, --help\t\t\tPrint this help message\n"
00033            << "-v, --version\t\t\tPrint the version number\n"
00034            << "-c, --credits\t\t\tPrint the credits\n\n"
00035            << "    --verbose\t\t\tStart the application in verbose mode\n"
00036            << ITALIC
00037            << "          \t\t\tNote: Verbose flag should be passed first!\n\n"
00038            << RESET << BOLD << "Filenames:\n"
00039            << RESET << "----------\n"
00040            << "The json files to be processed into batch files.\n"
00041            << "Multiple files should be seperated by spaces!\n\n";
00042     exit(0);
00043 }
00044 void CommandLineHandler::printVersion() {
00045     LOG_INFO << "Printing version number...";
```

```
00046     OUTPUT « config::PROJECT_NAME « " v" « config::MAJOR_VERSION « "."
00047            « config::MINOR_VERSION « "." « config::PATCH_VERSION « "\n";
00048     exit(0);
00049 }
00050 void CommandLineHandler::printCredits() {
00051     LOG_INFO « "Printing credits...";
00052     OUTPUT « BOLD « "Project information:\n"
00053            « RESET « "----------\n"
00054            « CYAN « BOLD « config::PROJECT_NAME « RESET « " v"
00055            « config::MAJOR_VERSION « "." « config::MINOR_VERSION « "."
00056            « config::PATCH_VERSION « "\n"
00057            « "\n"
00058            « config::DESCRIPTION « "\n"
00059            « "\n"
00060            « GREEN « "Authors: " « RESET « ITALIC « config::AUTHORS « RESET
00061            « "\n"
00062            « GREEN « "Documentation: " « RESET « ITALIC
00063            « config::HOMEPAGE_URL « RESET « GREEN « "\nContact: " « RESET
00064            « ITALIC « "simon21.blum@gmail.com" « "\n";
00065     exit(0);
00066 }
00067
00068 std::tuple<std::optional<std::string>, std::vector<std::string» CommandLineHandler::parseArguments(
00069             int argc, char* argv[]) {
00070     LOG_INFO « "Parsing arguments...";
00071     std::vector<std::string> files;
00072     std::optional<std::string> outDir;
00073
00074     while (true) {
00075         int optIndex = -1;
00076         struct option longOption = {};
00077         const auto result = getopt_long(argc, argv, "hvco:", options, &optIndex);
00078
00079         if (result == -1) {
00080             LOG_INFO « "End of options reached";
00081             break;
00082         }
00083
00084         switch (result) {
00085             case '?':
00086                 LOG_ERROR « "Invalid Option (argument)\n";
00087                 CommandLineHandler::printHelp();
00088
00089             case 'h':
00090                 LOG_INFO « "Help option detected";
00091                 CommandLineHandler::printHelp();
00092
00093             case 'v':
00094                 LOG_INFO « "Version option detected";
00095                 CommandLineHandler::printVersion();
00096
00097             case 'c':
00098                 LOG_INFO « "Credit option detected";
00099                 CommandLineHandler::printCredits();
00100
00101             case 'o':
00102                 LOG_INFO « "Output option detected";
00103                 outDir = optarg;
00104                 break;
00105
00106             case 0:
00107                 LOG_INFO « "Long option without short version detected";
00108                 longOption = options[optIndex];
00109                 LOG_INFO « "Option: " « longOption.name « " given";
00110
00111                 if (strcmp(longOption.name, "verbose") == 0) {
00112                     logging::setVerboseMode(true);
00113                     LOG_INFO « "Verbose mode activated";
00114                 }
00115
00116                 break;
00117
00118             default:
00119                 LOG_ERROR « "Default case for options reached!";
00120                 break;
00121         }
00122     }
00123
00124     LOG_INFO « "Options have been parsed";
00125     LOG_INFO « "Checking for arguments...";
00126
00127     while (optind < argc) {
00128         LOG_INFO « "Adding file: " « argv[optind];
00129         files.emplace_back(argv[optind++]);
00130     }
00131
00132     LOG_DEBUG « files.size();
```

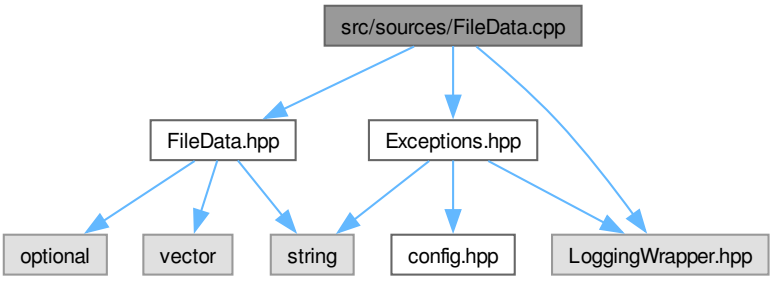
```
00133     LOG_INFO « "Arguments and options have been parsed";
00134     return {outDir, files};
00135 }
00136 } // namespace cli
```

10.24 src/sources/FileData.cpp File Reference

Implementation of the FileData class.

```
#include "FileData.hpp"
#include "Exceptions.hpp"
#include "LoggingWrapper.hpp"
```

Include dependency graph for FileData.cpp:



Namespaces

- namespace parsing

  The namespace containing everything relevant to parsing.

10.24.1 Detailed Description

Implementation of the FileData class.

Author

Elena Schwarzbach, Sonia Sinacci

Date

2024-04-26

Version

0.1.6

See also

src/include/FileData.hpp

Copyright

See LICENSE file

Definition in file FileData.cpp.

10.25 FileData.cpp

Go to the documentation of this file.

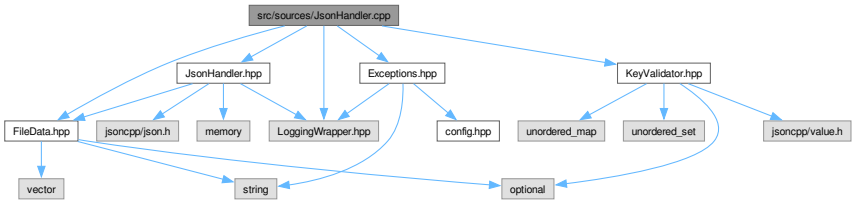
```
00001
00013 #include "FileData.hpp"
00014 #include "Exceptions.hpp"
00015 #include "LoggingWrapper.hpp"
00016
00017 namespace parsing {
00018 void FileData::setOutputFile(std::string &newOutputfile) {
00019     LOG_INFO « "Setting outputfile to...";
00020
00021     // If no value for key "outputfile"
00022     if (newOutputfile.empty()) {
00023         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00024         throw exceptions::InvalidValueException("outputfile",
00025                                                 "Outputfile can't be empty!");
00026     }
00027
00028     // If outputfile is already set
00029     if (!this->outputfile.empty()) {
00030         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00031         throw exceptions::InvalidValueException("outputfile",
00032                                                 "Outputfile is already set!");
00033     }
00034
00035     // If outputfile does not end with ".bat"
00036     if (!newOutputfile.ends_with(".bat")) {
00037         newOutputfile += ".bat";
00038         LOG_WARNING « "Outputfile does not end with \".bat\", adding it now: "
00039                     « newOutputfile;
00040     }
00041
00042     this->outputfile = newOutputfile;
00043     LOG_INFO « "Outputfile set to: " « this->outputfile « "\n";
00044 }
00045
00046 void FileData::setApplication(const std::string &newApplication) {
00047     if (newApplication.empty()) {
00048         LOG_INFO « "newApplication empty, returning";
00049         return;
00050     }
00051
00052     LOG_INFO « "Setting application to: " « newApplication « "\n";
00053     this->application.emplace(newApplication);
00054 }
00055
00056 void FileData::addCommand(const std::string &command) {
00057     if (command.empty()) {
00058         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00059         throw exceptions::InvalidValueException("command",
00060                                                 "Command value is empty!");
00061     }
00062
00063     LOG_INFO « "Adding command: " « command « "\n";
00064     this->commands.push_back(command);
00065 }
00066
00067 void FileData::addEnvironmentVariable(const std::string &name,
00068                                       const std::string &value) {
00069     if (name.empty()) {
00070         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00071         throw exceptions::InvalidValueException("name", "Name value is empty!");
00072     }
00073
00074     if (value.empty()) {
00075         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00076         throw exceptions::InvalidValueException("key", "Key value is empty");
00077     }
00078
00079     LOG_INFO « "Adding environment variable: " « name « "=" « value « "\n";
00080     this->environmentVariables.emplace_back(name, value);
00081 }
00082
00083 void FileData::addPathValue(const std::string &pathValue) {
00084     if (pathValue.empty()) {
00085         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00086         throw exceptions::InvalidValueException("path", "Path value is empty");
00087     }
00088
00089     LOG_INFO « "Adding path value: " « pathValue « "\n";
00090     this->pathValues.push_back(pathValue);
00091 }
00092 } // namespace parsing
```

10.26 src/sources/JsonHandler.cpp File Reference

Implementation of the JsonHandler class.

```
#include "JsonHandler.hpp"
#include "Exceptions.hpp"
#include "FileData.hpp"
#include "KeyValidator.hpp"
#include "LoggingWrapper.hpp"
```

Include dependency graph for JsonHandler.cpp:



Namespaces

- namespace parsing

  The namespace containing everything relevant to parsing.

10.26.1 Detailed Description

Implementation of the JsonHandler class.

Author

Elena Schwarzbach, Sonia Sinacci

Date

2024-04-16

Version

0.1.6

See also

src/include/JsonHandler.hpp

Copyright

See LICENSE file

Definition in file JsonHandler.cpp.

10.27 JsonHandler.cpp

Go to the documentation of this file.

```
00001
00013 #include "JsonHandler.hpp"
00014 #include "Exceptions.hpp"
00015 #include "FileData.hpp"
00016 #include "KeyValidator.hpp"
00017 #include "LoggingWrapper.hpp"
00018
00019 namespace parsing {
00020 JsonHandler::JsonHandler(const std::string &filename) {
00021     LOG_INFO << "Initializing JSONHandler with filename: " << filename << "\n";
00022     this->root = parseFile(filename);
00023 }
00024
00025 std::shared_ptr<Json::Value> JsonHandler::parseFile(const std::string &filename)
00026
00027 {
00028     LOG_INFO << "Parsing file: " << filename << "\n";
00029     std::ifstream file(filename);
00030     Json::Value newRoot;
00031
00032     // Json::Reader.parse() returns false if parsing fails
00033     if (Json::Reader reader; !reader.parse(file, newRoot)) {
00034         throw exceptions::ParsingException(filename);
00035     }
00036
00037     // Validate keys
00038     // Check for errors
00039     if (auto errors = KeyValidator::getInstance().validateKeys(newRoot, filename);
00040         !errors.empty()) {
00041         throw exceptions::InvalidKeyException(errors);
00042     }
00043
00044     LOG_INFO << "File \"" << filename << "\" has been parsed\n";
00045     return std::make_shared<Json::Value>(newRoot);
00046 }
00047
00048 std::shared_ptr<FileData> JsonHandler::getFileData() {
00049     LOG_INFO << "Creating FileData object for return...\n";
00050     return this->createFileData();
00051 }
00052
00053 std::shared_ptr<FileData> JsonHandler::createFileData() {
00054     LOG_INFO << "Creating FileData object...\n";
00055     this->data = std::make_shared<FileData>();
00056     this->assignOutputFile();
00057     this->assignHideShell();
00058     this->assignApplication();
00059     this->assignEntries();
00060     return this->data;
00061 }
00062
00063 void JsonHandler::assignOutputFile() const {
00064     LOG_INFO << "Assigning outputfile...\n";
00065     std::string outputFile = this->root->get("outputfile", "").asString();
00066     this->data->setOutputFile(outputFile);
00067 }
00068
00069 void JsonHandler::assignHideShell() const {
00070     LOG_INFO << "Assigning hide shell...\n";
00071     // If the 'hideshell' key is not given, it defaults to false
00072     this->data->setHideShell(this->root->get("hideshell", false).asBool());
00073 }
00074
00075 void JsonHandler::assignApplication() const {
00076     LOG_INFO << "Assigning application...\n";
00077     this->data->setApplication(this->root->get("application", "").asString());
00078 }
00079
00080 void JsonHandler::assignEntries() const {
00081     LOG_INFO << "Assigning entries...\n";
00082
00083     for (const auto &entry : this->root->get("entries", "")) {
00084         std::string entryType = entry.get("type", "").asString();
00085
00086         if (entryType == "EXE") {
00087             LOG_INFO << "Calling function to assign command...\n";
00088             this->assignCommand(entry);
00089         }
00090         else if (entryType == "ENV") {
00091             LOG_INFO << "Calling function to assign environment variable...\n";
00092             this->assignEnvironmentVariable(entry);
00093         }
```

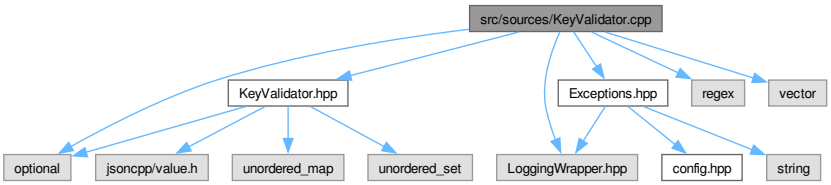
```
00094         else if (entryType == "PATH") {
00095             LOG_INFO « "Calling function to assign path value...\n";
00096             this->assignPathValue(entry);
00097         }
00098         else {
00099             // Due to validation beforehand - this should never be reached!
00100             throw exceptions::UnreachableCodeException(
00101                         "Unknown entries should be caught by KeyValidator!\nPlease report "
00102                         "this bug!");
00103         }
00104     }
00105 }
00106
00107 void JsonHandler::assignCommand(const Json::Value &entry) const {
00108     LOG_INFO « "Assigning command...\n";
00109     this->data->addCommand(entry.get("command", "").asString());
00110 }
00111
00112 void JsonHandler::assignEnvironmentVariable(const Json::Value &entry) const {
00113     LOG_INFO « "Assigning environment variable...\n";
00114     std::string key = entry.get("key", "").asString();
00115     std::string value = entry.get("value", "").asString();
00116     this->data->addEnvironmentVariable(key, value);
00117 }
00118
00119 void JsonHandler::assignPathValue(const Json::Value &entry) const {
00120     LOG_INFO « "Assigning path value...\n";
00121     this->data->addPathValue(entry.get("path", "").asString());
00122 }
00123 } // namespace parsing
```

10.28 src/sources/KeyValidator.cpp File Reference

Implementation for the KeyValidator class.

```
#include "KeyValidator.hpp"
#include "Exceptions.hpp"
#include "LoggingWrapper.hpp"
#include <optional>
#include <regex>
#include <vector>
```

Include dependency graph for KeyValidator.cpp:



Namespaces

- namespace parsing

  The namespace containing everything relevant to parsing.

10.28.1 Detailed Description

Implementation for the KeyValidator class.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

See also

src/include/KeyValidator.hpp

Copyright

See LICENSE file

Definition in file KeyValidator.cpp.

10.29 KeyValidator.cpp

Go to the documentation of this file.

```
00001
00012 #include "KeyValidator.hpp"
00013 #include "Exceptions.hpp"
00014 #include "LoggingWrapper.hpp"
00015 #include <optional>
00016 #include <regex>
00017 #include <vector>
00018
00019 namespace parsing {
00020 KeyValidator &KeyValidator::getInstance() {
00021     static KeyValidator keyValidator;
00022     LOG_INFO « "Returning KeyValidator instance!";
00023     return keyValidator;
00024 }
00025
00026 std::vector<std::tuple<int, std::string» KeyValidator::validateKeys(
00027             const Json::Value &root,
00028             const std::string &filename) {
00029     std::vector<std::tuple<int, std::string» wrongKeys =
00030                 getWrongKeys(root, filename);
00031
00032     // Inline declaration to prevent leaking in outer scope
00033     for (Json::Value entries = root.get("entries", "");
00034          const auto &entry : entries) {
00035         const auto entryKeys = entry.getMemberNames();
00036         // Create a set of the entry keys for faster lookup (O(1) instead of O(n))
00037         std::unordered_set<std::string> entryKeysSet(entryKeys.begin(),
00038                                                      entryKeys.end());
00039         const auto wrongEntries = validateEntries(filename, entryKeysSet);
00040         // Combine wrong keys
00041         wrongKeys.insert(wrongKeys.end(), wrongEntries.begin(), wrongEntries.end());
00042         // Validate that each entry has it's necessary keys
00043         validateTypes(filename, entry, entryKeysSet);
00044     }
```

```
00045
00046     return wrongKeys;
00047 }
00048
00049 std::vector<std::tuple<int, std::string» KeyValidator::getWrongKeys(
00050             const Json::Value &root,
00051             const std::string &filename) const {
00052     std::vector<std::tuple<int, std::string» wrongKeys = {};
00053
00054     for (const auto &key : root.getMemberNames()) {
00055         if (!validKeys.contains(key)) {
00056             const auto error = getUnknownKeyLine(filename, key);
00057
00058             if (!error.has_value()) {
00059                 LOG_ERROR « "Unable to find line of wrong key!";
00060                 continue;
00061             }
00062
00063             // If the line can't be found, add -1 as line number
00064             wrongKeys.emplace_back(error.value_or(-1), key);
00065         }
00066     }
00067
00068     return wrongKeys;
00069 }
00070
00071 std::vector<std::tuple<int, std::string» KeyValidator::validateEntries(
00072             const std::string &filename,
00073             const std::unordered_set<std::string> &entryKeys) const {
00074     std::vector<std::tuple<int, std::string» wrongKeys = {};
00075
00076     for (const auto &key : entryKeys) {
00077         if (!validEntryKeys.contains(key)) {
00078             const auto error = getUnknownKeyLine(filename, key);
00079
00080             if (!error.has_value()) {
00081                 LOG_ERROR « "Unable to find line of wrong key!";
00082                 continue;
00083             }
00084
00085             wrongKeys.emplace_back(error.value_or(-1), key);
00086         }
00087     }
00088
00089     return wrongKeys;
00090 }
00091
00092 void KeyValidator::validateTypes(
00093             const std::string &filename, const Json::Value &entry,
00094             const std::unordered_set<std::string> &entryKeys) {
00095     // Gett the type of the entry - error if not found
00096     const std::string type = entry.get("type", "ERROR").asString();
00097
00098     // If the type is not found, throw an exception
00099     if (type == "ERROR") {
00100         throw exceptions::MissingTypeException();
00101         // If the type is not known, throw an exception
00102         // @note This should already have been checked
00103     }
00104     else if (typeToKeys.contains(type)) {
00105         const std::optional<int> line =
00106                     getUnknownKeyLine(filename, std::string(type));
00107
00108         if (!line.has_value()) {
00109             LOG_INFO « "Unable to find line of wrong type!";
00110         }
00111
00112         throw exceptions::InvalidTypeException(std::string(type), line.value());
00113         // If the type is known, check if all necessary keys are present
00114     }
00115     else {
00116         for (const auto &key : typeToKeys[type]) {
00117             if (entryKeys.contains(key)) {
00118                 throw exceptions::MissingKeyException(key, std::string(type));
00119             }
00120         }
00121     }
00122 }
00123
00124 std::optional<int> KeyValidator::getUnknownKeyLine(const std::string &filename,
00125                                                    const std::string &wrongKey) {
00126     std::ifstream file(filename);
00127
00128     if (!file.is_open()) {
00129         LOG_ERROR « "File not open!";
00130         return std::nullopt;
00131     }
```

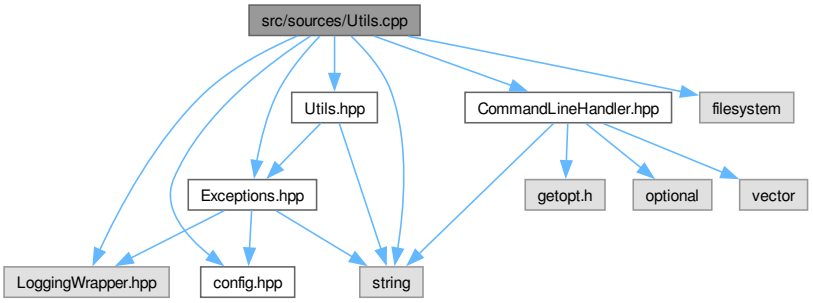
```
00132
00133     std::string line;
00134     // Create a regex pattern that matches the wrong key whole word
00135     const std::regex wrongKeyPattern("\\b" + wrongKey + "\\b");
00136
00137     for (int lineNumber = 1; std::getline(file, line); ++lineNumber) {
00138         if (std::regex_search(line, wrongKeyPattern)) {
00139             return lineNumber;
00140         }
00141     }
00142
00143     return std::nullopt;
00144 }
00145
00146 } // namespace parsing
```

10.30 src/sources/Utils.cpp File Reference

Implementation for the Utils class.

```
#include "Utils.hpp"
#include "CommandLineHandler.hpp"
#include "Exceptions.hpp"
#include "config.hpp"
#include <LoggingWrapper.hpp>
#include <filesystem>
#include <string>
```

Include dependency graph for Utils.cpp:



Namespaces

- namespace utilities

  Includes all utilities.

10.30.1 Detailed Description

Implementation for the Utils class.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

This file includes the implementation for the Utils class.

See also

src/include/utility/Utilities.hpp

Copyright

See LICENSE file

Definition in file Utils.cpp.

10.31 Utils.cpp

Go to the documentation of this file.

```
00001
00015 #include "Utils.hpp"
00016 #include "CommandLineHandler.hpp"
00017 #include "Exceptions.hpp"
00018 #include "config.hpp"
00019
00020 #include <LoggingWrapper.hpp>
00021 #include <filesystem>
00022 #include <string>
00023
00024 namespace utilities {
00025 void Utils::setupEasyLogging(const std::string &configFile) {
00026     el::Configurations conf(configFile);
00027     el::Loggers::reconfigureAllLoggers(conf);
00028     LOG_INFO « "Running " « config::PROJECT_NAME « " v"
00029              « config::MAJOR_VERSION « "." « config::MINOR_VERSION « "."
00030              « config::PATCH_VERSION;
00031     LOG_INFO « "For more Information checkout " « config::HOMEPAGE_URL;
00032     LOG_INFO « "EasyLogging has been setup!";
00033 }
00034 bool Utils::askToContinue(const std::string &prompt) {
00035     std::string userInput;
00036     LOG_INFO « "Asking for user Confirmation to continue...";
00037     OUTPUT « cli::BOLD « prompt « cli::RESET;
00038
00039     do {
00040         std::cin » userInput;
00041         std::ranges::transform(userInput, userInput.begin(), ::tolower);
00042
00043         if (userInput != "y" && userInput != "yes" && userInput != "n" &&
00044             userInput != "no") {
00045             LOG_INFO « "Wrong user input!";
00046             OUTPUT « cli::ITALIC « "Please enter Y/Yes or N/No!\n" « cli::RESET;
00047             continue;
00048         }
00049
00050         break;
00051     } while (true);
00052
00053     return userInput == "y" || userInput == "yes";
00054 }
00055 void Utils::checkConfigFile(const std::string &configFile) {
00056     if (!std::filesystem::is_regular_file(configFile)) {
00057         std::cerr « cli::RED « cli::BOLD
00058                  « "Fatal: Easylogging configuration file not found at:\n"
00059                  « cli::RESET « cli::ITALIC « "\n\t\"" « configFile « "\"\n\n"
```

```
00060                   « cli::RESET;
00061         std::cout « "Aborting...\n";
00062         exit(1);
00063     }
00064 }
00065 const std::string &Utils::checkDirectory(std::string &directory) {
00066     if (!directory.empty() && directory.back() != '/' &&
00067         directory.back() != '\\') {
00068         directory += '/';
00069     }
00070
00071     if (!std::filesystem::exists(directory)) {
00072         throw exceptions::NoSuchDirException(directory);
00073     }
00074
00075     return directory;
00076 }
00077 bool Utils::handleParseException(const exceptions::CustomException &e,
00078                                  const std::vector<std::string>::iterator &file,
00079                                  const std::vector<std::string> &files) {
00080     OUTPUT « "\nThere has been a error while trying to parse \"" « *file
00081            « ":\n";
00082     LOG_ERROR « e.what();
00083
00084     if (std::next(file) != files.end() &&
00085         !utilities::Utils::askToContinue(
00086                     "Do you want to continue with the other files? (y/n) "
00087                     "")) {
00088         OUTPUT « "Aborting...";
00089         LOG_INFO « "Application ended by user Input";
00090         return false;
00091     }
00092
00093     std::cout « std::endl;
00094     return true;
00095 }
00096
00097 } // namespace utilities
```

Index