JSON2Batch

0.2.2

Generated on Fri Apr 26 2024 13:02:13 for JSON2Batch by Doxygen 1.9.8

Fri Apr 26 2024 13:02:13

Chapter 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 Build instructions

1.1.1 Linux

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

1.1.2 Windows

@TODO Fix Windows

1.2 Documentation

The documentation for this project can be found here.

1.3 License

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

1.4 Other

Chapter 2

Todo List

Member exceptions::FailedToOpenFileException::FailedToOpenFileException (const std::string &file)

Documentation

Member exceptions::NoSuchDirException::NoSuchDirException (const std::string &dir)

Documentation

Chapter 3

Topic Index

3.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Topic Documentation

8.1 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 9

Namespace Documentation

9.1 cli Namespace Reference

Includes everything regarding the CLI.

Classes

- class CommandLineHandler

  Responsible for the Command Line Interface.

Variables

- static const struct option options [ ]

9.1.1 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

9.1.2 Variable Documentation

9.1.2.1 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.

9.2 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
- constexpr auto HOMEPAGE_URL

9.2.1 Detailed Description

Namespace used for general project information.

9.2.2 Variable Documentation

9.2.2.1 AUTHORS

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

Initial value:

```
=
            "Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci"
```

Definition at line 34 of file config.hpp.

9.2.2.2 DESCRIPTION

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

Definition at line 32 of file config.hpp.

9.2.2.3 EXECUTABLE_NAME

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

Definition at line 28 of file config.hpp.

9.2.2.4 HOMEPAGE_URL

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

Initial value:

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

Definition at line 36 of file config.hpp.

9.2.2.5 LOG_CONFIG

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

Initial value:

```
= "/home/simon/1_Coding/projectJsonToBat/"
                                  "build/Release/config/easylogging.conf"
```

Definition at line 26 of file config.hpp.

9.2.2.6 MAJOR_VERSION

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

Definition at line 29 of file config.hpp.

9.2.2.7 MINOR_VERSION

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

Definition at line 30 of file config.hpp.

9.2.2.8 PATCH_VERSION

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

Definition at line 31 of file config.hpp.

9.2.2.9 PROJECT_NAME

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

Definition at line 33 of file config.hpp.

9.3 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.

9.3.1 Detailed Description

Namespace used for customized exceptions.

9.4 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.

9.4.1 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

9.5 utilities Namespace Reference

Includes all utilities.

Classes

- class Utils

  Responsible for utility function.

9.5.1 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 10

Class Documentation

10.1 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 ()
  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

10.1.1 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.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 BatchCreator()

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

Initializes the BatchCreator.

Creates a stringstream and calls the createBatch() function

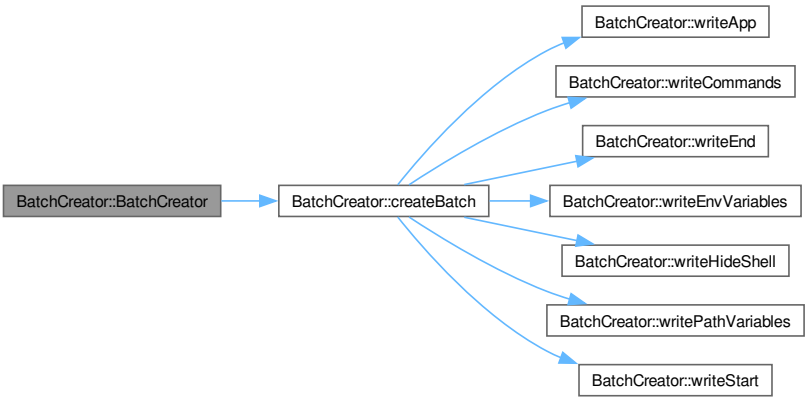
Parameters

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:



10.1.3 Member Function Documentation

10.1.3.1 createBatch()

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

Creates the batch stream.

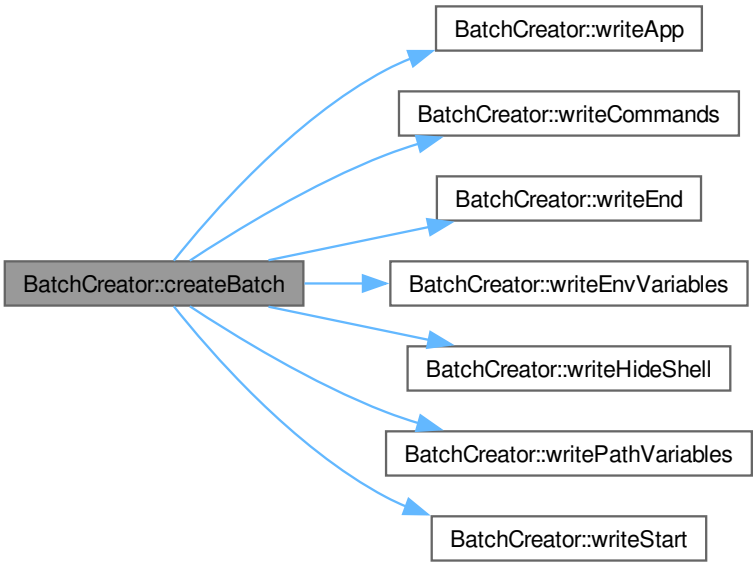
< FileData object

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

Definition at line 26 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:

10.1.3.2 getDataStream()

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

Returns the stringstream.

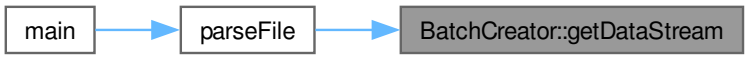
Returns

A shared pointer to the stringstream

Definition at line 46 of file BatchCreator.hpp.

References dataStream.

Here is the caller graph for this function:



10.1.3.3 writeApp()

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

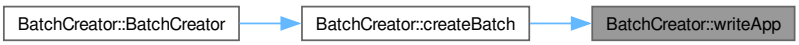
If an application is given, it is started at the end.

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

Definition at line 87 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:



10.1.3.4 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 56 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:

10.1.3.5 writeEnd()

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

Writes the end of the batch file.

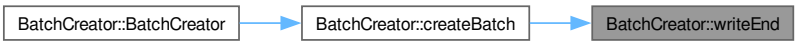
Writes the end of the batch file, which is always the same:

- @ECHO ON

Definition at line 103 of file BatchCreator.cpp.

References dataStream.

Here is the caller graph for this function:



10.1.3.6 writeEnvVariables()

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

Set's environment variables.

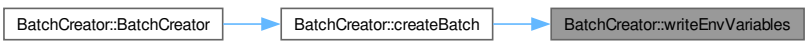
Set's the envirment variables for the batch. Those originiate from the "ENV" entry in the json file with the following syntax:

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

Definition at line 66 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:



10.1.3.7 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 44 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:

10.1.3.8 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 75 of file BatchCreator.cpp.

References dataStream, and fileData.

Here is the caller graph for this function:

10.1.3.9 writeStart()

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

Wirtes the start of the batch file.

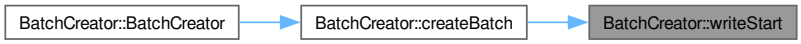
Writes the start of the batch file, which is always the same:

- setzt ECHO off
- startet cmd.exe

Definition at line 38 of file BatchCreator.cpp.

References dataStream.

Here is the caller graph for this function:



10.1.4 Member Data Documentation

10.1.4.1 dataStream

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

Definition at line 52 of file BatchCreator.hpp.

10.1.4.2 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

10.2 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.

10.2.1 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.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 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.

10.2.2.2 ~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.

10.2.3 Member Function Documentation

10.2.3.1 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

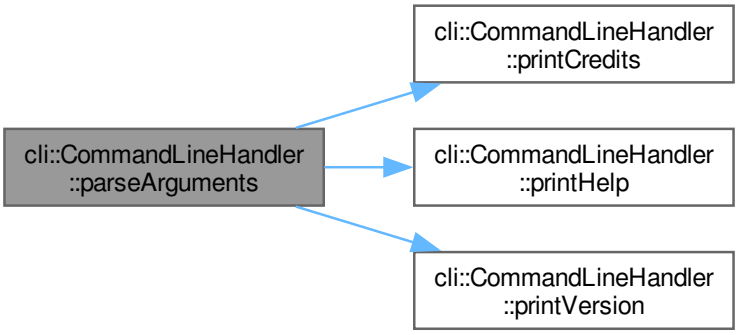
Returns

Returns a tuple containing the output directory and the files

Definition at line 72 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:

10.2.3.2 printCredits()

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

Prints the credits message.

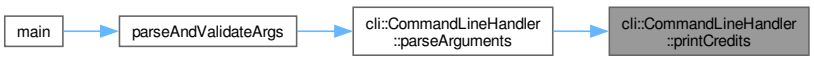
Note

This function ends the application.

Definition at line 52 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:



10.2.3.3 printHelp()

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

Prints the help message.

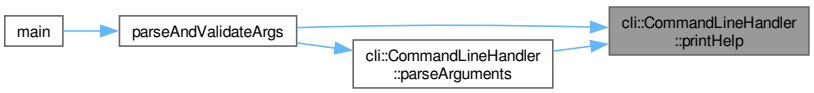
Note

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:



10.2.3.4 printVersion()

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

Prints the version message.

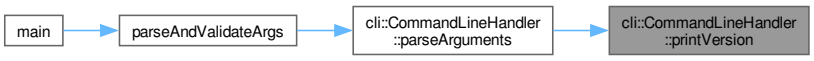
Note

This function ends the application.

Definition at line 45 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:

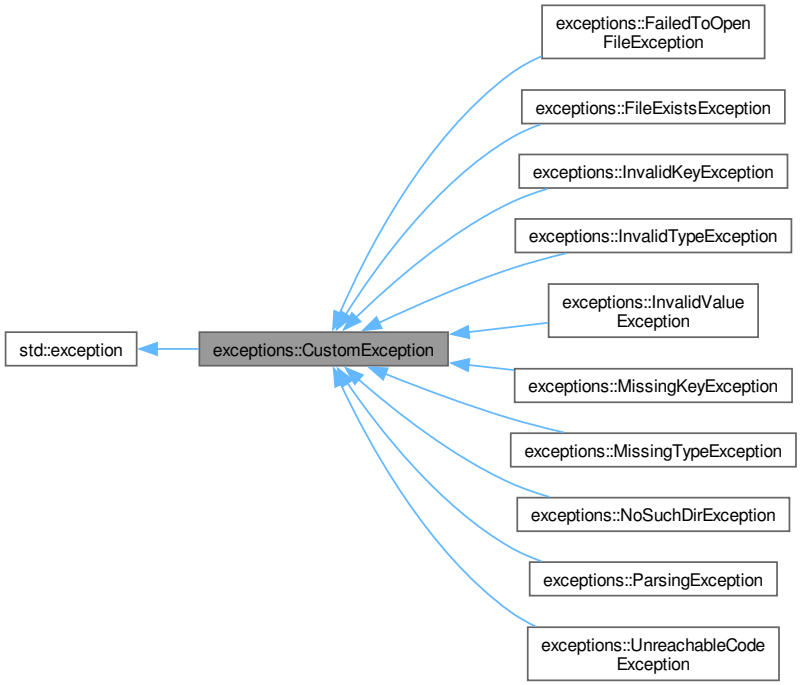
- src/include/CommandLineHandler.hpp
- src/sources/CommandLineHandler.cpp

10.3 exceptions::CustomException Class Reference

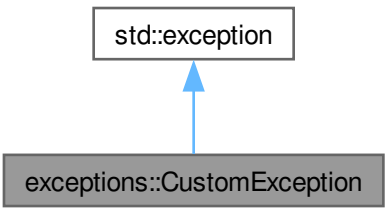
Base class for all custom exceptions.

`#include <Exceptions.hpp>`

Inheritance diagram for exceptions::CustomException:



Collaboration diagram for exceptions::CustomException:



Public Member Functions

- const char ∗ what () const noexcept override

10.3.1 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 31 of file Exceptions.hpp.

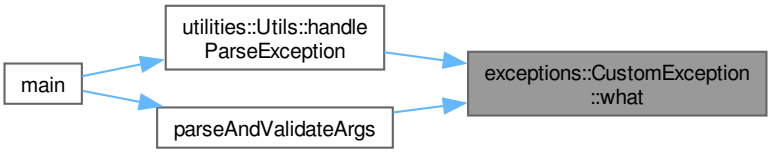
10.3.2 Member Function Documentation

10.3.2.1 what()

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

Definition at line 33 of file Exceptions.hpp.

Here is the caller graph for this function:



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

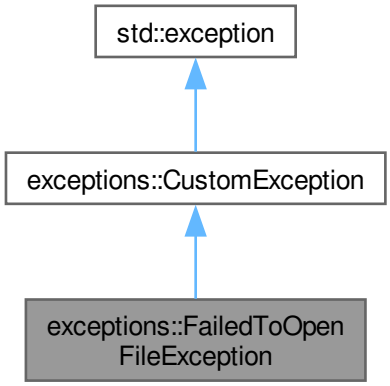
- src/include/Exceptions.hpp

10.4 exceptions::FailedToOpenFileException Class Reference

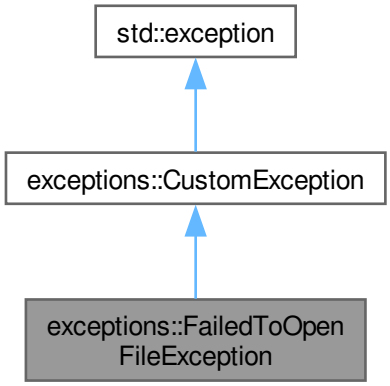
Exception for when a file can't be opened.

```
#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

10.4.1 Detailed Description

Exception for when a file can't be opened.

Definition at line 255 of file Exceptions.hpp.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 FailedToOpenFileException()

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

Todo Documentation

Definition at line 261 of file Exceptions.hpp.

References message.

10.4.3 Member Function Documentation

10.4.3.1 what()

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

Definition at line 265 of file Exceptions.hpp.

References message.

10.4.4 Member Data Documentation

10.4.4.1 message

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

Definition at line 257 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

10.5 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

10.5.1 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.

10.5.2 Member Function Documentation

10.5.2.1 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 58 of file FileData.cpp.

References commands.

10.5.2.2 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 70 of file FileData.cpp.

References environmentVariables.

10.5.2.3 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 87 of file FileData.cpp.

References pathValues.

10.5.2.4 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.

10.5.2.5 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.

10.5.2.6 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.

10.5.2.7 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.

10.5.2.8 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.

10.5.2.9 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.

10.5.2.10 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 47 of file FileData.cpp.

References application.

10.5.2.11 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.

10.5.2.12 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.

10.5.3 Member Data Documentation

10.5.3.1 application

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

Definition at line 153 of file FileData.hpp.

10.5.3.2 commands

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

Definition at line 154 of file FileData.hpp.

10.5.3.3 environmentVariables

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

Definition at line 156 of file FileData.hpp.

10.5.3.4 hideShell

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

Definition at line 152 of file FileData.hpp.

10.5.3.5 outputfile

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

Definition at line 151 of file FileData.hpp.

10.5.3.6 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:

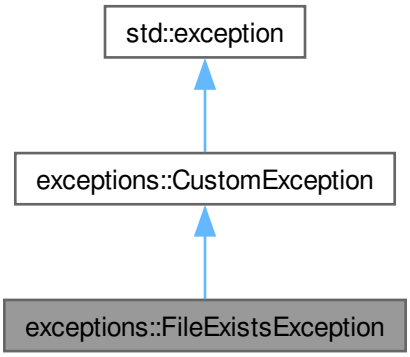
- src/include/FileData.hpp
- src/sources/FileData.cpp

10.6 exceptions::FileExistsException Class Reference

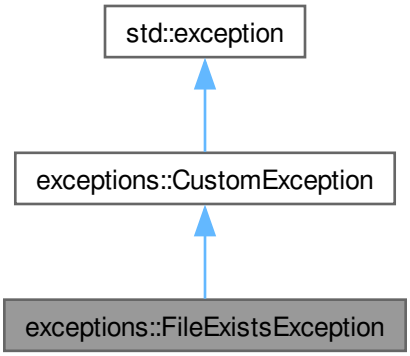
Exception for an already exisiting outputfile.

`#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

10.6.1 Detailed Description

Exception for an already exisiting outputfile.

Definition at line 70 of file Exceptions.hpp.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 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 76 of file Exceptions.hpp.

References file, and message.

10.6.3 Member Function Documentation

10.6.3.1 what()

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

Definition at line 88 of file Exceptions.hpp.

References message.

10.6.4 Member Data Documentation

10.6.4.1 file

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

Definition at line 72 of file Exceptions.hpp.

10.6.4.2 message

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

Definition at line 73 of file Exceptions.hpp.

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

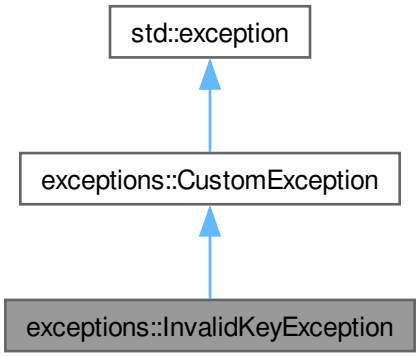
- src/include/Exceptions.hpp

10.7 exceptions::InvalidKeyException Class Reference

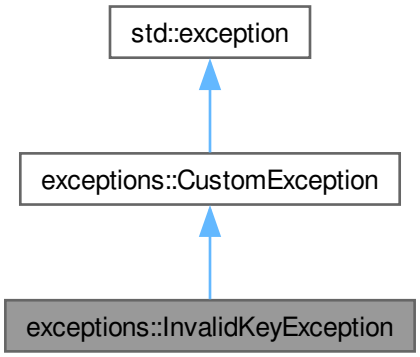
Exception for invalid keys.

```
#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!"

10.7.1 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 131 of file Exceptions.hpp.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 InvalidKeyException()

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

Definition at line 136 of file Exceptions.hpp.

References message.

10.7.3 Member Function Documentation

10.7.3.1 what()

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

Definition at line 145 of file Exceptions.hpp.

References message.

10.7.4 Member Data Documentation

10.7.4.1 message

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

Definition at line 133 of file Exceptions.hpp.

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

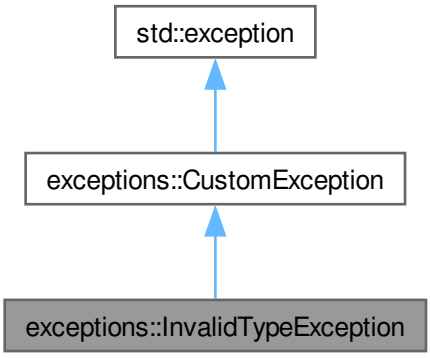
- src/include/Exceptions.hpp

10.8 exceptions::InvalidTypeException Class Reference

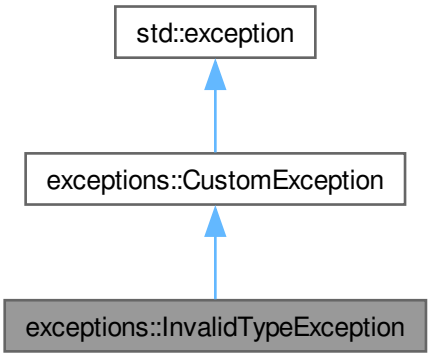
Exception for invalid types.

`#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

10.8.1 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 158 of file Exceptions.hpp.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 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 164 of file Exceptions.hpp.

References message, and type.

10.8.3 Member Function Documentation

10.8.3.1 what()

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

Definition at line 175 of file Exceptions.hpp.

References message.

10.8.4 Member Data Documentation

10.8.4.1 message

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

Definition at line 161 of file Exceptions.hpp.

10.8.4.2 type

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

Definition at line 160 of file Exceptions.hpp.

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

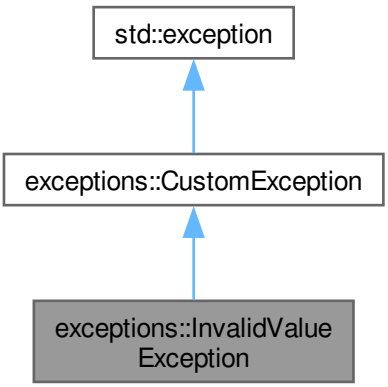
- src/include/Exceptions.hpp

10.9 exceptions::InvalidValueException Class Reference

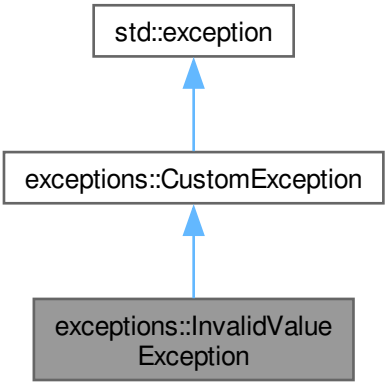
Exception for an ivalid (usually empty) value field.

```
#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

10.9.1 Detailed Description

Exception for an ivalid (usually empty) value field.

Definition at line 97 of file Exceptions.hpp.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 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 103 of file Exceptions.hpp.

References key, and message.

10.9.3 Member Function Documentation

10.9.3.1 what()

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

Definition at line 115 of file Exceptions.hpp.

References message.

10.9.4 Member Data Documentation

10.9.4.1 key

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

Definition at line 99 of file Exceptions.hpp.

10.9.4.2 message

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

Definition at line 100 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

10.10 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

10.10.1 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.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 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.

10.10.2.2 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:

10.10.3 Member Function Documentation

10.10.3.1 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 80 of file JsonHandler.cpp.

References data, and root.

Here is the caller graph for this function:

10.10.3.2 assignCommand()

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

Assigns an command to this->data.

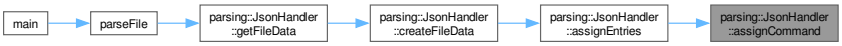
Parameters

entry	The entry with the command

Definition at line 114 of file JsonHandler.cpp.

References data.

Here is the caller graph for this function:



10.10.3.3 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

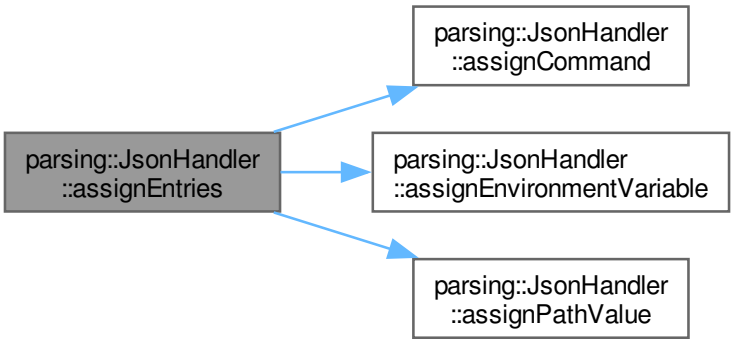
Exceptions

exceptions::UnreachableCodeException	

Definition at line 86 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:

10.10.3.4 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 120 of file JsonHandler.cpp.

References data.

Here is the caller graph for this function:

10.10.3.5 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 73 of file JsonHandler.cpp.

References data, and root.

Here is the caller graph for this function:

10.10.3.6 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 66 of file JsonHandler.cpp.

References data, and root.

Here is the caller graph for this function:

10.10.3.7 assignPathValue()

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

Assigns a path value to this->data.

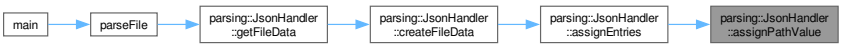
Parameters

entry	The entry with the path value

Definition at line 128 of file JsonHandler.cpp.

References data.

Here is the caller graph for this function:



10.10.3.8 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.

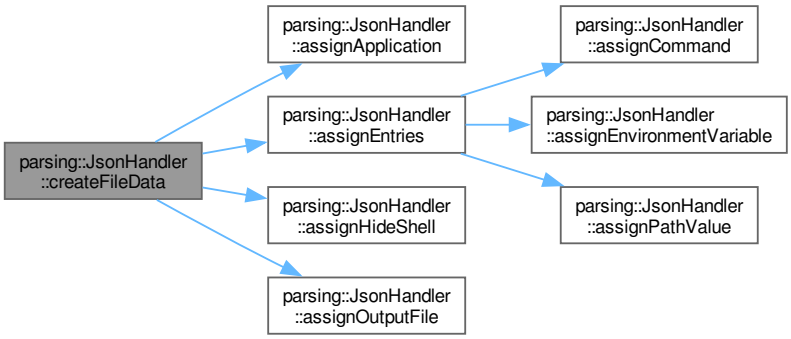
Returns

Pointer to the created instance of FileData

Definition at line 55 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:

10.10.3.9 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.

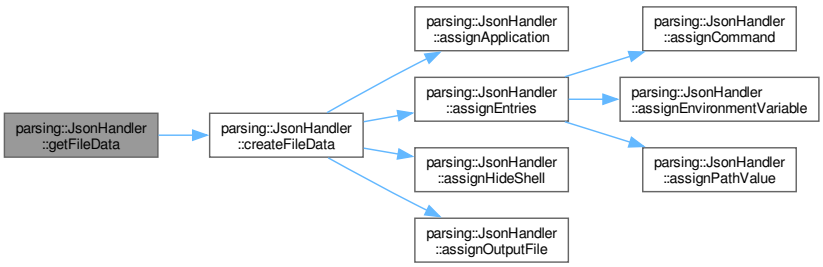
Returns

Pointer to the FileData Object with the parsed data from json

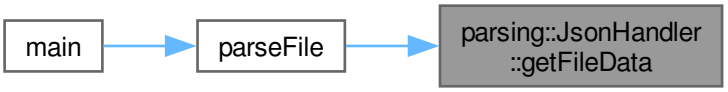
Definition at line 49 of file JsonHandler.cpp.

References createFileData().

Here is the call graph for this function:



Here is the caller graph for this function:



10.10.3.10 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()

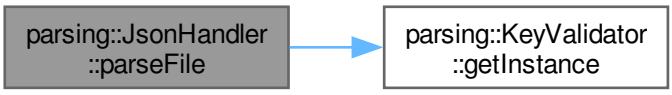
Exceptions

exceptions::ParsingException	
exceptions::InvalidKeyException	

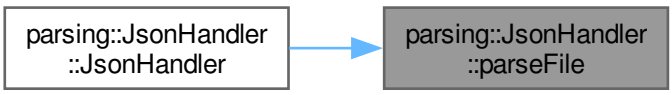
Definition at line 26 of file JsonHandler.cpp.

References parsing::KeyValidator::getInstance().

Here is the call graph for this function:



Here is the caller graph for this function:



10.10.4 Member Data Documentation

10.10.4.1 data

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

Definition at line 155 of file JsonHandler.hpp.

10.10.4.2 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

10.11 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

10.11.1 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.

10.11.2 Member Function Documentation

10.11.2.1 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:

10.11.2.2 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

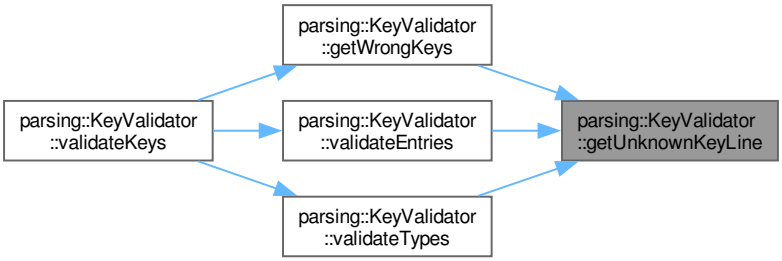
filename	The filename which should contain the key
wrongKey	The key to be searched for

Returns

The line of the key, if it was found

Definition at line 129 of file KeyValidator.cpp.

Here is the caller graph for this function:



10.11.2.3 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.

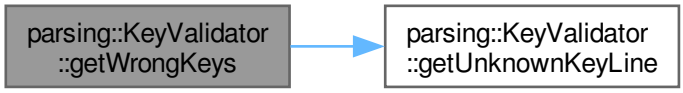
Returns

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

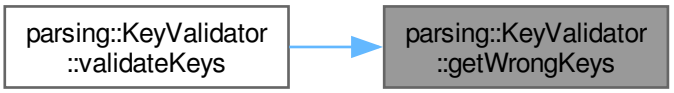
Definition at line 51 of file KeyValidator.cpp.

References getUnknownKeyLine(), and validKeys.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.2.4 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

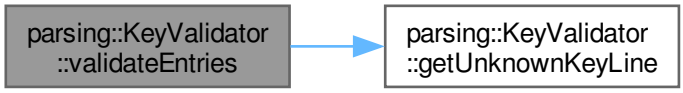
Returns

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

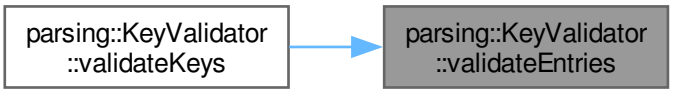
Definition at line 74 of file KeyValidator.cpp.

References getUnknownKeyLine(), and validEntryKeys.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.2.5 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.

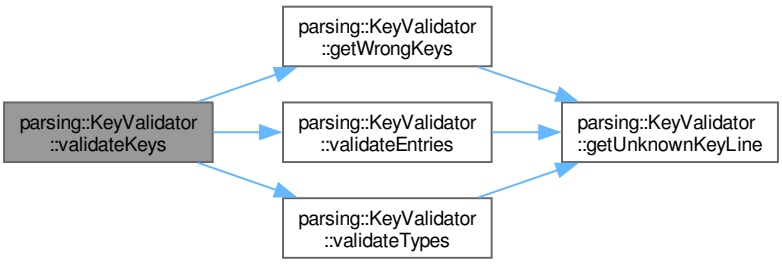
Returns

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

Definition at line 27 of file KeyValidator.cpp.

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

Here is the call graph for this function:



10.11.2.6 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

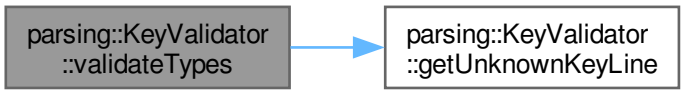
Exceptions

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

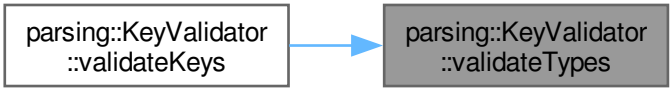
Definition at line 96 of file KeyValidator.cpp.

References getUnknownKeyLine(), and typeToKeys.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3 Member Data Documentation

10.11.3.1 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.

10.11.3.2 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.

10.11.3.3 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:

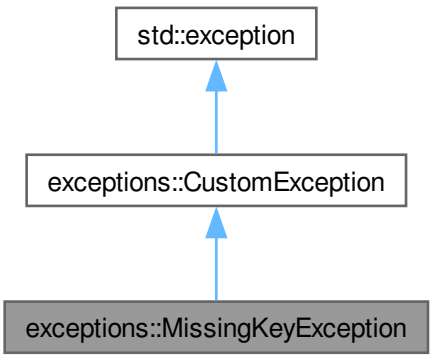
- src/include/KeyValidator.hpp
- src/sources/KeyValidator.cpp

10.12 exceptions::MissingKeyException Class Reference

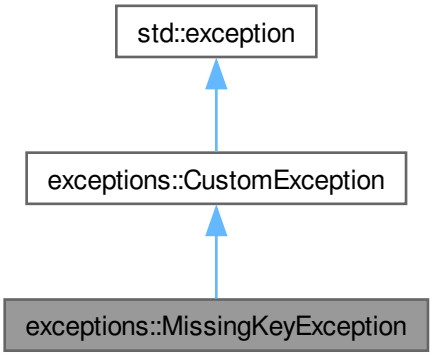
Exception for missing keys within entries.

```
#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

10.12.1 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 187 of file Exceptions.hpp.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 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 194 of file Exceptions.hpp.

References key, message, and type.

10.12.3 Member Function Documentation

10.12.3.1 what()

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

Definition at line 206 of file Exceptions.hpp.

References message.

10.12.4 Member Data Documentation

10.12.4.1 key

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

Definition at line 191 of file Exceptions.hpp.

10.12.4.2 message

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

Definition at line 189 of file Exceptions.hpp.

10.12.4.3 type

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

Definition at line 190 of file Exceptions.hpp.

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

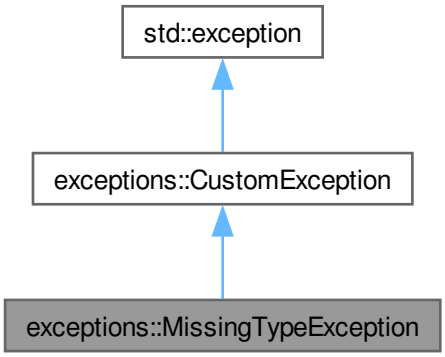
- src/include/Exceptions.hpp

10.13 exceptions::MissingTypeException Class Reference

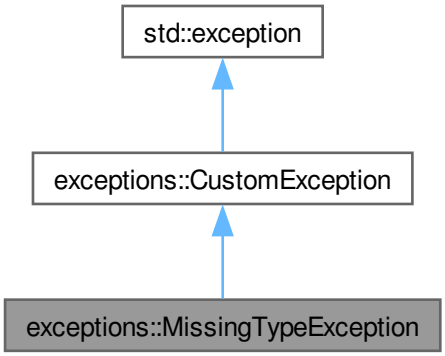
Exception for missing types of entries.

`#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!"

10.13.1 Detailed Description

Exception for missing types of entries.

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

Definition at line 217 of file Exceptions.hpp.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 MissingTypeException()

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

Definition at line 222 of file Exceptions.hpp.

References message.

10.13.3 Member Function Documentation

10.13.3.1 what()

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

Definition at line 225 of file Exceptions.hpp.

References message.

10.13.4 Member Data Documentation

10.13.4.1 message

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

Definition at line 219 of file Exceptions.hpp.

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

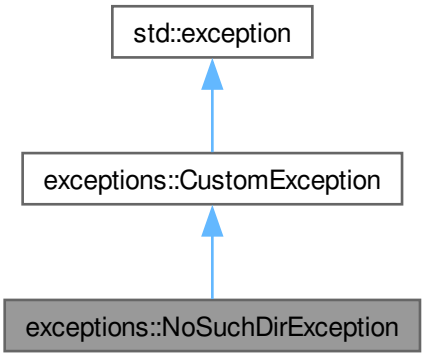
- src/include/Exceptions.hpp

10.14 exceptions::NoSuchDirException Class Reference

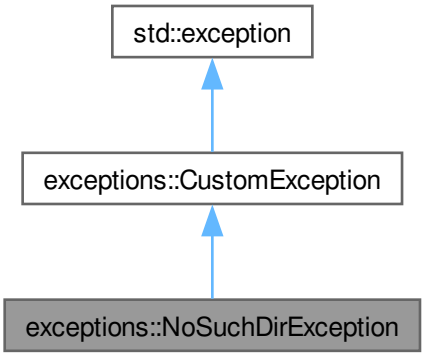
Exception for when a directory does not exist.

`#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

10.14.1 Detailed Description

Exception for when a directory does not exist.

Definition at line 274 of file Exceptions.hpp.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 NoSuchDirException()

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

Todo Documentation

Definition at line 280 of file Exceptions.hpp.

References message.

10.14.3 Member Function Documentation

10.14.3.1 what()

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

Definition at line 284 of file Exceptions.hpp.

References message.

10.14.4 Member Data Documentation

10.14.4.1 message

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

Definition at line 276 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

10.15 options Struct Reference

The struct containing all possible options.

`#include <CommandLineHandler.hpp>`

10.15.1 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:

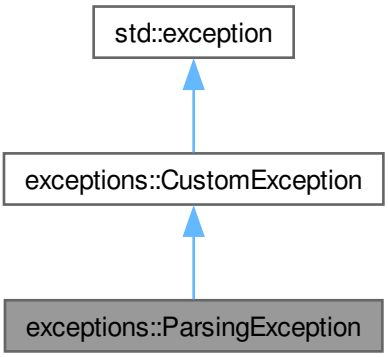
- src/include/CommandLineHandler.hpp

10.16 exceptions::ParsingException Class Reference

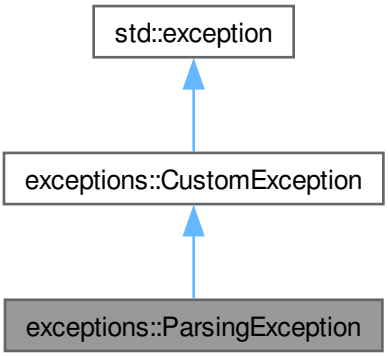
Exception for syntax errors within the json file.

`#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

10.16.1 Detailed Description

Exception for syntax errors within the json file.

Definition at line 42 of file Exceptions.hpp.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 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 48 of file Exceptions.hpp.

References file, and message.

10.16.3 Member Function Documentation

10.16.3.1 what()

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

Definition at line 61 of file Exceptions.hpp.

References message.

10.16.4 Member Data Documentation

10.16.4.1 file

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

Definition at line 44 of file Exceptions.hpp.

10.16.4.2 message

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

Definition at line 45 of file Exceptions.hpp.

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

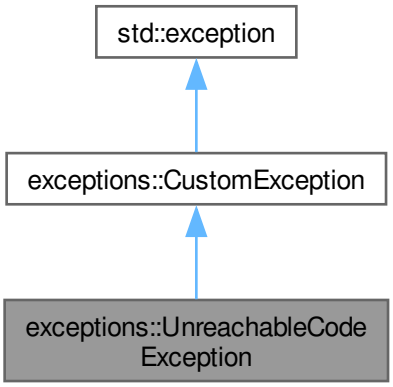
- src/include/Exceptions.hpp

10.17 exceptions::UnreachableCodeException Class Reference

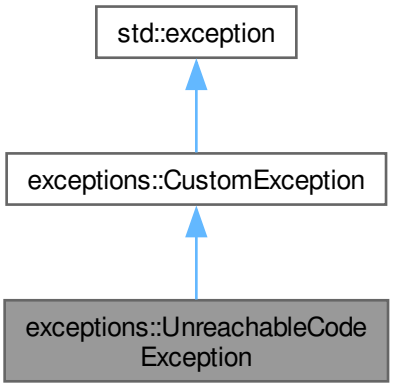
Exception for when the application reaches code it shouldn't reach.

```
#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

10.17.1 Detailed Description

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

Definition at line 234 of file Exceptions.hpp.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 UnreachableCodeException()

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

Definition at line 239 of file Exceptions.hpp.

References config::EXECUTABLE_NAME, and message.

10.17.3 Member Function Documentation

10.17.3.1 what()

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

Definition at line 246 of file Exceptions.hpp.

References message.

10.17.4 Member Data Documentation

10.17.4.1 message

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

Definition at line 236 of file Exceptions.hpp.

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

- src/include/Exceptions.hpp

10.18 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.

10.18.1 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.

10.18.2 Member Function Documentation

10.18.2.1 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

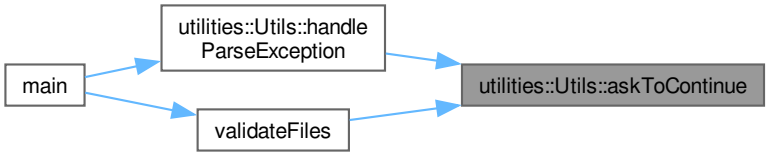
prompt	(Optional) A custom prompt to be used.

Returns

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

Definition at line 35 of file Utils.cpp.

Here is the caller graph for this function:



10.18.2.2 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 57 of file Utils.cpp.

Here is the caller graph for this function:

10.18.2.3 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 68 of file Utils.cpp.

Here is the caller graph for this function:

10.18.2.4 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

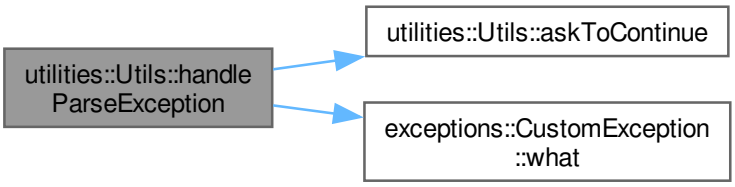
Returns

Returns true if the user wants to continue and false otherwise

Definition at line 81 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:

10.18.2.5 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.

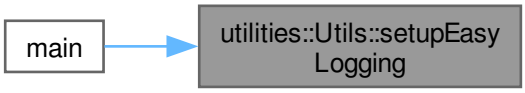
Parameters

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 11

File Documentation

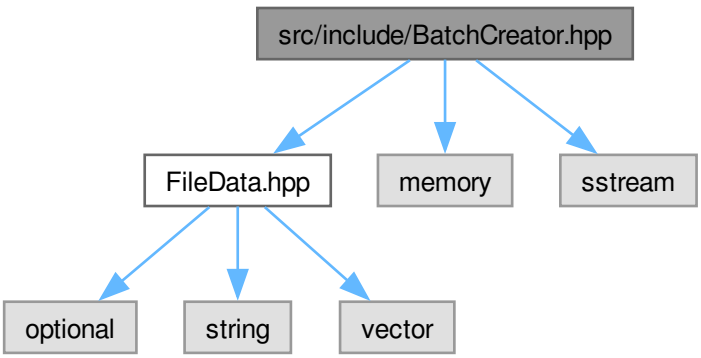
11.1 README.md File Reference

11.2 src/include/BatchCreator.hpp File Reference

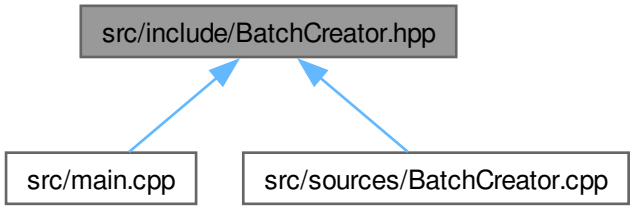
Contains the BatchCreator class.

```
#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.

11.2.1 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.

11.3 BatchCreator.hpp

Go to the documentation of this file.

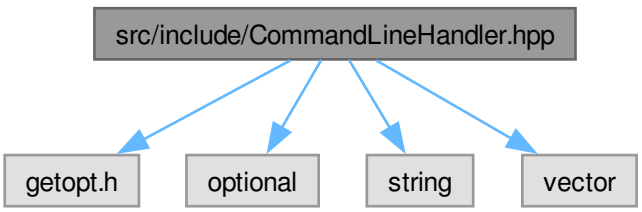
```
00001
00016 #include "FileData.hpp"
00017 #include <memory>
00018 #include <sstream>
00019
00029 class BatchCreator {
00030   public:
00039     explicit BatchCreator(std::shared_ptr<parsing::FileData> fileData);
00040
00046     [[nodiscard]] std::shared_ptr<std::stringstream> getDataStream() const {
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();
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 };
```

11.4 src/include/CommandLineHandler.hpp File Reference

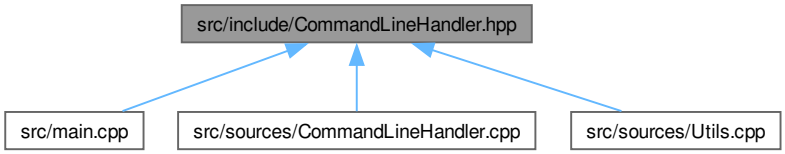
Responsible for the Command Line Interface.

```
#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 [ ]

11.4.1 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.

11.5 CommandLineHandler.hpp

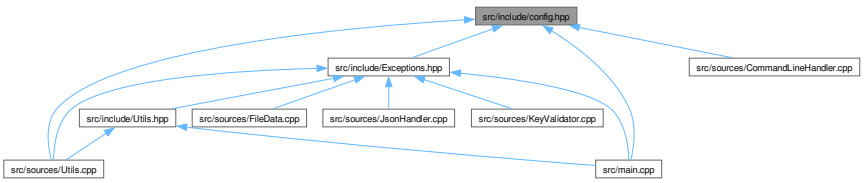
Go to the documentation of this file.

```
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
```

11.6 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
- constexpr auto config::HOMEPAGE_URL

11.6.1 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.

11.7 config.hpp

Go to the documentation of this file.

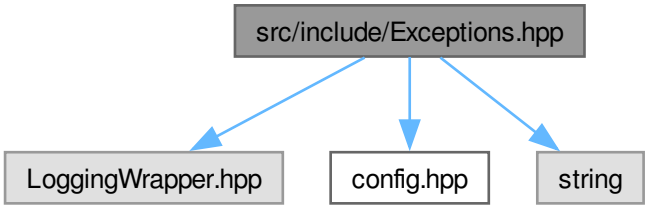
```
00001
00016 // This file is autogenerated. Changes will be overwritten
00017
00018 #ifndef CONFIG_HPP
00019 #define CONFIG_HPP
00020
00025 namespace config {
00026 inline constexpr auto LOG_CONFIG = "/home/simon/1_Coding/projectJsonToBat/"
00027                                    "build/Release/config/easylogging.conf";
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 =
00035             "Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci";
00036 inline constexpr auto HOMEPAGE_URL =
00037             "https://dhbwprojectsit23.github.io/JSON2Bat";
00038 } // namespace config
00039
00040 #endif
```

11.8 src/include/Exceptions.hpp File Reference

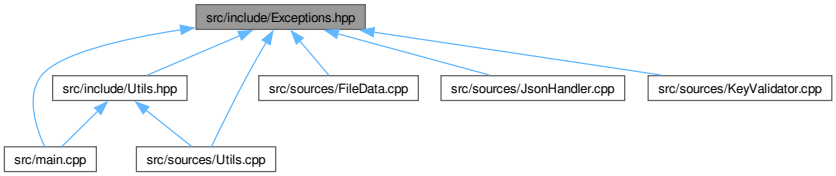
Contains all the custom exceptions used in the project.

```
#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.

11.8.1 Detailed Description

Contains all the custom exceptions used in the project.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

Copyright

See LICENSE file

Definition in file Exceptions.hpp.

11.9 Exceptions.hpp

Go to the documentation of this file.

```
00001
00010 #ifndef EXCEPTIONS_HPP
00011 #define EXCEPTIONS_HPP
00012
00013 #include "LoggingWrapper.hpp"
00014 #include "config.hpp"
00015 #include <string>
00016
00021 namespace exceptions {
00031 class CustomException : public std::exception {
00032   public:
00033     [[nodiscard]] const char* what() const noexcept override {
00034         return "Base Exception";
00035     }
00036 };
00037
00042 class ParsingException : public CustomException {
00043   private:
00044     const std::string file;
00045     std::string message;
00046
00047   public:
00048     explicit ParsingException(const std::string &file) : file(file) {
00054         std::stringstream ss;
00055         ss « "Error while trying to parse \"" « file « "\"!\n"
00056            « "There most likely is a syntax error within the \".json\" file.";
00057         this->message = ss.str();
00058         LOG_INFO « "ParsingException: " « message;
00059     }
00060
00061     [[nodiscard]] const char* what() const noexcept override {
00062         return message.c_str();
00063     }
00064 };
00065
00070 class FileExistsException : public CustomException {
00071   private:
00072     const std::string file;
00073     std::string message;
00074
00075   public:
00076     explicit FileExistsException(const std::string &file) : file(file) {
00082         std::stringstream ss;
00083         ss « "The outputfile \"" « file « "\" already exists!";
00084         this->message = ss.str();
00085         LOG_INFO « "BatchExistsException: " « message;
00086     }
00087
00088     [[nodiscard]] const char* what() const noexcept override {
00089         return message.c_str();
00090     }
00091 };
00092
00097 class InvalidValueException : public CustomException {
00098   private:
00099     const std::string key;
00100     std::string message;
00101
00102   public:
00103     InvalidValueException(const std::string &key, const std::string &issue)
00104         : key(key) {
00110         std::stringstream ss;
00111         ss « "Error at key \"" « key « "\"! " « issue;
00112         this->message = ss.str();
00113         LOG_INFO « "InvalidValueException: " « message;
00114     }
00115     [[nodiscard]] const char* what() const noexcept override {
00116         return message.c_str();
00117     }
00118 };
00119
00131 class InvalidKeyException : public CustomException {
00132   private:
00133     std::string message = "Invalid key found!";
00134
00135   public:
00136     explicit InvalidKeyException(
00137                 const std::vector<std::tuple<int, std::string» &keys) {
00138         LOG_INFO « "InvalidKeyException: " « message;
00139
00140         for (const auto &[line, key] : keys) {
00141             LOG_WARNING « "Invalid key found at line " « line « ": \"" « key
```

```
00142                         « "\"!";
00143         }
00144     }
00145     [[nodiscard]] const char* what() const noexcept override {
00146         return message.c_str();
00147     }
00148 };
00149
00158 class InvalidTypeException : public CustomException {
00159   private:
00160     const std::string type;
00161     std::string message;
00162
00163   public:
00164     InvalidTypeException(const std::string &type, int line) : type(type) {
00170         std::stringstream ss;
00171         ss « "Invalid type found at line " « line « ": \"" « type « "\"";
00172         this->message = ss.str();
00173         LOG_INFO « "InvalidTypeException: " « message;
00174     }
00175     [[nodiscard]] const char* what() const noexcept override {
00176         return message.c_str();
00177     }
00178 };
00179
00187 class MissingKeyException : public CustomException {
00188   private:
00189     std::string message;
00190     std::string type;
00191     std::string key;
00192
00193   public:
00194     MissingKeyException(const std::string &key, const std::string &type)
00195         : type(type), key(key) {
00201         std::stringstream ss;
00202         ss « "Missing key \"" « key « "\" for type \"" « type « "\"!";
00203         this->message = ss.str();
00204         LOG_INFO « "MissingKeyException: " « message;
00205     }
00206     [[nodiscard]] const char* what() const noexcept override {
00207         return message.c_str();
00208     }
00209 };
00210
00217 class MissingTypeException : public CustomException {
00218   private:
00219     std::string message = "Missing \"type\" key for at least one entry!";
00220
00221   public:
00222     MissingTypeException() {
00223         LOG_INFO « "MissingTypeException: " « message;
00224     }
00225     [[nodiscard]] const char* what() const noexcept override {
00226         return message.c_str();
00227     }
00228 };
00229
00234 class UnreachableCodeException : public CustomException {
00235   private:
00236     std::string message;
00237
00238   public:
00239     explicit UnreachableCodeException(const std::string &message)
00240         : message(message) {
00241         OUTPUT « "This exception happened due to a bug in the application!\n"
00242                « "Please report this bug! See " « config::EXECUTABLE_NAME
00243                « " -c for contact information.\n";
00244         LOG_INFO « "UnreachableCodeException: " « message;
00245     }
00246     [[nodiscard]] const char* what() const noexcept override {
00247         return message.c_str();
00248     }
00249 };
00250
00255 class FailedToOpenFileException : public CustomException {
00256   private:
00257     std::string message;
00258
00260   public:
00261     explicit FailedToOpenFileException(const std::string &file) {
00262         message = "Failed to open file: " + file;
00263         LOG_INFO « "FailedToOpenFileException: " « message;
00264     }
00265     [[nodiscard]] const char* what() const noexcept override {
00266         return message.c_str();
00267     }
00268 };
```

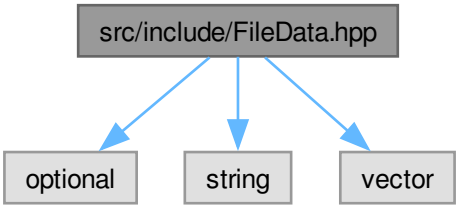
```
00269
00274 class NoSuchDirException : public CustomException {
00275   private:
00276     std::string message;
00277
00279   public:
00280     explicit NoSuchDirException(const std::string &dir) {
00281         message = "No such directory: " + dir;
00282         LOG_INFO « "NoSuchDirException: " « message;
00283     }
00284     [[nodiscard]] const char* what() const noexcept override {
00285         return message.c_str();
00286     }
00287 };
00288
00289 } // namespace exceptions
00290
00291 #endif
```

11.10 src/include/FileData.hpp File Reference

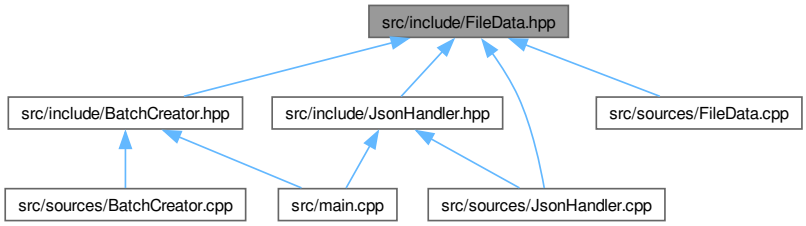
This file contains the FileData class.

```
#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.

11.10.1 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.

11.11 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,
```

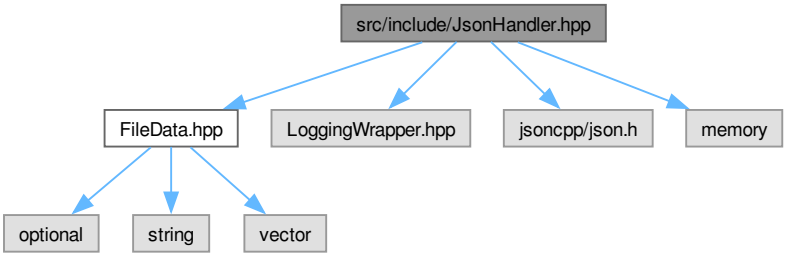
```
00087                                const std::string &value);
00088
00099     void addPathValue(const std::string &pathValue);
00100
00105     [[nodiscard]] const std::string &getOutputFile() const {
00106         return outputfile;
00107     }
00108
00113     [[nodiscard]] bool getHideShell() 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
```

11.12 src/include/JsonHandler.hpp File Reference

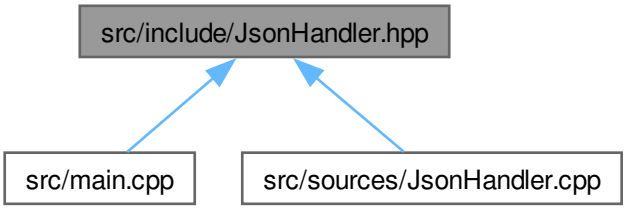
This file contains the JsonHandler class.

```
#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.

11.12.1 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.

11.13 JsonHandler.hpp

Go to the documentation of this file.

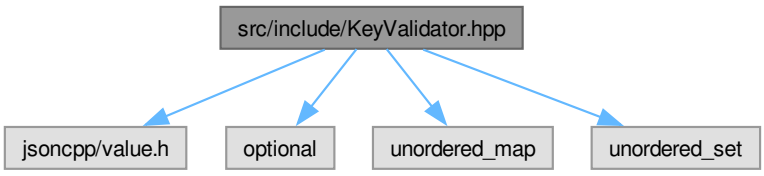
```
00001
00015 #ifndef JSONHANDLER_HPP
00016 #define JSONHANDLER_HPP
00017
00018 #include "FileData.hpp"
00019 #include "LoggingWrapper.hpp"
00020 #include <jsoncpp/json.h>
00021
00022 #include <memory>
00023
00036 namespace parsing {
00037
00047 class JsonHandler {
00048   public:
00055     JsonHandler() {
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
```

11.14 src/include/KeyValidator.hpp File Reference

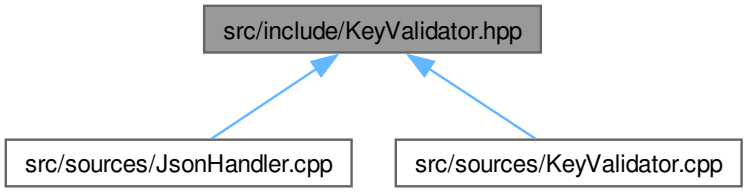
This file contains the KeyValidator class.

```
#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.

11.14.1 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.

11.15 KeyValidator.hpp

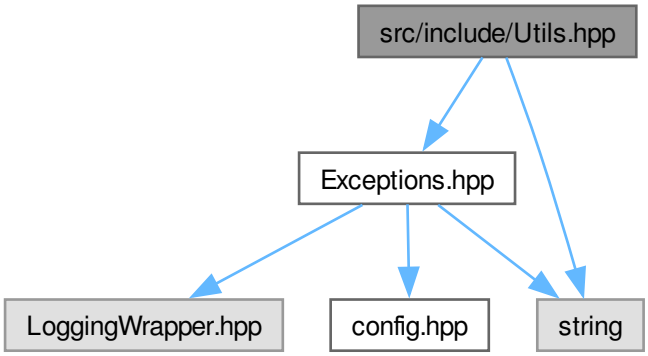
Go to the documentation of this file.

```
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
```

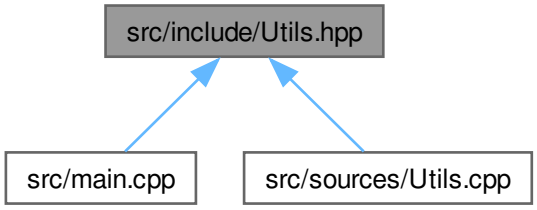
11.16 src/include/Utils.hpp File Reference

```
#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.

11.17 Utils.hpp

Go to the documentation of this file.

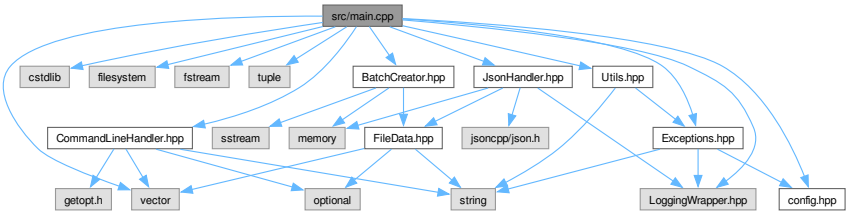
```
00001
00018 #ifndef UTILITIES_HPP
00019 #define UTILITIES_HPP
00020
00021 #include "Exceptions.hpp"
00022 #include <string>
00023
00033 namespace utilities {
00034
00042 class Utils {
00043   public:
00051     static void setupEasyLogging(const std::string &configFile);
00052
00066     static bool
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
```

11.18 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.
- const 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.

11.18.1 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.

11.18.2 Function Documentation

11.18.2.1 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

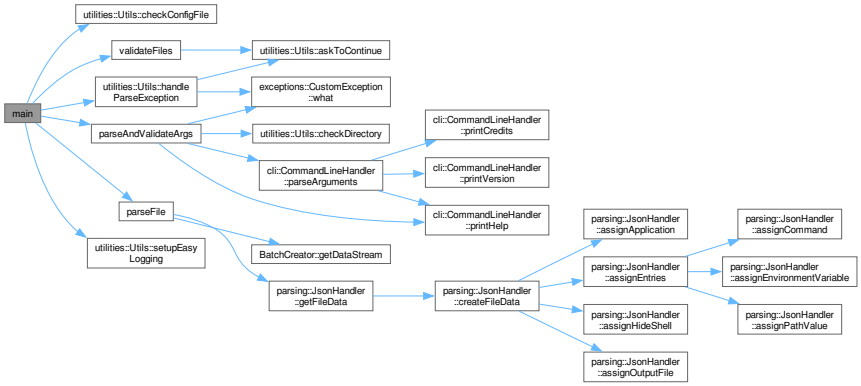
Returns

Returns 0 on success, 1 on failure

Definition at line 68 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:



11.18.2.2 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

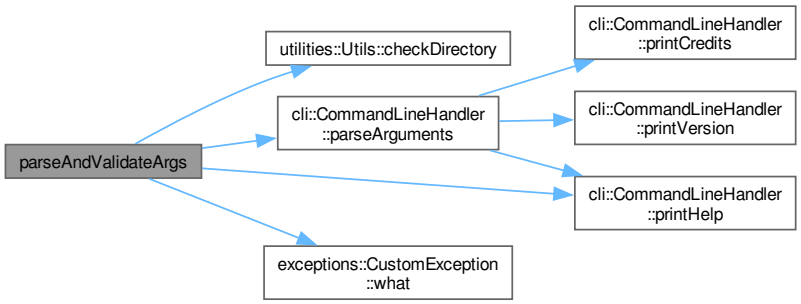
Returns

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

Definition at line 105 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:

11.18.2.3 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

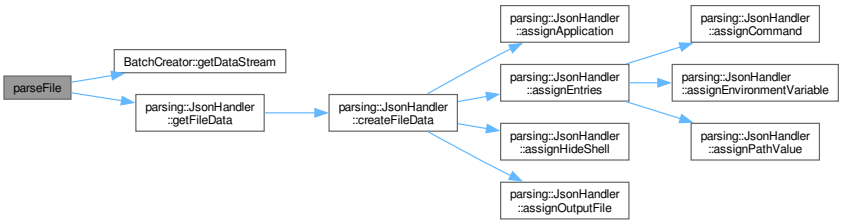
Parameters

file	The file to be parsed

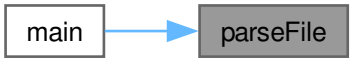
Definition at line 177 of file main.cpp.

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

Here is the call graph for this function:



Here is the caller graph for this function:



11.18.2.4 validateFiles()

```
const 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

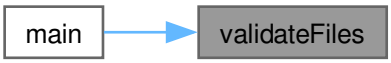
A vector containing the valid files

Definition at line 135 of file main.cpp.

References utilities::Utils::askToContinue().

Here is the call graph for this function:

Here is the caller graph for this function:



11.19 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 const std::vector<std::string>
00045 validateFiles(const std::vector<std::string> &files);
00046
00053 void parseFile(const std::string &file, const std::string &outputDirectory);
00054
00068 int main(int argc, char* argv[])
00069 {
00070     // Setup logging
00071     utilities::Utils::checkConfigFile(config::LOG_CONFIG);
00072     utilities::Utils::setupEasyLogging(config::LOG_CONFIG);
00073     // Parse and validate arguments
00074     auto [files, outDir] = parseAndValidateArgs(argc, argv);
00075     OUTPUT « cli::BOLD « "Parsing the following files:\n" « cli::RESET;
00076
00077     for (const auto &file : files) {
00078         OUTPUT « "\t - " « file « "\n";
00079     }
00080
00081     files = validateFiles(files);
00082
00083     // Main parsing loop
00084     for (auto file = files.begin(); file != files.end(); ++file) {
00085         OUTPUT « cli::ITALIC « "\nParsing file: " « *file « "...\n"
```

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

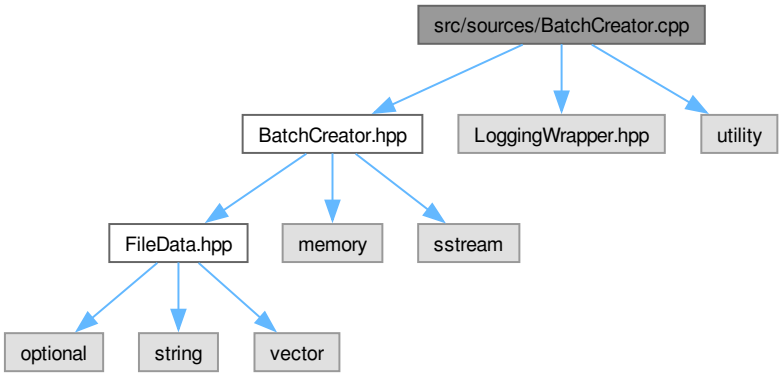
```
00173     validFiles.shrink_to_fit();
00174     return validFiles;
00175 }
00176
00177 void parseFile(const std::string &file, const std::string &outputDirectory)
00178 {
00179     parsing::JsonHandler jsonHandler(file);
00180     const auto fileData = jsonHandler.getFileData();
00181     BatchCreator batchCreator(fileData);
00182     const std::shared_ptr<std::stringstream> dataStream =
00183                 batchCreator.getDataStream();
00184     // Full filename is output directory + output file
00185     const std::string outputFileName =
00186                 outputDirectory + fileData->getOutputFile();
00187     std::ofstream outFile(outputFileName);
00188
00189     if (!outFile.good()) {
00190         throw exceptions::FailedToOpenFileException(outputFileName);
00191     }
00192
00193     outFile « dataStream->str();
00194     OUTPUT « "Done with files!\n";
00195 }
00196
00197 // Initialize easylogging++
00198 // Moved to bottom because it messed with doxygen
00199 INITIALIZE_EASYLOGGINGPP
```

11.20 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:



11.20.1 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.

11.21 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 {
00021     LOG_INFO « "Initializing BatchCreator";
00022     this->dataStream = std::make_shared<std::stringstream>();
00023     this->createBatch();
00024 }
00025
00026 void BatchCreator::createBatch()
00027 {
00028     LOG_INFO « "Creating Batch file";
00029     this->writeStart();
00030     this->writeHideShell();
00031     this->writeCommands();
00032     this->writeEnvVariables();
00033     this->writePathVariables();
00034     this->writeApp();
00035     this->writeEnd();
00036 }
00037
00038 void BatchCreator::writeStart() const
00039 {
00040     LOG_INFO « "writing Start of Batch";
00041     *this->dataStream « "@ECHO OFF\r\nC:\\Windows\\System32\\cmd.exe ";
00042 }
00043
00044 void BatchCreator::writeHideShell() const
00045 {
00046     if (this->fileData->getHideShell()) {
00047         LOG_INFO « "writing hide Shell";
00048         *this->dataStream « "/c ";
00049     }
00050     else {
00051         LOG_INFO « "writing show Shell";
00052         *this->dataStream « "/k ";
00053     }
00054 }
00055
00056 void BatchCreator::writeCommands() const
00057 {
00058     LOG_INFO « "writing Commands";
00059     *this->dataStream « "\"";
00060
```

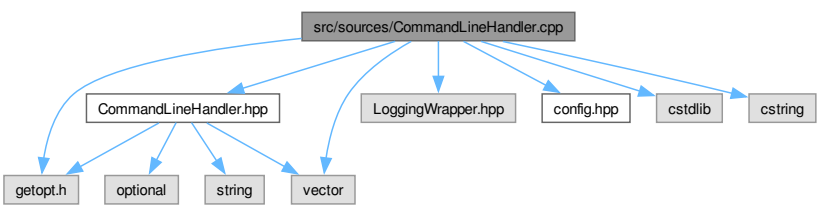
```
00061     for (const std::string &command : this->fileData->getCommands()) {
00062         *this->dataStream << command << " && ";
00063     }
00064 }
00065
00066 void BatchCreator::writeEnvVariables() const
00067 {
00068     LOG_INFO << "writing Environment Variables";
00069
00070     for (const auto &[key, value] : this->fileData->getEnvironmentVariables()) {
00071         *this->dataStream << "set " << key << "=" << value << " && ";
00072     }
00073 }
00074
00075 void BatchCreator::writePathVariables() const
00076 {
00077     LOG_INFO << "writing Path Variables";
00078     *this->dataStream << "set path=";
00079
00080     for (const std::string &path : this->fileData->getPathValues()) {
00081         *this->dataStream << path << ";";
00082     }
00083
00084     *this->dataStream << "%path%";
00085 }
00086
00087 void BatchCreator::writeApp() const
00088 {
00089     std::string appName = this->fileData->getOutputFile();
00090     appName = appName.substr(0, appName.find('.'));
00091
00092     if (this->fileData->getApplication().has_value()) {
00093         LOG_INFO << "writing start Application";
00094         *this->dataStream << " && start \"" << appName << "\" "
00095                           << this->fileData->getApplication().value() << "\"\r\n";
00096     }
00097     else {
00098         LOG_INFO << "writing not start Application";
00099         *this->dataStream << "\"\r\n";
00100     }
00101 }
00102
00103 void BatchCreator::writeEnd() const
00104 {
00105     *this->dataStream << "@ECHO ON";
00106 }
```

11.22 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.

11.22.1 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.

11.23 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 {
00024     LOG_INFO << "Printing help message...";
00025     OUTPUT << BOLD << "Usage:\n"
00026            << RESET << "----------\n"
00027            << config::EXECUTABLE_NAME << " [options] [filenames]\n"
00028            << "\n"
00029            << BOLD << "Options:\n"
00030            << RESET << "----------\n"
00031            << "-o, --outdir\t [path]\t\tOutput the batch file to the given "
00032            "dir\n"
00033            << "-h, --help\t\t\tPrint this help message\n"
00034            << "-v, --version\t\t\tPrint the version number\n"
00035            << "-c, --credits\t\t\tPrint the credits\n\n"
```

```
00036             « "    --verbose\t\t\tStart the application in verbose mode\n"
00037             « ITALIC
00038             « "           \t\t\tNote: Verbose flag should be passed first!\n\n"
00039             « RESET « BOLD « "Filenames:\n"
00040             « RESET « "----------\n"
00041             « "The json files to be processed into batch files.\n"
00042             « "Multiple files should be seperated by spaces!\n\n";
00043     exit(0);
00044 }
00045 void CommandLineHandler::printVersion()
00046 {
00047     LOG_INFO « "Printing version number...";
00048     OUTPUT « config::PROJECT_NAME « " v" « config::MAJOR_VERSION « "."
00049            « config::MINOR_VERSION « "." « config::PATCH_VERSION « "\n";
00050     exit(0);
00051 }
00052 void CommandLineHandler::printCredits()
00053 {
00054     LOG_INFO « "Printing credits...";
00055     OUTPUT « BOLD « "Project information:\n"
00056            « RESET « "----------\n"
00057            « CYAN « BOLD « config::PROJECT_NAME « RESET « " v"
00058            « config::MAJOR_VERSION « "." « config::MINOR_VERSION « "."
00059            « config::PATCH_VERSION « "\n"
00060            « "\n"
00061            « config::DESCRIPTION « "\n"
00062            « "\n"
00063            « GREEN « "Authors: " « RESET « ITALIC « config::AUTHORS « RESET
00064            « "\n"
00065            « GREEN « "Documentation: " « RESET « ITALIC
00066            « config::HOMEPAGE_URL « RESET « GREEN « "\nContact: " « RESET
00067            « ITALIC « "simon21.blum@gmail.com" « "\n";
00068     exit(0);
00069 }
00070
00071 std::tuple<std::optional<std::string>, std::vector<std::string»
00072 CommandLineHandler::parseArguments(
00073             int argc, char* argv[])
00074 {
00075     LOG_INFO « "Parsing arguments...";
00076     std::vector<std::string> files;
00077     std::optional<std::string> outDir;
00078
00079     while (true) {
00080         int optIndex = -1;
00081         struct option longOption = {};
00082         const auto result = getopt_long(argc, argv, "hvco:", options, &optIndex);
00083
00084         if (result == -1) {
00085             LOG_INFO « "End of options reached";
00086             break;
00087         }
00088
00089         switch (result) {
00090             case '?':
00091                 LOG_ERROR « "Invalid Option (argument)\n";
00092                 CommandLineHandler::printHelp();
00093
00094             case 'h':
00095                 LOG_INFO « "Help option detected";
00096                 CommandLineHandler::printHelp();
00097
00098             case 'v':
00099                 LOG_INFO « "Version option detected";
00100                 CommandLineHandler::printVersion();
00101
00102             case 'c':
00103                 LOG_INFO « "Credit option detected";
00104                 CommandLineHandler::printCredits();
00105
00106             case 'o':
00107                 LOG_INFO « "Output option detected";
00108                 outDir = optarg;
00109                 break;
00110
00111             case 0:
00112                 LOG_INFO « "Long option without short version detected";
00113                 longOption = options[optIndex];
00114                 LOG_INFO « "Option: " « longOption.name « " given";
00115
00116                 if (strcmp(longOption.name, "verbose") == 0) {
00117                     logging::setVerboseMode(true);
00118                     LOG_INFO « "Verbose mode activated";
00119                 }
00120
00121                 break;
00122
```

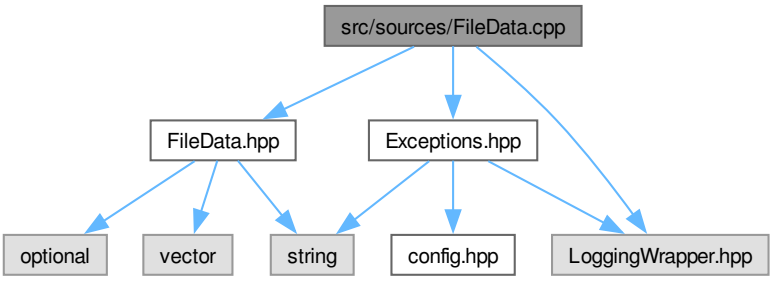
```
00123             default:
00124                 LOG_ERROR « "Default case for options reached!";
00125                 break;
00126         }
00127     }
00128
00129     LOG_INFO « "Options have been parsed";
00130     LOG_INFO « "Checking for arguments...";
00131
00132     while (optind < argc) {
00133         LOG_INFO « "Adding file: " « argv[optind];
00134         files.emplace_back(argv[optind++]);
00135     }
00136
00137     LOG_DEBUG « files.size();
00138     LOG_INFO « "Arguments and options have been parsed";
00139     return {outDir, files};
00140 }
00141 } // namespace cli
```

11.24 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.

11.24.1 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.

11.25 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 {
00020     LOG_INFO « "Setting outputfile to...";
00021
00022     // If no value for key "outputfile"
00023     if (newOutputfile.empty()) {
00024         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00025         throw exceptions::InvalidValueException("outputfile",
00026                                                 "Outputfile can't be empty!");
00027     }
00028
00029     // If outputfile is already set
00030     if (!this->outputfile.empty()) {
00031         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00032         throw exceptions::InvalidValueException("outputfile",
00033                                                 "Outputfile is already set!");
00034     }
00035
00036     // If outputfile does not end with ".bat"
00037     if (!newOutputfile.ends_with(".bat")) {
00038         newOutputfile += ".bat";
00039         LOG_WARNING « "Outputfile does not end with \".bat\", adding it now: "
00040                     « newOutputfile;
00041     }
00042
00043     this->outputfile = newOutputfile;
00044     LOG_INFO « "Outputfile set to: " « this->outputfile « "\n";
00045 }
00046
00047 void FileData::setApplication(const std::string &newApplication)
00048 {
00049     if (newApplication.empty()) {
00050         LOG_INFO « "newApplication empty, returning";
00051         return;
00052     }
00053
00054     LOG_INFO « "Setting application to: " « newApplication « "\n";
00055     this->application.emplace(newApplication);
00056 }
00057
00058 void FileData::addCommand(const std::string &command)
00059 {
00060     if (command.empty()) {
```

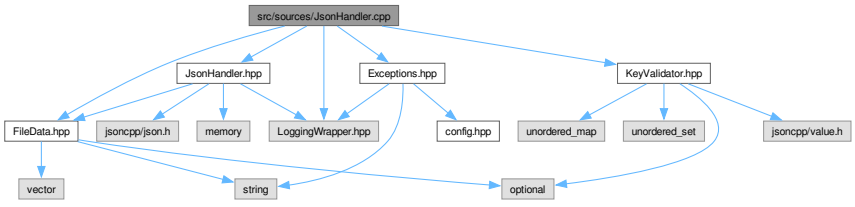
```
00061         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00062         throw exceptions::InvalidValueException("command",
00063                                                 "Command value is empty!");
00064     }
00065
00066     LOG_INFO « "Adding command: " « command « "\n";
00067     this->commands.push_back(command);
00068 }
00069
00070 void FileData::addEnvironmentVariable(const std::string &name,
00071                                       const std::string &value)
00072 {
00073     if (name.empty()) {
00074         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00075         throw exceptions::InvalidValueException("name", "Name value is empty!");
00076     }
00077
00078     if (value.empty()) {
00079         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00080         throw exceptions::InvalidValueException("key", "Key value is empty");
00081     }
00082
00083     LOG_INFO « "Adding environment variable: " « name « "=" « value « "\n";
00084     this->environmentVariables.emplace_back(name, value);
00085 }
00086
00087 void FileData::addPathValue(const std::string &pathValue)
00088 {
00089     if (pathValue.empty()) {
00090         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00091         throw exceptions::InvalidValueException("path", "Path value is empty");
00092     }
00093
00094     LOG_INFO « "Adding path value: " « pathValue « "\n";
00095     this->pathValues.push_back(pathValue);
00096 }
00097 } // namespace parsing
```

11.26 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.

11.26.1 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.

11.27 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 {
00022     LOG_INFO « "Initializing JSONHandler with filename: " « filename « "\n";
00023     this->root = parseFile(filename);
00024 }
00025
00026 std::shared_ptr<Json::Value> JsonHandler::parseFile(const std::string &filename)
00027
00028 {
00029     LOG_INFO « "Parsing file: " « filename « "\n";
00030     std::ifstream file(filename);
00031     Json::Value newRoot;
00032
00033     // Json::Reader.parse() returns false if parsing fails
00034     if (Json::Reader reader; !reader.parse(file, newRoot)) {
00035         throw exceptions::ParsingException(filename);
00036     }
00037
00038     // Validate keys
00039     // Check for errors
00040     if (auto errors = KeyValidator::getInstance().validateKeys(newRoot, filename);
00041         !errors.empty()) {
00042         throw exceptions::InvalidKeyException(errors);
00043     }
00044
00045     LOG_INFO « "File \"" « filename « "\" has been parsed\n";
```

```
00046     return std::make_shared<Json::Value>(newRoot);
00047 }
00048
00049 std::shared_ptr<FileData> JsonHandler::getFileData()
00050 {
00051     LOG_INFO << "Creating FileData object for return...\n";
00052     return this->createFileData();
00053 }
00054
00055 std::shared_ptr<FileData> JsonHandler::createFileData()
00056 {
00057     LOG_INFO << "Creating FileData object...\n";
00058     this->data = std::make_shared<FileData>();
00059     this->assignOutputFile();
00060     this->assignHideShell();
00061     this->assignApplication();
00062     this->assignEntries();
00063     return this->data;
00064 }
00065
00066 void JsonHandler::assignOutputFile() const
00067 {
00068     LOG_INFO << "Assigning outputfile...\n";
00069     std::string outputFile = this->root->get("outputfile", "").asString();
00070     this->data->setOutputFile(outputFile);
00071 }
00072
00073 void JsonHandler::assignHideShell() const
00074 {
00075     LOG_INFO << "Assigning hide shell...\n";
00076     // If the 'hideshell' key is not given, it defaults to false
00077     this->data->setHideShell(this->root->get("hideshell", false).asBool());
00078 }
00079
00080 void JsonHandler::assignApplication() const
00081 {
00082     LOG_INFO << "Assigning application...\n";
00083     this->data->setApplication(this->root->get("application", "").asString());
00084 }
00085
00086 void JsonHandler::assignEntries() const
00087 {
00088     LOG_INFO << "Assigning entries...\n";
00089
00090     for (const auto &entry : this->root->get("entries", "")) {
00091         std::string entryType = entry.get("type", "").asString();
00092
00093         if (entryType == "EXE") {
00094             LOG_INFO << "Calling function to assign command...\n";
00095             this->assignCommand(entry);
00096         }
00097         else if (entryType == "ENV") {
00098             LOG_INFO << "Calling function to assign environment variable...\n";
00099             this->assignEnvironmentVariable(entry);
00100         }
00101         else if (entryType == "PATH") {
00102             LOG_INFO << "Calling function to assign path value...\n";
00103             this->assignPathValue(entry);
00104         }
00105         else {
00106             // Due to validation beforehand - this should never be reached!
00107             throw exceptions::UnreachableCodeException(
00108                         "Unknown entries should be caught by KeyValidator!\nPlease report "
00109                         "this bug!");
00110         }
00111     }
00112 }
00113
00114 void JsonHandler::assignCommand(const Json::Value &entry) const
00115 {
00116     LOG_INFO << "Assigning command...\n";
00117     this->data->addCommand(entry.get("command", "").asString());
00118 }
00119
00120 void JsonHandler::assignEnvironmentVariable(const Json::Value &entry) const
00121 {
00122     LOG_INFO << "Assigning environment variable...\n";
00123     std::string key = entry.get("key", "").asString();
00124     std::string value = entry.get("value", "").asString();
00125     this->data->addEnvironmentVariable(key, value);
00126 }
00127
00128 void JsonHandler::assignPathValue(const Json::Value &entry) const
00129 {
00130     LOG_INFO << "Assigning path value...\n";
00131     this->data->addPathValue(entry.get("path", "").asString());
00132 }
```

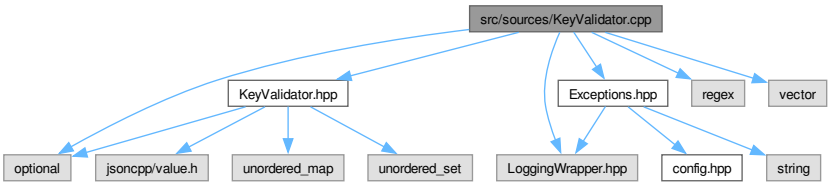
```
00133 } // namespace parsing
```

11.28 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.

11.28.1 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.

11.29 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 {
00022     static KeyValidator keyValidator;
00023     LOG_INFO « "Returning KeyValidator instance!";
00024     return keyValidator;
00025 }
00026
00027 std::vector<std::tuple<int, std::string» KeyValidator::validateKeys(
00028             const Json::Value &root,
00029             const std::string &filename)
00030 {
00031     std::vector<std::tuple<int, std::string» wrongKeys =
00032                 getWrongKeys(root, filename);
00033
00034     // Inline declaration to prevent leaking in outer scope
00035     for (Json::Value entries = root.get("entries", "");
00036          const auto &entry : entries) {
00037         const auto entryKeys = entry.getMemberNames();
00038         // Create a set of the entry keys for faster lookup (O(1) instead of O(n))
00039         std::unordered_set<std::string> entryKeysSet(entryKeys.begin(),
00040                                                      entryKeys.end());
00041         const auto wrongEntries = validateEntries(filename, entryKeysSet);
00042         // Combine wrong keys
00043         wrongKeys.insert(wrongKeys.end(), wrongEntries.begin(), wrongEntries.end());
00044         // Validate that each entry has it's necessary keys
00045         validateTypes(filename, entry, entryKeysSet);
00046     }
00047
00048     return wrongKeys;
00049 }
00050
00051 std::vector<std::tuple<int, std::string» KeyValidator::getWrongKeys(
00052             const Json::Value &root,
00053             const std::string &filename) const
00054 {
00055     std::vector<std::tuple<int, std::string» wrongKeys = {};
00056
00057     for (const auto &key : root.getMemberNames()) {
00058         if (!validKeys.contains(key)) {
00059             const auto error = getUnknownKeyLine(filename, key);
00060
00061             if (!error.has_value()) {
00062                 LOG_ERROR « "Unable to find line of wrong key!";
00063                 continue;
00064             }
00065
00066             // If the line can't be found, add -1 as line number
00067             wrongKeys.emplace_back(error.value_or(-1), key);
00068         }
00069     }
00070
00071     return wrongKeys;
00072 }
00073
00074 std::vector<std::tuple<int, std::string» KeyValidator::validateEntries(
00075             const std::string &filename,
00076             const std::unordered_set<std::string> &entryKeys) const
00077 {
00078     std::vector<std::tuple<int, std::string» wrongKeys = {};
00079
00080     for (const auto &key : entryKeys) {
00081         if (!validEntryKeys.contains(key)) {
00082             const auto error = getUnknownKeyLine(filename, key);
00083
00084             if (!error.has_value()) {
00085                 LOG_ERROR « "Unable to find line of wrong key!";
00086                 continue;
00087             }
00088
00089             wrongKeys.emplace_back(error.value_or(-1), key);
00090         }
00091     }
00092
```

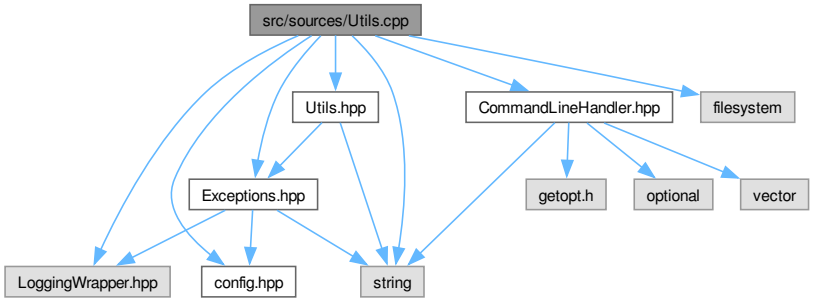
```
00093     return wrongKeys;
00094 }
00095
00096 void KeyValidator::validateTypes(
00097             const std::string &filename, const Json::Value &entry,
00098             const std::unordered_set<std::string> &entryKeys)
00099 {
00100     // Gett the type of the entry - error if not found
00101     const std::string type = entry.get("type", "ERROR").asString();
00102
00103     // If the type is not found, throw an exception
00104     if (type == "ERROR") {
00105         throw exceptions::MissingTypeException();
00106         // If the type is not known, throw an exception
00107         // @note This should already have been checked
00108     }
00109     else if (typeToKeys.contains(type)) {
00110         const std::optional<int> line =
00111                     getUnknownKeyLine(filename, std::string(type));
00112
00113         if (!line.has_value()) {
00114             LOG_INFO << "Unable to find line of wrong type!";
00115         }
00116
00117         throw exceptions::InvalidTypeException(std::string(type), line.value());
00118         // If the type is known, check if all necessary keys are present
00119     }
00120     else {
00121         for (const auto &key : typeToKeys[type]) {
00122             if (entryKeys.contains(key)) {
00123                 throw exceptions::MissingKeyException(key, std::string(type));
00124             }
00125         }
00126     }
00127 }
00128
00129 std::optional<int> KeyValidator::getUnknownKeyLine(const std::string &filename,
00130                                                    const std::string &wrongKey)
00131 {
00132     std::ifstream file(filename);
00133
00134     if (!file.is_open()) {
00135         LOG_ERROR << "File not open!";
00136         return std::nullopt;
00137     }
00138
00139     std::string line;
00140     // Create a regex pattern that matches the wrong key whole word
00141     const std::regex wrongKeyPattern("\\b" + wrongKey + "\\b");
00142
00143     for (int lineNumber = 1; std::getline(file, line); ++lineNumber) {
00144         if (std::regex_search(line, wrongKeyPattern)) {
00145             return lineNumber;
00146         }
00147     }
00148
00149     return std::nullopt;
00150 }
00151
00152 } // namespace parsing
```

11.30 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.

11.30.1 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.

11.31 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 {
00027     el::Configurations conf(configFile);
00028     el::Loggers::reconfigureAllLoggers(conf);
00029     LOG_INFO « "Running " « config::PROJECT_NAME « " v"
00030              « config::MAJOR_VERSION « "." « config::MINOR_VERSION « "."
00031              « config::PATCH_VERSION;
00032     LOG_INFO « "For more Information checkout " « config::HOMEPAGE_URL;
00033     LOG_INFO « "EasyLogging has been setup!";
00034 }
00035 bool Utils::askToContinue(const std::string &prompt)
00036 {
00037     std::string userInput;
00038     LOG_INFO « "Asking for user Confirmation to continue...";
00039     OUTPUT « cli::BOLD « prompt « cli::RESET;
00040
00041     do {
00042         std::cin » userInput;
00043         std::ranges::transform(userInput, userInput.begin(), ::tolower);
00044
00045         if (userInput != "y" && userInput != "yes" && userInput != "n" &&
00046             userInput != "no") {
00047             LOG_INFO « "Wrong user input!";
00048             OUTPUT « cli::ITALIC « "Please enter Y/Yes or N/No!\n" « cli::RESET;
00049             continue;
00050         }
00051
00052         break;
00053     } while (true);
00054
00055     return userInput == "y" || userInput == "yes";
00056 }
00057 void Utils::checkConfigFile(const std::string &configFile)
00058 {
00059     if (!std::filesystem::is_regular_file(configFile)) {
00060         std::cerr « cli::RED « cli::BOLD
00061                   « "Fatal: Easylogging configuration file not found at:\n"
00062                   « cli::RESET « cli::ITALIC « "\n\t\"" « configFile « "\"\n\n"
00063                   « cli::RESET;
00064         std::cout « "Aborting...\n";
00065         exit(1);
00066     }
00067 }
00068 const std::string &Utils::checkDirectory(std::string &directory)
00069 {
00070     if (!directory.empty() && directory.back() != '/' &&
00071         directory.back() != '\\') {
00072         directory += '/';
00073     }
00074
00075     if (!std::filesystem::exists(directory)) {
00076         throw exceptions::NoSuchDirException(directory);
00077     }
00078
00079     return directory;
00080 }
00081 bool Utils::handleParseException(const exceptions::CustomException &e,
00082                                  const std::vector<std::string>::iterator &file,
00083                                  const std::vector<std::string> &files)
00084 {
00085     OUTPUT « "\nThere has been a error while trying to parse \"" « *file
00086            « ":\n";
00087     LOG_ERROR « e.what();
00088
00089     if (std::next(file) != files.end() &&
00090         !utilities::Utils::askToContinue(
00091                     "Do you want to continue with the other files? (y/n) "
00092                     "")) {
00093         OUTPUT « "Aborting...";
00094         LOG_INFO « "Application ended by user Input";
00095         return false;
```

```
00096     }
00097
00098     std::cout « std::endl;
00099     return true;
00100 }
00101
00102 } // namespace utilities
```

Index