

JSON2Batch

0.3.1

Generated on Sat Apr 27 2024 13:51:21 for JSON2Batch by Doxygen 1.9.8

Sat Apr 27 2024 13:51:21

1 JSON2Batch	1
1.1 Table of Contents	1
1.2 Build Instructions	1
1.2.1 Linux	1
1.2.1.1 UNIX Compiler Compatibility	2
1.2.2 Windows	2
1.2.3 Build with MinGW	2
1.2.4 Build with Ninja	2
1.2.5 Generating Documentation	2
1.3 Documentation	2
1.3.1 Project Structure	2
1.4 External Libraries	3
1.4.1 easylogging++	3
1.4.2 LoggingWrapper	3
1.4.3 jsoncpp	3
1.5 License	3
2 Topic Index	5
2.1 Topics	5
3 Namespace Index	7
3.1 Namespace List	7
4 Hierarchical Index	9
4.1 Class Hierarchy	9
5 Class Index	11
5.1 Class List	11
6 File Index	13
6.1 File List	13
7 Topic Documentation	15
7.1 StyleHelpers	15
8 Namespace Documentation	17
8.1 cli Namespace Reference	17
8.1.1 Detailed Description	17
8.1.2 Variable Documentation	18
8.1.2.1 options	18
8.2 config Namespace Reference	18
8.2.1 Detailed Description	18
8.2.2 Variable Documentation	18
8.2.2.1 AUTHORS	18
8.2.2.2 DESCRIPTION	19

8.2.2.3 EXECUTABLE_NAME	19
8.2.2.4 HOMEPAGE_URL	19
8.2.2.5 LOG_CONFIG	19
8.2.2.6 MAJOR_VERSION	19
8.2.2.7 MINOR_VERSION	19
8.2.2.8 PATCH_VERSION	19
8.2.2.9 PROJECT_NAME	20
8.3 exceptions Namespace Reference	20
8.3.1 Detailed Description	20
8.4 parsing Namespace Reference	20
8.4.1 Detailed Description	21
8.5 utilities Namespace Reference	21
8.5.1 Detailed Description	21
9 Class Documentation	23
9.1 BatchCreator Class Reference	23
9.1.1 Detailed Description	24
9.1.2 Constructor & Destructor Documentation	24
9.1.2.1 BatchCreator()	24
9.1.3 Member Function Documentation	25
9.1.3.1 createBatch()	25
9.1.3.2 getDataStream()	26
9.1.3.3 writeApplication()	27
9.1.3.4 writeCommands()	27
9.1.3.5 writeEnd()	28
9.1.3.6 writeEnvVariables()	28
9.1.3.7 writeHideShell()	29
9.1.3.8 writePathVariables()	29
9.1.3.9 writeStart()	30
9.1.4 Member Data Documentation	30
9.1.4.1 dataStream	30
9.1.4.2 fileData	30
9.2 cli::CommandLineHandler Class Reference	30
9.2.1 Detailed Description	31
9.2.2 Constructor & Destructor Documentation	32
9.2.2.1 CommandLineHandler()	32
9.2.2.2 ~CommandLineHandler()	32
9.2.3 Member Function Documentation	32
9.2.3.1 parseArguments()	32
9.2.3.2 printCredits()	33
9.2.3.3 printHelp()	34
9.2.3.4 printVersion()	35

9.3 exceptions::ContainsBadCharacterException Class Reference	35
9.3.1 Detailed Description	36
9.3.2 Constructor & Destructor Documentation	36
9.3.2.1 ContainsBadCharacterException()	36
9.3.3 Member Function Documentation	37
9.3.3.1 what()	37
9.3.4 Member Data Documentation	37
9.3.4.1 message	37
9.4 exceptions::CustomException Class Reference	37
9.4.1 Detailed Description	38
9.4.2 Member Function Documentation	38
9.4.2.1 what()	38
9.5 exceptions::FailedToOpenFileException Class Reference	39
9.5.1 Detailed Description	40
9.5.2 Constructor & Destructor Documentation	40
9.5.2.1 FailedToOpenFileException()	40
9.5.3 Member Function Documentation	40
9.5.3.1 what()	40
9.5.4 Member Data Documentation	40
9.5.4.1 message	40
9.6 parsing::FileData Class Reference	41
9.6.1 Detailed Description	41
9.6.2 Member Function Documentation	42
9.6.2.1 addCommand()	42
9.6.2.2 addEnvironmentVariable()	42
9.6.2.3 addPathValue()	42
9.6.2.4 getApplication()	43
9.6.2.5 getCommands()	43
9.6.2.6 getEnvironmentVariables()	44
9.6.2.7 getHideShell()	44
9.6.2.8 getOutputFile()	44
9.6.2.9 getPathValues()	44
9.6.2.10 setApplication()	44
9.6.2.11 setHideShell()	45
9.6.2.12 setOutputFile()	45
9.6.3 Member Data Documentation	45
9.6.3.1 application	45
9.6.3.2 commands	46
9.6.3.3 environmentVariables	46
9.6.3.4 hideShell	46
9.6.3.5 outputfile	46
9.6.3.6 pathValues	46

9.7 exceptions::FileExistsException Class Reference	47
9.7.1 Detailed Description	48
9.7.2 Constructor & Destructor Documentation	48
9.7.2.1 FileExistsException()	48
9.7.3 Member Function Documentation	48
9.7.3.1 what()	48
9.7.4 Member Data Documentation	48
9.7.4.1 file	48
9.7.4.2 message	49
9.8 exceptions::InvalidKeyException Class Reference	49
9.8.1 Detailed Description	50
9.8.2 Constructor & Destructor Documentation	50
9.8.2.1 InvalidKeyException()	50
9.8.3 Member Function Documentation	50
9.8.3.1 what()	50
9.8.4 Member Data Documentation	51
9.8.4.1 message	51
9.9 exceptions::InvalidTypeException Class Reference	51
9.9.1 Detailed Description	52
9.9.2 Constructor & Destructor Documentation	52
9.9.2.1 InvalidTypeException()	52
9.9.3 Member Function Documentation	52
9.9.3.1 what()	52
9.9.4 Member Data Documentation	53
9.9.4.1 message	53
9.9.4.2 type	53
9.10 exceptions::InvalidValueException Class Reference	53
9.10.1 Detailed Description	54
9.10.2 Constructor & Destructor Documentation	54
9.10.2.1 InvalidValueException()	54
9.10.3 Member Function Documentation	55
9.10.3.1 what()	55
9.10.4 Member Data Documentation	55
9.10.4.1 key	55
9.10.4.2 message	55
9.11 parsing::JsonHandler Class Reference	55
9.11.1 Detailed Description	56
9.11.2 Constructor & Destructor Documentation	57
9.11.2.1 JsonHandler() [1/2]	57
9.11.2.2 JsonHandler() [2/2]	57
9.11.3 Member Function Documentation	57
9.11.3.1 assignApplication()	57

9.11.3.2 assignCommand()	58
9.11.3.3 assignEntries()	59
9.11.3.4 assignEnvironmentVariable()	60
9.11.3.5 assignHideShell()	61
9.11.3.6 assignOutputFile()	61
9.11.3.7 assignPathValue()	62
9.11.3.8 containsBadCharacter()	63
9.11.3.9 createFileData()	64
9.11.3.10 getFileData()	64
9.11.3.11 parseFile()	65
9.11.4 Member Data Documentation	66
9.11.4.1 data	66
9.11.4.2 root	67
9.12 parsing::KeyValidator Class Reference	67
9.12.1 Detailed Description	68
9.12.2 Member Function Documentation	68
9.12.2.1 getInstance()	68
9.12.2.2 getUnknownKeyLine()	68
9.12.2.3 getWrongKeys()	69
9.12.2.4 validateEntries()	70
9.12.2.5 validateKeys()	71
9.12.2.6 validateTypes()	72
9.12.3 Member Data Documentation	73
9.12.3.1 typeToKeys	73
9.12.3.2 validEntryKeys	73
9.12.3.3 validKeys	74
9.13 exceptions::MissingKeyException Class Reference	74
9.13.1 Detailed Description	75
9.13.2 Constructor & Destructor Documentation	76
9.13.2.1 MissingKeyException()	76
9.13.3 Member Function Documentation	76
9.13.3.1 what()	76
9.13.4 Member Data Documentation	76
9.13.4.1 key	76
9.13.4.2 message	76
9.13.4.3 type	76
9.14 exceptions::MissingTypeException Class Reference	77
9.14.1 Detailed Description	78
9.14.2 Constructor & Destructor Documentation	78
9.14.2.1 MissingTypeException()	78
9.14.3 Member Function Documentation	78
9.14.3.1 what()	78

9.14.4 Member Data Documentation	78
9.14.4.1 message	78
9.15 exceptions::NoSuchDirException Class Reference	79
9.15.1 Detailed Description	80
9.15.2 Constructor & Destructor Documentation	80
9.15.2.1 NoSuchDirException()	80
9.15.3 Member Function Documentation	80
9.15.3.1 what()	80
9.15.4 Member Data Documentation	80
9.15.4.1 message	80
9.16 options Struct Reference	81
9.16.1 Detailed Description	81
9.17 exceptions::ParsingException Class Reference	81
9.17.1 Detailed Description	82
9.17.2 Constructor & Destructor Documentation	82
9.17.2.1 ParsingException()	82
9.17.3 Member Function Documentation	83
9.17.3.1 what()	83
9.17.4 Member Data Documentation	83
9.17.4.1 file	83
9.17.4.2 message	83
9.18 exceptions::UnreachableCodeException Class Reference	83
9.18.1 Detailed Description	84
9.18.2 Constructor & Destructor Documentation	84
9.18.2.1 UnreachableCodeException()	84
9.18.3 Member Function Documentation	85
9.18.3.1 what()	85
9.18.4 Member Data Documentation	85
9.18.4.1 message	85
9.19 utilities::Utils Class Reference	85
9.19.1 Detailed Description	86
9.19.2 Member Function Documentation	86
9.19.2.1 askToContinue()	86
9.19.2.2 checkConfigFile()	86
9.19.2.3 checkDirectory()	87
9.19.2.4 escapeString()	88
9.19.2.5 handleParseException()	88
9.19.2.6 setupEasyLogging()	89
10 File Documentation	91
10.1 README.md File Reference	91
10.2 src/include/BatchCreator.hpp File Reference	91

10.2.1 Detailed Description	92
10.3 BatchCreator.hpp	93
10.4 src/include/CommandLineHandler.hpp File Reference	93
10.4.1 Detailed Description	94
10.5 CommandLineHandler.hpp	95
10.6 src/include/config.hpp File Reference	95
10.6.1 Detailed Description	96
10.7 config.hpp	97
10.8 src/include/Exceptions.hpp File Reference	97
10.8.1 Detailed Description	98
10.9 Exceptions.hpp	99
10.10 src/include/FileData.hpp File Reference	101
10.10.1 Detailed Description	102
10.11 FileData.hpp	103
10.12 src/include/JsonHandler.hpp File Reference	103
10.12.1 Detailed Description	104
10.13 JsonHandler.hpp	105
10.14 src/include/KeyValidator.hpp File Reference	106
10.14.1 Detailed Description	107
10.15 KeyValidator.hpp	107
10.16 src/include/Utils.hpp File Reference	108
10.17 Utils.hpp	109
10.18 src/main.cpp File Reference	109
10.18.1 Detailed Description	110
10.18.2 Function Documentation	111
10.18.2.1 main()	111
10.18.2.2 parseAndValidateArgs()	111
10.18.2.3 parseFile()	112
10.18.2.4 validateFiles()	113
10.19 main.cpp	114
10.20 src/sources/BatchCreator.cpp File Reference	116
10.20.1 Detailed Description	117
10.21 BatchCreator.cpp	118
10.22 src/sources/CommandLineHandler.cpp File Reference	119
10.22.1 Detailed Description	119
10.23 CommandLineHandler.cpp	120
10.24 src/sources/FileData.cpp File Reference	121
10.24.1 Detailed Description	122
10.25 FileData.cpp	123
10.26 src/sources/JsonHandler.cpp File Reference	124
10.26.1 Detailed Description	124
10.27 JsonHandler.cpp	125

10.28 src/sources/KeyValidator.cpp File Reference	126
10.28.1 Detailed Description	127
10.29 KeyValidator.cpp	128
10.30 src/sources/Utils.cpp File Reference	129
10.30.1 Detailed Description	130
10.31 Utils.cpp	131
Index	133

Chapter 1

JSON2Batch

0.3.1

JSON2Batch was developed for 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 Table of Contents

1. Build Instructions
 - Linux
 - Windows
 - Generating Documentation
2. Documentation
 - Project Structure
3. External Libraries
 - easylogging++
 - LoggingWrapper
 - jsoncpp
4. License

1.2 Build Instructions

1.2.1 Linux

```
git clone -b release https://github.com/DHBWProjectsIT23/JSON2Bat/  
cd JSON2Bat  
cmake -S . -B build  
cmake --build build
```

1.2.1.1 UNIX Compiler Compatibility

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

1.2.2 Windows

The project has been tested on windows using MinGW and Ninja.

1.2.3 Build with MinGW

MinGW can be installed by following Steps 1 through 7 in this [tutorial](#).

```
git clone -b release https://github.com/DHBWProjectsIT23/JSON2Bat/  
cd JSON2Bat  
cmake -S . -B build -G "MinGW Makefiles"  
cmake --build build
```

The project was tested using MinGW with the above mentioned installation and using [this](#) GitHub Action.

1.2.4 Build with Ninja

The Ninja binary can be found [here](#). Alternatively Ninja can be build from [source](#).

```
git clone -b release https://github.com/DHBWProjectsIT23/JSON2Bat/  
cd JSON2Bat  
cmake -S . -B build -G "Ninja"  
cmake --build build
```

The project was tested using Ninja v1.12.0 on a local machine and using v1.10 using [this](#) GitHub Action.

1.2.5 Generating Documentation

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

```
git clone -b release https://github.com/DHBWProjectsIT23/JSON2Bat/  
cd JSON2Bat  
cmake -S . -B build  
cmake --build build --target doxygen_generate
```

1.3 Documentation

The documentation generated by doxygen for this project can be found [here](#). A PDF version can be found [here](#) and a short man page can be found [here](#). After building the project the man page can be accessed by:

```
man assets/man/json2batch.troff
```

1.3.1 Project Structure

The project directory is structured as follows:

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

1.4 External Libraries

1.4.1 easylogging++

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

1.4.2 LoggingWrapper

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

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

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

1.4.3 jsoncpp

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

1.5 License

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

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

StyleHelpers	15
------------------------	----

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cli	Includes everything regarding the CLI	17
config	Namespace used for general project information	18
exceptions	Namespace used for customized exceptions	20
parsing	The namespace containing everything relevant to parsing	20
utilities	Includes all utilities	21

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

BatchCreator	23
cli::CommandLineHandler	30
std::exception	
exceptions::CustomException	37
exceptions::ContainsBadCharacterException	35
exceptions::FailedToOpenFileException	39
exceptions::FileExistsException	47
exceptions::InvalidKeyException	49
exceptions::InvalidTypeException	51
exceptions::InvalidValueException	53
exceptions::MissingKeyException	74
exceptions::MissingTypeException	77
exceptions::NoSuchDirException	79
exceptions::ParsingException	81
exceptions::UnreachableCodeException	83
parsing::FileData	41
parsing::JsonHandler	55
parsing::KeyValidator	67
options	81
utilities::Utils	85

Chapter 5

Class Index

5.1 Class List

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

BatchCreator	Creates a batch file from a FileData obeject	23
cli::CommandLineHandler	Responsible for the Command Line Interface	30
exceptions::ContainsBadCharacterException	Exception for when a string contains bad characters	35
exceptions::CustomException	Base class for all custom exceptions	37
exceptions::FailedToOpenFileException	Exception for when a file can't be opened	39
parsing::FileData	This class contains all data from the json file	41
exceptions::FileExistsException	Exception for an already exisiting outputfile	47
exceptions::InvalidKeyException	Exception for invalid keys	49
exceptions::InvalidTypeException	Exception for invalid types	51
exceptions::InvalidValueException	Exception for an invalid (usually empty) value field	53
parsing::JsonHandler	This file reads all data from the json file	55
parsing::KeyValidator	Validates keys of a Json::Value object	67
exceptions::MissingKeyException	Exception for missing keys within entries	74
exceptions::MissingTypeException	Exception for missing types of entries	77
exceptions::NoSuchDirException	Exception for when a directory does not exist	79
options	The struct containing all possible options	81
exceptions::ParsingException	Exception for syntax errors within the json file	81
exceptions::UnreachableCodeException	Exception for when the application reaches code it shouldn't reach	83
utilities::Utils	Responsible for utility function	85

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	
Contains the main function	109
src/include/BatchCreator.hpp	
Contains the BatchCreator class	91
src/include/CommandLineHandler.hpp	
Responsible for the Command Line Interface	93
src/include/config.hpp	
Configures general project information	95
src/include/Exceptions.hpp	
Contains all the custom exceptions used in the project	97
src/include/FileData.hpp	
This file contains the FileData class	101
src/include/JsonHandler.hpp	
This file contains the JsonHandler class	103
src/include/KeyValidator.hpp	
This file contains the KeyValidator class	106
src/include/Utils.hpp	
.	108
src/sources/BatchCreator.cpp	
Contains the implementation of the BatchCreator class	116
src/sources/CommandLineHandler.cpp	
Implementation for the Command Line Interface	119
src/sources/FileData.cpp	
Implementation of the FileData class	121
src/sources/JsonHandler.cpp	
Implementation of the JsonHandler class	124
src/sources/KeyValidator.cpp	
Implementation for the KeyValidator class	126
src/sources/Utils.cpp	
Implementation for the Utils class	129

Chapter 7

Topic Documentation

7.1 StyleHelpers

Static variables to help with CLI styling.

Static variables to help with CLI styling.

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

Chapter 8

Namespace Documentation

8.1 cli Namespace Reference

Includes everything regarding the CLI.

Classes

- class [CommandLineHandler](#)
Responsible for the Command Line Interface.

Variables

- static const struct option [options](#) []

8.1.1 Detailed Description

Includes everything regarding the CLI.

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

See also

[CommandLineHandler](#)
[options](#)
[StyleHelpers](#)

8.1.2 Variable Documentation

8.1.2.1 options

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

Initial value:

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

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

8.2 config Namespace Reference

Namespace used for general project information.

Variables

- constexpr auto [LOG_CONFIG](#) = "/home/simon/1_Coding/projectJsonToBat/build/Debug/config/easylogging.conf"↩
- constexpr auto [EXECUTABLE_NAME](#) = "json2batch"
- constexpr auto [MAJOR_VERSION](#) = "0"
- constexpr auto [MINOR_VERSION](#) = "3"
- constexpr auto [PATCH_VERSION](#) = "1"
- constexpr auto [DESCRIPTION](#) = "A simple tool to convert json to batch."
- constexpr auto [PROJECT_NAME](#) = "JSON2Batch"
- constexpr auto [AUTHORS](#) = "@AUTHORS"
- constexpr auto [HOMEPAGE_URL](#) = "https://dhwprojectsit23.github.io/JSON2Bat"

8.2.1 Detailed Description

Namespace used for general project information.

8.2.2 Variable Documentation

8.2.2.1 AUTHORS

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

Definition at line 33 of file [config.hpp](#).

8.2.2.2 DESCRIPTION

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

Definition at line 31 of file [config.hpp](#).

8.2.2.3 EXECUTABLE_NAME

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

Definition at line 27 of file [config.hpp](#).

8.2.2.4 HOMEPAGE_URL

```
constexpr auto config::HOMEPAGE_URL = "https://dhwprojectsit23.github.io/JSON2Bat" [inline],  
[constexpr]
```

Definition at line 34 of file [config.hpp](#).

8.2.2.5 LOG_CONFIG

```
constexpr auto config::LOG_CONFIG = "/home/simon/1_Coding/projectJsonToBat/build/Debug/config/easylogging.↵  
conf" [inline], [constexpr]
```

Definition at line 26 of file [config.hpp](#).

8.2.2.6 MAJOR_VERSION

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

Definition at line 28 of file [config.hpp](#).

8.2.2.7 MINOR_VERSION

```
constexpr auto config::MINOR_VERSION = "3" [inline], [constexpr]
```

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

8.2.2.8 PATCH_VERSION

```
constexpr auto config::PATCH_VERSION = "1" [inline], [constexpr]
```

Definition at line 30 of file [config.hpp](#).

8.2.2.9 PROJECT_NAME

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

Definition at line 32 of file [config.hpp](#).

8.3 exceptions Namespace Reference

Namespace used for customized exceptions.

Classes

- class [ContainsBadCharacterException](#)
Exception for when a string contains bad characters.
- 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 existing outputfile.
- class [InvalidKeyException](#)
Exception for invalid keys.
- class [InvalidTypeException](#)
Exception for invalid types.
- class [InvalidValueException](#)
Exception for an invalid (usually empty) value field.
- class [MissingKeyException](#)
Exception for missing keys within entries.
- class [MissingTypeException](#)
Exception for missing types of entries.
- class [NoSuchDirException](#)
Exception for when a directory does not exist.
- class [ParsingException](#)
Exception for syntax errors within the json file.
- class [UnreachableCodeException](#)
Exception for when the application reaches code it shouldn't reach.

8.3.1 Detailed Description

Namespace used for customized exceptions.

8.4 parsing Namespace Reference

The namespace containing everything relevant to parsing.

Classes

- class [FileData](#)
This class contains all data from the json file.
- class [JsonHandler](#)
This file reads all data from the json file.
- class [KeyValidator](#)
Validates keys of a `Json::Value` object.

8.4.1 Detailed Description

The namespace containing everything relevant to parsing.

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

See also

[JsonHandler](#)
[FileData](#)
[KeyValidator](#)
[BatchCreator](#)

8.5 utilities Namespace Reference

Includes all utilities.

Classes

- class [Utils](#)
Responsible for utility function.

8.5.1 Detailed Description

Includes all utilities.

This namespace includes the [Utils](#) class with utility functions which can be used throughout the project.

See also

[Utils](#)

Chapter 9

Class Documentation

9.1 BatchCreator Class Reference

Creates a batch file from a FileData object.

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

Public Member Functions

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

Private Member Functions

- void [createBatch](#) () const
Creates the batch stream.
- void [writeStart](#) () const
Writes 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 [writeApplication](#) () const
If an application is given, it is started at the end.
- void [writeEnd](#) () const
Writes the end of the batch file.

Private Attributes

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

9.1.1 Detailed Description

Creates a batch file from a FileData obeject.

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

See also

[FileData](#)

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

9.1.2 Constructor & Destructor Documentation

9.1.2.1 BatchCreator()

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

Initializes the [BatchCreator](#).

Creates a stringstream and calls the [createBatch\(\)](#) function

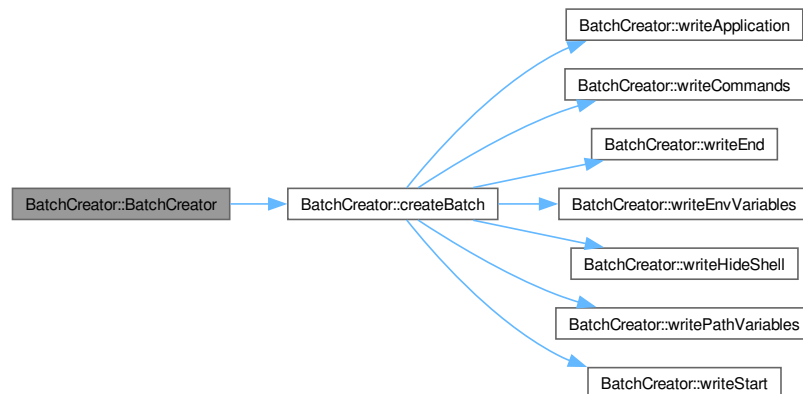
Parameters

<i>fileData</i>	A shared pointer to the FileData object
-----------------	-----------------------------------------

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

References [createBatch\(\)](#), and [dataStream](#).

Here is the call graph for this function:



9.1.3 Member Function Documentation

9.1.3.1 `createBatch()`

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

Creates the batch stream.

< `FileData` object

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

Definition at line 25 of file [BatchCreator.cpp](#).

References [writeApplication\(\)](#), [writeCommands\(\)](#), [writeEnd\(\)](#), [writeEnvVariables\(\)](#), [writeHideShell\(\)](#), [writePathVariables\(\)](#), and [writeStart\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.3.2 `getDataStream()`

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

Returns the stringstream.

Returns

A shared pointer to the stringstream

Definition at line 46 of file [BatchCreator.hpp](#).

References [dataStream](#).

Here is the caller graph for this function:



9.1.3.3 writeApplication()

```
void BatchCreator::writeApplication ( ) 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.

- {ReqFunc16}
- {ReqFunc25}

Definition at line 81 of file [BatchCreator.cpp](#).

References [dataStream](#), and [fileData](#).

Here is the caller graph for this function:



9.1.3.4 writeCommands()

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

Writes the commands to be executed.

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

- {ReqFunc20}
- {ReqFunc22}

Definition at line 53 of file [BatchCreator.cpp](#).

References [dataStream](#), and [fileData](#).

Here is the caller graph for this function:



9.1.3.5 writeEnd()

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

Writes the end of the batch file.

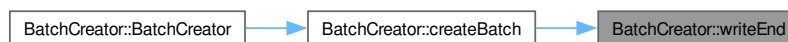
Writes the end of the batch file, which is always the same:

- @ECHO ON

Definition at line 99 of file [BatchCreator.cpp](#).

References [dataStream](#).

Here is the caller graph for this function:



9.1.3.6 writeEnvVariables()

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

Set's environment variables.

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

- Entry under "key" = Entry under "value"
- {ReqFunc20}
- {ReqFunc21}

Definition at line 62 of file [BatchCreator.cpp](#).

References [dataStream](#), and [fileData](#).

Here is the caller graph for this function:



9.1.3.7 writeHideShell()

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

Writes the visibility of the shell.

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

- {ReqFunc19}

Definition at line 42 of file [BatchCreator.cpp](#).

References [dataStream](#), and [fileData](#).

Here is the caller graph for this function:



9.1.3.8 writePathVariables()

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

Set's the path variables.

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

- {ReqFunc20}
- {ReqFunc23}

Definition at line 70 of file [BatchCreator.cpp](#).

References [dataStream](#), and [fileData](#).

Here is the caller graph for this function:



9.1.3.9 writeStart()

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

Writes the start of the batch file.

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

- setzt ECHO off
- startet cmd.exe

Definition at line 36 of file [BatchCreator.cpp](#).

References [dataStream](#).

Here is the caller graph for this function:



9.1.4 Member Data Documentation

9.1.4.1 dataStream

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

Definition at line 52 of file [BatchCreator.hpp](#).

9.1.4.2 fileData

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

< stringstream for the batch file

Definition at line 54 of file [BatchCreator.hpp](#).

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

- src/include/[BatchCreator.hpp](#)
- src/sources/[BatchCreator.cpp](#)

9.2 cli::CommandLineHandler Class Reference

Responsible for the Command Line Interface.

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


Public Member Functions

- [CommandLineHandler](#) ()=delete
The Constructor of the [CommandLineHandler](#) Class.
- [~CommandLineHandler](#) ()=delete
The Destructor of the [CommandLineHandler](#) Class.

Static Public Member Functions

- static void [printHelp](#) ()
Prints the help message.
- static void [printVersion](#) ()
Prints the version message.
- static void [printCredits](#) ()
Prints the credits message.
- static std::tuple< std::optional< std::string >, std::vector< std::string > > [parseArguments](#) (int argc, char *argv[])
Parses the Command Line Arguments.

9.2.1 Detailed Description

Responsible for the Command Line Interface.

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

Author

Simon Blum

Date

2024-04-18

Version

0.1.5

See also

[options](#)

Definition at line 56 of file [CommandLineHandler.hpp](#).

9.2.2 Constructor & Destructor Documentation

9.2.2.1 CommandLineHandler()

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

The Constructor of the [CommandLineHandler](#) Class.

Note

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

9.2.2.2 ~CommandLineHandler()

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

The Destructor of the [CommandLineHandler](#) Class.

Note

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

9.2.3 Member Function Documentation

9.2.3.1 parseArguments()

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

Parses the Command Line Arguments.

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

- {ReqFunc4}
- {ReqFunc5}
- {ReqNonFunc4}

Parameters

<i>argc</i>	The number of arguments given
<i>argv</i>	The arguments given

Returns

Returns a tuple containing the output directory and the files

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

References [printCredits\(\)](#), [printHelp\(\)](#), and [printVersion\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.2.3.2 printCredits()

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

Prints the credits message.

- {ReqFunc3}

Note

This function ends the application.

Definition at line 50 of file [CommandLineHandler.cpp](#).

References [config::AUTHORS](#), [config::DESCRIPTION](#), [config::HOMEPAGE_URL](#), [config::MAJOR_VERSION](#), [config::MINOR_VERSION](#), [config::PATCH_VERSION](#), and [config::PROJECT_NAME](#).

Here is the caller graph for this function:

**9.2.3.3 printHelp()**

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

Prints the help message.

- {ReqFunc1}
- {ReqFunc2}

Note

This function ends the application.

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

References [config::EXECUTABLE_NAME](#).

Here is the caller graph for this function:



9.2.3.4 printVersion()

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

Prints the version message.

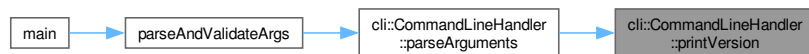
Note

This function ends the application.

Definition at line 44 of file [CommandLineHandler.cpp](#).

References [config::MAJOR_VERSION](#), [config::MINOR_VERSION](#), [config::PATCH_VERSION](#), and [config::PROJECT_NAME](#).

Here is the caller graph for this function:



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

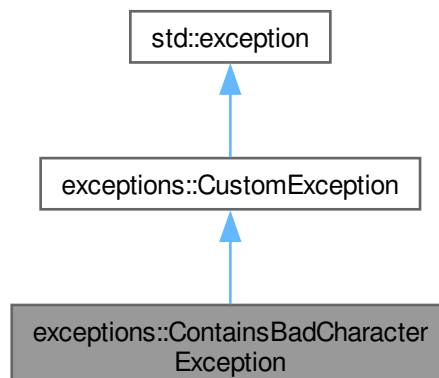
- [src/include/CommandLineHandler.hpp](#)
- [src/sources/CommandLineHandler.cpp](#)

9.3 exceptions::ContainsBadCharacterException Class Reference

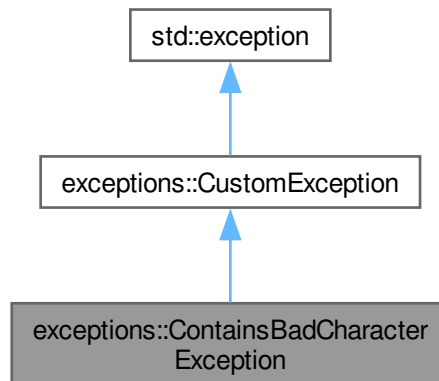
Exception for when a string contains bad characters.

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

Inheritance diagram for `exceptions::ContainsBadCharacterException`:



Collaboration diagram for exceptions::ContainsBadCharacterException:



Public Member Functions

- [ContainsBadCharacterException](#) (const std::string &value)
- const char * [what](#) () const noexcept override

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- std::string [message](#)

9.3.1 Detailed Description

Exception for when a string contains bad characters.

Definition at line 295 of file [Exceptions.hpp](#).

9.3.2 Constructor & Destructor Documentation

9.3.2.1 ContainsBadCharacterException()

```
exceptions::ContainsBadCharacterException::ContainsBadCharacterException (  
    const std::string & value ) [inline], [explicit]
```

Definition at line 300 of file [Exceptions.hpp](#).

References [message](#).

9.3.3 Member Function Documentation

9.3.3.1 what()

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

Definition at line 304 of file [Exceptions.hpp](#).

References [message](#).

9.3.4 Member Data Documentation

9.3.4.1 message

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

Definition at line 297 of file [Exceptions.hpp](#).

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

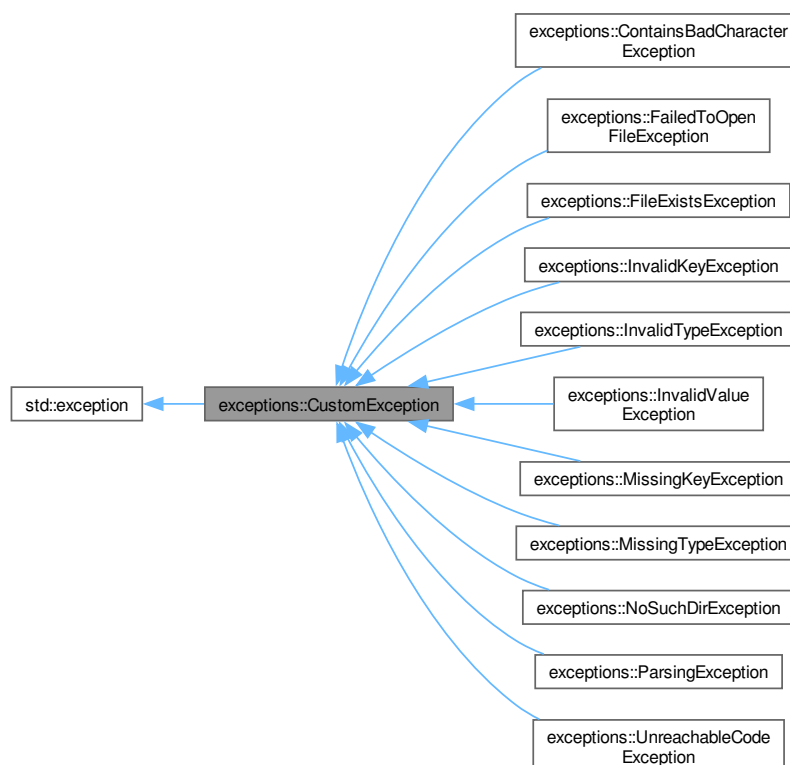
- [src/include/Exceptions.hpp](#)

9.4 exceptions::CustomException Class Reference

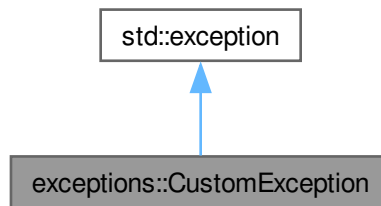
Base class for all custom exceptions.

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

Inheritance diagram for exceptions::CustomException:



Collaboration diagram for exceptions::CustomException:



Public Member Functions

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

9.4.1 Detailed Description

Base class for all custom exceptions.

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

See also

`std::exception`

Definition at line 35 of file [Exceptions.hpp](#).

9.4.2 Member Function Documentation

9.4.2.1 `what()`

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

Definition at line 37 of file [Exceptions.hpp](#).

Here is the caller graph for this function:



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

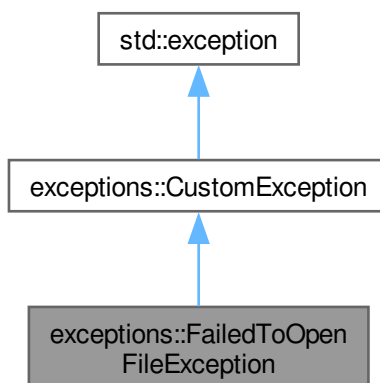
- `src/include/Exceptions.hpp`

9.5 exceptions::FailedToOpenFileException Class Reference

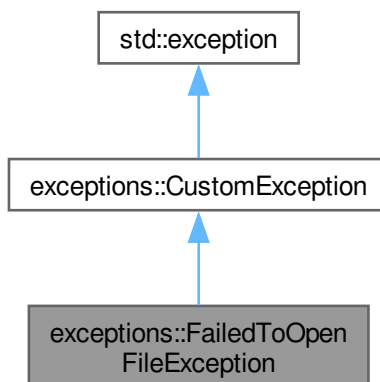
Exception for when a file can't be opened.

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

Inheritance diagram for exceptions::FailedToOpenFileException:



Collaboration diagram for exceptions::FailedToOpenFileException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

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

Private Attributes

- `std::string message`

9.5.1 Detailed Description

Exception for when a file can't be opened.

Definition at line [259](#) of file [Exceptions.hpp](#).

9.5.2 Constructor & Destructor Documentation

9.5.2.1 FailedToOpenFileException()

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

Definition at line [264](#) of file [Exceptions.hpp](#).

References [message](#).

9.5.3 Member Function Documentation

9.5.3.1 what()

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

Definition at line [268](#) of file [Exceptions.hpp](#).

References [message](#).

9.5.4 Member Data Documentation

9.5.4.1 message

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

Definition at line [261](#) of file [Exceptions.hpp](#).

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

- `src/include/Exceptions.hpp`

9.6 parsing::FileData Class Reference

This class contains all data from the json file.

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

Public Member Functions

- void [setOutputFile](#) (std::string &newOutputfile)
Setter for this->outputfile.
- void [setHideShell](#) (bool newHideShell)
Setter for this->hideshell.
- void [setApplication](#) (const std::string &newApplication)
Setter for this->application.
- void [addCommand](#) (const std::string &command)
Adds a given command to this->commands.
- void [addEnvironmentVariable](#) (const std::string &name, const std::string &value)
Adds a given tuple to this->environmentVariables.
- void [addPathValue](#) (const std::string &pathValue)
Add's a given value to this->pathValues.
- const std::string & [getOutputFile](#) () const
Getter for this->outputfile.
- bool [getHideShell](#) () const
Getter for this->hideShell.
- const std::optional< std::string > & [getApplication](#) () const
Getter for this->application.
- const std::vector< std::string > & [getCommands](#) () const
Getter for this->commands.
- const std::vector< std::tuple< std::string, std::string > > & [getEnvironmentVariables](#) () const
Getter for this->environmentVariables.
- const std::vector< std::string > & [getPathValues](#) () const
Getter for this->pathValues.

Private Attributes

- std::string [outputfile](#)
- bool [hideShell](#)
- std::optional< std::string > [application](#)
- std::vector< std::string > [commands](#)
- std::vector< std::tuple< std::string, std::string > > [environmentVariables](#)
- std::vector< std::string > [pathValues](#)

9.6.1 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.

- {ReqFunc14}

Definition at line 32 of file [FileData.hpp](#).

9.6.2 Member Function Documentation

9.6.2.1 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

<i>command</i>	The command to be added
----------------	-------------------------

Exceptions

exceptions::InvalidValueException	
---------------------------------------------------	--

Definition at line 56 of file [FileData.cpp](#).

References [commands](#).

9.6.2.2 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

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

Exceptions

exceptions::InvalidValueException	
---------------------------------------------------	--

Definition at line 67 of file [FileData.cpp](#).

References [environmentVariables](#).

9.6.2.3 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

<i>pathValue</i>	The value to be added
------------------	-----------------------

Exceptions

exceptions::InvalidValueException	
---------------------------------------------------	--

Definition at line 83 of file [FileData.cpp](#).

References [pathValues](#).

9.6.2.4 getApplication()

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

Getter for this->application.

Returns

The assigned application

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

References [application](#).

9.6.2.5 getCommands()

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

Getter for this->commands.

Returns

The vector of assigned commands

Definition at line 130 of file [FileData.hpp](#).

References [commands](#).

9.6.2.6 `getEnvironmentVariables()`

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

Getter for this->environmentVariables.

Returns

The vector of assigned env variables

Definition at line 139 of file [FileData.hpp](#).

References [environmentVariables](#).

9.6.2.7 `getHideShell()`

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

Getter for this->hideShell.

Returns

The assigned value for hideshow

Definition at line 114 of file [FileData.hpp](#).

References [hideShell](#).

9.6.2.8 `getOutputFile()`

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

Getter for this->outputfile.

Returns

The assigned outputfile

Definition at line 106 of file [FileData.hpp](#).

References [outputfile](#).

9.6.2.9 `getPathValues()`

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

Getter for this->pathValues.

Returns

The vector of assigned pathValues

Definition at line 147 of file [FileData.hpp](#).

References [pathValues](#).

9.6.2.10 `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

<i>newApplication</i>	The application to be set
-----------------------	---------------------------

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

References [application](#).

9.6.2.11 setHideShell()

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

Setter for this->hideshell.

Parameters

<i>newHideShell</i>	The hideshell value to be set
---------------------	-------------------------------

Definition at line 50 of file [FileData.hpp](#).

References [hideShell](#).

9.6.2.12 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

<i>newOutputfile</i>	The outputfile to be set
----------------------	--------------------------

Exceptions

exceptions::InvalidValueException	
---------------------------------------------------	--

Definition at line 18 of file [FileData.cpp](#).

References [outputfile](#).

9.6.3 Member Data Documentation

9.6.3.1 application

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

Definition at line 154 of file [FileData.hpp](#).

9.6.3.2 commands

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

Definition at line 156 of file [FileData.hpp](#).

9.6.3.3 environmentVariables

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

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

9.6.3.4 hideShell

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

Definition at line 153 of file [FileData.hpp](#).

9.6.3.5 outputfile

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

Definition at line 152 of file [FileData.hpp](#).

9.6.3.6 pathValues

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

Definition at line 160 of file [FileData.hpp](#).

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

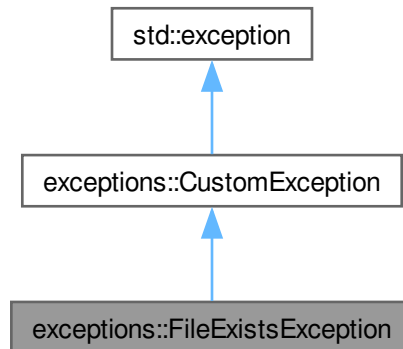
- [src/include/FileData.hpp](#)
- [src/sources/FileData.cpp](#)

9.7 exceptions::FileExistsException Class Reference

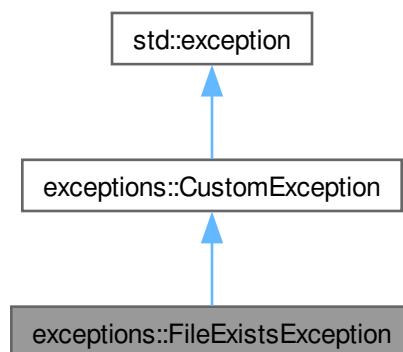
Exception for an already existing outputfile.

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

Inheritance diagram for exceptions::FileExistsException:



Collaboration diagram for exceptions::FileExistsException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

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

Private Attributes

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

9.7.1 Detailed Description

Exception for an already existing outputfile.

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

9.7.2 Constructor & Destructor Documentation

9.7.2.1 FileExistsException()

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

Note

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

Definition at line 80 of file [Exceptions.hpp](#).

References [file](#), and [message](#).

9.7.3 Member Function Documentation

9.7.3.1 what()

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

Definition at line 92 of file [Exceptions.hpp](#).

References [message](#).

9.7.4 Member Data Documentation

9.7.4.1 file

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

Definition at line 76 of file [Exceptions.hpp](#).

9.7.4.2 message

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

Definition at line 77 of file [Exceptions.hpp](#).

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

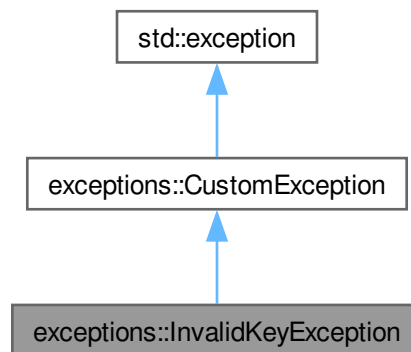
- [src/include/Exceptions.hpp](#)

9.8 exceptions::InvalidKeyException Class Reference

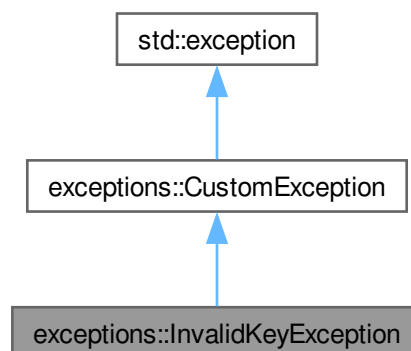
Exception for invalid keys.

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

Inheritance diagram for exceptions::InvalidKeyException:



Collaboration diagram for exceptions::InvalidKeyException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- std::string [message](#) = "Invalid key found!"

9.8.1 Detailed Description

Exception for invalid keys.

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

See also

[parsing::KeyValidator::validKeys](#)

[parsing::KeyValidator::validEntryKeys](#)

Definition at line 135 of file [Exceptions.hpp](#).

9.8.2 Constructor & Destructor Documentation

9.8.2.1 InvalidKeyException()

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

Definition at line 140 of file [Exceptions.hpp](#).

References [message](#).

9.8.3 Member Function Documentation

9.8.3.1 what()

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

Definition at line 149 of file [Exceptions.hpp](#).

References [message](#).

9.8.4 Member Data Documentation

9.8.4.1 message

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

Definition at line 137 of file [Exceptions.hpp](#).

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

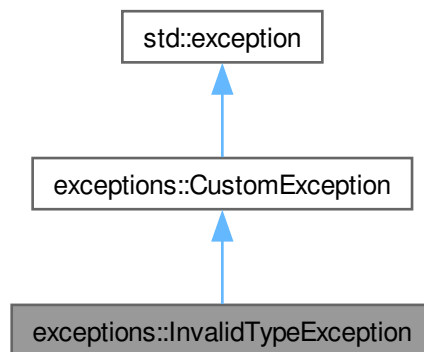
- [src/include/Exceptions.hpp](#)

9.9 exceptions::InvalidTypeException Class Reference

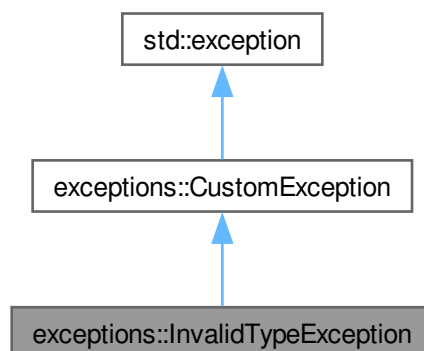
Exception for invalid types.

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

Inheritance diagram for exceptions::InvalidTypeException:



Collaboration diagram for exceptions::InvalidTypeException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- const std::string [type](#)
- std::string [message](#)

9.9.1 Detailed Description

Exception for invalid types.

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

Definition at line [162](#) of file [Exceptions.hpp](#).

9.9.2 Constructor & Destructor Documentation

9.9.2.1 InvalidTypeException()

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

Note

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

Definition at line [168](#) of file [Exceptions.hpp](#).

References [message](#), and [type](#).

9.9.3 Member Function Documentation

9.9.3.1 what()

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

Definition at line [179](#) of file [Exceptions.hpp](#).

References [message](#).

9.9.4 Member Data Documentation

9.9.4.1 message

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

Definition at line 165 of file [Exceptions.hpp](#).

9.9.4.2 type

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

Definition at line 164 of file [Exceptions.hpp](#).

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

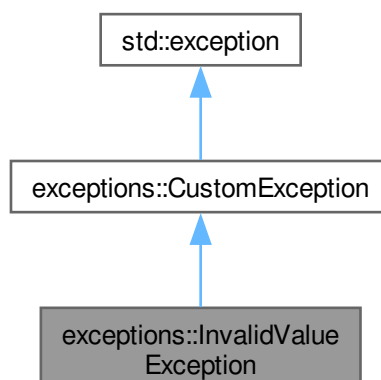
- [src/include/Exceptions.hpp](#)

9.10 exceptions::InvalidValueException Class Reference

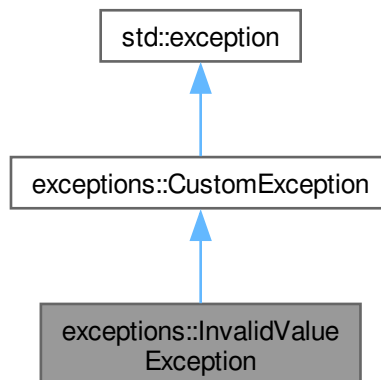
Exception for an ivalid (usually empty) value field.

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

Inheritance diagram for exceptions::InvalidValueException:



Collaboration diagram for exceptions::InvalidValueException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- const std::string [key](#)
- std::string [message](#)

9.10.1 Detailed Description

Exception for an ivalid (usually empty) value field.

Definition at line 101 of file [Exceptions.hpp](#).

9.10.2 Constructor & Destructor Documentation

9.10.2.1 InvalidValueException()

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

Note

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

Definition at line 107 of file [Exceptions.hpp](#).

References [key](#), and [message](#).

9.10.3 Member Function Documentation

9.10.3.1 what()

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

Definition at line 119 of file [Exceptions.hpp](#).

References [message](#).

9.10.4 Member Data Documentation

9.10.4.1 key

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

Definition at line 103 of file [Exceptions.hpp](#).

9.10.4.2 message

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

Definition at line 104 of file [Exceptions.hpp](#).

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

- [src/include/Exceptions.hpp](#)

9.11 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 hideshow 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.
- static bool `containsBadCharacter` (const std::string_view &str)
Check if a string contains a bad character.

Private Attributes

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

9.11.1 Detailed Description

This file reads all data from the json file.

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

See also

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

Definition at line 47 of file `JsonHandler.hpp`.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 JsonHandler() [1/2]

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

Constructor without arguments.

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

Definition at line 55 of file [JsonHandler.hpp](#).

9.11.2.2 JsonHandler() [2/2]

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

The constructor.

This constructor calls this->[parseFile\(\)](#) when called.

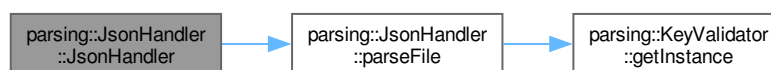
Parameters

<i>filename</i>	Name of the json file
-----------------	-----------------------

Definition at line 23 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



9.11.3 Member Function Documentation

9.11.3.1 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.

- {ReqFunc16}

Definition at line 84 of file [JsonHandler.cpp](#).

References [containsBadCharacter\(\)](#), [data](#), [utilities::Utils::escapeString\(\)](#), and [root](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.2 assignCommand()

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

Assigns an command to this->data.

- {ReqFunc12}

Parameters

<i>entry</i>	The entry with the command
--------------	----------------------------

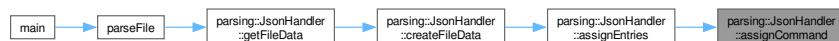
Definition at line 118 of file [JsonHandler.cpp](#).

References [containsBadCharacter\(\)](#), [data](#), and [utilities::Utils::escapeString\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.3 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.

- {ReqFunc10}

Parameters

<i>entry</i>	Json::Value containing an array with entries
--------------	----------------------------------------------

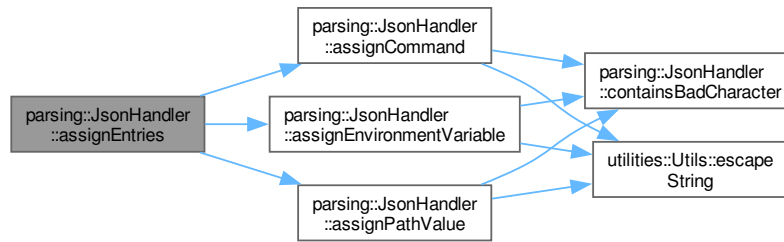
Exceptions

<i>exceptions::UnreachableCodeException</i>	
-------------------------------------------------------------	--

Definition at line 94 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.4 assignEnvironmentVariable()

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

Assigns an environmentVariable to this->data.

- {ReqFunc11}

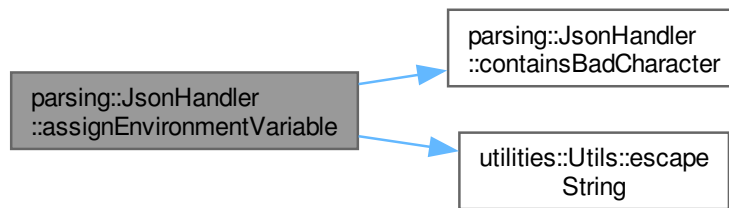
Parameters

<i>entry</i>	The entry with the environmentVariable
--------------	----------------------------------------

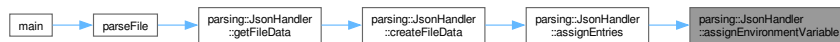
Definition at line 128 of file [JsonHandler.cpp](#).

References [containsBadCharacter\(\)](#), [data](#), and [utilities::Utils::escapeString\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.5 assignHideShell()

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

Assigns the hideshow value to this->data.

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

- {ReqFunc9}

Definition at line 78 of file [JsonHandler.cpp](#).

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

Here is the caller graph for this function:



9.11.3.6 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.

- {ReqFunc8}

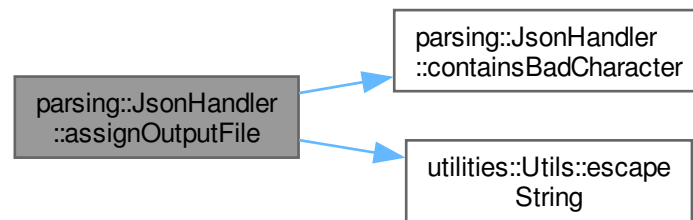
Exceptions

exceptions::FileExistsException

Definition at line 68 of file [JsonHandler.cpp](#).

References [containsBadCharacter\(\)](#), [data](#), [utilities::Utils::escapeString\(\)](#), and [root](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.7 assignPathValue()

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

Assigns a path value to this->data.

- {ReqFunc13}

Parameters

<i>entry</i>	The entry with the path value
--------------	-------------------------------

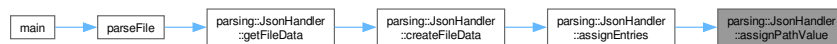
Definition at line 144 of file [JsonHandler.cpp](#).

References [containsBadCharacter\(\)](#), [data](#), and [utilities::Utils::escapeString\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.8 containsBadCharacter()

```
bool parsing::JsonHandler::containsBadCharacter (
    const std::string_view & str ) [static], [private]
```

Check if a string contains a bad character.

This method checks if a given string contains a bad character. Bad characters are declared in a set within the function. This is done to ensure, that no characters such as line breaks, break the later generated batch file.

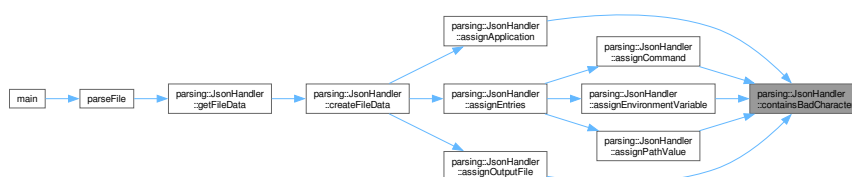
Parameters

<i>str</i>	The string to be checked
------------	--------------------------

@bool If the string contains a bad char or not

Definition at line 154 of file [JsonHandler.cpp](#).

Here is the caller graph for this function:



9.11.3.9 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.

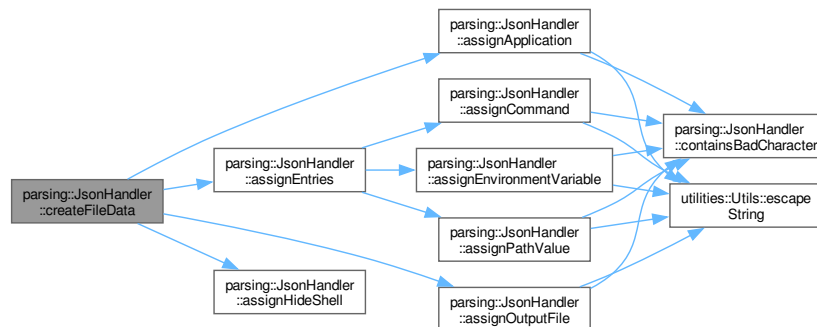
Returns

Pointer to the created instance of [FileData](#)

Definition at line 58 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.10 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.

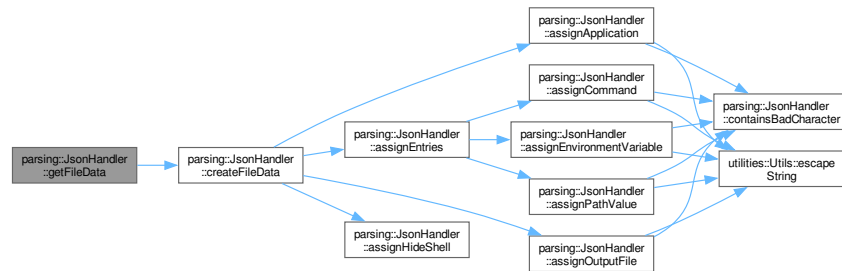
Returns

Pointer to the [FileData](#) Object with the parsed data from json

Definition at line 53 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:

**9.11.3.11 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

<i>filename</i>	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	

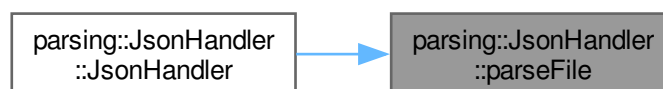
Definition at line 28 of file [JsonHandler.cpp](#).

References [parsing::KeyValidator::getInstance\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.4 Member Data Documentation

9.11.4.1 data

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

Definition at line 179 of file [JsonHandler.hpp](#).

9.11.4.2 root

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

Definition at line 178 of file [JsonHandler.hpp](#).

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

- src/include/[JsonHandler.hpp](#)
- src/sources/[JsonHandler.cpp](#)

9.12 parsing::KeyValidator Class Reference

Validates keys of a `Json::Value` object.

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

Public Member Functions

- `std::vector< std::tuple< int, std::string > >` [validateKeys](#) (const `Json::Value` &root, const `std::string` &filename)
Validate keys off a `Json::Value` object.

Static Public Member Functions

- static [KeyValidator](#) & [getInstance](#) ()
Get the instance of this class.

Private Member Functions

- `std::vector< std::tuple< int, std::string > >` [getWrongKeys](#) (const `Json::Value` &root, const `std::string` &filename) const
Retrieve the wrong keys from a `Json::Value` object.
- void [validateTypes](#) (const `std::string` &filename, const `Json::Value` &entry, const `std::unordered_set< std::string >` &entryKeys)
Validates types from the entries array.
- `std::vector< std::tuple< int, std::string > >` [validateEntries](#) (const `std::string` &filename, const `std::unordered_set< std::string >` &entryKeys) const
Validates that keys within the entries array are valid.

Static Private Member Functions

- static `std::optional< int >` [getUnknownKeyLine](#) (const `std::string` &filename, const `std::string` &wrongKey)
Get the line of an unknown key.

Private Attributes

- `std::unordered_set< std::string >` [validKeys](#)
- `std::unordered_set< std::string >` [validEntryKeys](#)
- `std::unordered_map< std::string_view, std::vector< std::string > >` [typeToKeys](#)

9.12.1 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.

- {ReqFunc17}

Definition at line 31 of file [KeyValidator.hpp](#).

9.12.2 Member Function Documentation

9.12.2.1 getInstance()

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

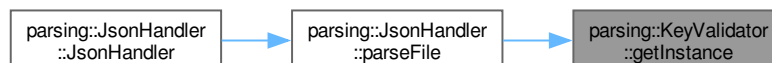
Get the instance of this class.

Returns

Reference to the instance of this class

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

Here is the caller graph for this function:



9.12.2.2 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

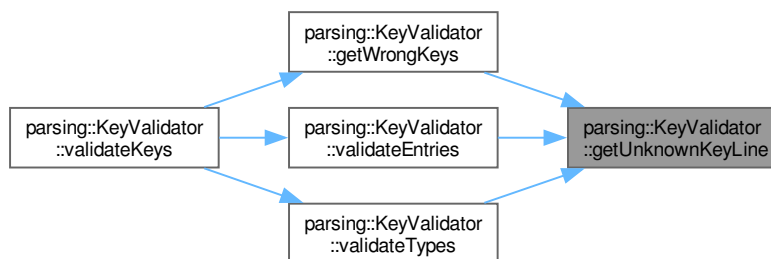
<i>filename</i>	The filename which should contain the key
<i>wrongKey</i>	The key to be searched for

Returns

The line of the key, if it was found

Definition at line 133 of file [KeyValidator.cpp](#).

Here is the caller graph for this function:



9.12.2.3 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

<i>root</i>	The <code>Json::Value</code> object to be validated.
<i>filename</i>	The filename from which 'root' is from.

Returns

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

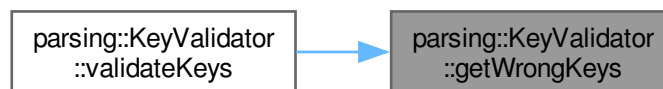
Definition at line 55 of file [KeyValidator.cpp](#).

References [getUnknownKeyLine\(\)](#), and [validKeys](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.2.4 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 method goes through each of the entries, and validates, that the keys are part of the `validEntryKeys` attribute.

Parameters

<i>filename</i>	The filename from which the entries are from
<i>entryKeys</i>	The keys of the entries

Returns

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

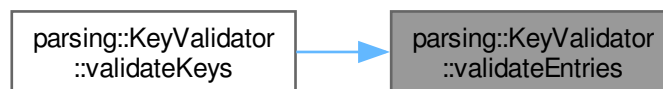
Definition at line 78 of file [KeyValidator.cpp](#).

References [getUnknownKeyLine\(\)](#), and [validEntryKeys](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.2.5 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

<i>root</i>	The <code>Json::Value</code> object to be validated.
<i>filename</i>	The filename from which 'root' is from.

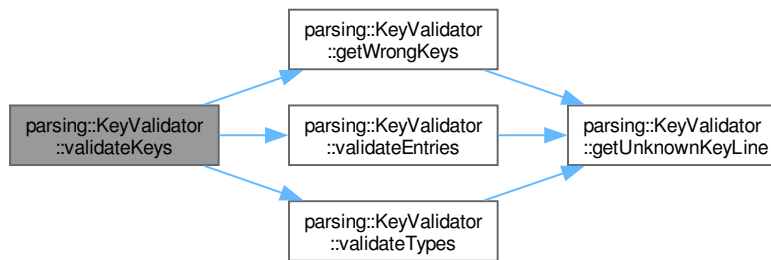
Returns

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:



9.12.2.6 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

<i>filename</i>	The filename from which 'entry' is from
<i>entry</i>	The entry to be validated
<i>entryKeys</i>	The keys of the entry

Exceptions

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

Definition at line 100 of file [KeyValidator.cpp](#).

References [getUnknownKeyLine\(\)](#), and [typeToKeys](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.3 Member Data Documentation

9.12.3.1 typeToKeys

```
std::unordered_map<std::string_view, std::vector<std::string> > parsing::KeyValidator::typeToKeys [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 145 of file [KeyValidator.hpp](#).

9.12.3.2 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 should improve lookup performance from O(n) to O(1)

Definition at line 138 of file [KeyValidator.hpp](#).

9.12.3.3 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 should improve lookup performance from $O(n)$ to $O(1)$

Definition at line 131 of file [KeyValidator.hpp](#).

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

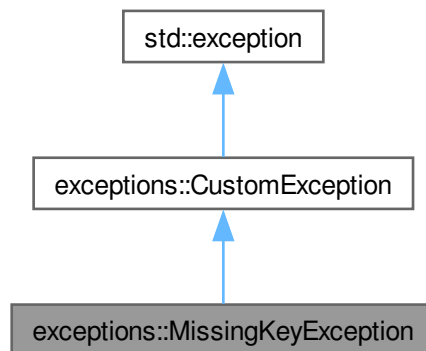
- [src/include/KeyValidator.hpp](#)
- [src/sources/KeyValidator.cpp](#)

9.13 exceptions::MissingKeyException Class Reference

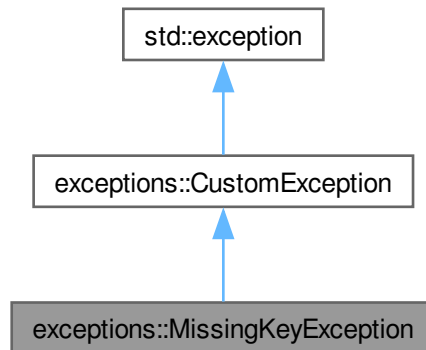
Exception for missing keys within entries.

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

Inheritance diagram for exceptions::MissingKeyException:



Collaboration diagram for exceptions::MissingKeyException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- std::string [message](#)
- std::string [type](#)
- std::string [key](#)

9.13.1 Detailed Description

Exception for missing keys within entries.

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

Definition at line 191 of file [Exceptions.hpp](#).

9.13.2 Constructor & Destructor Documentation

9.13.2.1 MissingKeyException()

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

Note

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

Definition at line 198 of file [Exceptions.hpp](#).

References [key](#), [message](#), and [type](#).

9.13.3 Member Function Documentation

9.13.3.1 what()

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

Definition at line 210 of file [Exceptions.hpp](#).

References [message](#).

9.13.4 Member Data Documentation

9.13.4.1 key

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

Definition at line 195 of file [Exceptions.hpp](#).

9.13.4.2 message

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

Definition at line 193 of file [Exceptions.hpp](#).

9.13.4.3 type

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

Definition at line 194 of file [Exceptions.hpp](#).

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

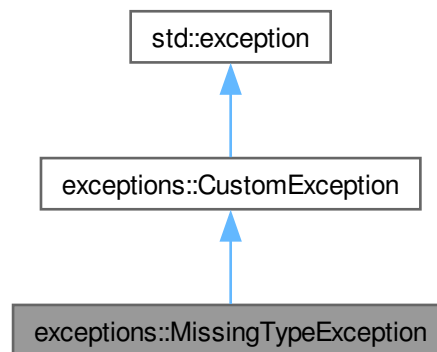
- [src/include/Exceptions.hpp](#)

9.14 exceptions::MissingTypeException Class Reference

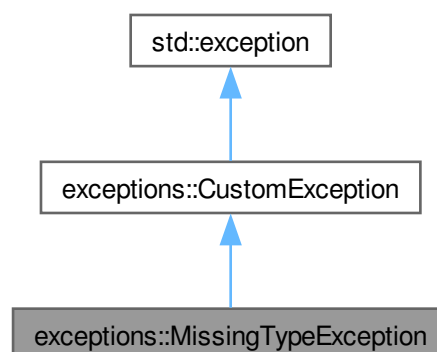
Exception for missing types of entries.

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

Inheritance diagram for exceptions::MissingTypeException:



Collaboration diagram for exceptions::MissingTypeException:



Public Member Functions

- [MissingTypeException](#) ()
- const char * [what](#) () const noexcept override

Public Member Functions inherited from [exceptions::CustomException](#)

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

Private Attributes

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

9.14.1 Detailed Description

Exception for missing types of entries.

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

Definition at line [221](#) of file [Exceptions.hpp](#).

9.14.2 Constructor & Destructor Documentation

9.14.2.1 `MissingTypeException()`

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

Definition at line [226](#) of file [Exceptions.hpp](#).

References [message](#).

9.14.3 Member Function Documentation

9.14.3.1 `what()`

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

Definition at line [229](#) of file [Exceptions.hpp](#).

References [message](#).

9.14.4 Member Data Documentation

9.14.4.1 `message`

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

Definition at line [223](#) of file [Exceptions.hpp](#).

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

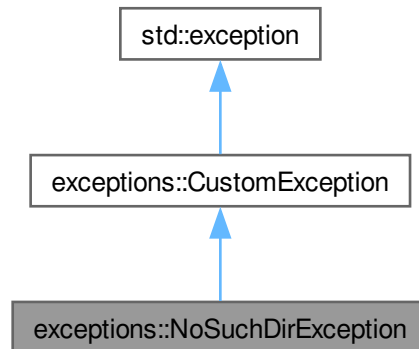
- `src/include/Exceptions.hpp`

9.15 exceptions::NoSuchDirException Class Reference

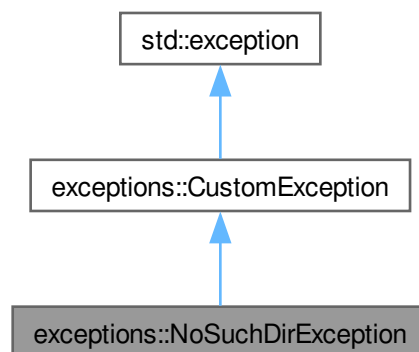
Exception for when a directory does not exist.

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

Inheritance diagram for exceptions::NoSuchDirException:



Collaboration diagram for exceptions::NoSuchDirException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

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

Private Attributes

- `std::string message`

9.15.1 Detailed Description

Exception for when a directory does not exist.

Definition at line 277 of file [Exceptions.hpp](#).

9.15.2 Constructor & Destructor Documentation

9.15.2.1 NoSuchDirException()

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

Definition at line 282 of file [Exceptions.hpp](#).

References [message](#).

9.15.3 Member Function Documentation

9.15.3.1 what()

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

Definition at line 286 of file [Exceptions.hpp](#).

References [message](#).

9.15.4 Member Data Documentation

9.15.4.1 message

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

Definition at line 279 of file [Exceptions.hpp](#).

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

- `src/include/Exceptions.hpp`

9.16 options Struct Reference

The struct containing all possible options.

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

9.16.1 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"

- {ReqNonFunc4}

See also

CommandLineHandler

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

- src/include/[CommandLineHandler.hpp](#)

9.17 exceptions::ParsingException Class Reference

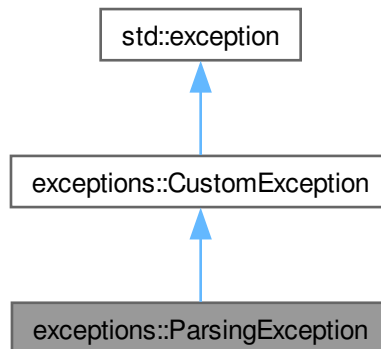
Exception for syntax errors within the json file.

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

Inheritance diagram for exceptions::ParsingException:



Collaboration diagram for exceptions::ParsingException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- const std::string [file](#)
- std::string [message](#)

9.17.1 Detailed Description

Exception for syntax errors within the json file.

Definition at line 46 of file [Exceptions.hpp](#).

9.17.2 Constructor & Destructor Documentation

9.17.2.1 ParsingException()

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

Note

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

Definition at line 52 of file [Exceptions.hpp](#).

References [file](#), and [message](#).

9.17.3 Member Function Documentation

9.17.3.1 what()

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

Definition at line 65 of file [Exceptions.hpp](#).

References [message](#).

9.17.4 Member Data Documentation

9.17.4.1 file

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

Definition at line 48 of file [Exceptions.hpp](#).

9.17.4.2 message

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

Definition at line 49 of file [Exceptions.hpp](#).

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

- [src/include/Exceptions.hpp](#)

9.18 exceptions::UnreachableCodeException Class Reference

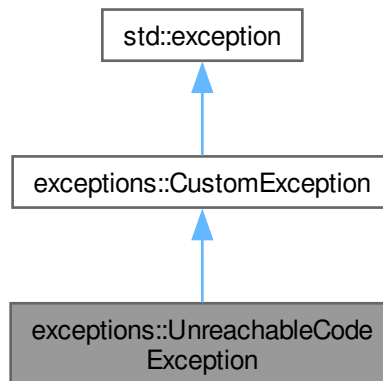
Exception for when the application reaches code it shouldn't reach.

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

Inheritance diagram for exceptions::UnreachableCodeException:



Collaboration diagram for exceptions::UnreachableCodeException:



Public Member Functions

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

Public Member Functions inherited from [exceptions::CustomException](#)

- const char * [what](#) () const noexcept override

Private Attributes

- std::string [message](#)

9.18.1 Detailed Description

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

Definition at line 238 of file [Exceptions.hpp](#).

9.18.2 Constructor & Destructor Documentation

9.18.2.1 UnreachableCodeException()

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

Definition at line 243 of file [Exceptions.hpp](#).

References [config::EXECUTABLE_NAME](#), and [message](#).

9.18.3 Member Function Documentation

9.18.3.1 what()

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

Definition at line 250 of file [Exceptions.hpp](#).

References [message](#).

9.18.4 Member Data Documentation

9.18.4.1 message

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

Definition at line 240 of file [Exceptions.hpp](#).

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

- [src/include/Exceptions.hpp](#)

9.19 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 std::exception &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.
- static std::string [escapeString](#) (const std::string &str)
Escape any unwanted escape sequences in a string.

9.19.1 Detailed Description

Responsible for utility function.

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

Definition at line 42 of file [Utils.hpp](#).

9.19.2 Member Function Documentation

9.19.2.1 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

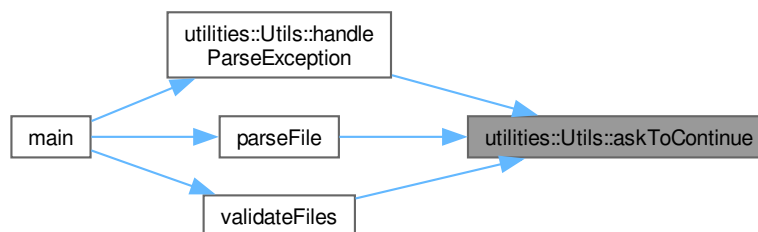
<i>prompt</i>	(Optional) A custom prompt to be used.
---------------	----------------------------------------

Returns

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

Definition at line 34 of file [Utils.cpp](#).

Here is the caller graph for this function:



9.19.2.2 checkConfigFile()

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

Checks if the easylogging-config file exists.

Parameters

<i>configFile</i>	The config file to be checked
-------------------	-------------------------------

Definition at line 55 of file [Utils.cpp](#).

Here is the caller graph for this function:

**9.19.2.3 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

<i>directory</i>	The directory to be checked
------------------	-----------------------------

Returns

The checked directory

Definition at line 65 of file [Utils.cpp](#).

Here is the caller graph for this function:



9.19.2.4 escapeString()

```
std::string utilities::Utils::escapeString (
    const std::string & str ) [static]
```

Escape any unwanted escape sequences in a string.

This function takes a string and escapes already existing escape sequences. E.g. `"\n"` would become `"\\n"`.

Parameters

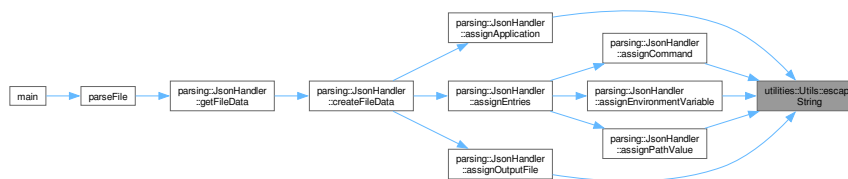
<i>str</i>	The string to be escaped
------------	--------------------------

Returns

The processed string

Definition at line 97 of file [Utils.cpp](#).

Here is the caller graph for this function:



9.19.2.5 handleParseException()

```
bool utilities::Utils::handleParseException (
    const std::exception & 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 readability in [main.cpp](#)

Parameters

<i>e</i>	The exception to be handled
<i>file</i>	The file which caused the exception
<i>files</i>	The list of files

Returns

Returns true if the user wants to continue and false otherwise

Definition at line 77 of file [Utils.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



9.19.2.6 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.

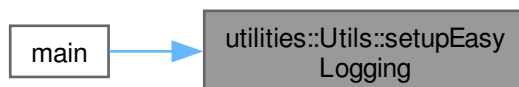
Parameters

<i>configFile</i>	The config file which is used
-------------------	-------------------------------

Definition at line 25 of file [Utils.cpp](#).

References [config::HOMEPAGE_URL](#), [config::MAJOR_VERSION](#), [config::MINOR_VERSION](#), [config::PATCH_VERSION](#), and [config::PROJECT_NAME](#).

Here is the caller graph for this function:



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

- [src/include/Utils.hpp](#)
- [src/sources/Utils.cpp](#)

Chapter 10

File Documentation

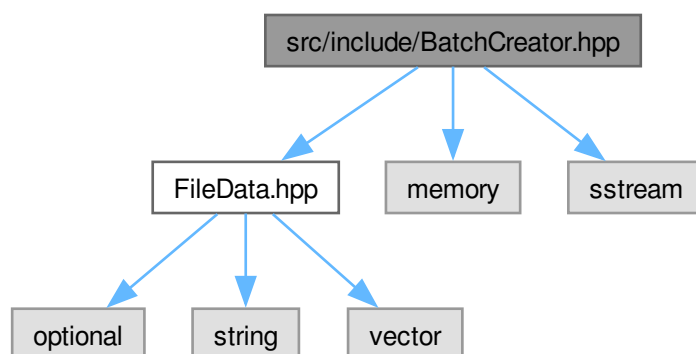
10.1 README.md File Reference

10.2 src/include/BatchCreator.hpp File Reference

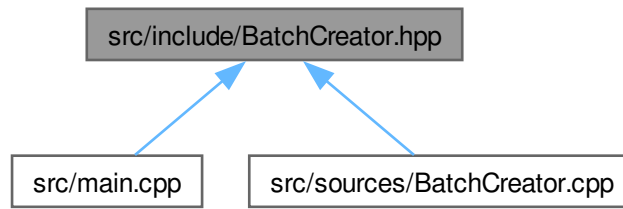
Contains the [BatchCreator](#) class.

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

Include dependency graph for BatchCreator.hpp:



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



Classes

- class [BatchCreator](#)
Creates a batch file from a `FileData` object.

10.2.1 Detailed Description

Contains the [BatchCreator](#) class.

Author

Maximilian Rodler

Date

2024-04-22

Version

0.2.1

See also

[BatchCreator](#)
[src/sources/BatchCreator.cpp](#)

Copyright

See LICENSE file

Definition in file [BatchCreator.hpp](#).

10.3 BatchCreator.hpp

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

```

00001
00016 #include "FileData.hpp"
00017 #include <memory>
00018 #include <sstream>
00019
00029 class BatchCreator {
00030 public:
00039     explicit BatchCreator(std::shared_ptr<parsing::FileData> fileData);
00040
00046     [[nodiscard]] std::shared_ptr<std::stringstream> getDataStream() const {
00047         return dataStream;
00048     }
00049
00050 private:
00051     std::shared_ptr<std::stringstream>
00052     dataStream;
00054     std::shared_ptr<parsing::FileData> fileData;
00063     void createBatch() const;
00064
00073     void writeStart() const;
00074
00082     void writeHideShell() const;
00083
00093     void writeCommands() const;
00094
00106     void writeEnvVariables() const;
00107
00116     void writePathVariables() const;
00117
00127     void writeApplication() const;
00128
00136     void writeEnd() const;
00137 };

```

10.4 src/include/CommandLineHandler.hpp File Reference

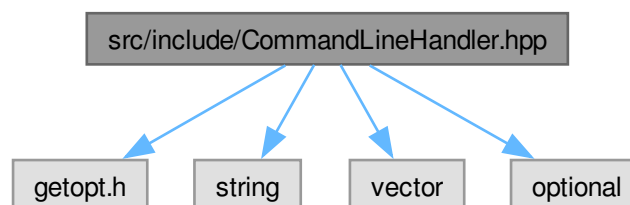
Responsible for the Command Line Interface.

```

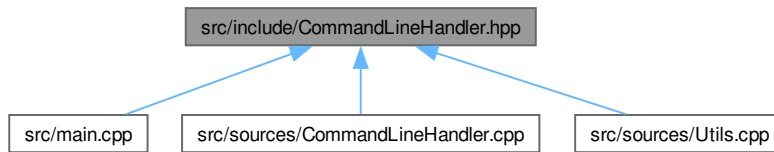
#include <getopt.h>
#include <string>
#include <vector>
#include <optional>

```

Include dependency graph for CommandLineHandler.hpp:



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



Classes

- class [cli::CommandLineHandler](#)
Responsible for the Command Line Interface.

Namespaces

- namespace [cli](#)
Includes everything regarding the CLI.

Variables

- static const struct option [cli::options](#) []

10.4.1 Detailed Description

Responsible for the Command Line Interface.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

This file is responsible for the Command Line Interface. As such it includes things such as the [CommandLineHandler](#) class, possible options and style helpers.

See also

[cli](#)
[CommandLineHandler](#)
[options](#)
[StyleHelpers](#)
[src/sources/CommandLineHandler.cpp](#)

Copyright

See LICENSE file

Definition in file [CommandLineHandler.hpp](#).

10.5 CommandLineHandler.hpp

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

```

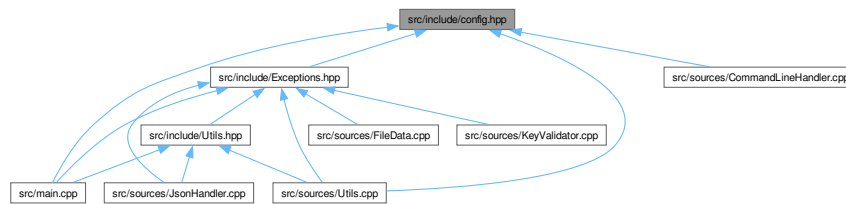
00001
00021 #ifndef COMMANDLINEHANDLER_HPP
00022 #define COMMANDLINEHANDLER_HPP
00023
00024
00025 #include <getopt.h>
00026 #include <string>
00027 #include <vector>
00028 #include <optional>
00029
00042 namespace cli {
00043
00056 class CommandLineHandler {
00057 public:
00066     [[noreturn]] static void printHelp();
00072     [[noreturn]] static void printVersion();
00080     [[noreturn]] static void printCredits();
00095     static std::tuple<std::optional<std::string>, std::vector<std::string>>
00096     parseArguments(int argc, char* argv[]);
00102     CommandLineHandler() = delete;
00108     ~CommandLineHandler() = delete;
00109 };
00110
00121 static const struct option options[] = {
00122     {"help", no_argument, nullptr, 'h'},
00123     {"version", no_argument, nullptr, 'v'},
00124     {"credits", no_argument, nullptr, 'c'},
00125     {"verbose", no_argument, nullptr, 0},
00126     {"outdir", required_argument, nullptr, 'o'},
00127     nullptr
00128 };
00129
00141 #ifdef IS_UNIX // CLI Formatting for Linux
00142 static const std::string CLEAR_TERMINAL = "\033[2J\033[1;1H";
00143 static const std::string RESET = "\033[0m";
00144 static const std::string RED = "\033[0;31m";
00145 static const std::string GREEN = "\033[0;32m";
00146 static const std::string YELLOW = "\033[0;33m";
00147 static const std::string BLUE = "\033[0;34m";
00148 static const std::string MAGENTA = "\033[0;35m";
00149 static const std::string CYAN = "\033[0;36m";
00150 static const std::string WHITE = "\033[0;37m";
00151 static const std::string BOLD = "\033[1m";
00152 static const std::string UNDERLINE = "\033[4m";
00153 static const std::string ITALIC = "\033[3m";
00154 //@note Windows doesn't support ANSI escape codes the same way
00155 #elif defined(IS_WINDOWS)
00156 static const std::string CLEAR_TERMINAL = "";
00157 static const std::string RESET = "";
00158 static const std::string RED = "";
00159 static const std::string GREEN = "";
00160 static const std::string YELLOW = "";
00161 static const std::string BLUE = "";
00162 static const std::string MAGENTA = "";
00163 static const std::string CYAN = "";
00164 static const std::string WHITE = "";
00165 static const std::string BOLD = "";
00166 static const std::string UNDERLINE = "";
00167 static const std::string ITALIC = "";
00168 #endif
00170 // end of group StyleHelpers
00171 } // namespace cli
00172
00173 #endif // COMMANDLINEHANDLER_HPP

```

10.6 src/include/config.hpp File Reference

Configures general project information.

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



Namespaces

- namespace [config](#)
Namespace used for general project information.

Variables

- constexpr auto [config::LOG_CONFIG](#) = "/home/simon/1_Coding/projectJsonToBat/build/Debug/config/easylogging.conf"
- constexpr auto [config::EXECUTABLE_NAME](#) = "json2batch"
- constexpr auto [config::MAJOR_VERSION](#) = "0"
- constexpr auto [config::MINOR_VERSION](#) = "3"
- constexpr auto [config::PATCH_VERSION](#) = "1"
- constexpr auto [config::DESCRIPTION](#) = "A simple tool to convert json to batch."
- constexpr auto [config::PROJECT_NAME](#) = "JSON2Batch"
- constexpr auto [config::AUTHORS](#) = "@AUTHORS"
- constexpr auto [config::HOMEPAGE_URL](#) = "https://dhwprojectsit23.github.io/JSON2Bat"

10.6.1 Detailed Description

Configures general project information.

Author

Simon Blum

Date

2024-04-18

Version

0.1.5

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

Note

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

Copyright

See LICENSE file

Definition in file [config.hpp](#).

10.7 config.hpp

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

```

00001
00016 // This file is autogenerated. Changes will be overwritten
00017
00018 #ifndef CONFIG_HPP
00019 #define CONFIG_HPP
00020
00025 namespace config {
00026 inline constexpr auto LOG_CONFIG =
    "/home/simon/1_Coding/projectJsonToBat/build/Debug/config/easylogging.conf";
00027 inline constexpr auto EXECUTABLE_NAME = "json2batch";
00028 inline constexpr auto MAJOR_VERSION = "0";
00029 inline constexpr auto MINOR_VERSION = "3";
00030 inline constexpr auto PATCH_VERSION = "1";
00031 inline constexpr auto DESCRIPTION = "A simple tool to convert json to batch.";
00032 inline constexpr auto PROJECT_NAME = "JSON2Batch";
00033 inline constexpr auto AUTHORS = "@AUTHORS";
00034 inline constexpr auto HOMEPAGE_URL = "https://dhwprojectsit23.github.io/JSON2Bat";
00035 } // namespace config
00036
00037 #endif

```

10.8 src/include/Exceptions.hpp File Reference

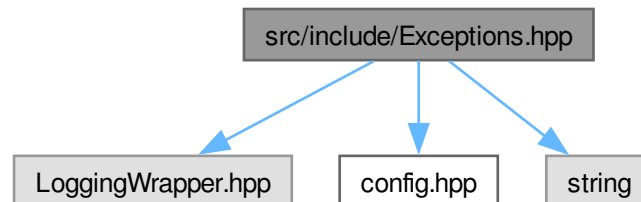
Contains all the custom exceptions used in the project.

```
#include "LoggingWrapper.hpp"
```

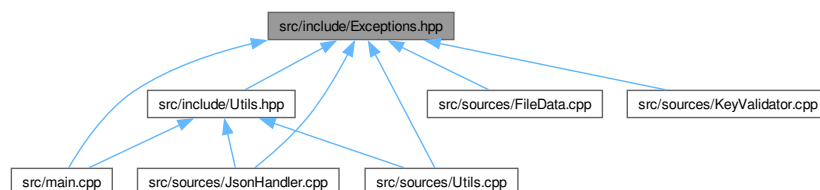
```
#include "config.hpp"
```

```
#include <string>
```

Include dependency graph for Exceptions.hpp:



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



Classes

- class [exceptions::CustomException](#)
Base class for all custom exceptions.
- class [exceptions::ParsingException](#)
Exception for syntax errors within the json file.
- class [exceptions::FileExistsException](#)
Exception for an already existing outputfile.
- class [exceptions::InvalidValueException](#)
Exception for an invalid (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.
- class [exceptions::ContainsBadCharacterException](#)
Exception for when a string contains bad characters.

Namespaces

- namespace [exceptions](#)
Namespace used for customized exceptions.

10.8.1 Detailed Description

Contains all the custom exceptions used in the project.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

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

Copyright

See LICENSE file

Definition in file [Exceptions.hpp](#).

10.9 Exceptions.hpp

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

```

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

```

```

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

```

```

00272
00277 class NoSuchDirException : public CustomException {
00278 private:
00279     std::string message;
00280
00281 public:
00282     explicit NoSuchDirException(const std::string &dir) {
00283         message = "No such directory: " + dir;
00284         LOG_INFO << "NoSuchDirException: " << message;
00285     }
00286     [[nodiscard]] const char *what() const noexcept override {
00287         return message.c_str();
00288     }
00289 };
00290
00295 class ContainsBadCharacterException : public CustomException {
00296 private:
00297     std::string message;
00298
00299 public:
00300     explicit ContainsBadCharacterException(const std::string &value) {
00301         message = "The value \"" + value + "\" contains bad characters!";
00302         LOG_INFO << "ContainsBadCharacterException: " << message;
00303     }
00304     [[nodiscard]] const char *what() const noexcept override {
00305         return message.c_str();
00306     }
00307 };
00308
00309 } // namespace exceptions
00310
00311 #endif

```

10.10 src/include/FileData.hpp File Reference

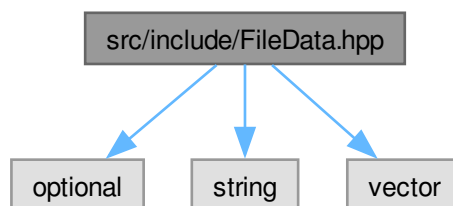
This file contains the FileData class.

```

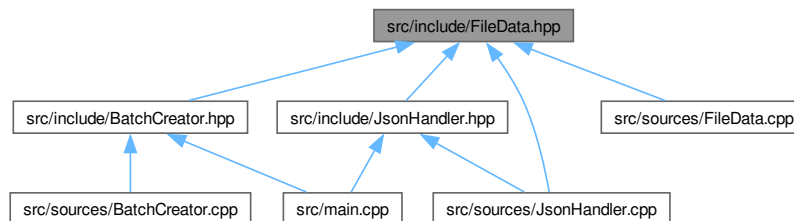
#include <optional>
#include <string>
#include <vector>

```

Include dependency graph for FileData.hpp:



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



Classes

- class [parsing::FileData](#)

This class contains all data from the json file.

Namespaces

- namespace [parsing](#)

The namespace containing everything relevant to parsing.

10.10.1 Detailed Description

This file contains the FileData class.

Author

Sonia Sinacci, Elena Schwartzbach

Date

16.04.2024

Version

0.1.5

See also

[parsing::FileData](#)

[src/sources/FileData.cpp](#)

Copyright

See LICENSE file

Definition in file [FileData.hpp](#).

10.11 FileData.hpp

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

```

00001
00015 #ifndef FILEDATA_HPP
00016 #define FILEDATA_HPP
00017
00018 #include <optional>
00019 #include <string>
00020 #include <vector>
00021
00022 namespace parsing {
00032 class FileData {
00033 public:
00044     void setOutputFile(std::string &newOutputfile);
00045
00050     void setHideShell(bool newHideShell) {
00051         this->hideShell = newHideShell;
00052     }
00053
00062     void setApplication(const std::string &newApplication);
00063
00074     void addCommand(const std::string &command);
00075
00087     void addEnvironmentVariable(const std::string &name,
00088                                 const std::string &value);
00089
00100     void addPathValue(const std::string &pathValue);
00101
00106     [[nodiscard]] const std::string &getOutputFile() const {
00107         return outputfile;
00108     }
00109
00114     [[nodiscard]] bool getHideShell() const {
00115         return hideShell;
00116     }
00117
00122     [[nodiscard]] const std::optional<std::string> &getApplication() const {
00123         return application;
00124     }
00125
00130     [[nodiscard]] const std::vector<std::string> &getCommands() const {
00131         return commands;
00132     }
00133
00138     [[nodiscard]] const std::vector<std::tuple<std::string, std::string> &
00139     getEnvironmentVariables() const {
00140         return environmentVariables;
00141     }
00142
00147     [[nodiscard]] const std::vector<std::string> &getPathValues() const {
00148         return pathValues;
00149     }
00150
00151 private:
00152     std::string outputfile;
00153     bool hideShell;
00154     std::optional<std::string> application;
00155     // {ReqFunc15}
00156     std::vector<std::string> commands;
00157     // Tuple<key, value> - {ReqFunc15}
00158     std::vector<std::tuple<std::string, std::string> > environmentVariables;
00159     // {ReqFunc15}
00160     std::vector<std::string> pathValues;
00161 };
00162 } // namespace parsing
00163
00164 #endif // FILEDATA_HPP

```

10.12 src/include/JsonHandler.hpp File Reference

This file contains the JsonHandler class.

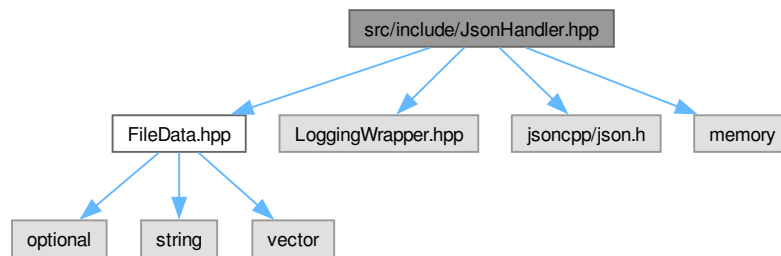
```

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

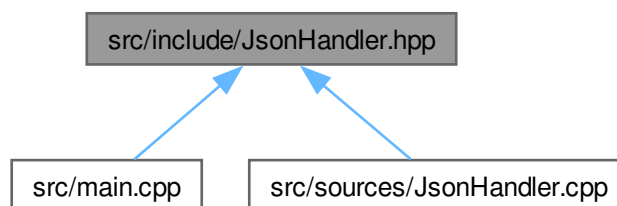
```

```
#include <memory>
```

Include dependency graph for JsonHandler.hpp:



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



Classes

- class `parsing::JsonHandler`

This file reads all data from the json file.

Namespaces

- namespace `parsing`

The namespace containing everything relevant to parsing.

10.12.1 Detailed Description

This file contains the `JsonHandler` class.

Author

Sonia Sinacci, Elena Schwartzbach

Date

23.04.2024

Version

0.1.5

See also

[parsing::JsonHandler](#)[src/sources/JsonHandler.cpp](#)

Copyright

See LICENSE file

Definition in file [JsonHandler.hpp](#).

10.13 JsonHandler.hpp

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

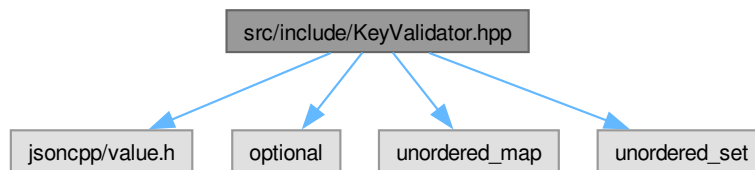
```
00001
00015 #ifndef JSONHANDLER_HPP
00016 #define JSONHANDLER_HPP
00017
00018 #include "FileData.hpp"
00019 #include "LoggingWrapper.hpp"
00020 #include <jsoncpp/json.h>
00021
00022 #include <memory>
00023
00036 namespace parsing {
00037
00047 class JsonHandler {
00048 public:
00055     JsonHandler() {
00056         LOG_INFO « "Initialising empty JsonHandler";
00057     }
00065     explicit JsonHandler(const std::string &filename);
00075     std::shared_ptr<FileData> getFileData();
00076
00077 private:
00093     [[nodiscard]] static std::shared_ptr<Json::Value>
00094     parseFile(const std::string &filename);
00104     void assignOutputFile() const;
00112     void assignHideShell() const;
00120     void assignApplication() const;
00133     void assignEntries() const;
00140     void assignCommand(const Json::Value &entry) const;
00147     void assignEnvironmentVariable(const Json::Value &entry) const;
00154     void assignPathValue(const Json::Value &entry) const;
00163     std::shared_ptr<FileData> createFileData();
00164
00177     [[nodiscard]] static bool containsBadCharacter(const std::string_view &str);
00178     std::shared_ptr<Json::Value> root;
00179     std::shared_ptr<FileData> data;
00180 };
00181 } // namespace parsing
00182
00183 #endif // JSONHANDLER_HPP
```

10.14 src/include/KeyValidator.hpp File Reference

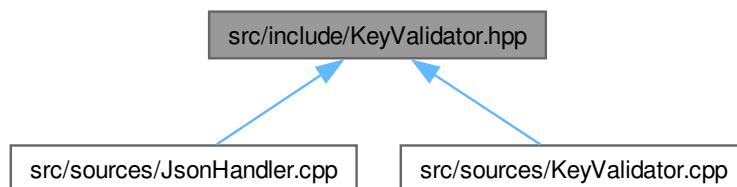
This file contains the KeyValidator class.

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

Include dependency graph for KeyValidator.hpp:



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



Classes

- class `parsing::KeyValidator`
Validates keys of a `Json::Value` object.

Namespaces

- namespace `parsing`
The namespace containing everything relevant to parsing.

10.14.1 Detailed Description

This file contains the KeyValidator class.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

See also

[parsing::KeyValidator](#)
[src/sources/KeyValidator.cpp](#)

Copyright

See LICENSE file

Definition in file [KeyValidator.hpp](#).

10.15 KeyValidator.hpp

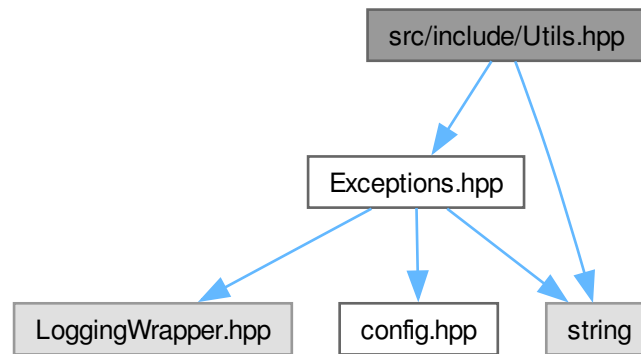
[Go to the documentation of this file.](#)

```
00001
00014 #ifndef KEYVALIDATOR_HPP
00015 #define KEYVALIDATOR_HPP
00016
00017 #include "jsoncpp/value.h"
00018 #include <optional>
00019 #include <unordered_map>
00020 #include <unordered_set>
00021 namespace parsing {
00031 class KeyValidator {
00032 public:
00038     static KeyValidator &getInstance();
00039
00054     std::vector<std::tuple<int, std::string>>
00055     validateKeys(const Json::Value &root, const std::string &filename);
00056
00057 private:
00070     std::vector<std::tuple<int, std::string>>
00071     getWrongKeys(const Json::Value &root, const std::string &filename) const;
00072
00092     void validateTypes(const std::string &filename, const Json::Value &entry,
00093                       const std::unordered_set<std::string> &entryKeys);
00094
00108     std::vector<std::tuple<int, std::string>>
00109     validateEntries(const std::string &filename,
00110                    const std::unordered_set<std::string> &entryKeys) const;
00111
00124     static std::optional<int> getUnknownKeyLine(const std::string &filename,
00125                                                  const std::string &wrongKey);
00126
00131     std::unordered_set<std::string> validKeys = {"outputfile", "hideshell",
00132         "entries", "application"};
00133 };
00138     std::unordered_set<std::string> validEntryKeys = {"type", "key", "value",
00139         "path", "command"};
00140 };
00141
00145     std::unordered_map<std::string_view, std::vector<std::string>> typeToKeys = {
00146         {"EXE", {"command"}}, {"PATH", {"path"}}, {"ENV", {"key", "value"}}
00147     };
00148 };
00149 } // namespace parsing
00150
00151 #endif
```

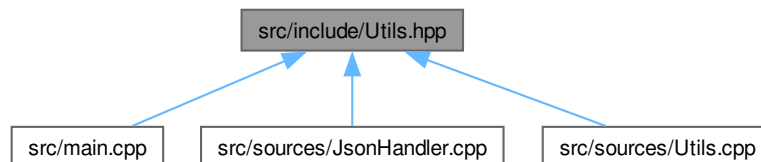
10.16 src/include/Utils.hpp File Reference

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

Include dependency graph for Utils.hpp:



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



Classes

- class `utilities::Utils`
Responsible for utility function.

Namespaces

- namespace `utilities`
Includes all utilities.

10.17 Utils.hpp

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

```

00001
00018 #ifndef UTILITIES_HPP
00019 #define UTILITIES_HPP
00020
00021 #include "Exceptions.hpp"
00022 #include <string>
00023
00033 namespace utilities {
00034
00042 class Utils {
00043 public:
00051     static void setupEasyLogging(const std::string &configFile);
00052
00066     static bool
00067     handleParseException(const std::exception &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
00110     static std::string escapeString(const std::string &str);
00111 };
00112 } // namespace utilities
00113
00114 #endif // UTILITIES_HPP

```

10.18 src/main.cpp File Reference

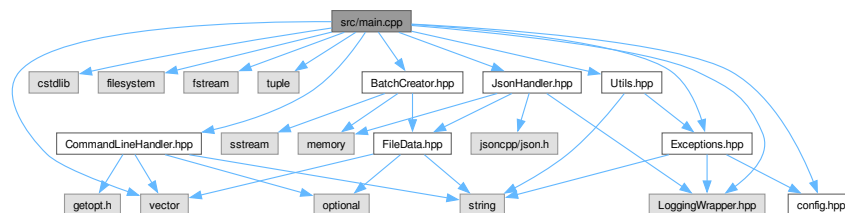
Contains the main function.

```

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

```

Include dependency graph for main.cpp:



Functions

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

10.18.1 Detailed Description

Contains the main function.

Author

Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci

Date

2024-04-26

Version

0.2.2

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

- {ReqOptFunc3} - Documentation is done using doxygen syntax
- {ReqOptFunc3} - All Classes, methods, function, namespaces and file are documented
- {ReqNonFunc5} - Source files are found under src/sources, header under src/include
- {ReqNonFunc6} - All header files can be included without paths
- {ReqNonFunc7} - Non source files are included
- {ReqNonFunc8} - All header files include a "ifndef/define/endif" block
- {ReqOptFunc5} - Every file has a top comment including the authors
- {ReqOptFunc6} - Logging is done using easylogging++ library
 - A self written wrapper is used, to allow for parallel output to the stdout and the logfile. Though we don't consider this wrapper part of the project itself and as such is placed within the directories for external libraries
- Formatting is done via astyle
- !{ReqOptFunc7} - No unit tests are included

Copyright

See LICENSE file

Definition in file [main.cpp](#).

Parameters

<i>argc</i>	Number of arguments provided
<i>argv</i>	The arguments provided

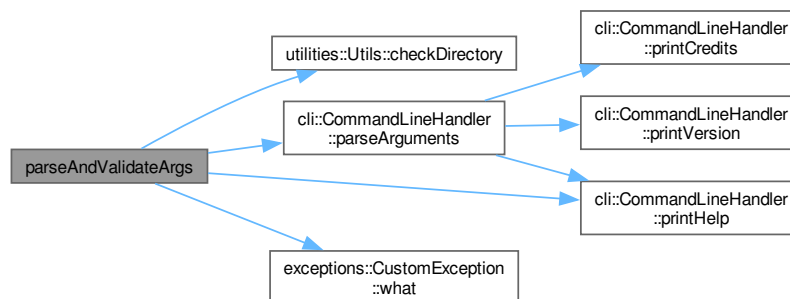
Returns

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

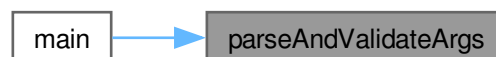
Definition at line 131 of file [main.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



10.18.2.3 parseFile()

```

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

```

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

Creates the Batch file from the given file

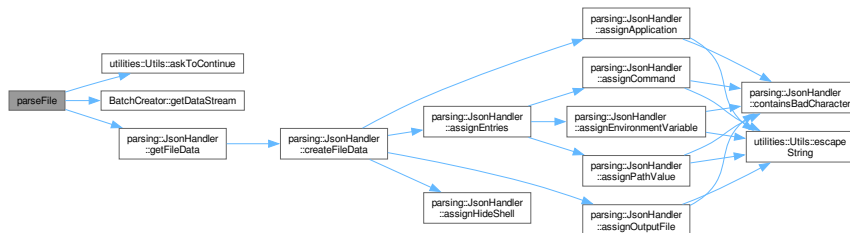
Parameters

<i>file</i>	The file to be parsed
-------------	-----------------------

Definition at line 199 of file [main.cpp](#).

References [utilities::Utils::askToContinue\(\)](#), [BatchCreator::getDataStream\(\)](#), and [parsing::JsonHandler::getFileData\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



10.18.2.4 validateFiles()

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

Checks if the files are valid.

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

Parameters

<i>files</i>	The files to be checked
	<ul style="list-style-type: none"> {ReqFunc5}

Returns

A vector containing the valid files

Definition at line 158 of file [main.cpp](#).

References [utilities::Utils::askToContinue\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



10.19 main.cpp

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

```

00001
00029 #include <LoggingWrapper.hpp>
00030 #include <cstdlib>
00031 #include <filesystem>
00032 #include <fstream>
00033 #include <tuple>
00034 #include <vector>
00035
00036 #include "BatchCreator.hpp"
00037 #include "CommandLineHandler.hpp"
00038 #include "Exceptions.hpp"
00039 #include "JsonHandler.hpp"
00040 #include "Utils.hpp"
00041 #include "config.hpp"
00042
00050 std::tuple<std::vector<std::string>, std::string>
00051 parseAndValidateArgs(int argc, char *argv[]);
00052
00062 std::vector<std::string> validateFiles(const std::vector<std::string> &files);
00063
00070 void parseFile(const std::string &file, const std::string &outputDirectory);
00071
00085 int main(int argc, char *argv[]) {
00086     // Setup logging
00087     utilities::Utils::checkConfigFile(config::LOG_CONFIG);
00088     utilities::Utils::setupEasyLogging(config::LOG_CONFIG);
00089     // Parse and validate arguments
00090     auto [files, outDir] = parseAndValidateArgs(argc, argv);
00091     OUTPUT « cli::BOLD « "Parsing the following files:\n" « cli::RESET;
  
```

```

00092
00093     for (const auto &file : files) {
00094         OUTPUT << "\t - " << file << "\n";
00095     }
00096
00097     files = validateFiles(files);
00098
00099     // Loop for {ReqFunc7}
00100     for (auto file = files.begin(); file != files.end(); ++file) {
00101         OUTPUT << cli::ITALIC << "\nParsing file: " << *file << "... \n"
00102             << cli::RESET;
00103
00104         try {
00105             parseFile(*file, outDir);
00106             // Only catch custom exceptions, other exceptions are fatal
00107         } catch (const exceptions::CustomException &e) {
00108             LOG_INFO << "Caught custom exception: " << typeid(e).name();
00109             if (utilities::Utils::handleParseException(e, file, files)) {
00110                 continue;
00111             }
00112
00113             exit(1);
00114         } catch (const Json::Exception &e) {
00115             LOG_INFO << "Caught Json exception: " << typeid(e).name();
00116             if (utilities::Utils::handleParseException(e, file, files)) {
00117                 continue;
00118             }
00119
00120             exit(1);
00121         }
00122     }
00123     OUTPUT << "Done parsing files!\n";
00124
00125     LOG_INFO << "Exiting...";
00126     return 0;
00127 }
00128
00129 std::tuple<std::vector<std::string>, std::string>
00130 parseAndValidateArgs(int argc, char *argv[]) {
00131     if (argc < 2) {
00132         LOG_ERROR << "No options given!";
00133         cli::CommandLineHandler::printHelp();
00134     }
00135
00136     auto [outOption, files] = cli::CommandLineHandler::parseArguments(argc, argv);
00137     // Set the output directory if given
00138     std::string outDir = outOption.value_or("");
00139
00140     if (!outDir.empty()) {
00141         try {
00142             outDir = utilities::Utils::checkDirectory(outDir);
00143         } catch (const exceptions::CustomException &e) {
00144             LOG_ERROR << e.what();
00145             exit(1);
00146         }
00147     }
00148
00149     if (files.empty()) {
00150         LOG_ERROR << "No files were given as arguments!";
00151         exit(1);
00152     }
00153
00154     return {files, outDir};
00155 }
00156
00157 std::vector<std::string> validateFiles(const std::vector<std::string> &files) {
00158     std::vector<std::string> validFiles;
00159     // Reserve space, to avoid reallocating with each valid file
00160     validFiles.reserve(files.size());
00161
00162     for (const std::filesystem::path file : files) {
00163         // Check that the file exists
00164         // {ReqFunc5}
00165         if (!std::filesystem::is_regular_file(file)) {
00166             LOG_ERROR << "The file \"" << file << "\" does not exist!";
00167
00168             if (files.size() > 1 && !utilities::Utils::askToContinue()) {
00169                 OUTPUT << "Aborting...\n";
00170                 LOG_INFO << "Application ended by user Input";
00171                 exit(1);
00172             }
00173
00174             continue;
00175         }
00176     }
00177
00178     // Check if the file ends in .json

```

```

00179         if (file.extension() != ".json") {
00180             LOG_WARNING « "The file \"\" « file « R "(" does not end in ".json")";
00181             OUTPUT « "If the file is not in JSON Format, continuing may "
00182                 "result in\nunexpected behaviour!\n";
00183
00184             if (!utilities::Utils::askToContinue()) {
00185                 OUTPUT « "Aborting...\n";
00186                 LOG_INFO « "Application ended by user Input";
00187                 exit(1);
00188             }
00189         }
00190
00191         validFiles.push_back(file.string());
00192     }
00193
00194     // Shrinks the vector if invalid files were found
00195     validFiles.shrink_to_fit();
00196     return validFiles;
00197 }
00198
00199 void parseFile(const std::string &file, const std::string &outputDirectory) {
00200     parsing::JsonHandler jsonHandler(file);
00201     const auto fileData = jsonHandler.getFileData();
00202     BatchCreator batchCreator(fileData);
00203     const std::shared_ptr<std::stringstream> dataStream =
00204         batchCreator.getDataStream();
00205     // Full filename is output directory + output file
00206     // {ReqFunc18}
00207     const std::string outputFileName =
00208         outputDirectory + fileData->getOutputFile();
00209
00210     if (std::filesystem::is_regular_file(outputFileName)) {
00211         if (!utilities::Utils::askToContinue(
00212             "The file already exists, do you want to overwrite it? (y/n) ")) {
00213             OUTPUT « "Skipping file...\n";
00214             return;
00215         }
00216         OUTPUT « "Overwriting file...\n";
00217     }
00218
00219     std::ofstream outFile(outputFileName);
00220
00221     if (!outFile.good()) {
00222         throw exceptions::FailedToOpenFileException(outputFileName);
00223     }
00224
00225     outFile « dataStream->str();
00226 }
00227
00228 // Initialize easylogging++
00229 // Moved to bottom because it messed with doxygen
00230 INITIALIZE_EASYLOGGINGPP

```

10.20 src/sources/BatchCreator.cpp File Reference

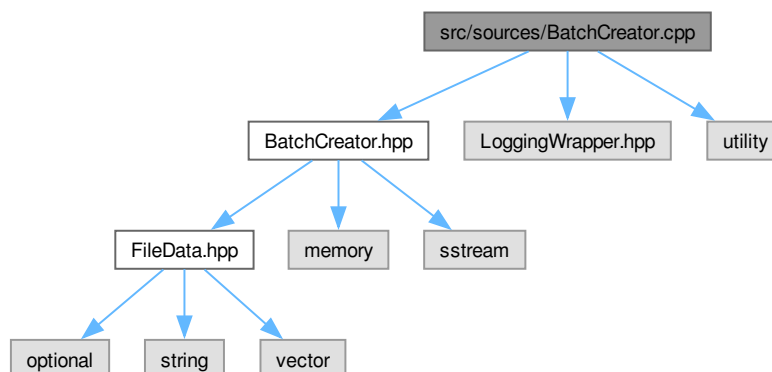
Contains the implementation of the [BatchCreator](#) class.

```

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

```

Include dependency graph for BatchCreator.cpp:



10.20.1 Detailed Description

Contains the implementation of the [BatchCreator](#) class.

Author

Maximilian Rodler

Date

22.04.2024

Version

0.2.2

See also

[src/include/BatchCreator.hpp](#)

Copyright

See LICENSE file

Definition in file [BatchCreator.cpp](#).

10.21 BatchCreator.cpp

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

```

00001
00013 #include "BatchCreator.hpp"
00014 #include "LoggingWrapper.hpp"
00015 #include <utility>
00016
00017
00018 BatchCreator::BatchCreator(std::shared_ptr<parsing::FileData> fileData)
00019     : fileData(std::move(fileData)) {
00020     LOG_INFO << "Initializing BatchCreator";
00021     this->dataStream = std::make_shared<std::stringstream>();
00022     this->createBatch();
00023 }
00024
00025 void BatchCreator::createBatch() const {
00026     LOG_INFO << "Creating Batch file";
00027     this->writeStart();
00028     this->writeHideShell();
00029     this->writeCommands();
00030     this->writeEnvVariables();
00031     this->writePathVariables();
00032     this->writeApplication();
00033     this->writeEnd();
00034 }
00035
00036 void BatchCreator::writeStart() const {
00037     LOG_INFO << "writing Start of Batch";
00038     // {ReqFunc24} - \r\n
00039     *this->dataStream << "@ECHO OFF\r\nC:\\Windows\\System32\\cmd.exe ";
00040 }
00041
00042 void BatchCreator::writeHideShell() const {
00043     if (this->fileData->getHideShell()) {
00044         LOG_INFO << "writing hide Shell";
00045         *this->dataStream << "/c ";
00046     }
00047     else {
00048         LOG_INFO << "writing show Shell";
00049         *this->dataStream << "/k ";
00050     }
00051 }
00052
00053 void BatchCreator::writeCommands() const {
00054     LOG_INFO << "writing Commands";
00055     *this->dataStream << "\n";
00056
00057     for (const std::string &command : this->fileData->getCommands()) {
00058         *this->dataStream << command << " && ";
00059     }
00060 }
00061
00062 void BatchCreator::writeEnvVariables() const {
00063     LOG_INFO << "writing Environment Variables";
00064
00065     for (const auto &[key, value] : this->fileData->getEnvironmentVariables()) {
00066         *this->dataStream << "set " << key << "=" << value << " && ";
00067     }
00068 }
00069
00070 void BatchCreator::writePathVariables() const {
00071     LOG_INFO << "writing Path Variables";
00072     *this->dataStream << "set path=";
00073
00074     for (const std::string &path : this->fileData->getPathValues()) {
00075         *this->dataStream << path << ";";
00076     }
00077
00078     *this->dataStream << "%path%";
00079 }
00080
00081 void BatchCreator::writeApplication() const {
00082     std::string appName = this->fileData->getOutputFile();
00083     appName = appName.substr(0, appName.find('.'));
00084
00085     if (this->fileData->getApplication().has_value()) {
00086         LOG_INFO << "writing start Application";
00087         *this->dataStream << " && start \"" << appName
00088             << "\" "
00089             // {ReqFunc24} - \r\n
00090             << this->fileData->getApplication().value() << "\"\r\n";
00091     }
00092     else {
00093         LOG_INFO << "writing not start Application";
00094     }
00095 }

```



```

00094         // {ReqFunc24} - \r\n
00095         *this->dataStream << "\\r\n";
00096     }
00097 }
00098
00099 void BatchCreator::writeEnd() const {
00100     *this->dataStream << "@ECHO ON";
00101 }

```

10.22 src/sources/CommandLineHandler.cpp File Reference

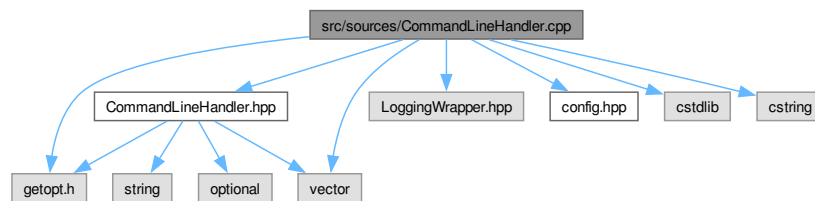
Implementation for the Command Line Interface.

```

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

```

Include dependency graph for CommandLineHandler.cpp:



Namespaces

- namespace `cli`
Includes everything regarding the CLI.

10.22.1 Detailed Description

Implementation for the Command Line Interface.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2


```

00075     int optIndex = -1;
00076     struct option longOption = {};
00077     const auto result = getopt_long(argc, argv, "hvco:", options, &optIndex);
00078
00079     if (result == -1) {
00080         LOG_INFO « "End of options reached";
00081         break;
00082     }
00083
00084     switch (result) {
00085     case '?':
00086         LOG_ERROR « "Invalid Option (argument)";
00087         CommandLineHandler::printHelp();
00088
00089     case 'h':
00090         LOG_INFO « "Help option detected";
00091         CommandLineHandler::printHelp();
00092
00093     case 'v':
00094         LOG_INFO « "Version option detected";
00095         CommandLineHandler::printVersion();
00096
00097     case 'c':
00098         LOG_INFO « "Credit option detected";
00099         CommandLineHandler::printCredits();
00100
00101     case 'o':
00102         LOG_INFO « "Output option detected";
00103         outDir = optarg;
00104         break;
00105
00106     case 0:
00107         LOG_INFO « "Long option without short version detected";
00108         longOption = options[optIndex];
00109         LOG_INFO « "Option: " « longOption.name « " given";
00110
00111         if (strcmp(longOption.name, "verbose") == 0) {
00112             logging::setVerboseMode(true);
00113             LOG_INFO « "Verbose mode activated";
00114         }
00115
00116         break;
00117
00118     default:
00119         LOG_ERROR « "Default case for options reached!";
00120         break;
00121     }
00122 }
00123
00124 LOG_INFO « "Options have been parsed";
00125 LOG_INFO « "Checking for arguments...";
00126
00127 // Loop for {reqFunc5}
00128 while (optind < argc) {
00129     LOG_INFO « "Adding file: " « argv[optind];
00130     // Vector for {reqFunc7}
00131     files.emplace_back(argv[optind++]);
00132 }
00133
00134 LOG_INFO « "Arguments and options have been parsed";
00135 return {outDir, files};
00136 }
00137 } // namespace cli

```

10.24 src/sources/FileData.cpp File Reference

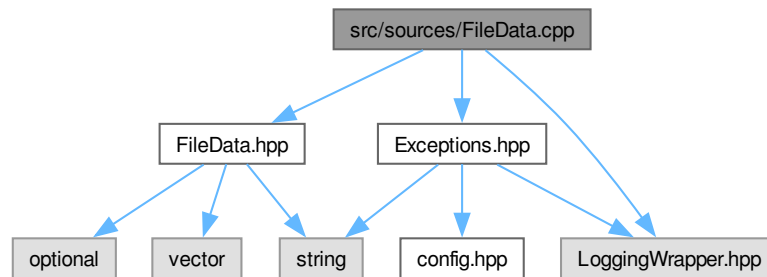
Implementation of the FileData class.

```

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

```

Include dependency graph for FileData.cpp:



Namespaces

- namespace [parsing](#)

The namespace containing everything relevant to parsing.

10.24.1 Detailed Description

Implementation of the FileData class.

Author

Elena Schwarzbach, Sonia Sinacci

Date

2024-04-26

Version

0.1.6

See also

[src/include/FileData.hpp](#)

Copyright

See LICENSE file

Definition in file [FileData.cpp](#).

10.25 FileData.cpp

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

```

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

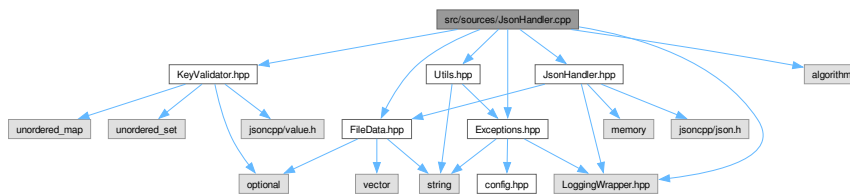
```

10.26 src/sources/JsonHandler.cpp File Reference

Implementation of the JsonHandler class.

```
#include "JsonHandler.hpp"
#include "Exceptions.hpp"
#include "FileData.hpp"
#include "KeyValidator.hpp"
#include "LoggingWrapper.hpp"
#include "Utils.hpp"
#include <algorithm>
```

Include dependency graph for JsonHandler.cpp:



Namespaces

- namespace [parsing](#)
The namespace containing everything relevant to parsing.

10.26.1 Detailed Description

Implementation of the JsonHandler class.

Author

Elena Schwarzbach, Sonia Sinacci

Date

2024-04-16

Version

0.1.6

See also

[src/include/JsonHandler.hpp](#)

Copyright

See LICENSE file

Definition in file [JsonHandler.cpp](#).

10.27 JsonHandler.cpp

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

```

00001
00013 #include "JsonHandler.hpp"
00014 #include "Exceptions.hpp"
00015 #include "FileData.hpp"
00016 #include "KeyValidator.hpp"
00017 #include "LoggingWrapper.hpp"
00018 #include "Utils.hpp"
00019
00020 #include <algorithm>
00021
00022 namespace parsing {
00023 JsonHandler::JsonHandler(const std::string &filename) {
00024     LOG_INFO << "Initializing JSONHandler with filename: " << filename << "\n";
00025     this->root = parseFile(filename);
00026 }
00027
00028 std::shared_ptr<Json::Value> JsonHandler::parseFile(const std::string &filename)
00029 {
00030     LOG_INFO << "Parsing file: " << filename << "\n";
00031     // Can open files anywhere with relative/absolute path
00032     // - {ReqFunc5}
00033     std::ifstream file(filename);
00034     Json::Value newRoot;
00035
00036     // Json::Reader.parse() returns false if parsing fails
00037     if (Json::Reader reader; !reader.parse(file, newRoot)) {
00038         throw exceptions::ParsingException(filename);
00039     }
00040
00041     // Validate keys
00042     // Check for errors
00043     if (auto errors = KeyValidator::getInstance().validateKeys(newRoot, filename);
00044         !errors.empty()) {
00045         throw exceptions::InvalidKeyException(errors);
00046     }
00047
00048     LOG_INFO << "File \"" << filename << "\" has been parsed\n";
00049     return std::make_shared<Json::Value>(newRoot);
00050 }
00051
00052 std::shared_ptr<FileData> JsonHandler::getFileData() {
00053     LOG_INFO << "Creating FileData object for return...\n";
00054     return this->createFileData();
00055 }
00056
00057 std::shared_ptr<FileData> JsonHandler::createFileData() {
00058     LOG_INFO << "Creating FileData object...\n";
00059     this->data = std::make_shared<FileData>();
00060     this->assignOutputFile();
00061     this->assignHideShell();
00062     this->assignApplication();
00063     this->assignEntries();
00064     return this->data;
00065 }
00066
00067 void JsonHandler::assignOutputFile() const {
00068     LOG_INFO << "Assigning outputfile...\n";
00069     std::string outputFile = this->root->get("outputfile", "").asString();
00070     if (containsBadCharacter(outputFile)) {
00071         outputFile = utilities::Utils::escapeString(outputFile);
00072         throw exceptions::ContainsBadCharacterException(outputFile);
00073     }
00074     this->data->setOutputFile(outputFile);
00075 }
00076
00077 void JsonHandler::assignHideShell() const {
00078     LOG_INFO << "Assigning hide shell...\n";
00079     // If the 'hideshell' key is not given, it defaults to false
00080     this->data->setHideShell(this->root->get("hideshell", false).asBool());
00081 }
00082
00083 void JsonHandler::assignApplication() const {
00084     LOG_INFO << "Assigning application...\n";
00085     std::string application = this->root->get("application", "").asString();
00086     if (containsBadCharacter(application)) {
00087         application = utilities::Utils::escapeString(application);
00088         throw exceptions::ContainsBadCharacterException(application);
00089     }
00090     this->data->setApplication(application);
00091 }
00092
00093

```

```

00094 void JsonHandler::assignEntries() const {
00095     LOG_INFO << "Assigning entries...\n";
00096
00097     for (const auto &entry : this->root->get("entries", "")) {
00098         std::string entryType = entry.get("type", "").asString();
00099
00100         if (entryType == "EXE") {
00101             LOG_INFO << "Calling function to assign command...\n";
00102             this->assignCommand(entry);
00103         } else if (entryType == "ENV") {
00104             LOG_INFO << "Calling function to assign environment variable...\n";
00105             this->assignEnvironmentVariable(entry);
00106         } else if (entryType == "PATH") {
00107             LOG_INFO << "Calling function to assign path value...\n";
00108             this->assignPathValue(entry);
00109         } else {
00110             // Due to validation beforehand - this should never be reached!
00111             throw exceptions::UnreachableCodeException(
00112                 "Unknown entries should be caught by KeyValidator!\nPlease report "
00113                 "this bug!");
00114         }
00115     }
00116 }
00117
00118 void JsonHandler::assignCommand(const Json::Value &entry) const {
00119     LOG_INFO << "Assigning command...\n";
00120     std::string command = entry.get("command", "").asString();
00121     if (containsBadCharacter(command)) {
00122         command = utilities::Utils::escapeString(command);
00123         throw exceptions::ContainsBadCharacterException(command);
00124     }
00125     this->data->addCommand(command);
00126 }
00127
00128 void JsonHandler::assignEnvironmentVariable(const Json::Value &entry) const {
00129     LOG_INFO << "Assigning environment variable...\n";
00130     std::string key = entry.get("key", "").asString();
00131     std::string value = entry.get("value", "").asString();
00132
00133     if (containsBadCharacter(key)) {
00134         key = utilities::Utils::escapeString(key);
00135         throw exceptions::ContainsBadCharacterException(key);
00136     }
00137     if (containsBadCharacter(value)) {
00138         value = utilities::Utils::escapeString(value);
00139         throw exceptions::ContainsBadCharacterException(value);
00140     }
00141     this->data->addEnvironmentVariable(key, value);
00142 }
00143
00144 void JsonHandler::assignPathValue(const Json::Value &entry) const {
00145     LOG_INFO << "Assigning path value...\n";
00146     std::string path = entry.get("path", "").asString();
00147     if (containsBadCharacter(path)) {
00148         path = utilities::Utils::escapeString(path);
00149         throw exceptions::ContainsBadCharacterException(path);
00150     }
00151     this->data->addPathValue(path);
00152 }
00153
00154 bool JsonHandler::containsBadCharacter(const std::string_view &str) {
00155     // Set of characters which may not be in the string
00156     static const std::unordered_set<char> badChars = {
00157         '\n', '\t', '\r', '\0', '\x1A', '|', ';', '<', '>', '!', '%', '"', '\\'
00158     };
00159
00160     // Lambda function which returns true, if the char is bad
00161     auto isBadCharacter = [](char c) {
00162         return badChars.contains(c);
00163     };
00164
00165     return std::ranges::any_of(str, isBadCharacter);
00166 }
00167 }
00168 } // namespace parsing

```

10.28 src/sources/KeyValidator.cpp File Reference

Implementation for the KeyValidator class.

```

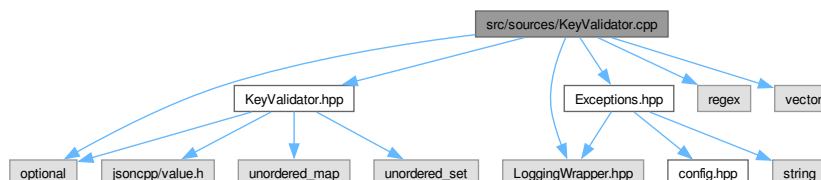
#include "KeyValidator.hpp"
#include "Exceptions.hpp"

```



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

Include dependency graph for KeyValidator.cpp:



Namespaces

- namespace [parsing](#)

The namespace containing everything relevant to parsing.

10.28.1 Detailed Description

Implementation for the `KeyValidator` class.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

See also

[src/include/KeyValidator.hpp](#)

Copyright

See LICENSE file

Definition in file [KeyValidator.cpp](#).

10.29 KeyValidator.cpp

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

```

00001
00012 #include "KeyValidator.hpp"
00013 #include "Exceptions.hpp"
00014 #include "LoggingWrapper.hpp"
00015 #include <optional>
00016 #include <regex>
00017 #include <vector>
00018
00019 namespace parsing {
00020 KeyValidator &KeyValidator::getInstance() {
00021     static KeyValidator keyValidator;
00022     LOG_INFO << "Returning KeyValidator instance!";
00023     return keyValidator;
00024 }
00025
00026 std::vector<std::tuple<int, std::string>
00027 KeyValidator::validateKeys(const Json::Value &root,
00028                             const std::string &filename) {
00029     LOG_INFO << "Validating keys for file " << filename;
00030     std::vector<std::tuple<int, std::string> wrongKeys =
00031         getWrongKeys(root, filename);
00032
00033     // Inline declaration to prevent leaking in outer scope
00034     for (Json::Value entries = root.get("entries", "");
00035          const auto &entry : entries) {
00036         LOG_INFO << "Validating entry";
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
00042         const auto wrongEntries = validateEntries(filename, entryKeysSet);
00043
00044         // Combine wrong keys
00045         wrongKeys.insert(wrongKeys.end(), wrongEntries.begin(), wrongEntries.end());
00046
00047         LOG_INFO << "Validating types for entry";
00048         validateTypes(filename, entry, entryKeysSet);
00049     }
00050
00051     return wrongKeys;
00052 }
00053
00054 std::vector<std::tuple<int, std::string>
00055 KeyValidator::getWrongKeys(const Json::Value &root,
00056                             const std::string &filename) const {
00057     std::vector<std::tuple<int, std::string> wrongKeys = {};
00058
00059     LOG_INFO << "Checcking for wrong keys in file " << filename << "!";
00060     for (const auto &key : root.getMemberNames()) {
00061         if (!validKeys.contains(key)) {
00062             LOG_WARNING << "Found wrong key " << key << "!";
00063             const auto error = getUnknownKeyLine(filename, key);
00064
00065             if (!error.has_value()) {
00066                 LOG_ERROR << "Unable to find line of wrong key!";
00067                 continue;
00068             }
00069
00070             // If the line can't be found, add -1 as line number
00071             wrongKeys.emplace_back(error.value_or(-1), key);
00072         }
00073     }
00074
00075     return wrongKeys;
00076 }
00077
00078 std::vector<std::tuple<int, std::string> KeyValidator::validateEntries(
00079     const std::string &filename,
00080     const std::unordered_set<std::string> &entryKeys) const {
00081     std::vector<std::tuple<int, std::string> wrongKeys = {};
00082
00083     for (const auto &key : entryKeys) {
00084         LOG_INFO << "Checking key " << key << "!";
00085         if (!validEntryKeys.contains(key)) {
00086             const auto error = getUnknownKeyLine(filename, key);
00087
00088             if (!error.has_value()) {
00089                 LOG_ERROR << "Unable to find line of wrong key!";
00090                 continue;
00091             }
00092         }

```

```

00093         wrongKeys.emplace_back(error.value_or(-1), key);
00094     }
00095 }
00096
00097 return wrongKeys;
00098 }
00099
00100 void KeyValidator::validateTypes(
00101     const std::string &filename, const Json::Value &entry,
00102     const std::unordered_set<std::string> &entryKeys) {
00103     // Get the type of the entry - error if not found
00104     const std::string type = entry.get("type", "ERROR").asString();
00105     LOG_INFO << "Validating type " << type;
00106
00107     // If the type is not found, throw an exception
00108     if (type == "ERROR") {
00109         throw exceptions::MissingTypeException();
00110         // If the type is not known, throw an exception
00111         // @note This should already have been checked
00112     } else if (!typeToKeys.contains(type)) {
00113         const std::optional<int> line =
00114             getUnknownKeyLine(filename, std::string(type));
00115
00116         if (!line.has_value()) {
00117             LOG_INFO << "Unable to find line of wrong type!";
00118         }
00119
00120         throw exceptions::InvalidTypeException(std::string(type), line.value());
00121         // If the type is known, check if all necessary keys are present
00122     } else {
00123         for (const auto &key : typeToKeys[type]) {
00124             LOG_INFO << "Checking key " << key << " for type " << type;
00125             if (!entryKeys.contains(key)) {
00126                 throw exceptions::MissingKeyException(key, type);
00127             }
00128         }
00129     }
00130 }
00131
00132 std::optional<int>
00133 KeyValidator::getUnknownKeyLine(const std::string &filename,
00134                                 const std::string &wrongKey) {
00135     std::ifstream file(filename);
00136     LOG_INFO << "Checking for key " << wrongKey << " in file " << filename;
00137
00138     if (!file.is_open()) {
00139         LOG_ERROR << "File not open!";
00140         return std::nullopt;
00141     }
00142
00143     std::string line;
00144     // Create a regex pattern that matches the wrong key whole word
00145     const std::regex wrongKeyPattern("\\b" + wrongKey + "\\b");
00146
00147     for (int lineNumber = 1; std::getline(file, line); ++lineNumber) {
00148         if (std::regex_search(line, wrongKeyPattern)) {
00149             LOG_INFO << "Found key " << wrongKey << " in line " << lineNumber;
00150
00151             return lineNumber;
00152         }
00153     }
00154
00155     return std::nullopt;
00156 }
00157
00158 } // namespace parsing

```

10.30 src/sources/Utils.cpp File Reference

Implementation for the Utils class.

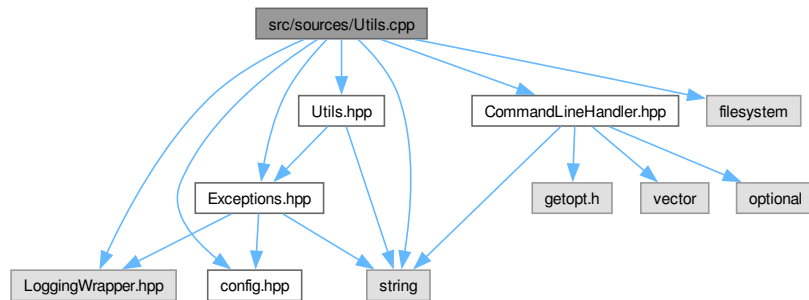
```

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

```

```
#include <string>
```

Include dependency graph for Utils.cpp:



Namespaces

- namespace [utilities](#)
Includes all utilities.

10.30.1 Detailed Description

Implementation for the Utils class.

Author

Simon Blum

Date

2024-04-26

Version

0.2.2

This file includes the implementation for the Utils class.

See also

`src/include/utility/Utilities.hpp`

Copyright

See LICENSE file

Definition in file [Utils.cpp](#).

10.31 Utils.cpp

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

```

00001
00015 #include "Utils.hpp"
00016 #include "CommandLineHandler.hpp"
00017 #include "Exceptions.hpp"
00018 #include "config.hpp"
00019
00020 #include <LoggingWrapper.hpp>
00021 #include <filesystem>
00022 #include <string>
00023
00024 namespace utilities {
00025 void Utils::setupEasyLogging(const std::string &configFile) {
00026     el::Configurations conf(configFile);
00027     el::Loggers::reconfigureAllLoggers(conf);
00028     LOG_INFO << "Running " << config::PROJECT_NAME << " v"
00029             << config::MAJOR_VERSION << "." << config::MINOR_VERSION << "."
00030             << config::PATCH_VERSION;
00031     LOG_INFO << "For more Information checkout " << config::HOMEPAGE_URL;
00032     LOG_INFO << "EasyLogging has been setup!";
00033 }
00034 bool Utils::askToContinue(const std::string &prompt) {
00035     std::string userInput;
00036     LOG_INFO << "Asking for user Confirmation to continue...";
00037     OUTPUT << cli::BOLD << prompt << cli::RESET;
00038
00039     do {
00040         std::cin >> userInput;
00041         std::ranges::transform(userInput, userInput.begin(), ::tolower);
00042
00043         if (userInput != "y" && userInput != "yes" && userInput != "n" &&
00044             userInput != "no") {
00045             LOG_INFO << "Wrong user input!";
00046             OUTPUT << cli::ITALIC << "Please enter Y/Yes or N/No!\n" << cli::RESET;
00047             continue;
00048         }
00049
00050         break;
00051     } while (true);
00052
00053     return userInput == "y" || userInput == "yes";
00054 }
00055 void Utils::checkConfigFile(const std::string &configFile) {
00056     if (!std::filesystem::is_regular_file(configFile)) {
00057         std::cerr << cli::RED << cli::BOLD
00058             << "Fatal: Easylogging configuration file not found at:\n"
00059             << cli::RESET << cli::ITALIC << "\n\t\"" << configFile << "\"\n\n"
00060             << cli::RESET;
00061         std::cout << "Aborting...\n";
00062         exit(1);
00063     }
00064 }
00065 const std::string &Utils::checkDirectory(std::string &directory) {
00066     if (!directory.empty() && directory.back() != '/' &&
00067         directory.back() != '\\') {
00068         directory += '/';
00069     }
00070
00071     if (!std::filesystem::exists(directory)) {
00072         throw exceptions::NoSuchDirException(directory);
00073     }
00074
00075     return directory;
00076 }
00077 bool Utils::handleParseException(const std::exception &e,
00078     const std::vector<std::string>::iterator &file,
00079     const std::vector<std::string> &files) {
00080     OUTPUT << "\nThere has been a error while trying to parse \"" << *file
00081         << ":\n";
00082     LOG_ERROR << e.what();
00083
00084     if (std::next(file) != files.end() &&
00085         !utilities::Utils::askToContinue(
00086             "Do you want to continue with the other files? (y/n) "
00087             "")) {
00088         OUTPUT << "Aborting...";
00089         LOG_INFO << "Application ended by user Input";
00090         return false;
00091     }
00092
00093     std::cout << std::endl;
00094     return true;
00095 }

```

```
00096
00097 std::string Utils::escapeString(const std::string &str) {
00098     // Map of characters to their escape sequences
00099     static const std::unordered_map<char, std::string> escapeSequences = {
00100         {'\\', "\\\\"}, // Replace backslash with double backslash
00101         {'\n', "\\n"}, // Replace newline with backslash-n
00102         {'\t', "\\t"}, // Replace tab with backslash-t
00103         {'\x1A', "\\x1A"}, // Replace end of file with backslash-x1A
00104         {'\r', "\\r"} // Replace carriage return with backslash-r
00105     };
00106
00107     std::ostringstream escapedStream;
00108     for (char c : str) {
00109         // Replace a character with it's counterpart, if it is in the map
00110         if (escapeSequences.contains(c)) {
00111             escapedStream << escapeSequences.at(c);
00112         } else {
00113             escapedStream << c;
00114         }
00115     }
00116     return escapedStream.str();
00117 }
00118
00119 } // namespace utilities
```

Index

- ~CommandLineHandler
 - cli::CommandLineHandler, [32](#)
- addCommand
 - parsing::FileData, [42](#)
- addEnvironmentVariable
 - parsing::FileData, [42](#)
- addPathValue
 - parsing::FileData, [42](#)
- application
 - parsing::FileData, [45](#)
- askToContinue
 - utilities::Utils, [86](#)
- assignApplication
 - parsing::JsonHandler, [57](#)
- assignCommand
 - parsing::JsonHandler, [58](#)
- assignEntries
 - parsing::JsonHandler, [59](#)
- assignEnvironmentVariable
 - parsing::JsonHandler, [60](#)
- assignHideShell
 - parsing::JsonHandler, [61](#)
- assignOutputFile
 - parsing::JsonHandler, [61](#)
- assignPathValue
 - parsing::JsonHandler, [62](#)
- AUTHORS
 - config, [18](#)
- BatchCreator, [23](#)
 - BatchCreator, [24](#)
 - createBatch, [25](#)
 - dataStream, [30](#)
 - fileData, [30](#)
 - getDataStream, [26](#)
 - writeApplication, [27](#)
 - writeCommands, [27](#)
 - writeEnd, [27](#)
 - writeEnvVariables, [28](#)
 - writeHideShell, [28](#)
 - writePathVariables, [29](#)
 - writeStart, [29](#)
- checkConfigFile
 - utilities::Utils, [86](#)
- checkDirectory
 - utilities::Utils, [87](#)
- cli, [17](#)
 - options, [18](#)
- cli::CommandLineHandler, [30](#)
 - ~CommandLineHandler, [32](#)
 - CommandLineHandler, [32](#)
 - parseArguments, [32](#)
 - printCredits, [33](#)
 - printHelp, [34](#)
 - printVersion, [34](#)
- CommandLineHandler
 - cli::CommandLineHandler, [32](#)
- commands
 - parsing::FileData, [46](#)
- config, [18](#)
 - AUTHORS, [18](#)
 - DESCRIPTION, [18](#)
 - EXECUTABLE_NAME, [19](#)
 - HOMEPAGE_URL, [19](#)
 - LOG_CONFIG, [19](#)
 - MAJOR_VERSION, [19](#)
 - MINOR_VERSION, [19](#)
 - PATCH_VERSION, [19](#)
 - PROJECT_NAME, [19](#)
- containsBadCharacter
 - parsing::JsonHandler, [63](#)
- ContainsBadCharacterException
 - exceptions::ContainsBadCharacterException, [36](#)
- createBatch
 - BatchCreator, [25](#)
- createFileData
 - parsing::JsonHandler, [63](#)
- data
 - parsing::JsonHandler, [66](#)
- dataStream
 - BatchCreator, [30](#)
- DESCRIPTION
 - config, [18](#)
- environmentVariables
 - parsing::FileData, [46](#)
- escapeString
 - utilities::Utils, [87](#)
- exceptions, [20](#)
- exceptions::ContainsBadCharacterException, [35](#)
 - ContainsBadCharacterException, [36](#)
 - message, [37](#)
 - what, [37](#)
- exceptions::CustomException, [37](#)
 - what, [38](#)
- exceptions::FailedToOpenFileException, [39](#)
 - FailedToOpenFileException, [40](#)

- message, 40
 - what, 40
- exceptions::FileExistsException, 47
 - file, 48
 - FileExistsException, 48
 - message, 48
 - what, 48
- exceptions::InvalidKeyException, 49
 - InvalidKeyException, 50
 - message, 51
 - what, 50
- exceptions::InvalidTypeException, 51
 - InvalidTypeException, 52
 - message, 53
 - type, 53
 - what, 52
- exceptions::InvalidValueException, 53
 - InvalidValueException, 54
 - key, 55
 - message, 55
 - what, 55
- exceptions::MissingKeyException, 74
 - key, 76
 - message, 76
 - MissingKeyException, 76
 - type, 76
 - what, 76
- exceptions::MissingTypeException, 77
 - message, 78
 - MissingTypeException, 78
 - what, 78
- exceptions::NoSuchDirException, 79
 - message, 80
 - NoSuchDirException, 80
 - what, 80
- exceptions::ParsingException, 81
 - file, 83
 - message, 83
 - ParsingException, 82
 - what, 83
- exceptions::UnreachableCodeException, 83
 - message, 85
 - UnreachableCodeException, 84
 - what, 85
- EXECUTABLE_NAME
 - config, 19
- FailedToOpenFileException
 - exceptions::FailedToOpenFileException, 40
- file
 - exceptions::FileExistsException, 48
 - exceptions::ParsingException, 83
- fileData
 - BatchCreator, 30
- FileExistsException
 - exceptions::FileExistsException, 48
- getApplication
 - parsing::FileData, 43
- getCommands
 - parsing::FileData, 43
- getDataStream
 - BatchCreator, 26
- getEnvironmentVariables
 - parsing::FileData, 43
- getFileData
 - parsing::JsonHandler, 64
- getHideShell
 - parsing::FileData, 44
- getInstance
 - parsing::KeyValidator, 68
- getOutputFile
 - parsing::FileData, 44
- getPathValues
 - parsing::FileData, 44
- getUnknownKeyLine
 - parsing::KeyValidator, 68
- getWrongKeys
 - parsing::KeyValidator, 69
- handleParseException
 - utilities::Utils, 88
- hideShell
 - parsing::FileData, 46
- Homepage_URL
 - config, 19
- InvalidKeyException
 - exceptions::InvalidKeyException, 50
- InvalidTypeException
 - exceptions::InvalidTypeException, 52
- InvalidValueException
 - exceptions::InvalidValueException, 54
- JSON2Batch, 1
- JsonHandler
 - parsing::JsonHandler, 57
- key
 - exceptions::InvalidValueException, 55
 - exceptions::MissingKeyException, 76
- LOG_CONFIG
 - config, 19
- main
 - main.cpp, 111
- main.cpp
 - main, 111
 - parseAndValidateArgs, 111
 - parseFile, 112
 - validateFiles, 113
- MAJOR_VERSION
 - config, 19
- message
 - exceptions::ContainsBadCharacterException, 37
 - exceptions::FailedToOpenFileException, 40
 - exceptions::FileExistsException, 48
 - exceptions::InvalidKeyException, 51

- exceptions::InvalidTypeException, 53
- exceptions::InvalidValueException, 55
- exceptions::MissingKeyException, 76
- exceptions::MissingTypeException, 78
- exceptions::NoSuchDirException, 80
- exceptions::ParsingException, 83
- exceptions::UnreachableCodeException, 85
- MINOR_VERSION
 - config, 19
- MissingKeyException
 - exceptions::MissingKeyException, 76
- MissingTypeException
 - exceptions::MissingTypeException, 78
- NoSuchDirException
 - exceptions::NoSuchDirException, 80
- options, 18
 - cli, 18
- outputfile
 - parsing::FileData, 46
- parseAndValidateArgs
 - main.cpp, 111
- parseArguments
 - cli::CommandLineHandler, 32
- parseFile
 - main.cpp, 112
 - parsing::JsonHandler, 65
- parsing, 20
- parsing::FileData, 41
 - addCommand, 42
 - addEnvironmentVariable, 42
 - addPathValue, 42
 - application, 45
 - commands, 46
 - environmentVariables, 46
 - getApplication, 43
 - getCommands, 43
 - getEnvironmentVariables, 43
 - getHideShell, 44
 - getOutputFile, 44
 - getPathValues, 44
 - hideShell, 46
 - outputfile, 46
 - pathValues, 46
 - setApplication, 44
 - setHideShell, 45
 - setOutputFile, 45
- parsing::JsonHandler, 55
 - assignApplication, 57
 - assignCommand, 58
 - assignEntries, 59
 - assignEnvironmentVariable, 60
 - assignHideShell, 61
 - assignOutputFile, 61
 - assignPathValue, 62
 - containsBadCharacter, 63
 - createFileData, 63
 - data, 66
 - getFileData, 64
 - JsonHandler, 57
 - parseFile, 65
 - root, 66
- parsing::KeyValidator, 67
 - getInstance, 68
 - getUnknownKeyLine, 68
 - getWrongKeys, 69
 - typeToKeys, 73
 - validateEntries, 70
 - validateKeys, 71
 - validateTypes, 72
 - validEntryKeys, 73
 - validKeys, 73
- ParsingException
 - exceptions::ParsingException, 82
- PATCH_VERSION
 - config, 19
- pathValues
 - parsing::FileData, 46
- printCredits
 - cli::CommandLineHandler, 33
- printHelp
 - cli::CommandLineHandler, 34
- printVersion
 - cli::CommandLineHandler, 34
- PROJECT_NAME
 - config, 19
- README.md, 91
- root
 - parsing::JsonHandler, 66
- setApplication
 - parsing::FileData, 44
- setHideShell
 - parsing::FileData, 45
- setOutputFile
 - parsing::FileData, 45
- setupEasyLogging
 - utilities::Utils, 89
- src/include/BatchCreator.hpp, 91, 93
- src/include/CommandLineHandler.hpp, 93, 95
- src/include/config.hpp, 95, 97
- src/include/Exceptions.hpp, 97, 99
- src/include/FileData.hpp, 101, 103
- src/include/JsonHandler.hpp, 103, 105
- src/include/KeyValidator.hpp, 106, 107
- src/include/Utils.hpp, 108, 109
- src/main.cpp, 109, 114
- src/sources/BatchCreator.cpp, 116, 118
- src/sources/CommandLineHandler.cpp, 119, 120
- src/sources/FileData.cpp, 121, 123
- src/sources/JsonHandler.cpp, 124, 125
- src/sources/KeyValidator.cpp, 126, 128
- src/sources/Utils.cpp, 129, 131
- StyleHelpers, 15

- type
 - exceptions::InvalidTypeException, [53](#)
 - exceptions::MissingKeyException, [76](#)
- typeToKeys
 - parsing::KeyValidator, [73](#)
- UnreachableCodeException
 - exceptions::UnreachableCodeException, [84](#)
- utilities, [21](#)
- utilities::Utils, [85](#)
 - askToContinue, [86](#)
 - checkConfigFile, [86](#)
 - checkDirectory, [87](#)
 - escapeString, [87](#)
 - handleParseException, [88](#)
 - setupEasyLogging, [89](#)
- validateEntries
 - parsing::KeyValidator, [70](#)
- validateFiles
 - main.cpp, [113](#)
- validateKeys
 - parsing::KeyValidator, [71](#)
- validateTypes
 - parsing::KeyValidator, [72](#)
- validEntryKeys
 - parsing::KeyValidator, [73](#)
- validKeys
 - parsing::KeyValidator, [73](#)
- what
 - exceptions::ContainsBadCharacterException, [37](#)
 - exceptions::CustomException, [38](#)
 - exceptions::FailedToOpenFileException, [40](#)
 - exceptions::FileExistsException, [48](#)
 - exceptions::InvalidKeyException, [50](#)
 - exceptions::InvalidTypeException, [52](#)
 - exceptions::InvalidValueException, [55](#)
 - exceptions::MissingKeyException, [76](#)
 - exceptions::MissingTypeException, [78](#)
 - exceptions::NoSuchDirException, [80](#)
 - exceptions::ParsingException, [83](#)
 - exceptions::UnreachableCodeException, [85](#)
- writeApplication
 - BatchCreator, [27](#)
- writeCommands
 - BatchCreator, [27](#)
- writeEnd
 - BatchCreator, [27](#)
- writeEnvVariables
 - BatchCreator, [28](#)
- writeHideShell
 - BatchCreator, [28](#)
- writePathVariables
 - BatchCreator, [29](#)
- writeStart
 - BatchCreator, [29](#)