

JSON2Batch

0.2.2

Generated on Fri Apr 26 2024 10:00:36 for JSON2Batch by Doxygen 1.9.8

Fri Apr 26 2024 10:00:36



<b>1 JSON2Batch</b>	<b>1</b>
1.1 JSON2Batch . . . . .	1
<b>2 Todo List</b>	<b>3</b>
<b>3 Topic Index</b>	<b>5</b>
3.1 Topics . . . . .	5
<b>4 Namespace Index</b>	<b>7</b>
4.1 Namespace List . . . . .	7
<b>5 Hierarchical Index</b>	<b>9</b>
5.1 Class Hierarchy . . . . .	9
<b>6 Class Index</b>	<b>11</b>
6.1 Class List . . . . .	11
<b>7 File Index</b>	<b>13</b>
7.1 File List . . . . .	13
<b>8 Topic Documentation</b>	<b>15</b>
8.1 StyleHelpers . . . . .	15
<b>9 Namespace Documentation</b>	<b>17</b>
9.1 cli Namespace Reference . . . . .	17
9.1.1 Detailed Description . . . . .	17
9.1.2 Variable Documentation . . . . .	18
9.1.2.1 options . . . . .	18
9.2 config Namespace Reference . . . . .	18
9.2.1 Variable Documentation . . . . .	18
9.2.1.1 AUTHORS . . . . .	18
9.2.1.2 DESCRIPTION . . . . .	18
9.2.1.3 EXECUTABLE_NAME . . . . .	19
9.2.1.4 HOMEPAGE_URL . . . . .	19
9.2.1.5 LOG_CONFIG . . . . .	19
9.2.1.6 MAJOR_VERSION . . . . .	19
9.2.1.7 MINOR_VERSION . . . . .	19
9.2.1.8 PATCH_VERSION . . . . .	19
9.2.1.9 PROJECT_NAME . . . . .	19
9.3 exceptions Namespace Reference . . . . .	20
9.3.1 Detailed Description . . . . .	20
9.4 parsing Namespace Reference . . . . .	20
9.4.1 Detailed Description . . . . .	21
9.5 utilities Namespace Reference . . . . .	21
9.5.1 Detailed Description . . . . .	21

<b>10 Class Documentation</b>	<b>23</b>
10.1 BatchCreator Class Reference	23
10.1.1 Detailed Description	24
10.1.2 Constructor & Destructor Documentation	24
10.1.2.1 BatchCreator()	24
10.1.3 Member Function Documentation	24
10.1.3.1 createBatch()	24
10.1.3.2 getDataStream()	25
10.1.3.3 writeApp()	26
10.1.3.4 writeCommands()	26
10.1.3.5 writeEnd()	27
10.1.3.6 writeEnvVariables()	27
10.1.3.7 writeHideShell()	28
10.1.3.8 writePathVariables()	28
10.1.3.9 writeStart()	29
10.1.4 Member Data Documentation	29
10.1.4.1 dataStream	29
10.1.4.2 fileData	29
10.2 cli::CommandLineHandler Class Reference	29
10.2.1 Detailed Description	30
10.2.2 Constructor & Destructor Documentation	31
10.2.2.1 CommandLineHandler()	31
10.2.2.2 ~CommandLineHandler()	31
10.2.3 Member Function Documentation	31
10.2.3.1 parseArguments()	31
10.2.3.2 printCredits()	32
10.2.3.3 printHelp()	33
10.2.3.4 printVersion()	34
10.3 exceptions::CustomException Class Reference	34
10.3.1 Detailed Description	35
10.3.2 Member Function Documentation	36
10.3.2.1 what()	36
10.4 exceptions::FailedToOpenFileException Class Reference	36
10.4.1 Detailed Description	37
10.4.2 Constructor & Destructor Documentation	37
10.4.2.1 FailedToOpenFileException()	37
10.4.3 Member Function Documentation	38
10.4.3.1 what()	38
10.4.4 Member Data Documentation	38
10.4.4.1 message	38
10.5 parsing::FileData Class Reference	38
10.5.1 Detailed Description	39

10.5.2 Member Function Documentation	39
10.5.2.1 addCommand()	39
10.5.2.2 addEnvironmentVariable()	39
10.5.2.3 addPathValue()	40
10.5.2.4 getApplication()	40
10.5.2.5 getCommands()	41
10.5.2.6 getEnvironmentVariables()	41
10.5.2.7 getHideShell()	41
10.5.2.8 getOutputFile()	41
10.5.2.9 getPathValues()	42
10.5.2.10 setApplication()	42
10.5.2.11 setHideShell()	42
10.5.2.12 setOutputFile()	42
10.5.3 Member Data Documentation	43
10.5.3.1 application	43
10.5.3.2 commands	43
10.5.3.3 environmentVariables	43
10.5.3.4 hideShell	43
10.5.3.5 outputfile	44
10.5.3.6 pathValues	44
10.6 exceptions::FileExistsException Class Reference	44
10.6.1 Detailed Description	45
10.6.2 Constructor & Destructor Documentation	45
10.6.2.1 FileExistsException()	45
10.6.3 Member Function Documentation	46
10.6.3.1 what()	46
10.6.4 Member Data Documentation	46
10.6.4.1 file	46
10.6.4.2 message	46
10.7 exceptions::InvalidKeyException Class Reference	46
10.7.1 Detailed Description	47
10.7.2 Constructor & Destructor Documentation	48
10.7.2.1 InvalidKeyException()	48
10.7.3 Member Function Documentation	48
10.7.3.1 what()	48
10.7.4 Member Data Documentation	48
10.7.4.1 message	48
10.8 exceptions::InvalidTypeException Class Reference	48
10.8.1 Detailed Description	50
10.8.2 Constructor & Destructor Documentation	50
10.8.2.1 InvalidTypeException()	50
10.8.3 Member Function Documentation	50

10.8.3.1 what()	50
10.8.4 Member Data Documentation	50
10.8.4.1 message	50
10.8.4.2 type	51
10.9 exceptions::InvalidValueException Class Reference	51
10.9.1 Detailed Description	52
10.9.2 Constructor & Destructor Documentation	52
10.9.2.1 InvalidValueException()	52
10.9.3 Member Function Documentation	52
10.9.3.1 what()	52
10.9.4 Member Data Documentation	53
10.9.4.1 key	53
10.9.4.2 message	53
10.10 parsing::JsonHandler Class Reference	53
10.10.1 Detailed Description	54
10.10.2 Constructor & Destructor Documentation	54
10.10.2.1 JsonHandler() [1/2]	54
10.10.2.2 JsonHandler() [2/2]	54
10.10.3 Member Function Documentation	55
10.10.3.1 assignApplication()	55
10.10.3.2 assignCommand()	55
10.10.3.3 assignEntries()	56
10.10.3.4 assignEnvironmentVariable()	57
10.10.3.5 assignHideShell()	57
10.10.3.6 assignOutputFile()	58
10.10.3.7 assignPathValue()	58
10.10.3.8 createFileData()	59
10.10.3.9 getFileData()	59
10.10.3.10 parseFile()	60
10.10.4 Member Data Documentation	61
10.10.4.1 data	61
10.10.4.2 root	62
10.11 parsing::KeyValidator Class Reference	62
10.11.1 Detailed Description	63
10.11.2 Member Function Documentation	63
10.11.2.1 getInstance()	63
10.11.2.2 getUnknownKeyLine()	63
10.11.2.3 getWrongKeys()	64
10.11.2.4 validateEntries()	65
10.11.2.5 validateKeys()	66
10.11.2.6 validateTypes()	67
10.11.3 Member Data Documentation	67

10.11.3.1 typeToKeys . . . . .	67
10.11.3.2 validEntryKeys . . . . .	68
10.11.3.3 validKeys . . . . .	68
10.12 exceptions::MissingKeyException Class Reference . . . . .	68
10.12.1 Detailed Description . . . . .	69
10.12.2 Constructor & Destructor Documentation . . . . .	70
10.12.2.1 MissingKeyException() . . . . .	70
10.12.3 Member Function Documentation . . . . .	70
10.12.3.1 what() . . . . .	70
10.12.4 Member Data Documentation . . . . .	70
10.12.4.1 key . . . . .	70
10.12.4.2 message . . . . .	70
10.12.4.3 type . . . . .	70
10.13 exceptions::MissingTypeException Class Reference . . . . .	71
10.13.1 Detailed Description . . . . .	72
10.13.2 Constructor & Destructor Documentation . . . . .	72
10.13.2.1 MissingTypeException() . . . . .	72
10.13.3 Member Function Documentation . . . . .	72
10.13.3.1 what() . . . . .	72
10.13.4 Member Data Documentation . . . . .	72
10.13.4.1 message . . . . .	72
10.14 exceptions::NoSuchDirException Class Reference . . . . .	73
10.14.1 Detailed Description . . . . .	74
10.14.2 Constructor & Destructor Documentation . . . . .	74
10.14.2.1 NoSuchDirException() . . . . .	74
10.14.3 Member Function Documentation . . . . .	74
10.14.3.1 what() . . . . .	74
10.14.4 Member Data Documentation . . . . .	74
10.14.4.1 message . . . . .	74
10.15 options Struct Reference . . . . .	75
10.15.1 Detailed Description . . . . .	75
10.16 exceptions::ParsingException Class Reference . . . . .	75
10.16.1 Detailed Description . . . . .	76
10.16.2 Constructor & Destructor Documentation . . . . .	76
10.16.2.1 ParsingException() . . . . .	76
10.16.3 Member Function Documentation . . . . .	77
10.16.3.1 what() . . . . .	77
10.16.4 Member Data Documentation . . . . .	77
10.16.4.1 file . . . . .	77
10.16.4.2 message . . . . .	77
10.17 exceptions::UnreachableCodeException Class Reference . . . . .	77
10.17.1 Detailed Description . . . . .	78

10.17.2 Constructor & Destructor Documentation	78
10.17.2.1 UnreachableCodeException()	78
10.17.3 Member Function Documentation	79
10.17.3.1 what()	79
10.17.4 Member Data Documentation	79
10.17.4.1 message	79
10.18 utilities::Utils Class Reference	79
10.18.1 Detailed Description	79
10.18.2 Member Function Documentation	79
10.18.2.1 askToContinue()	79
10.18.2.2 checkConfigFile()	80
10.18.2.3 checkDirectory()	81
10.18.2.4 handleParseException()	81
10.18.2.5 setupEasyLogging()	82
<b>11 File Documentation</b>	<b>83</b>
11.1 README.md File Reference	83
11.2 src/include/BatchCreator.hpp File Reference	83
11.2.1 Detailed Description	84
11.3 BatchCreator.hpp	85
11.4 src/include/CommandLineHandler.hpp File Reference	85
11.4.1 Detailed Description	86
11.5 CommandLineHandler.hpp	87
11.6 src/include/config.hpp File Reference	87
11.6.1 Detailed Description	89
11.7 config.hpp	89
11.8 src/include/Exceptions.hpp File Reference	90
11.8.1 Detailed Description	91
11.9 Exceptions.hpp	91
11.10 src/include/FileData.hpp File Reference	93
11.10.1 Detailed Description	94
11.11 FileData.hpp	95
11.12 src/include/JsonHandler.hpp File Reference	96
11.12.1 Detailed Description	97
11.13 JsonHandler.hpp	97
11.14 src/include/KeyValidator.hpp File Reference	98
11.14.1 Detailed Description	99
11.15 KeyValidator.hpp	99
11.16 src/include/Utils.hpp File Reference	100
11.17 Utils.hpp	101
11.18 src/main.cpp File Reference	101
11.18.1 Detailed Description	102



11.18.2 Function Documentation	102
11.18.2.1 main()	102
11.18.2.2 parseAndValidateArgs()	103
11.18.2.3 parseFile()	104
11.18.2.4 validateFiles()	105
11.19 main.cpp	106
11.20 src/sources/BatchCreator.cpp File Reference	107
11.21 BatchCreator.cpp	108
11.22 src/sources/CommandLineHandler.cpp File Reference	109
11.22.1 Detailed Description	110
11.23 CommandLineHandler.cpp	110
11.24 src/sources/FileData.cpp File Reference	112
11.24.1 Detailed Description	112
11.25 FileData.cpp	113
11.26 src/sources/JsonHandler.cpp File Reference	114
11.26.1 Detailed Description	114
11.27 JsonHandler.cpp	115
11.28 src/sources/KeyValidator.cpp File Reference	116
11.28.1 Detailed Description	116
11.29 KeyValidator.cpp	117
11.30 src/sources/Utils.cpp File Reference	118
11.30.1 Detailed Description	119
11.31 Utils.cpp	120
<b>Index</b>	<b>121</b>



# Chapter 1

## JSON2Batch

*This file is autogenerated. Changes will be overwritten*

### 1.1 JSON2Batch

**Todo** Update [README.md](#)

Beschreibung: A simple tool to convert json to batch.

Version: 0.2.2

Autoren: Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci

Documentation: <https://dhbwprojectsit23.github.io/JSON2Bat>

#### Aktueller Plan:

- Verantwortlichkeiten zugewiesen
- "Sprint" bis ?

#### Verantwortlichkeiten:

- [CMake](#) &#8594 Simon
- [JsonParsing](#) &#8594 Elena und Sonia
- [Batch Creation](#) &#8594 Max
- [CLI](#) &#8594 Simon

#### Andere Arbeitspakete

- Error Handling
- Unit Tests
- Code Quality
- Documentation

#### Bezüglich Code Quality

- Kein using namespace

- Nur main im Global Namespace

### Wichtige Commands

Branch wechseln

- git checkout -b NEUERBRANCH Pushen
- git push origin zum pullen
- git pull --prune

### Kurze Doxygen Übersicht

**Achtung: Die Leerzeichen zwischen @ und dem Wort dürfen nicht in den Code, sind nur da, damit Doxygen die nicht aufnimmt! /\*\***

- @ brief Kurze Beschreibung
- @ details Längere
- @ todo
- @ bug
- @ param PARAMETERNAME was der macht
- @ return was die funktion return
- @ throws \*\*/

## Chapter 2

# Todo List

### File [BatchCreator.hpp](#)

Documentation in english

### Member [BatchCreator::getDataStream](#) () const

Documentation

### Member [cli::CommandLineHandler::parseArguments](#) (int argc, char \*argv[])

Update documentation

### Member [exceptions::FailedToOpenFileException::FailedToOpenFileException](#) (const std::string &file)

Documentation

### Member [exceptions::NoSuchDirException::NoSuchDirException](#) (const std::string &dir)

Documentation

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

Documentation

Refactoring

### page [Main Page](#)

Update [README.md](#)

### Namespace [parsing](#)

Document – map/set for efficient

### Member [parsing::KeyValidator::getUnknownKeyLine](#) (const std::string &filename, const std::string &wrongKey)

Documentation

### Member [parsing::KeyValidator::validateEntries](#) (const std::string &filename, const std::unordered\_set<std::string> &entryKeys) const

Documentation

### Member [utilities::Utils::checkDirectory](#) (std::string &directory)

documentation



# Chapter 3

## Topic Index

### 3.1 Topics

Here is a list of all topics with brief descriptions:

StyleHelpers . . . . .	15
------------------------	----





## Chapter 4

# Namespace Index

### 4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">cli</a>	Includes everything regarding the CLI . . . . .	17
<a href="#">config</a>	. . . . .	18
<a href="#">exceptions</a>	Namespace used for customized exceptions . . . . .	20
<a href="#">parsing</a>	The namespace containing everything relevant to parsing . . . . .	20
<a href="#">utilities</a>	Includes all utilities . . . . .	21



## Chapter 5

# Hierarchical Index

### 5.1 Class Hierarchy

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

BatchCreator . . . . .	23
cli::CommandLineHandler . . . . .	29
std::exception	
exceptions::CustomException . . . . .	34
exceptions::FailedToOpenFileException . . . . .	36
exceptions::FileExistsException . . . . .	44
exceptions::InvalidKeyException . . . . .	46
exceptions::InvalidTypeException . . . . .	48
exceptions::InvalidValueException . . . . .	51
exceptions::MissingKeyException . . . . .	68
exceptions::MissingTypeException . . . . .	71
exceptions::NoSuchDirException . . . . .	73
exceptions::ParsingException . . . . .	75
exceptions::UnreachableCodeException . . . . .	77
parsing::FileData . . . . .	38
parsing::JsonHandler . . . . .	53
parsing::KeyValidator . . . . .	62
options . . . . .	75
utilities::Utils . . . . .	79



# Chapter 6

## Class Index

### 6.1 Class List

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

<a href="#">BatchCreator</a>	
Erstellt Batch Datei . . . . .	23
<a href="#">cli::CommandLineHandler</a>	
Responsible for the Command Line Interface . . . . .	29
<a href="#">exceptions::CustomException</a>	
Base class for all custom exceptions . . . . .	34
<a href="#">exceptions::FailedToOpenFileException</a>	
. . . . .	36
<a href="#">parsing::FileData</a>	
This class contains all data from the json file . . . . .	38
<a href="#">exceptions::FileExistsException</a>	
Exception for an already existing outputfile . . . . .	44
<a href="#">exceptions::InvalidKeyException</a>	
Exception for invalid keys . . . . .	46
<a href="#">exceptions::InvalidTypeException</a>	
Exception for invalid types . . . . .	48
<a href="#">exceptions::InvalidValueException</a>	
Exception for an invalid (usually empty) value field . . . . .	51
<a href="#">parsing::JsonHandler</a>	
This file reads all data from the json file . . . . .	53
<a href="#">parsing::KeyValidator</a>	
Validates keys of a Json::Value object . . . . .	62
<a href="#">exceptions::MissingKeyException</a>	
Exception for missing keys within entries . . . . .	68
<a href="#">exceptions::MissingTypeException</a>	
Exception for missing types of entries . . . . .	71
<a href="#">exceptions::NoSuchDirException</a>	
. . . . .	73
<a href="#">options</a>	
The struct containing all possible options . . . . .	75
<a href="#">exceptions::ParsingException</a>	
Exception for syntax errors within the json file . . . . .	75
<a href="#">exceptions::UnreachableCodeException</a>	
Exception for when the application reaches code it shouldn't reach . . . . .	77
<a href="#">utilities::Utils</a>	
Responsible for utility function . . . . .	79



# Chapter 7

## File Index

### 7.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	
Contains the main function . . . . .	101
src/include/BatchCreator.hpp	
Creates batch file . . . . .	83
src/include/CommandLineHandler.hpp	
Responsible for the Command Line Interface . . . . .	85
src/include/config.hpp	
Configures general project information . . . . .	87
src/include/Exceptions.hpp	
Contains all the custom exceptions used in the project . . . . .	90
src/include/FileData.hpp	
This file contains the FileData class . . . . .	93
src/include/JsonHandler.hpp	
This file contains the JsonHandler class . . . . .	96
src/include/KeyValidator.hpp	
This file contains the KeyValidator class . . . . .	98
src/include/Utils.hpp	
. . . . .	100
src/sources/BatchCreator.cpp	
. . . . .	107
src/sources/CommandLineHandler.cpp	
Implementation for the Command Line Interface . . . . .	109
src/sources/FileData.cpp	
. . . . .	112
src/sources/JsonHandler.cpp	
. . . . .	114
src/sources/KeyValidator.cpp	
. . . . .	116
src/sources/Utils.cpp	
Implementation for the Utils class . . . . .	118





## Chapter 8

# Topic Documentation

### 8.1 StyleHelpers

Static variables to help with CLI styling.

Static variables to help with CLI styling.

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



## Chapter 9

# Namespace Documentation

### 9.1 cli Namespace Reference

Includes everything regarding the CLI.

#### Classes

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

#### Variables

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

#### 9.1.1 Detailed Description

Includes everything regarding the CLI.

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

#### See also

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

## 9.1.2 Variable Documentation

### 9.1.2.1 options

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

**Initial value:**

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

Definition at line 117 of file [CommandLineHandler.hpp](#).

## 9.2 config Namespace Reference

### Variables

- constexpr auto [LOG\\_CONFIG](#)
- constexpr auto [EXECUTABLE\\_NAME](#) = "json2batch"
- constexpr auto [MAJOR\\_VERSION](#) = "0"
- constexpr auto [MINOR\\_VERSION](#) = "2"
- constexpr auto [PATCH\\_VERSION](#) = "2"
- constexpr auto [DESCRIPTION](#) = "A simple tool to convert json to batch."
- constexpr auto [PROJECT\\_NAME](#) = "JSON2Batch"
- constexpr auto [AUTHORS](#)
- constexpr auto [HOMEPAGE\\_URL](#)

## 9.2.1 Variable Documentation

### 9.2.1.1 AUTHORS

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

**Initial value:**

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

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

### 9.2.1.2 DESCRIPTION

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

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

### 9.2.1.3 EXECUTABLE\_NAME

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

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

### 9.2.1.4 HOMEPAGE\_URL

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

**Initial value:**

```
=  
    "https://dhwprojectsit23.github.io/JSON2Bat"
```

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

### 9.2.1.5 LOG\_CONFIG

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

**Initial value:**

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

Definition at line 24 of file [config.hpp](#).

### 9.2.1.6 MAJOR\_VERSION

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

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

### 9.2.1.7 MINOR\_VERSION

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

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

### 9.2.1.8 PATCH\_VERSION

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

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

### 9.2.1.9 PROJECT\_NAME

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

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

## 9.3 exceptions Namespace Reference

Namespace used for customized exceptions.

### Classes

- class [CustomException](#)  
*Base class for all custom exceptions.*
- class [FailedToOpenFileException](#)
- 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 ivalid (usually empty) value field.*
- class [MissingKeyException](#)  
*Exception for missing keys within entries.*
- class [MissingTypeException](#)  
*Exception for missing types of entries.*
- class [NoSuchDirException](#)
- class [ParsingException](#)  
*Exception for syntax errors within the json file.*
- class [UnreachableCodeException](#)  
*Exception for when the application reaches code it shouldn't reach.*

### 9.3.1 Detailed Description

Namespace used for customized exceptions.

## 9.4 parsing Namespace Reference

The namespace containing everything relevant to parsing.

### Classes

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

### 9.4.1 Detailed Description

The namespace containing everything relevant to parsing.

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

See also

[JsonHandler](#)

[FileData](#)

[KeyValidator](#)

[BatchCreator](#)

**Todo** Document – map/set for efficient

## 9.5 utilities Namespace Reference

Includes all utilities.

### Classes

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

### 9.5.1 Detailed Description

Includes all utilities.

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

See also

[Utils](#)





# Chapter 10

## Class Documentation

### 10.1 BatchCreator Class Reference

Erstellt Batch Datei.

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

#### Public Member Functions

- [BatchCreator](#) (std::shared\_ptr< [parsing::FileData](#) > [fileData](#))  
*Initialisiert [BatchCreator](#).*
- std::shared\_ptr< std::stringstream > [getDataStream](#) () const

#### Private Member Functions

- void [createBatch](#) ()  
*Setzt batch Datei zusammen.*
- void [writeStart](#) () const  
*Anfang der Batch Datei.*
- void [writeHideShell](#) () const  
*Sichtbarkeit Konsole.*
- void [writeCommands](#) () const  
*Befehle ausführen.*
- void [writeEnvVariables](#) () const  
*Umgebungsvariablen setzten.*
- void [writePathVariables](#) () const  
*Pfade setzten.*
- void [writeApp](#) () const  
*Öffnet Anwendung falls gewünscht.*
- void [writeEnd](#) () const  
*Ende der Batch Datei.*

#### Private Attributes

- std::shared\_ptr< std::stringstream > [dataStream](#)
- std::shared\_ptr< [parsing::FileData](#) > [fileData](#)

### 10.1.1 Detailed Description

Erstellt Batch Datei.

Wandelt Elemente aus JSON-Datei in Batch-Format um

See also

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

### 10.1.2 Constructor & Destructor Documentation

#### 10.1.2.1 BatchCreator()

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

Initialisiert [BatchCreator](#).

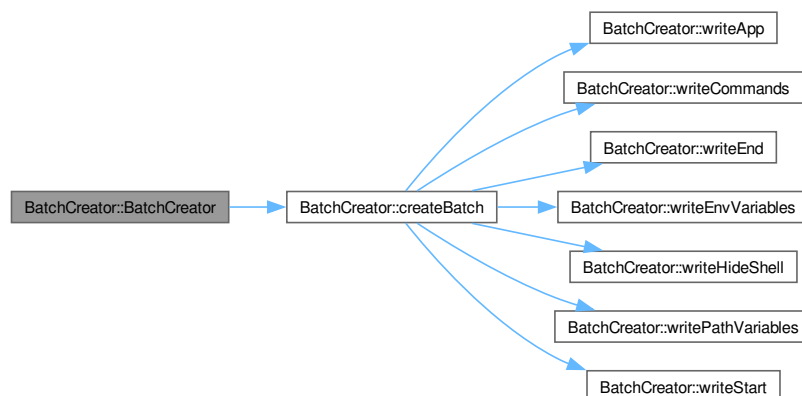
Parameters

<i>filename</i>	
-----------------	--

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

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

Here is the call graph for this function:



### 10.1.3 Member Function Documentation

#### 10.1.3.1 createBatch()

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

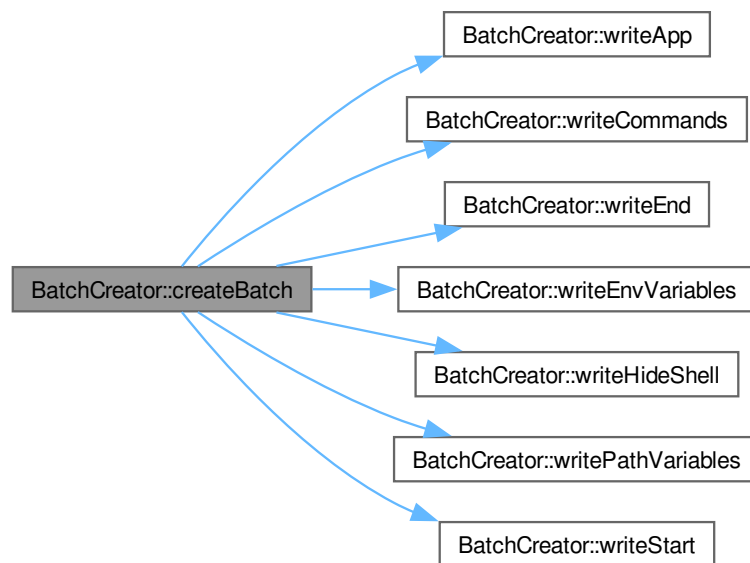
Setzt batch Datei zusammen.

Beinhaltet Aufrufe der einzelnen Komponenten der batch Datei

Definition at line 24 of file [BatchCreator.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.1.3.2 getDataStream()

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

**Todo** Documentation

Definition at line 39 of file [BatchCreator.hpp](#).

References [dataStream](#).

Here is the caller graph for this function:



### 10.1.3.3 writeApp()

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

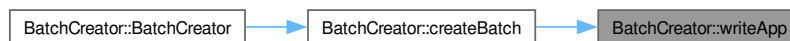
Öffnet Anwendung falls gewünscht.

Öffnet Anwendung, falls unter "application" gegeben Wird unter dem Namen aus "outputfile" gestartet

Definition at line 76 of file [BatchCreator.cpp](#).

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

Here is the caller graph for this function:



### 10.1.3.4 writeCommands()

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

Befehle ausführen.

Führt Befehle aus: Zu finden unter "EXE" als "command"

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

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

Here is the caller graph for this function:



### 10.1.3.5 writeEnd()

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

Ende der Batch Datei.

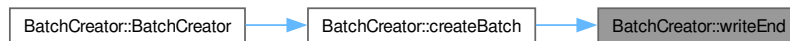
Schreibt den teil der Batch Datei der immer gleich ist

- setzt ECHO OFF

Definition at line 89 of file [BatchCreator.cpp](#).

References [dataStream](#).

Here is the caller graph for this function:



### 10.1.3.6 writeEnvVariables()

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

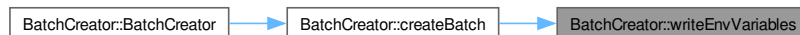
Umgebungsvariablen setzten.

Setzt Umgebungsvariablen aus "ENV" nach folgender Syntax: Eintrag unter "key" = Eintrag unter "value"

Definition at line 60 of file [BatchCreator.cpp](#).

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

Here is the caller graph for this function:



### 10.1.3.7 writeHideShell()

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

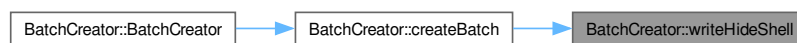
Sichtbarkeit Konsole.

Zeigt bzw. versteckt Konsolenausgabe

Definition at line 41 of file [BatchCreator.cpp](#).

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

Here is the caller graph for this function:



### 10.1.3.8 writePathVariables()

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

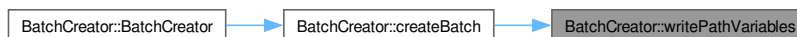
Pfade setzten.

Verknüpft die unter "PATH" angegebenen Pfade mit dem Systempfad Setzt Pfad

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

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

Here is the caller graph for this function:



### 10.1.3.9 writeStart()

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

Anfang der Batch Datei.

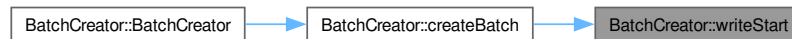
Schreibt den Teil der Batch Datei der immer gleich ist.

- setzt ECHO off
- startet cmd.exe

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

References [dataStream](#).

Here is the caller graph for this function:



## 10.1.4 Member Data Documentation

### 10.1.4.1 dataStream

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

Definition at line 44 of file [BatchCreator.hpp](#).

### 10.1.4.2 fileData

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

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

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

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

## 10.2 cli::CommandLineHandler Class Reference

Responsible for the Command Line Interface.

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

## Public Member Functions

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

## Static Public Member Functions

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

## 10.2.1 Detailed Description

Responsible for the Command Line Interface.

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

### Author

Simon Blum

### Date

2024-04-18

### Version

0.1.5

### See also

[options](#)

Definition at line 53 of file [CommandLineHandler.hpp](#).



## 10.2.2 Constructor & Destructor Documentation

### 10.2.2.1 CommandLineHandler()

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

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

#### Note

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

### 10.2.2.2 ~CommandLineHandler()

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

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

#### Note

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

## 10.2.3 Member Function Documentation

### 10.2.3.1 parseArguments()

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

Parses the Command Line Arguments.

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

#### Parameters

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

#### Exceptions

<i>std::logic_error</i>	
-------------------------	--

#### Returns

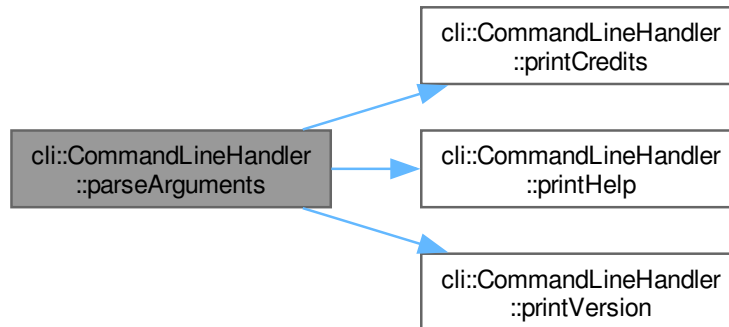
Returns a vector of strings containing all filenames.

**Todo** Update documentation

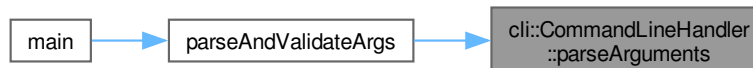
Definition at line 70 of file [CommandLineHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.2.3.2 printCredits()

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

Prints the credits message.

Prints the credits message when called.

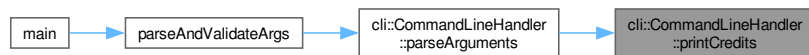
**Note**

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:

**10.2.3.3 printHelp()**

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

Prints the help message.

Prints the help message when called.

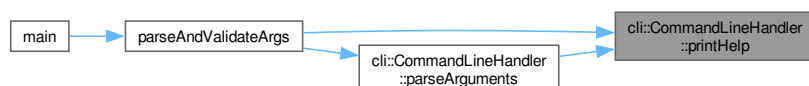
**Note**

This function ends the application.

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

References [config::EXECUTABLE\\_NAME](#).

Here is the caller graph for this function:



#### 10.2.3.4 printVersion()

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

Prints the version message.

Prints the version message when called.

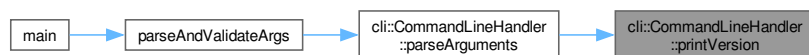
##### Note

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](#)

## 10.3 exceptions::CustomException Class Reference

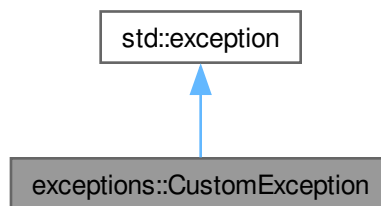
Base class for all custom exceptions.

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

Inheritance diagram for exceptions::CustomException:



Collaboration diagram for exceptions::CustomException:



### Public Member Functions

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

### 10.3.1 Detailed Description

Base class for all custom exceptions.

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

See also

`std::exception`

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

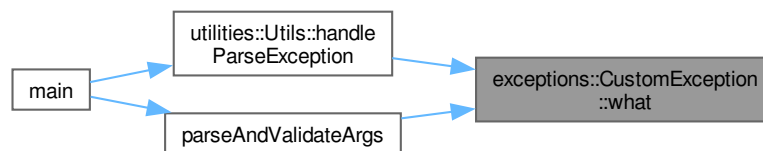
## 10.3.2 Member Function Documentation

### 10.3.2.1 `what()`

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

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

Here is the caller graph for this function:



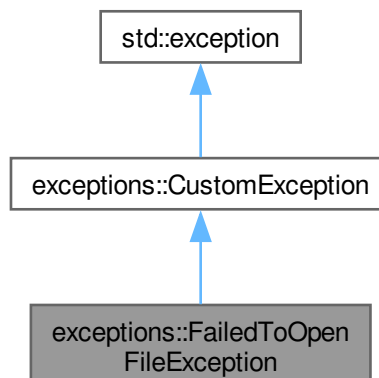
The documentation for this class was generated from the following file:

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

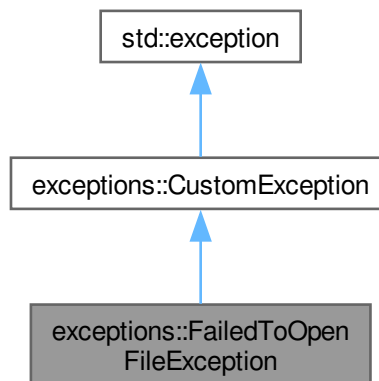
## 10.4 `exceptions::FailedToOpenFileException` Class Reference

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

Inheritance diagram for `exceptions::FailedToOpenFileException`:



Collaboration diagram for exceptions::FailedToOpenFileException:



#### Public Member Functions

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

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

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

#### Private Attributes

- std::string [message](#)

### 10.4.1 Detailed Description

Definition at line 246 of file [Exceptions.hpp](#).

### 10.4.2 Constructor & Destructor Documentation

#### 10.4.2.1 FailedToOpenFileException()

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

**Todo** Documentation

Definition at line 252 of file [Exceptions.hpp](#).

References [message](#).

### 10.4.3 Member Function Documentation

#### 10.4.3.1 what()

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

Definition at line 256 of file [Exceptions.hpp](#).

References [message](#).

### 10.4.4 Member Data Documentation

#### 10.4.4.1 message

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

Definition at line 248 of file [Exceptions.hpp](#).

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

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

## 10.5 parsing::FileData Class Reference

This class contains all data from the json file.

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

#### Public Member Functions

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



### Private Attributes

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

## 10.5.1 Detailed Description

This class contains all data from the json file.

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

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

## 10.5.2 Member Function Documentation

### 10.5.2.1 addCommand()

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

Adds a given command to this->commands.

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

#### Parameters

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

#### Exceptions

<a href="#">exceptions::InvalidValueException</a>	
---	--

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

References [commands](#).

### 10.5.2.2 addEnvironmentVariable()

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

Adds a given tuple to this->environmentVariables.

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

## Parameters

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

## Exceptions

<a href="#">exceptions::InvalidValueException</a>	
---	--

Definition at line 66 of file [FileData.cpp](#).

References [environmentVariables](#).

### 10.5.2.3 addPathValue()

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

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

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

## Parameters

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

## Exceptions

<a href="#">exceptions::InvalidValueException</a>	
---	--

Definition at line 82 of file [FileData.cpp](#).

References [pathValues](#).

### 10.5.2.4 getApplication()

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

Getter for this->application.

## Returns

The assigned application

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

References [application](#).

### 10.5.2.5 getCommands()

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

Getter for this->commands.

#### Returns

The vector of assigned commands

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

References [commands](#).

### 10.5.2.6 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 131 of file [FileData.hpp](#).

References [environmentVariables](#).

### 10.5.2.7 getHideShell()

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

Getter for this->hideShell.

#### Returns

The assigned value for hideshow

Definition at line 108 of file [FileData.hpp](#).

References [hideShell](#).

### 10.5.2.8 getOutputFile()

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

Getter for this->outputfile.

#### Returns

The assigned outputfile

Definition at line 102 of file [FileData.hpp](#).

References [outputfile](#).

### 10.5.2.9 getPathValues()

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

Getter for this->pathValues.

#### Returns

The vector of assigned pathValues

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

References [pathValues](#).

### 10.5.2.10 setApplication()

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

Setter for this->application.

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

#### Parameters

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

Definition at line 45 of file [FileData.cpp](#).

References [application](#).

### 10.5.2.11 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 48 of file [FileData.hpp](#).

References [hideShell](#).

### 10.5.2.12 setOutputFile()

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

Setter for this->outputfile.

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

#### Parameters

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

#### Exceptions

<a href="#">exceptions::InvalidValueException</a>	
---	--

Definition at line 17 of file [FileData.cpp](#).

References [outputfile](#).

## 10.5.3 Member Data Documentation

### 10.5.3.1 application

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

Definition at line 146 of file [FileData.hpp](#).

### 10.5.3.2 commands

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

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

### 10.5.3.3 environmentVariables

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

Definition at line 148 of file [FileData.hpp](#).

### 10.5.3.4 hideShell

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

Definition at line 145 of file [FileData.hpp](#).

### 10.5.3.5 outputfile

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

Definition at line 144 of file [FileData.hpp](#).

### 10.5.3.6 pathValues

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

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

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

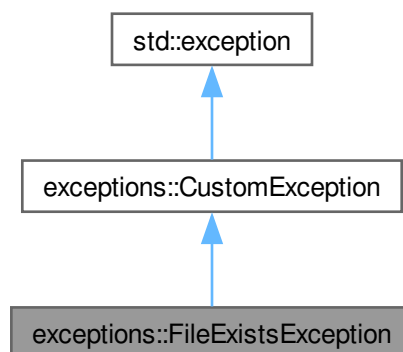
- [src/include/FileData.hpp](#)
- [src/sources/FileData.cpp](#)

## 10.6 exceptions::FileExistsException Class Reference

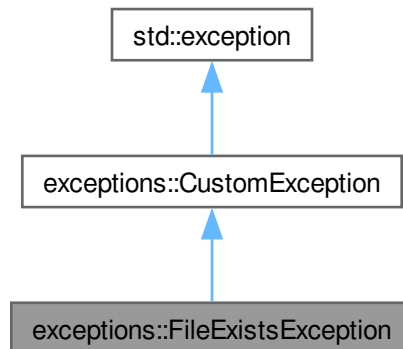
Exception for an already exisiting outputfile.

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

Inheritance diagram for exceptions::FileExistsException:



Collaboration diagram for exceptions::FileExistsException:



#### Public Member Functions

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

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

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

#### Private Attributes

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

### 10.6.1 Detailed Description

Exception for an already existing outputfile.

Definition at line 69 of file [Exceptions.hpp](#).

### 10.6.2 Constructor & Destructor Documentation

#### 10.6.2.1 FileExistsException()

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

#### Note

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

Definition at line 75 of file [Exceptions.hpp](#).

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

### 10.6.3 Member Function Documentation

#### 10.6.3.1 what()

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

Definition at line 87 of file [Exceptions.hpp](#).

References [message](#).

### 10.6.4 Member Data Documentation

#### 10.6.4.1 file

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

Definition at line 71 of file [Exceptions.hpp](#).

#### 10.6.4.2 message

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

Definition at line 72 of file [Exceptions.hpp](#).

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

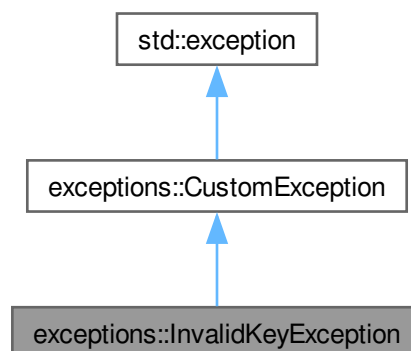
- [src/include/Exceptions.hpp](#)

## 10.7 exceptions::InvalidKeyException Class Reference

Exception for invalid keys.

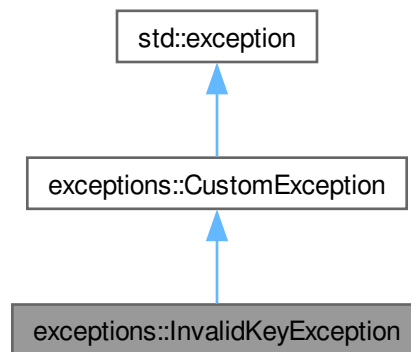
```
#include <Exceptions.hpp>
```

Inheritance diagram for exceptions::InvalidKeyException:





Collaboration diagram for exceptions::InvalidKeyException:



### Public Member Functions

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

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

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

### Private Attributes

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

## 10.7.1 Detailed Description

Exception for invalid keys.

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

See also

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

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

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

## 10.7.2 Constructor & Destructor Documentation

### 10.7.2.1 InvalidKeyException()

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

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

References [message](#).

## 10.7.3 Member Function Documentation

### 10.7.3.1 what()

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

Definition at line 143 of file [Exceptions.hpp](#).

References [message](#).

## 10.7.4 Member Data Documentation

### 10.7.4.1 message

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

Definition at line 132 of file [Exceptions.hpp](#).

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

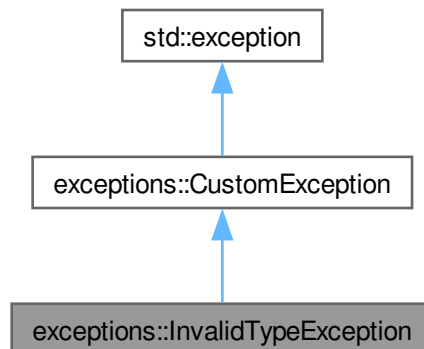
- [src/include/Exceptions.hpp](#)

## 10.8 exceptions::InvalidTypeException Class Reference

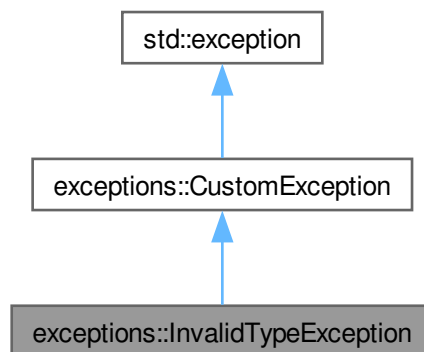
Exception for invalid types.

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

Inheritance diagram for exceptions::InvalidTypeException:



Collaboration diagram for exceptions::InvalidTypeException:



### Public Member Functions

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

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

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

### Private Attributes

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

### 10.8.1 Detailed Description

Exception for invalid types.

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

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

### 10.8.2 Constructor & Destructor Documentation

#### 10.8.2.1 InvalidTypeException()

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

##### Note

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

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

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

### 10.8.3 Member Function Documentation

#### 10.8.3.1 what()

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

Definition at line [173](#) of file [Exceptions.hpp](#).

References [message](#).

### 10.8.4 Member Data Documentation

#### 10.8.4.1 message

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

Definition at line [159](#) of file [Exceptions.hpp](#).

#### 10.8.4.2 type

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

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

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

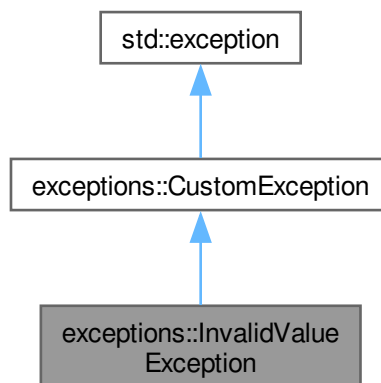
- [src/include/Exceptions.hpp](#)

## 10.9 exceptions::InvalidValueException Class Reference

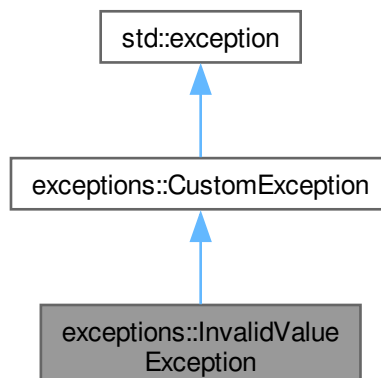
Exception for an ivalid (usually empty) value field.

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

Inheritance diagram for exceptions::InvalidValueException:



Collaboration diagram for exceptions::InvalidValueException:



## Public Member Functions

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

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

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

## Private Attributes

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

### 10.9.1 Detailed Description

Exception for an ivalid (usually empty) value field.

Definition at line 96 of file [Exceptions.hpp](#).

### 10.9.2 Constructor & Destructor Documentation

#### 10.9.2.1 InvalidValueException()

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

#### Note

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

Definition at line 102 of file [Exceptions.hpp](#).

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

### 10.9.3 Member Function Documentation

#### 10.9.3.1 what()

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

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

References [message](#).

## 10.9.4 Member Data Documentation

### 10.9.4.1 key

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

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

### 10.9.4.2 message

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

Definition at line 99 of file [Exceptions.hpp](#).

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

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

## 10.10 parsing::JsonHandler Class Reference

This file reads all data from the json file.

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

### Public Member Functions

- [JsonHandler](#) ()  
*Constructor without arguments.*
- [JsonHandler](#) (const std::string &filename)  
*The constructor.*
- std::shared\_ptr< [FileData](#) > [getFileData](#) ()  
*Retrieve the data from the json file.*

### Private Member Functions

- void [assignOutputFile](#) () const  
*Assigns the outputfile to this->data.*
- void [assignHideShell](#) () const  
*Assigns the 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.*

## Private Attributes

- std::shared\_ptr< Json::Value > [root](#)
- std::shared\_ptr< [FileData](#) > [data](#)

### 10.10.1 Detailed Description

This file reads all data from the json file.

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

See also

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

Definition at line [45](#) of file [JsonHandler.hpp](#).

### 10.10.2 Constructor & Destructor Documentation

#### 10.10.2.1 JsonHandler() [1/2]

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

Constructor without arguments.

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

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

#### 10.10.2.2 JsonHandler() [2/2]

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

The constructor.

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

Parameters

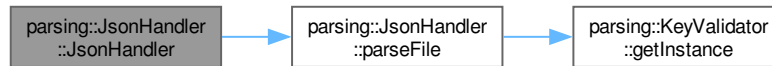
<i>filename</i>	Name of the json file
-----------------	-----------------------



Definition at line 19 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



## 10.10.3 Member Function Documentation

### 10.10.3.1 assignApplication()

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

Assigns application to this->data.

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

Definition at line 75 of file [JsonHandler.cpp](#).

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

Here is the caller graph for this function:



### 10.10.3.2 assignCommand()

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

Assigns an command to this->data.

#### Parameters

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

Definition at line 104 of file [JsonHandler.cpp](#).

References [data](#).

Here is the caller graph for this function:



### 10.10.3.3 assignEntries()

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

Assigns entries to this->data.

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

#### Parameters

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

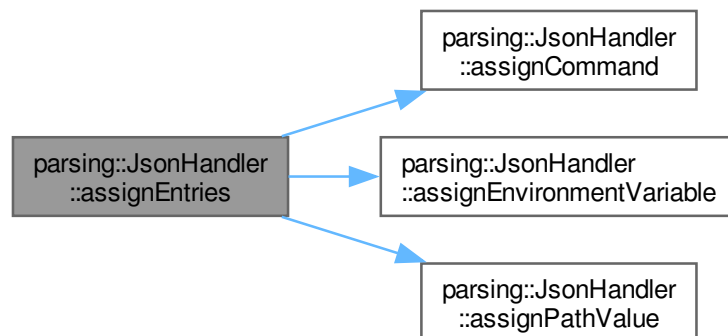
#### Exceptions

<a href="#">exceptions::UnreachableCodeException</a>	
--	--

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.10.3.4 assignEnvironmentVariable()

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

Assigns an environmentVariable to this->data.

##### Parameters

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

Definition at line 109 of file [JsonHandler.cpp](#).

References [data](#).

Here is the caller graph for this function:



#### 10.10.3.5 assignHideShell()

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

Assigns the hideshell value to this->data.

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

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

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

Here is the caller graph for this function:



### 10.10.3.6 assignOutputFile()

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

Assigns the outputfile to this->data.

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

#### Exceptions

<a href="#">exceptions::FileExistsException</a>
---

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

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

Here is the caller graph for this function:



### 10.10.3.7 assignPathValue()

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

Assigns a path value to this->data.

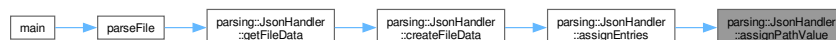
#### Parameters

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

Definition at line 116 of file [JsonHandler.cpp](#).

References [data](#).

Here is the caller graph for this function:



### 10.10.3.8 createFileData()

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

Creates the [FileData](#) instance.

Instantiates the [FileData](#) instance, calls all nessecary functions and returns a shared pointer to it.

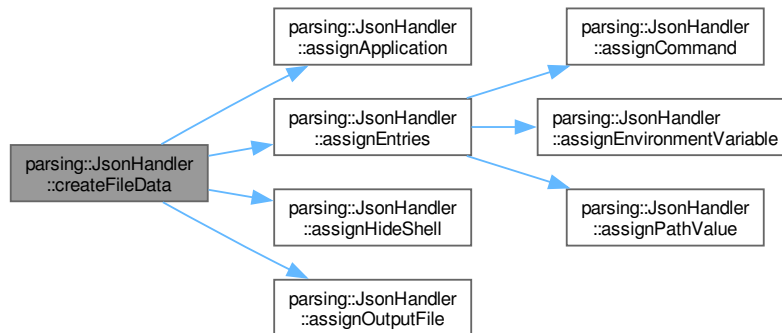
#### Returns

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

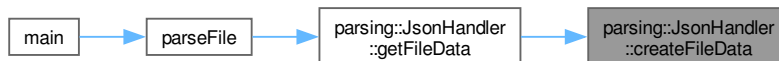
Definition at line 52 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.10.3.9 getFileData()

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

Retrieve the data from the json file.

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

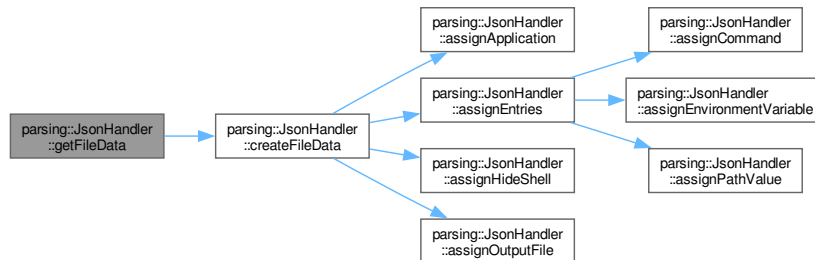
**Returns**

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:

**10.10.3.10 parseFile()**

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

Parses the given json file.

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

**Parameters**

<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

<a href="#">exceptions::ParsingException</a>	
<a href="#">exceptions::InvalidKeyException</a>	

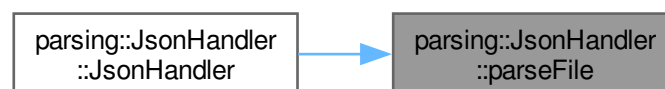
Definition at line 24 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.10.4 Member Data Documentation

### 10.10.4.1 data

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

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

### 10.10.4.2 root

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

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

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

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

## 10.11 parsing::KeyValidator Class Reference

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

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

### Public Member Functions

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

*Validate keys off a `Json::Value` object.*

### Static Public Member Functions

- static [KeyValidator](#) & [getInstance](#) ()

*Get the instance of this class.*

### Private Member Functions

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

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

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

*Validates types from the entries array.*

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

*Validates that an entries 'type' key is valid.*

### Static Private Member Functions

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

### Private Attributes

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



### 10.11.1 Detailed Description

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

This class is singleton. That way when multiple files are parsed with the application, the `validKeys` and `validEntryKeys` field only have to be allocated once.

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

### 10.11.2 Member Function Documentation

#### 10.11.2.1 getInstance()

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

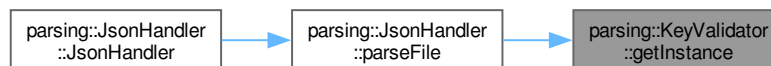
Get the instance of this class.

##### Returns

Reference to the instance of this class

Definition at line 21 of file [KeyValidator.cpp](#).

Here is the caller graph for this function:



#### 10.11.2.2 getUnknownKeyLine()

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

##### Parameters

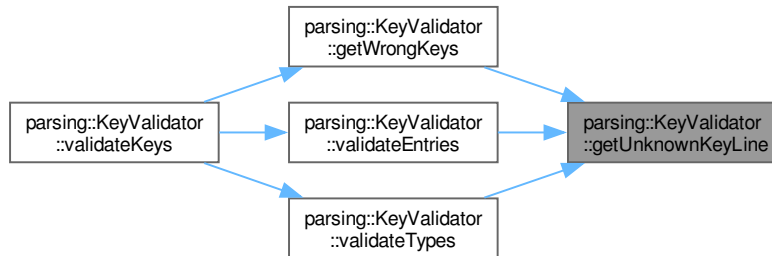
<i>filename</i>	
<i>wrongKey</i>	

##### Returns

**Todo** Documentation

Definition at line 117 of file [KeyValidator.cpp](#).

Here is the caller graph for this function:



### 10.11.2.3 getWrongKeys()

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

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

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

#### Parameters

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

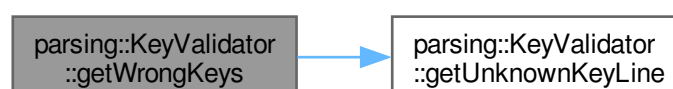
#### Returns

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

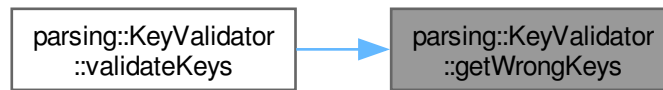
Definition at line 51 of file [KeyValidator.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.2.4 validateEntries()

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

Validates that an entries 'type' key is valid.

##### Parameters

<i>filename</i>	
<i>entryKeys</i>	

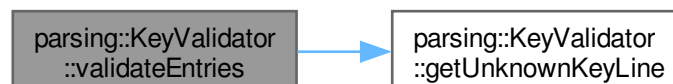
##### Returns

**Todo** Documentation

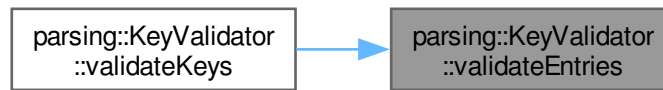
Definition at line 71 of file [KeyValidator.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.11.2.5 validateKeys()

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

Validate keys off a `Json::Value` object.

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

#### Parameters

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

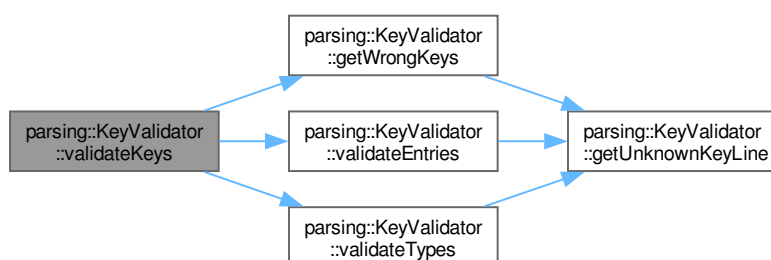
#### Returns

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

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

References [getWrongKeys\(\)](#), [validateEntries\(\)](#), and [validateTypes\(\)](#).

Here is the call graph for this function:



### 10.11.2.6 validateTypes()

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

Validates types from the entries array.

Makes sure that each type has it's according keys, needed to parse it.

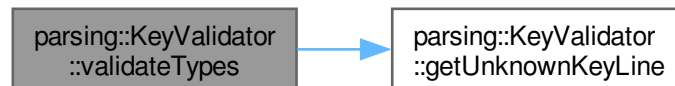
#### Parameters

<i>filename</i>	The filename from which 'entry' is from
<i>entry</i>	
<i>entryKeys</i>	

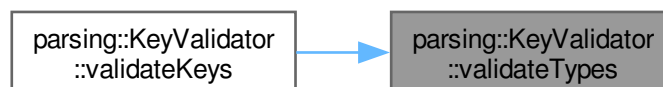
Definition at line 92 of file [KeyValidator.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.11.3 Member Data Documentation

### 10.11.3.1 typeToKeys

```
std::unordered_map<std::string_view, std::vector<std::string> > parsing::KeyValidator::typeToKeys [private]
```

#### Initial value:

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

Definition at line 113 of file [KeyValidator.hpp](#).

### 10.11.3.2 validEntryKeys

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

**Initial value:**

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

Definition at line 110 of file [KeyValidator.hpp](#).

### 10.11.3.3 validKeys

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

**Initial value:**

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

Definition at line 108 of file [KeyValidator.hpp](#).

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

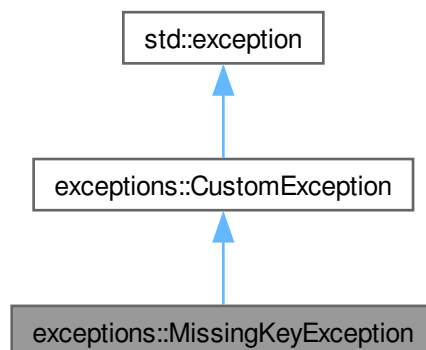
- [src/include/KeyValidator.hpp](#)
- [src/sources/KeyValidator.cpp](#)

## 10.12 exceptions::MissingKeyException Class Reference

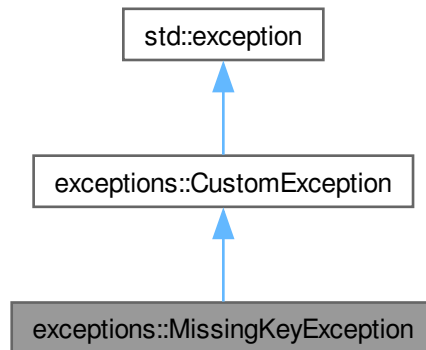
Exception for missing keys within entries.

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

Inheritance diagram for exceptions::MissingKeyException:



Collaboration diagram for exceptions::MissingKeyException:



### Public Member Functions

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

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

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

### Private Attributes

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

## 10.12.1 Detailed Description

Exception for missing keys within entries.

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

Definition at line 185 of file [Exceptions.hpp](#).

## 10.12.2 Constructor & Destructor Documentation

### 10.12.2.1 MissingKeyException()

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

#### Note

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

Definition at line 192 of file [Exceptions.hpp](#).

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

## 10.12.3 Member Function Documentation

### 10.12.3.1 what()

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

Definition at line 204 of file [Exceptions.hpp](#).

References [message](#).

## 10.12.4 Member Data Documentation

### 10.12.4.1 key

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

Definition at line 189 of file [Exceptions.hpp](#).

### 10.12.4.2 message

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

Definition at line 187 of file [Exceptions.hpp](#).

### 10.12.4.3 type

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

Definition at line 188 of file [Exceptions.hpp](#).

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

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

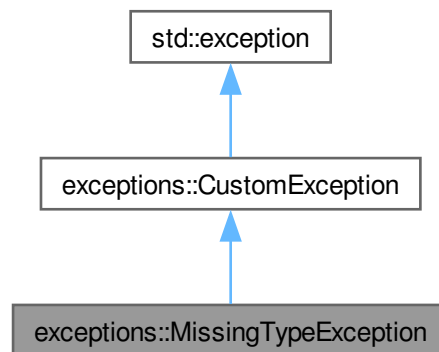


## 10.13 exceptions::MissingTypeException Class Reference

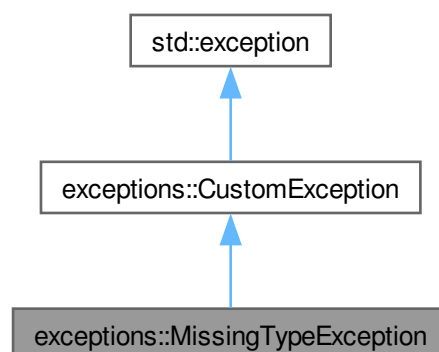
Exception for missing types of entries.

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

Inheritance diagram for exceptions::MissingTypeException:



Collaboration diagram for exceptions::MissingTypeException:



### Public Member Functions

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

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

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

## Private Attributes

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

### 10.13.1 Detailed Description

Exception for missing types of entries.

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

Definition at line [215](#) of file [Exceptions.hpp](#).

### 10.13.2 Constructor & Destructor Documentation

#### 10.13.2.1 `MissingTypeException()`

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

Definition at line [220](#) of file [Exceptions.hpp](#).

References [message](#).

### 10.13.3 Member Function Documentation

#### 10.13.3.1 `what()`

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

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

References [message](#).

### 10.13.4 Member Data Documentation

#### 10.13.4.1 `message`

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

Definition at line [217](#) of file [Exceptions.hpp](#).

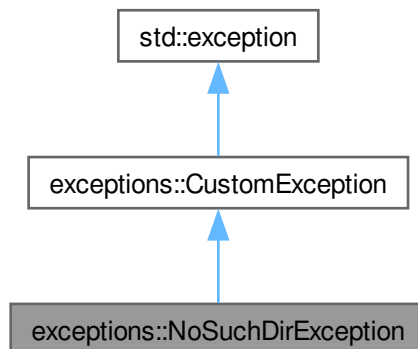
The documentation for this class was generated from the following file:

- `src/include/Exceptions.hpp`

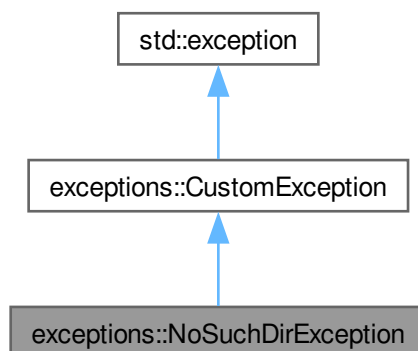
## 10.14 exceptions::NoSuchDirException Class Reference

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

Inheritance diagram for exceptions::NoSuchDirException:



Collaboration diagram for exceptions::NoSuchDirException:



### Public Member Functions

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

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

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

### Private Attributes

- `std::string` [message](#)

## 10.14.1 Detailed Description

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

## 10.14.2 Constructor & Destructor Documentation

### 10.14.2.1 NoSuchDirException()

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

[Todo](#) Documentation

Definition at line [267](#) of file [Exceptions.hpp](#).

References [message](#).

## 10.14.3 Member Function Documentation

### 10.14.3.1 what()

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

Definition at line [271](#) of file [Exceptions.hpp](#).

References [message](#).

## 10.14.4 Member Data Documentation

### 10.14.4.1 message

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

Definition at line [263](#) of file [Exceptions.hpp](#).

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

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

## 10.15 options Struct Reference

The struct containing all possible options.

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

### 10.15.1 Detailed Description

The struct containing all possible options.

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

See also

CommandLineHandler

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

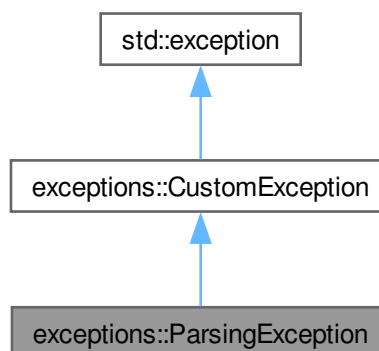
- [src/include/CommandLineHandler.hpp](#)

## 10.16 exceptions::ParsingException Class Reference

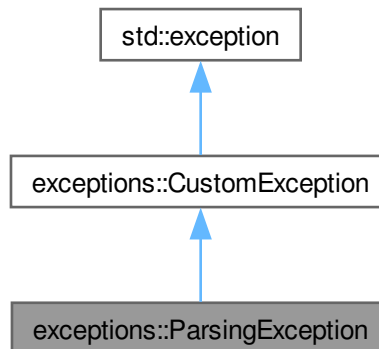
Exception for syntax errors within the json file.

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

Inheritance diagram for exceptions::ParsingException:



Collaboration diagram for exceptions::ParsingException:



#### Public Member Functions

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

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

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

#### Private Attributes

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

### 10.16.1 Detailed Description

Exception for syntax errors within the json file.

Definition at line 41 of file [Exceptions.hpp](#).

### 10.16.2 Constructor & Destructor Documentation

#### 10.16.2.1 ParsingException()

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

#### Note

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

Definition at line 47 of file [Exceptions.hpp](#).

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

### 10.16.3 Member Function Documentation

#### 10.16.3.1 what()

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

Definition at line 60 of file [Exceptions.hpp](#).

References [message](#).

### 10.16.4 Member Data Documentation

#### 10.16.4.1 file

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

Definition at line 43 of file [Exceptions.hpp](#).

#### 10.16.4.2 message

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

Definition at line 44 of file [Exceptions.hpp](#).

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

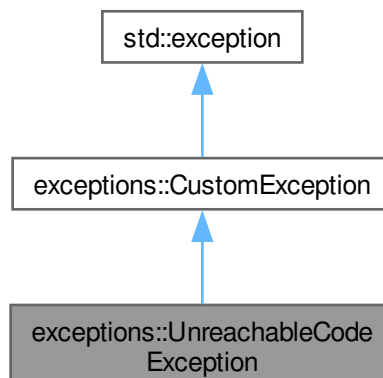
- [src/include/Exceptions.hpp](#)

## 10.17 exceptions::UnreachableCodeException Class Reference

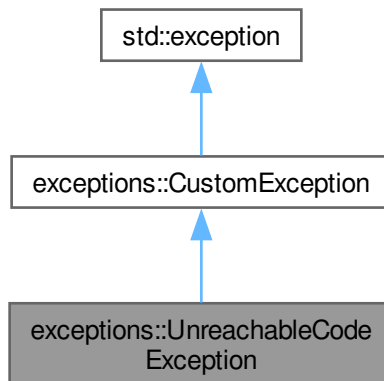
Exception for when the application reaches code it shouldn't reach.

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

Inheritance diagram for exceptions::UnreachableCodeException:



Collaboration diagram for exceptions::UnreachableCodeException:



#### Public Member Functions

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

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

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

#### Private Attributes

- std::string [message](#)

### 10.17.1 Detailed Description

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

Definition at line 232 of file [Exceptions.hpp](#).

### 10.17.2 Constructor & Destructor Documentation

#### 10.17.2.1 UnreachableCodeException()

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

Definition at line 237 of file [Exceptions.hpp](#).

References [message](#).



## 10.17.3 Member Function Documentation

### 10.17.3.1 what()

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

Definition at line 241 of file [Exceptions.hpp](#).

References [message](#).

## 10.17.4 Member Data Documentation

### 10.17.4.1 message

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

Definition at line 234 of file [Exceptions.hpp](#).

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

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

## 10.18 utilities::Utils Class Reference

Responsible for utility function.

```
#include <Utils.hpp>
```

### Static Public Member Functions

- static void [setupEasyLogging](#) (const std::string &configFile)  
*Set up easylogging.*
- static bool [handleParseException](#) (const exceptions::CustomException &e, const std::vector< std::string >↵  
::iterator &file, const std::vector< std::string > &files)
- 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 std::string & [checkDirectory](#) (std::string &directory)

### 10.18.1 Detailed Description

Responsible for utility function.

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

Definition at line 40 of file [Utils.hpp](#).

## 10.18.2 Member Function Documentation

### 10.18.2.1 askToContinue()

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

Asks if the user wants to continue.

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

**Parameters**

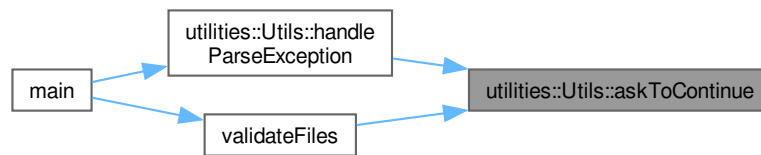
<i>prompt</i>	(Optional) A custom prompt to be used.
---------------	--

**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:

**10.18.2.2 checkConfigFile()**

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

Checks if the easylogging-config file exists.

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

Here is the caller graph for this function:



### 10.18.2.3 checkDirectory()

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

[Todo](#) documentation

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

Here is the caller graph for this function:



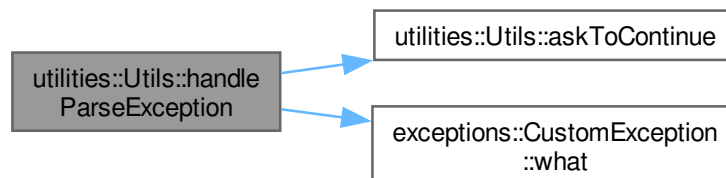
### 10.18.2.4 handleParseException()

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

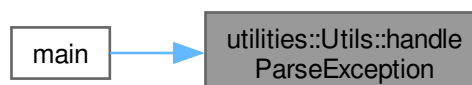
Definition at line 76 of file [Utils.cpp](#).

References [askToContinue\(\)](#), and [exceptions::CustomException::what\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.18.2.5 setupEasyLogging()

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

Set up easylogging.

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

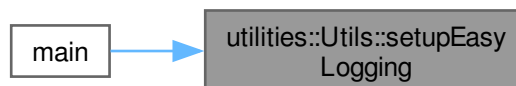
#### Parameters

<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 11

## File Documentation

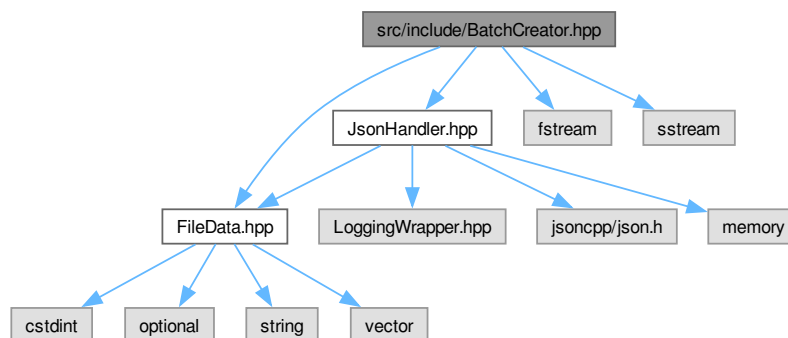
### 11.1 README.md File Reference

### 11.2 src/include/BatchCreator.hpp File Reference

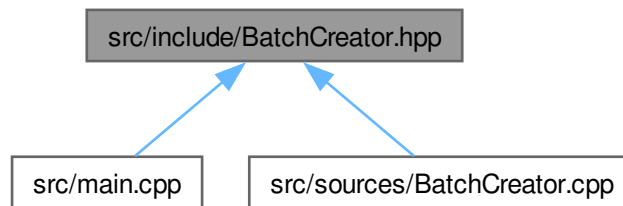
Creates batch file.

```
#include "FileData.hpp"  
#include "JsonHandler.hpp"  
#include <fstream>  
#include <sstream>
```

Include dependency graph for BatchCreator.hpp:



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



## Classes

- class [BatchCreator](#)  
*Erstellt Batch Datei.*

### 11.2.1 Detailed Description

Creates batch file.

#### Author

Maximilian Rodler

#### Date

22.04.2024

#### Version

#### Copyright

See LICENSE file

**Todo** Documentation in english

#### Author

Maximilian Rodler

#### Date

22.04.2024

#### Version

Creates batch file from Arguments in JSON

#### Copyright

See LICENSE file

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

## 11.3 BatchCreator.hpp

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

```

00001
00014 #include "FileData.hpp"
00015 #include "JsonHandler.hpp"
00016 #include <fstream>
00017 #include <sstream>
00018
00027 class BatchCreator {
00028 public:
00036     explicit BatchCreator(std::shared_ptr<parsing::FileData> fileData);
00037
00039     [[nodiscard]] std::shared_ptr<std::stringstream> getDataStream() const {
00040         return dataStream;
00041     }
00042
00043 private:
00044     std::shared_ptr<std::stringstream> dataStream;
00045
00046     std::shared_ptr<parsing::FileData> fileData;
00047
00053     void createBatch();
00054
00062     void writeStart() const;
00063
00069     void writeHideShell() const;
00070
00077     void writeCommands() const;
00078
00085     void writeEnvVariables() const;
00086
00093     void writePathVariables() const;
00094
00101     void writeApp() const;
00102
00109     void writeEnd() const;
00110 };

```

## 11.4 src/include/CommandLineHandler.hpp File Reference

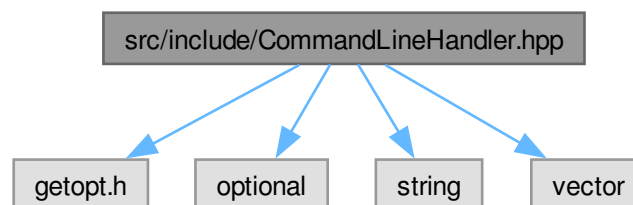
Responsible for the Command Line Interface.

```

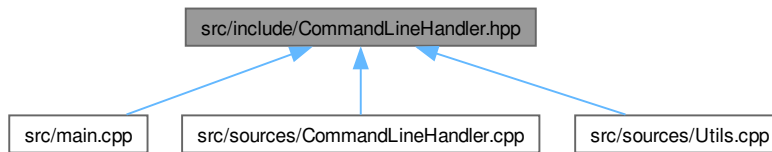
#include <getopt.h>
#include <optional>
#include <string>
#include <vector>

```

Include dependency graph for CommandLineHandler.hpp:



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



## Classes

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

## Namespaces

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

## Variables

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

### 11.4.1 Detailed Description

Responsible for the Command Line Interface.

#### Author

Simon Blum

#### Date

2024-04-18

#### Version

0.1.5

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](#)

#### Copyright

See LICENSE file

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



## 11.5 CommandLineHandler.hpp

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

```

00001
00019 #ifndef COMMANDLINEHANDLER_HPP
00020 #define COMMANDLINEHANDLER_HPP
00021
00022 #include <getopt.h>
00023 #include <optional>
00024 #include <string>
00025 #include <vector>
00026
00039 namespace cli {
00040
00053 class CommandLineHandler {
00054 public:
00062     [[noreturn]] static void printHelp();
00070     [[noreturn]] static void printVersion();
00078     [[noreturn]] static void printCredits();
00092     static std::tuple<std::optional<std::string>, std::vector<std::string>
00093     parseArguments(int argc, char *argv[]);
00099     CommandLineHandler() = delete;
00105     ~CommandLineHandler() = delete;
00106 };
00107
00117 static const struct option options[] = {
00118     {"help", no_argument, nullptr, 'h'},
00119     {"version", no_argument, nullptr, 'v'},
00120     {"credits", no_argument, nullptr, 'c'},
00121     {"verbose", no_argument, nullptr, 0},
00122     {"outdir", required_argument, nullptr, 'o'},
00123     nullptr
00124     // Brief/verbose
00125     // Output dir
00126 };
00127
00139 #ifdef IS_UNIX // CLI Formatting for Linux
00140 static const std::string CLEAR_TERMINAL = "\033[2J\033[1H";
00141 static const std::string RESET = "\033[0m";
00142 static const std::string RED = "\033[0;31m";
00143 static const std::string GREEN = "\033[0;32m";
00144 static const std::string YELLOW = "\033[0;33m";
00145 static const std::string BLUE = "\033[0;34m";
00146 static const std::string MAGENTA = "\033[0;35m";
00147 static const std::string CYAN = "\033[0;36m";
00148 static const std::string WHITE = "\033[0;37m";
00149 static const std::string BOLD = "\033[1m";
00150 static const std::string UNDERLINE = "\033[4m";
00151 static const std::string ITALIC = "\033[3m";
00152 #elif defined(
00153     IS_WINDOWS) // Windows doesn't support ANSI escape codes the same way
00154 static const std::string CLEAR_TERMINAL = "";
00155 static const std::string RESET = "";
00156 static const std::string RED = "";
00157 static const std::string GREEN = "";
00158 static const std::string YELLOW = "";
00159 static const std::string BLUE = "";
00160 static const std::string MAGENTA = "";
00161 static const std::string CYAN = "";
00162 static const std::string WHITE = "";
00163 static const std::string BOLD = "";
00164 static const std::string UNDERLINE = "";
00165 static const std::string ITALIC = "";
00166 #endif
00168 // end of group StyleHelpers
00169 } // namespace cli
00170
00171 #endif // COMMANDLINEHANDLER_HPP

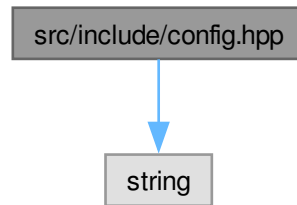
```

## 11.6 src/include/config.hpp File Reference

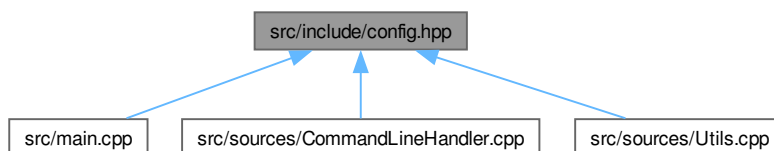
Configures general project information.

```
#include <string>
```

Include dependency graph for config.hpp:



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



## Namespaces

- namespace [config](#)

## Variables

- constexpr auto [config::LOG\\_CONFIG](#)
- constexpr auto [config::EXECUTABLE\\_NAME](#) = "json2batch"
- constexpr auto [config::MAJOR\\_VERSION](#) = "0"
- constexpr auto [config::MINOR\\_VERSION](#) = "2"
- constexpr auto [config::PATCH\\_VERSION](#) = "2"
- constexpr auto [config::DESCRIPTION](#) = "A simple tool to convert json to batch."
- constexpr auto [config::PROJECT\\_NAME](#) = "JSON2Batch"
- constexpr auto [config::AUTHORS](#)
- constexpr auto [config::HOMEPAGE\\_URL](#)

## 11.6.1 Detailed Description

Configures general project information.

### Author

Simon Blum

### Date

2024-04-18

### Version

0.1.5

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

### Note

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

### Copyright

See LICENSE file

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

## 11.7 config.hpp

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

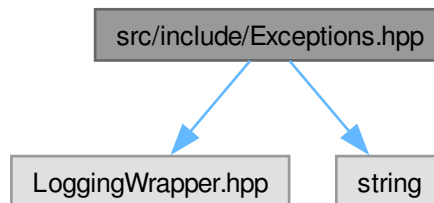
```
00001
00016 // This file is autogenerated. Changes will be overwritten
00017
00018 #ifndef CONFIG_HPP
00019 #define CONFIG_HPP
00020
00021 #include <string>
00022
00023 namespace config {
00024 inline constexpr auto LOG_CONFIG = "/home/simon/l_Coding/projectJsonToBat/"
00025                                     "build/Release/config/easylogging.conf";
00026 inline constexpr auto EXECUTABLE_NAME = "json2batch";
00027 inline constexpr auto MAJOR_VERSION = "0";
00028 inline constexpr auto MINOR_VERSION = "2";
00029 inline constexpr auto PATCH_VERSION = "2";
00030 inline constexpr auto DESCRIPTION = "A simple tool to convert json to batch.";
00031 inline constexpr auto PROJECT_NAME = "JSON2Batch";
00032 inline constexpr auto AUTHORS =
00033     "Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci";
00034 inline constexpr auto HOMEPAGE_URL =
00035     "https://dhwprojectsit23.github.io/JSON2Bat";
00036 } // namespace config
00037
00038 #endif
```

## 11.8 src/include/Exceptions.hpp File Reference

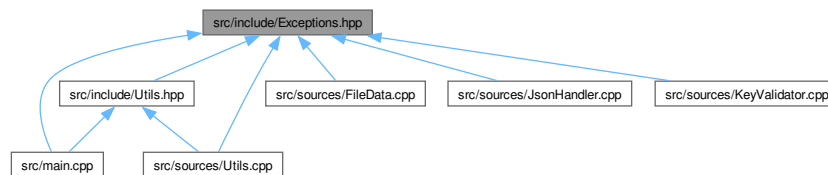
Contains all the custom exceptions used in the project.

```
#include "LoggingWrapper.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](#)
- class [exceptions::NoSuchDirException](#)

## Namespaces

- namespace [exceptions](#)

*Namespace used for customized exceptions.*

### 11.8.1 Detailed Description

Contains all the custom exceptions used in the project.

#### Author

Simon Blum

#### Date

23.04.2024

#### Version

0.1.6

#### Copyright

See LICENSE file

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

## 11.9 Exceptions.hpp

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

```
00001
00010 #ifndef EXCEPTIONS_HPP
00011 #define EXCEPTIONS_HPP
00012
00013 #include "LoggingWrapper.hpp"
00014 #include <string>
00015
00020 namespace exceptions {
00030 class CustomException : public std::exception {
00031 public:
00032     [[nodiscard]] const char *what() const noexcept override {
00033         return "Base Exception";
00034     }
00035 };
00036
00041 class ParsingException : public CustomException {
00042 private:
00043     const std::string file;
00044     std::string message;
00045
00046 public:
00047     explicit ParsingException(const std::string &file) : file(file) {
00053         std::stringstream ss;
00054         ss << "Error while trying to parse \"" << file << "\"!\n"
00055             << "There most likely is a syntax error within the \".json\" file.";
00056         this->message = ss.str();
00057         LOG_INFO << "ParsingException: " << message;
00058     }
00059
00060     [[nodiscard]] const char *what() const noexcept override {
00061         return message.c_str();
00062     }
00063 }
```

```

00062     }
00063 };
00064
00069 class FileExistsException : public CustomException {
00070 private:
00071     const std::string file;
00072     std::string message;
00073
00074 public:
00075     explicit FileExistsException(const std::string &file) : file(file) {
00081         std::stringstream ss;
00082         ss << "The outputfile \"" << file << "\" already exists!";
00083         this->message = ss.str();
00084         LOG_INFO << "BatchExistsException: " << message;
00085     }
00086
00087     [[nodiscard]] const char *what() const noexcept override {
00088         return message.c_str();
00089     }
00090 };
00091
00096 class InvalidValueException : public CustomException {
00097 private:
00098     const std::string key;
00099     std::string message;
00100
00101 public:
00102     InvalidValueException(const std::string &key, const std::string &issue)
00103         : key(key) {
00109         std::stringstream ss;
00110         ss << "Error at key \"" << key << "\"! " << issue;
00111         this->message = ss.str();
00112         LOG_INFO << "InvalidValueException: " << message;
00113     }
00114     [[nodiscard]] const char *what() const noexcept override {
00115         return message.c_str();
00116     }
00117 };
00118
00130 class InvalidKeyException : public CustomException {
00131 private:
00132     std::string message = "Invalid key found!";
00133
00134 public:
00135     explicit InvalidKeyException(
00136         const std::vector<std::tuple<int, std::string>> &keys) {
00137         LOG_INFO << "InvalidKeyException: " << message;
00138         for (const auto &[line, key] : keys) {
00139             LOG_WARNING << "Invalid key found at line " << line << ": \"" << key
00140                 << "\"!";
00141         }
00142     }
00143     [[nodiscard]] const char *what() const noexcept override {
00144         return message.c_str();
00145     }
00146 };
00147
00156 class InvalidTypeException : public CustomException {
00157 private:
00158     const std::string type;
00159     std::string message;
00160
00161 public:
00162     InvalidTypeException(const std::string &type, int line) : type(type) {
00168         std::stringstream ss;
00169         ss << "Invalid type found at line " << line << ": \"" << type << "\"";
00170         this->message = ss.str();
00171         LOG_INFO << "InvalidTypeException: " << message;
00172     }
00173     [[nodiscard]] const char *what() const noexcept override {
00174         return message.c_str();
00175     }
00176 };
00177
00185 class MissingKeyException : public CustomException {
00186 private:
00187     std::string message;
00188     std::string type;
00189     std::string key;
00190
00191 public:
00192     MissingKeyException(const std::string &key, const std::string &type)
00193         : type(type), key(key) {
00199         std::stringstream ss;
00200         ss << "Missing key \"" << key << "\" for type \"" << type << "\"!";
00201         this->message = ss.str();
00202         LOG_INFO << "MissingKeyException: " << message;

```

```

00203     }
00204     [[nodiscard]] const char *what() const noexcept override {
00205         return message.c_str();
00206     }
00207 };
00208
00215 class MissingTypeException : public CustomException {
00216 private:
00217     std::string message = "Missing \"type\" key for at least one entry!";
00218 public:
00219     MissingTypeException() {
00220         LOG_INFO << "MissingTypeException: " << message;
00221     }
00222     [[nodiscard]] const char *what() const noexcept override {
00223         return message.c_str();
00224     }
00225 };
00226
00232 class UnreachableCodeException : public CustomException {
00233 private:
00234     std::string message;
00235 public:
00236     explicit UnreachableCodeException(const std::string &message)
00237         : message(message) {
00238         LOG_INFO << "UnreachableCodeException: " << message;
00239     }
00240     [[nodiscard]] const char *what() const noexcept override {
00241         return message.c_str();
00242     }
00243 };
00244
00246 class FailedToOpenFileException : public CustomException {
00247 private:
00248     std::string message;
00249 public:
00250     explicit FailedToOpenFileException(const std::string &file) {
00251         message = "Failed to open file: " + file;
00252         LOG_INFO << "FailedToOpenFileException: " << message;
00253     }
00254     [[nodiscard]] const char *what() const noexcept override {
00255         return message.c_str();
00256     }
00257 };
00258
00261 class NoSuchDirException : public CustomException {
00262 private:
00263     std::string message;
00264 public:
00265     explicit NoSuchDirException(const std::string &dir) {
00266         message = "No such directory: " + dir;
00267         LOG_INFO << "NoSuchDirException: " << message;
00268     }
00269     [[nodiscard]] const char *what() const noexcept override {
00270         return message.c_str();
00271     }
00272 };
00273
00276 } // namespace exceptions
00277
00278 #endif

```

## 11.10 src/include/FileData.hpp File Reference

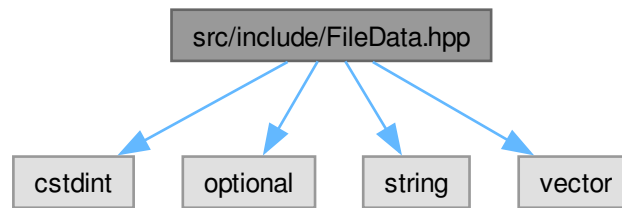
This file contains the FileData class.

```

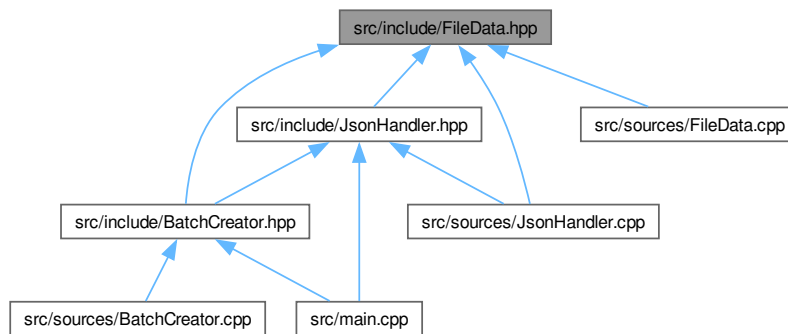
#include <cstdint>
#include <optional>
#include <string>
#include <vector>

```

Include dependency graph for `FileData.hpp`:



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



## Classes

- class [parsing::FileData](#)  
*This class contains all data from the json file.*

## Namespaces

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

### 11.10.1 Detailed Description

This file contains the `FileData` class.

#### Author

Sonia Sinacci, Elena Schwartzbach



**Date**

16.04.2024

**Version**

0.1.5

**See also**[parsing::FileData](#)**Copyright**

See LICENSE file

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

## 11.11 FileData.hpp

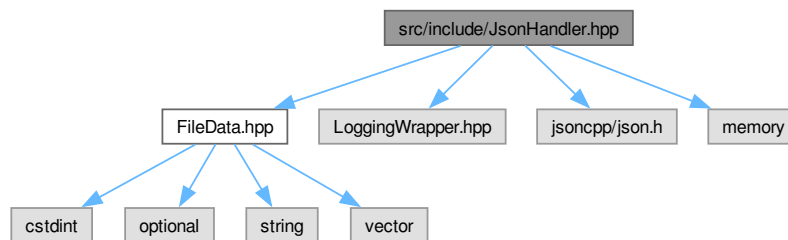
[Go to the documentation of this file.](#)

```
00001
00013 #ifndef FILEDATA_HPP
00014 #define FILEDATA_HPP
00015
00016 #include <cstdint>
00017 #include <optional>
00018 #include <string>
00019 #include <vector>
00020
00021 namespace parsing {
00030 class FileData {
00031 public:
00042     void setOutputFile(std::string &newOutputfile);
00043
00048     void setHideShell(bool newHideShell) { this->hideShell = newHideShell; }
00049
00058     void setApplication(const std::string &newApplication);
00059
00070     void addCommand(const std::string &command);
00071
00083     void addEnvironmentVariable(const std::string &name,
00084                                 const std::string &value);
00085
00096     void addPathValue(const std::string &pathValue);
00097
00102     [[nodiscard]] const std::string &getOutputFile() const { return outputfile; }
00103
00108     [[nodiscard]] bool getHideShell() const { return hideShell; }
00109
00114     [[nodiscard]] const std::optional<std::string> &getApplication() const {
00115         return application;
00116     }
00117
00122     [[nodiscard]] const std::vector<std::string> &getCommands() const {
00123         return commands;
00124     }
00125
00130     [[nodiscard]] const std::vector<std::tuple<std::string, std::string> &
00131     getEnvironmentVariables() const {
00132         return environmentVariables;
00133     }
00134
00139     [[nodiscard]] const std::vector<std::string> &getPathValues() const {
00140         return pathValues;
00141     }
00142
00143 private:
00144     std::string outputfile;
00145     bool hideShell;
00146     std::optional<std::string> application;
00147     std::vector<std::string> commands;
00148     std::vector<std::tuple<std::string, std::string> > environmentVariables;
00149     std::vector<std::string> pathValues;
00150 };
00151 } // namespace parsing
00152
00153 #endif // FILEDATA_HPP
```

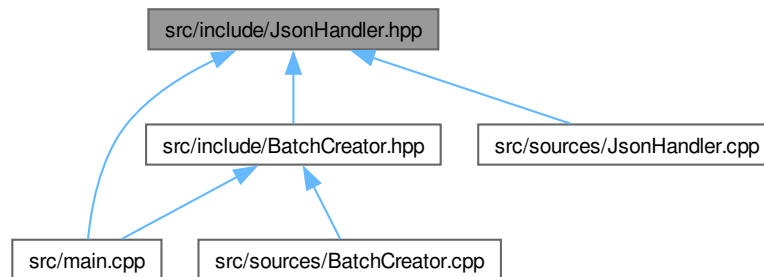
## 11.12 src/include/JsonHandler.hpp File Reference

This file contains the JsonHandler class.

```
#include "FileData.hpp"
#include "LoggingWrapper.hpp"
#include <jsoncpp/json.h>
#include <memory>
Include dependency graph for JsonHandler.hpp:
```



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



### Classes

- class `parsing::JsonHandler`  
*This file reads all data from the json file.*

### Namespaces

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

### 11.12.1 Detailed Description

This file contains the JsonHandler class.

#### Author

Sonia Sinacci, Elena Schwartzbach

#### Date

23.04.2024

#### Version

0.1.5

#### See also

[parsing::JsonHandler](#)

#### Copyright

See LICENSE file

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

## 11.13 JsonHandler.hpp

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

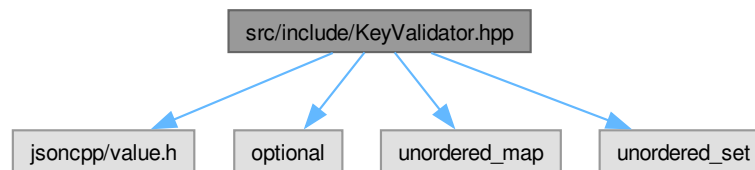
```
00001
00013 #ifndef JSONHANDLER_HPP
00014 #define JSONHANDLER_HPP
00015
00016 #include "FileData.hpp"
00017 #include "LoggingWrapper.hpp"
00018 #include <jsoncpp/json.h>
00019
00020 #include <memory>
00021
00034 namespace parsing {
00035
00045 class JsonHandler {
00046 public:
00053     JsonHandler() {
00054         LOG_INFO « "Initialising empty JsonHandler";
00055     }
00063     explicit JsonHandler(const std::string &filename);
00073     std::shared_ptr<FileData> getFileData();
00074
00075 private:
00091     [[nodiscard]] static std::shared_ptr<Json::Value>
00092     parseFile(const std::string &filename);
00101     void assignOutputFile() const;
00108     void assignHideShell() const;
00115     void assignApplication() const;
00127     void assignEntries() const;
00132     void assignCommand(const Json::Value &entry) const;
00137     void assignEnvironmentVariable(const Json::Value &entry) const;
00142     void assignPathValue(const Json::Value &entry) const;
00151     std::shared_ptr<FileData> createFileData();
00152     std::shared_ptr<Json::Value> root;
00153     std::shared_ptr<FileData> data;
00154 };
00155 } // namespace parsing
00156
00157 #endif // JSONHANDLER_HPP
```

## 11.14 src/include/KeyValidator.hpp File Reference

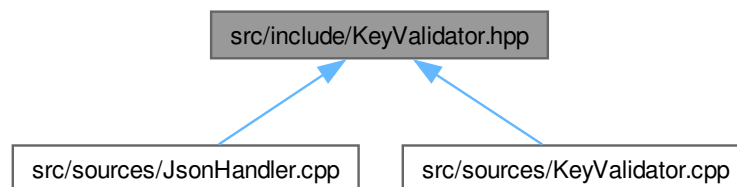
This file contains the KeyValidator class.

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

Include dependency graph for KeyValidator.hpp:



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



### Classes

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

### Namespaces

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

### 11.14.1 Detailed Description

This file contains the KeyValidator class.

#### Author

Simon Blum

#### Date

21.04.2024

#### Version

0.1.6

#### See also

[parsing::KeyValidator](#)

#### Copyright

See LICENSE file

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

## 11.15 KeyValidator.hpp

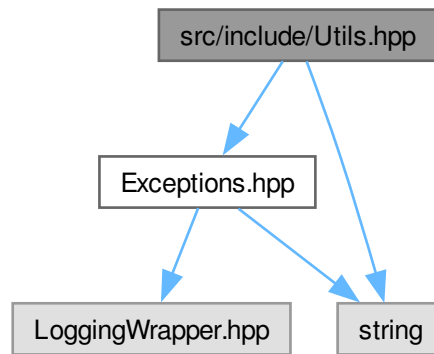
[Go to the documentation of this file.](#)

```
00001
00012 #ifndef KEYVALIDATOR_HPP
00013 #define KEYVALIDATOR_HPP
00014
00015 #include "jsoncpp/value.h"
00016 #include <optional>
00017 #include <unordered_map>
00018 #include <unordered_set>
00019 namespace parsing {
00020 class KeyValidator {
00021 public:
00022     static KeyValidator &getInstance();
00023
00024     std::vector<std::tuple<int, std::string>
00025     validateKeys(const Json::Value &root, const std::string &filename);
00026
00027 private:
00028     std::vector<std::tuple<int, std::string>
00029     getWrongKeys(const Json::Value &root, const std::string &filename) const;
00030
00031     void validateTypes(const std::string &filename, const Json::Value &entry,
00032         const std::unordered_set<std::string> &entryKeys);
00033
00034     std::vector<std::tuple<int, std::string>
00035     validateEntries(const std::string &filename,
00036         const std::unordered_set<std::string> &entryKeys) const;
00037
00038     static std::optional<int> getUnknownKeyLine(const std::string &filename,
00039         const std::string &wrongKey);
00040
00041     std::unordered_set<std::string> validKeys = {"outputfile", "hideshell",
00042         "entries", "application"};
00043     std::unordered_set<std::string> validEntryKeys = {"type", "key", "value",
00044         "path", "command"};
00045
00046     std::unordered_map<std::string_view, std::vector<std::string> typeToKeys = {
00047         {"EXE", {"command"}}, {"PATH", {"path"}}, {"ENV", {"key", "value"}}};
00048 };
00049 } // namespace parsing
00050 #endif
```

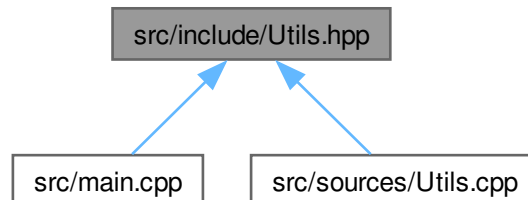
## 11.16 src/include/Utils.hpp File Reference

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

Include dependency graph for Utils.hpp:



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



### Classes

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

### Namespaces

- namespace `utilities`  
*Includes all utilities.*

## 11.17 Utils.hpp

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

```

00001
00016 #ifndef UTILITIES_HPP
00017 #define UTILITIES_HPP
00018
00019 #include "Exceptions.hpp"
00020 #include <string>
00021
00031 namespace utilities {
00032
00040 class Utils {
00041 public:
00049     static void setupEasyLogging(const std::string &configFile);
00050
00051     static bool
00052     handleParseException(const exceptions::CustomException &e,
00053                          const std::vector<std::string>::iterator &file,
00054                          const std::vector<std::string> &files);
00055
00063     static bool
00064     askToContinue(const std::string &prompt = "Do you want to continue? (Y/N)\n");
00065
00069     static void checkConfigFile(const std::string &configFile);
00070
00072     static std::string &checkDirectory(std::string &directory);
00073 };
00074 } // namespace utilities
00075
00076 #endif // UTILITIES_HPP

```

## 11.18 src/main.cpp File Reference

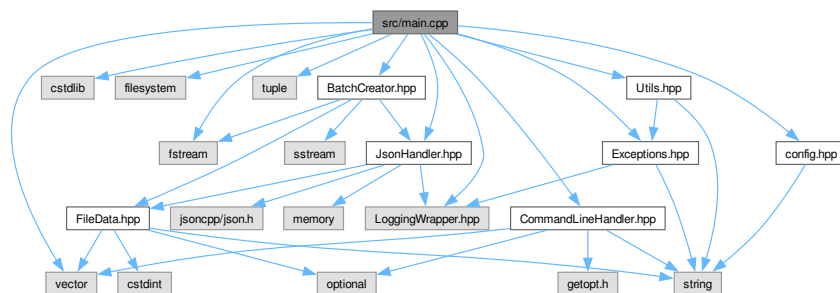
Contains the main function.

```

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

```

Include dependency graph for main.cpp:



## Functions

- INITIALIZE\_EASYLOGGINGPP std::tuple< std::vector< std::string >, std::string > [parseAndValidateArgs](#) (int argc, char \*argv[])
- std::vector< std::string > [validateFiles](#) (const std::vector< std::string > &files)
- void [parseFile](#) (const std::string &file, const std::string &outputDirectory)
- int [main](#) (int argc, char \*argv[])  
*Main function of the program.*

### 11.18.1 Detailed Description

Contains the main function.

#### Author

Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci

#### Date

2024-04-18

#### Version

0.1.5

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

#### Copyright

See LICENSE file

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

### 11.18.2 Function Documentation

#### 11.18.2.1 main()

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

Main function of the program.

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



## Parameters

<i>argc</i>	The number of arguments given
<i>argv</i>	Th command line arguments given

## Returns

Returns 0 on success, 1 on failure

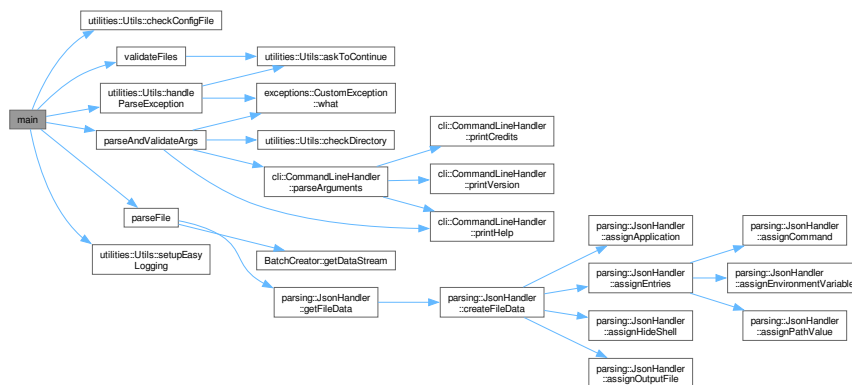
**Todo** Documentation

Refactoring

Definition at line 72 of file [main.cpp](#).

References [utilities::Utils::checkConfigFile\(\)](#), [utilities::Utils::handleParseException\(\)](#), [config::LOG\\_CONFIG](#), [parseAndValidateArgs\(\)](#), [parseFile\(\)](#), [utilities::Utils::setupEasyLogging\(\)](#), and [validateFiles\(\)](#).

Here is the call graph for this function:



## 11.18.2.2 parseAndValidateArgs()

```

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

```

## Parameters

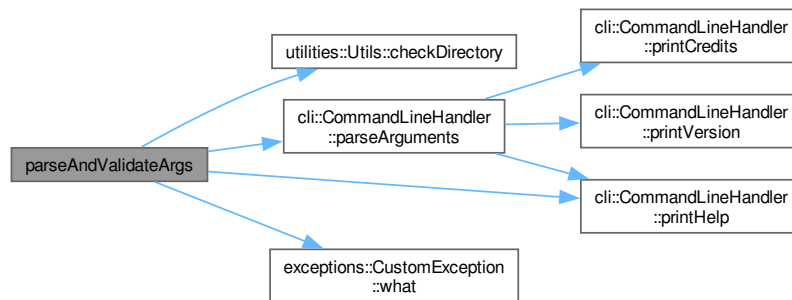
<i>argc</i>	
<i>argv</i>	

## Returns

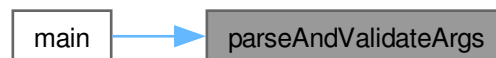
Definition at line 100 of file [main.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 11.18.2.3 parseFile()

```

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

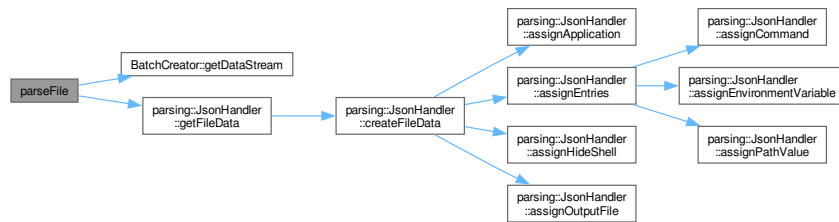
#### Parameters

<i>file</i>	
-------------	--

Definition at line 150 of file [main.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 11.18.2.4 validateFiles()

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

##### Parameters

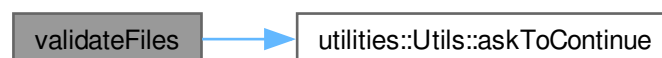
<i>files</i>	
--------------	--

##### Returns

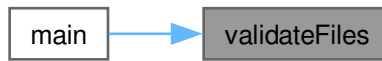
Definition at line 122 of file [main.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



## 11.19 main.cpp

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

```

00001
00013 #include <LoggingWrapper.hpp>
00014 #include <cstdlib>
00015 #include <filesystem>
00016 #include <fstream>
00017 #include <tuple>
00018 #include <vector>
00019
00020 #include "BatchCreator.hpp"
00021 #include "CommandLineHandler.hpp"
00022 #include "Exceptions.hpp"
00023 #include "JsonHandler.hpp"
00024 #include "Utils.hpp"
00025 #include "config.hpp"
00026
00027 INITIALIZE_EASYLOGGINGPP
00028
00037 std::tuple<std::vector<std::string>, std::string>
00038 parseAndValidateArgs(int argc, char *argv[]);
00039
00047 std::vector<std::string> validateFiles(const std::vector<std::string> &files);
00048
00055 void parseFile(const std::string &file, const std::string &outputDirectory);
00056
00072 int main(int argc, char *argv[]) {
00073
00074     utilities::Utils::checkConfigFile(config::LOG_CONFIG);
00075     utilities::Utils::setupEasyLogging(config::LOG_CONFIG);
00076     auto [files, outDir] = parseAndValidateArgs(argc, argv);
00077     OUTPUT « cli::BOLD « "Parsing the following files:\n" « cli::RESET;
00078     for (const auto &file : files) {
00079         OUTPUT « "\t - " « file « "\n";
00080     }
00081
00082     files = validateFiles(files);
00083     for (auto file = files.begin(); file != files.end(); ++file) {
00084         OUTPUT « cli::ITALIC « "\nParsing file: " « *file « "... \n"
00085             « cli::RESET;
00086         try {
00087             parseFile(*file, outDir);
00088         } catch (const exceptions::CustomException &e) {
00089             if (utilities::Utils::handleParseException(e, file, files)) {
00090                 continue;
00091             }
00092             exit(1);
00093         }
00094     }
00095     LOG_INFO « "Exiting...";
00096     return 0;
00097 }
00098
00099 std::tuple<std::vector<std::string>, std::string>
00100 parseAndValidateArgs(int argc, char *argv[]) {
00101     if (argc < 2) {
00102         LOG_ERROR « "No options given!\n";
00103         cli::CommandLineHandler::printHelp();
00104     }
00105     auto [outOption, files] = cli::CommandLineHandler::parseArguments(argc, argv);
00106     std::string outDir = outOption.value_or("");
00107     if (!outDir.empty()) {
00108         try {

```

```

00109     outDir = utilities::Utils::checkDirectory(outDir);
00110 } catch (const exceptions::CustomException &e) {
00111     LOG_ERROR << e.what();
00112     exit(1);
00113 }
00114 }
00115 if (files.empty()) {
00116     LOG_ERROR << "No files were given as arguments!\n";
00117     exit(1);
00118 }
00119 return {files, outDir};
00120 }
00121
00122 std::vector<std::string> validateFiles(const std::vector<std::string> &files) {
00123     std::vector<std::string> validFiles;
00124     validFiles.reserve(files.size());
00125     for (const std::filesystem::path file : files) {
00126         if (!std::filesystem::is_regular_file(file)) {
00127             LOG_ERROR << "The file \"" << file << "\" does not exist!\n";
00128             if (files.size() > 1 && !utilities::Utils::askToContinue()) {
00129                 OUTPUT << "Aborting...\n";
00130                 LOG_INFO << "Application ended by user Input";
00131                 exit(1);
00132             }
00133             continue;
00134         }
00135         if (file.extension() != ".json") {
00136             LOG_WARNING << "The file \"" << file << "\" does not end in \".json\"\n";
00137             OUTPUT << "If the file is not in JSON Format, continuing may "
00138                 << "result in\nunexpected behaviour!\n";
00139             if (!utilities::Utils::askToContinue()) {
00140                 OUTPUT << "Aborting...\n";
00141                 LOG_INFO << "Application ended by user Input";
00142                 exit(1);
00143             }
00144         }
00145         validFiles.push_back(file);
00146     }
00147     return validFiles;
00148 }
00149
00150 void parseFile(const std::string &file, const std::string &outputDirectory) {
00151     parsing::JsonHandler jsonHandler(file);
00152     const auto fileData = jsonHandler.getFileData();
00153     BatchCreator batchCreator(fileData);
00154     const std::shared_ptr<std::stringstream> dataStream =
00155         batchCreator.getDataStream();
00156     const std::string outputFileName =
00157         outputDirectory + fileData->getOutputFile();
00158     std::ofstream outFile(outputFileName);
00159     if (!outFile.good()) {
00160         throw exceptions::FailedToOpenFileException(outputFileName);
00161     }
00162     outFile << dataStream->str();
00163     OUTPUT << "Done with files!\n";
00164 }

```

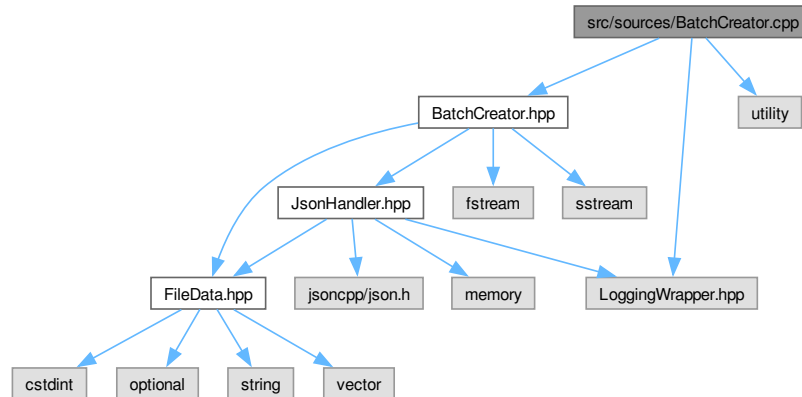
## 11.20 src/sources/BatchCreator.cpp File Reference

```

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

```

Include dependency graph for BatchCreator.cpp:



## 11.21 BatchCreator.cpp

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

```

00001
00012 #include "BatchCreator.hpp"
00013
00014 #include "LoggingWrapper.hpp"
00015 #include <utility>
00016
00017 BatchCreator::BatchCreator(std::shared_ptr<parsing::FileData> fileData)
00018 : fileData(std::move(fileData)) {
00019     LOG_INFO << "Initializing BatchCreator";
00020     this->dataStream = std::make_shared<std::stringstream>();
00021     this->createBatch();
00022 }
00023
00024 void BatchCreator::createBatch() {
00025     LOG_INFO << "Creating Batch file";
00026
00027     this->writeStart();
00028     this->writeHideShell();
00029     this->writeCommands();
00030     this->writeEnvVariables();
00031     this->writePathVariables();
00032     this->writeApp();
00033     this->writeEnd();
00034 }
00035
00036 void BatchCreator::writeStart() const {
00037     LOG_INFO << "writing Start of Batch";
00038     *this->dataStream << "@ECHO OFF\r\nC:\\Windows\\System32\\cmd.exe ";
00039 }
00040
00041 void BatchCreator::writeHideShell() const {
00042     if (this->fileData->getHideShell()) {
00043         LOG_INFO << "writing hide Shell";
00044         *this->dataStream << "/c ";
00045     } else {
00046         LOG_INFO << "writing show Shell";
00047         *this->dataStream << "/k ";
00048     }
00049 }
00050
00051
00052 void BatchCreator::writeCommands() const {
00053     LOG_INFO << "writing Commands";
00054     *this->dataStream << "\"";
00055     for (const std::string &command : this->fileData->getCommands()) {
00056         *this->dataStream << command << " && ";
00057     }
00058 }

```

```

00059
00060 void BatchCreator::writeEnvVariables() const {
00061     LOG_INFO << "writing Environment Variables";
00062     for (const auto &[key, value] : this->fileData->getEnvironmentVariables()) {
00063         *this->dataStream << "set " << key << "=" << value << " && ";
00064     }
00065 }
00066
00067 void BatchCreator::writePathVariables() const {
00068     LOG_INFO << "writing Path Variables";
00069     *this->dataStream << "set path=";
00070     for (const std::string &path : this->fileData->getPathValues()) {
00071         *this->dataStream << path << ";";
00072     }
00073     *this->dataStream << "%path%";
00074 }
00075
00076 void BatchCreator::writeApp() const {
00077     std::string appName = this->fileData->getOutputFile();
00078     appName = appName.substr(0, appName.find('.'));
00079     if (this->fileData->getApplication().has_value()) {
00080         LOG_INFO << "writing start Application";
00081         *this->dataStream << " && start \"" << appName << "\" "
00082             << this->fileData->getApplication().value() << "\"\r\n";
00083     } else {
00084         LOG_INFO << "writing not start Application";
00085         *this->dataStream << "\"\r\n";
00086     }
00087 }
00088
00089 void BatchCreator::writeEnd() const {
00090     *this->dataStream << "@ECHO ON";
00091 }

```

## 11.22 src/sources/CommandLineHandler.cpp File Reference

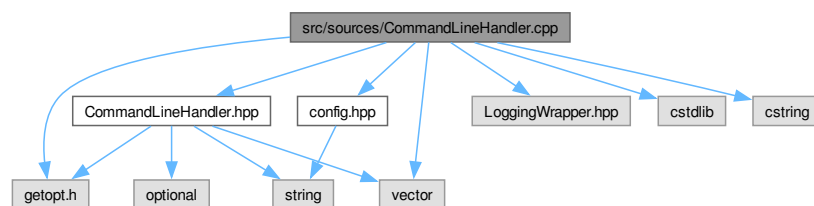
Implementation for the Command Line Interface.

```

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

```

Include dependency graph for CommandLineHandler.cpp:



### Namespaces

- namespace `cli`

*Includes everything regarding the CLI.*





```

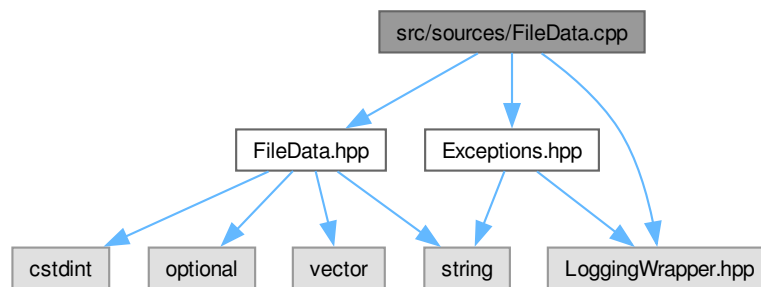
00046 OUTPUT « config::PROJECT_NAME « " v" « config::MAJOR_VERSION « "."
00047 « config::MINOR_VERSION « "." « config::PATCH_VERSION « "\n";
00048 exit(0);
00049 }
00050 void CommandLineHandler::printCredits() {
00051 LOG_INFO « "Printing credits...";
00052 OUTPUT « BOLD « "Project information:\n"
00053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

```

```
00133     return {outDir, files};  
00134 }  
00135 } // namespace cli
```

## 11.24 src/sources/FileData.cpp File Reference

```
#include "FileData.hpp"  
#include "Exceptions.hpp"  
#include "LoggingWrapper.hpp"  
Include dependency graph for FileData.cpp:
```



### Namespaces

- namespace [parsing](#)

*The namespace containing everything relevant to parsing.*

### 11.24.1 Detailed Description

Author

Date

Version

Copyright

See LICENSE file

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

## 11.25 FileData.cpp

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

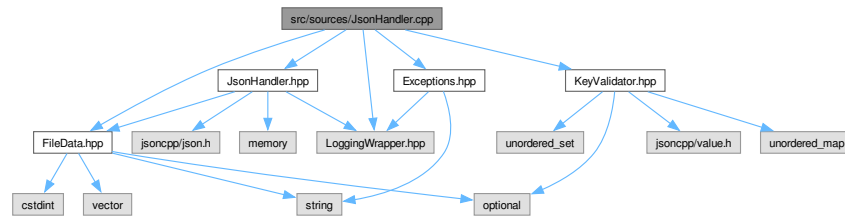
```

00001
00012 #include "FileData.hpp"
00013 #include "Exceptions.hpp"
00014 #include "LoggingWrapper.hpp"
00015
00016 namespace parsing {
00017 void FileData::setOutputfile(std::string &newOutputfile) {
00018     LOG_INFO << "Setting outputfile to...";
00019
00020     // If no value for key "outputfile"
00021     if (newOutputfile.empty()) {
00022         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00023         throw exceptions::InvalidValueException("outputfile",
00024         "Outputfile can't be empty!");
00025     }
00026
00027     // If outputfile is already set
00028     if (!this->outputfile.empty()) {
00029         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00030         throw exceptions::InvalidValueException("outputfile",
00031         "Outputfile is already set!");
00032     }
00033
00034     // If outputfile does not end with ".bat"
00035     if (!newOutputfile.ends_with(".bat")) {
00036         newOutputfile += ".bat";
00037         LOG_WARNING << "Outputfile does not end with \".bat\", adding it now: "
00038         << newOutputfile;
00039     }
00040
00041     this->outputfile = newOutputfile;
00042     LOG_INFO << "Outputfile set to: " << this->outputfile << "\n";
00043 }
00044
00045 void FileData::setApplication(const std::string &newApplication) {
00046     if (newApplication.empty()) {
00047         LOG_INFO << "newApplication empty, returning";
00048         return;
00049     }
00050
00051     LOG_INFO << "Setting application to: " << newApplication << "\n";
00052     this->application.emplace(newApplication);
00053 }
00054
00055 void FileData::addCommand(const std::string &command) {
00056     if (command.empty()) {
00057         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00058         throw exceptions::InvalidValueException("command",
00059         "Command value is empty!");
00060     }
00061
00062     LOG_INFO << "Adding command: " << command << "\n";
00063     this->commands.push_back(command);
00064 }
00065
00066 void FileData::addEnvironmentVariable(const std::string &name,
00067                                     const std::string &value) {
00068     if (name.empty()) {
00069         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00070         throw exceptions::InvalidValueException("name", "Name value is empty!");
00071     }
00072
00073     if (value.empty()) {
00074         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00075         throw exceptions::InvalidValueException("key", "Key value is empty");
00076     }
00077
00078     LOG_INFO << "Adding environment variable: " << name << "=" << value << "\n";
00079     this->environmentVariables.emplace_back(name, value);
00080 }
00081
00082 void FileData::addPathValue(const std::string &pathValue) {
00083     if (pathValue.empty()) {
00084         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00085         throw exceptions::InvalidValueException("path", "Path value is empty");
00086     }
00087
00088     LOG_INFO << "Adding path value: " << pathValue << "\n";
00089     this->pathValues.push_back(pathValue);
00090 }
00091 } // namespace parsing

```

## 11.26 src/sources/JsonHandler.cpp File Reference

```
#include "JsonHandler.hpp"
#include "Exceptions.hpp"
#include "FileData.hpp"
#include "KeyValidator.hpp"
#include "LoggingWrapper.hpp"
Include dependency graph for JsonHandler.cpp:
```



### Namespaces

- namespace [parsing](#)

*The namespace containing everything relevant to parsing.*

### 11.26.1 Detailed Description

Author

Date

Version

Copyright

See LICENSE file

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

## 11.27 JsonHandler.cpp

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

```

00001
00012 #include "JsonHandler.hpp"
00013 #include "Exceptions.hpp"
00014 #include "FileData.hpp"
00015 #include "KeyValidator.hpp"
00016 #include "LoggingWrapper.hpp"
00017
00018 namespace parsing {
00019 JsonHandler::JsonHandler(const std::string &filename) {
00020     LOG_INFO << "Initializing JSONHandler with filename: " << filename << "\n";
00021     this->root = parseFile(filename);
00022 }
00023
00024 std::shared_ptr<Json::Value> JsonHandler::parseFile(const std::string &filename)
00025 {
00026     LOG_INFO << "Parsing file: " << filename << "\n";
00027     std::ifstream file(filename);
00028     Json::Value newRoot;
00029
00030     // Json::Reader.parse() returns false if parsing fails
00031     if (Json::Reader reader; !reader.parse(file, newRoot)) {
00032         throw exceptions::ParsingException(filename);
00033     }
00034
00035     // Validate keys
00036     // Check for errors
00037     if (auto errors = KeyValidator::getInstance().validateKeys(newRoot, filename);
00038         !errors.empty()) {
00039         throw exceptions::InvalidKeyException(errors);
00040     }
00041
00042     LOG_INFO << "File \"" << filename << "\" has been parsed\n";
00043     return std::make_shared<Json::Value>(newRoot);
00044 }
00045
00046 std::shared_ptr<FileData> JsonHandler::getFileData() {
00047     LOG_INFO << "Creating FileData object for return...\n";
00048     return this->createFileData();
00049 }
00050
00051 std::shared_ptr<FileData> JsonHandler::createFileData() {
00052     LOG_INFO << "Creating FileData object...\n";
00053     this->data = std::make_shared<FileData>();
00054     this->assignOutputFile();
00055     this->assignHideShell();
00056     this->assignApplication();
00057     this->assignEntries();
00058     return this->data;
00059 }
00060
00061 void JsonHandler::assignOutputFile() const {
00062     LOG_INFO << "Assigning outputfile...\n";
00063     std::string outputFile = this->root->get("outputfile", "").asString();
00064
00065     this->data->setOutputFile(outputFile);
00066 }
00067
00068 void JsonHandler::assignHideShell() const {
00069     LOG_INFO << "Assigning hide shell...\n";
00070     // If the 'hideshell' key is not given, it defaults to false
00071     this->data->setHideShell(this->root->get("hideshell", false).asBool());
00072 }
00073
00074 void JsonHandler::assignApplication() const {
00075     LOG_INFO << "Assigning application...\n";
00076     this->data->setApplication(this->root->get("application", "").asString());
00077 }
00078
00079 void JsonHandler::assignEntries() const {
00080     LOG_INFO << "Assigning entries...\n";
00081
00082     for (const auto &entry : this->root->get("entries", "")) {
00083         std::string entryType = entry.get("type", "").asString();
00084
00085         if (entryType == "EXE") {
00086             LOG_INFO << "Calling function to assign command...\n";
00087             this->assignCommand(entry);
00088         } else if (entryType == "ENV") {
00089             LOG_INFO << "Calling function to assign environment variable...\n";
00090             this->assignEnvironmentVariable(entry);
00091         } else if (entryType == "PATH") {
00092

```

```

00093     LOG_INFO « "Calling function to assign path value...\n";
00094     this->assignPathValue(entry);
00095 } else {
00096     // Due to validation beforehand - this should never be reached!
00097     throw exceptions::UnreachableCodeException(
00098         "Unknown entries should be caught by KeyValidator!\nPlease report "
00099         "this bug!");
00100 }
00101 }
00102 }
00103
00104 void JsonHandler::assignCommand(const Json::Value &entry) const {
00105     LOG_INFO « "Assigning command...\n";
00106     this->data->addCommand(entry.get("command", "").asString());
00107 }
00108
00109 void JsonHandler::assignEnvironmentVariable(const Json::Value &entry) const {
00110     LOG_INFO « "Assigning environment variable...\n";
00111     std::string key = entry.get("key", "").asString();
00112     std::string value = entry.get("value", "").asString();
00113     this->data->addEnvironmentVariable(key, value);
00114 }
00115
00116 void JsonHandler::assignPathValue(const Json::Value &entry) const {
00117     LOG_INFO « "Assigning path value...\n";
00118     this->data->addPathValue(entry.get("path", "").asString());
00119 }
00120 } // namespace parsing

```

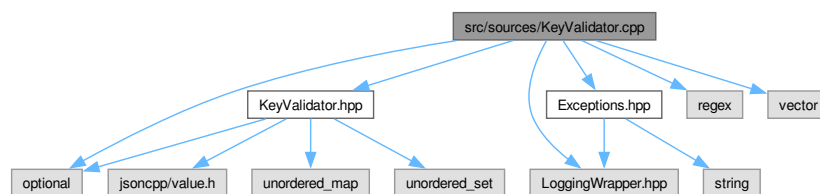
## 11.28 src/sources/KeyValidator.cpp File Reference

```

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

```

Include dependency graph for KeyValidator.cpp:



### Namespaces

- namespace `parsing`

*The namespace containing everything relevant to parsing.*

### 11.28.1 Detailed Description

Author

Date

Version

Copyright

See LICENSE file

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

## 11.29 KeyValidator.cpp

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

```
00001
00011 #include "KeyValidator.hpp"
00012 #include "Exceptions.hpp"
00013 #include "LoggingWrapper.hpp"
00014 #include <optional>
00015 #include <regex>
00016 #include <vector>
00017
00020 namespace parsing {
00021 KeyValidator &KeyValidator::getInstance() {
00022     static KeyValidator keyValidator;
00023     LOG_INFO « "Returning KeyValidator instance!";
00024     return keyValidator;
00025 }
00026
00027 std::vector<std::tuple<int, std::string>>
00028 KeyValidator::validateKeys(const Json::Value &root,
00029                             const std::string &filename) {
00030
00031     std::vector<std::tuple<int, std::string>> wrongKeys =
00032         getWrongKeys(root, filename);
00033
00034     for (Json::Value entries = root.get("entries", "");
00035          const auto &entry : entries) {
00036
00037         const auto entryKeys = entry.getMemberNames();
00038         std::unordered_set<std::string> entryKeysSet(entryKeys.begin(),
00039                                                       entryKeys.end());
00040
00041         const auto wrongEntries = validateEntries(filename, entryKeysSet);
00042         wrongKeys.insert(wrongKeys.end(), wrongEntries.begin(), wrongEntries.end());
00043         // Validate that each entry has it's necessary keys
00044         validateTypes(filename, entry, entryKeysSet);
00045     }
00046
00047     return wrongKeys;
00048 }
00049
00050 std::vector<std::tuple<int, std::string>>
00051 KeyValidator::getWrongKeys(const Json::Value &root,
00052                             const std::string &filename) const {
00053     std::vector<std::tuple<int, std::string>> wrongKeys = {};
00054
00055     for (const auto &key : root.getMemberNames()) {
00056         if (!validKeys.contains(key)) {
00057             const auto error = getUnknownKeyLine(filename, key);
00058
00059             if (!error.has_value()) {
00060                 LOG_ERROR « "Unable to find line of wrong key!";
00061                 continue;
00062             }
00063
00064             wrongKeys.emplace_back(error.value_or(-1), key);
00065         }
00066     }
00067 }
```

```

00068     return wrongKeys;
00069 }
00070
00071 std::vector<std::tuple<int, std::string>> KeyValidator::validateEntries(
00072     const std::string &filename,
00073     const std::unordered_set<std::string> &entryKeys) const {
00074     std::vector<std::tuple<int, std::string>> wrongKeys = {};
00075
00076     for (const auto &key : entryKeys) {
00077         if (!validEntryKeys.contains(key)) {
00078             const auto error = getUnknownKeyLine(filename, key);
00079
00080             if (!error.has_value()) {
00081                 LOG_ERROR « "Unable to find line of wrong key!";
00082                 continue;
00083             }
00084
00085             wrongKeys.emplace_back(error.value(), key);
00086         }
00087     }
00088
00089     return wrongKeys;
00090 }
00091
00092 void KeyValidator::validateTypes(
00093     const std::string &filename, const Json::Value &entry,
00094     const std::unordered_set<std::string> &entryKeys) {
00095     const std::string type = entry.get("type", "ERROR").asString();
00096
00097     if (type == "ERROR") {
00098         throw exceptions::MissingTypeException();
00099     } else if (typeToKeys.contains(type)) {
00100         const std::optional<int> line =
00101             getUnknownKeyLine(filename, std::string(type));
00102
00103         if (!line.has_value()) {
00104             LOG_INFO « "Unable to find line of wrong type!";
00105         }
00106         throw exceptions::InvalidTypeException(std::string(type), line.value());
00107     } else {
00108         for (const auto &key : typeToKeys[type]) {
00109             if (entryKeys.contains(key)) {
00110                 throw exceptions::MissingKeyException(key, std::string(type));
00111             }
00112         }
00113     }
00114 }
00115
00116 std::optional<int>
00117 KeyValidator::getUnknownKeyLine(const std::string &filename,
00118                                 const std::string &wrongKey) {
00119     std::ifstream file(filename);
00120
00121     if (!file.is_open()) {
00122         LOG_ERROR « "File not open!";
00123         return std::nullopt;
00124     }
00125
00126     std::string line;
00127     const std::regex wrongKeyPattern("\\b" + wrongKey + "\\b");
00128
00129     for (int lineNumber = 1; std::getline(file, line); ++lineNumber) {
00130         if (std::regex_search(line, wrongKeyPattern)) {
00131             return lineNumber;
00132         }
00133     }
00134     return std::nullopt;
00135 }
00136
00137 } // namespace parsing

```

## 11.30 src/sources/Utils.cpp File Reference

Implementation for the Utils class.

```

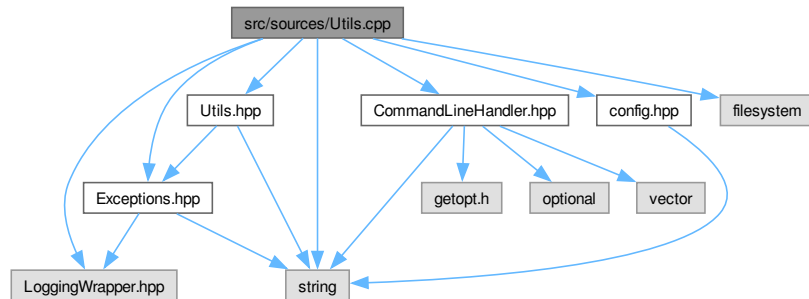
#include "Utils.hpp"
#include "CommandLineHandler.hpp"
#include "Exceptions.hpp"
#include "config.hpp"

```



```
#include <LoggingWrapper.hpp>
#include <filesystem>
#include <string>
```

Include dependency graph for Utils.cpp:



## Namespaces

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

### 11.30.1 Detailed Description

Implementation for the Utils class.

#### Author

Simon Blum

#### Date

2024-04-18

#### Version

0.1.5

This file includes the implementation for the Utils class.

#### See also

`src/include/utility/Utilities.hpp`

#### Copyright

See LICENSE file

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

## 11.31 Utils.cpp

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

```

00001
00015 #include "Utils.hpp"
00016 #include "CommandLineHandler.hpp"
00017 #include "Exceptions.hpp"
00018 #include "config.hpp"
00019
00020 #include <LoggingWrapper.hpp>
00021 #include <filesystem>
00022 #include <string>
00023
00024 namespace utilities {
00025 void Utils::setupEasyLogging(const std::string &configFile) {
00026     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 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     return directory;
00075 }
00076 bool Utils::handleParseException(const exceptions::CustomException &e,
00077     const std::vector<std::string>::iterator &file,
00078     const std::vector<std::string> &files) {
00079     OUTPUT << "\nThere has been a error while trying to parse \"" << *file
00080         << ":\n";
00081     LOG_ERROR << e.what();
00082
00083     if (std::next(file) != files.end() &&
00084         !utilities::Utils::askToContinue(
00085             "Do you want to continue with the other files? (y/n) "
00086             "")) {
00087         OUTPUT << "Aborting...";
00088         LOG_INFO << "Application ended by user Input";
00089         return false;
00090     }
00091     std::cout << std::endl;
00092     return true;
00093 }
00094
00095 } // namespace utilities

```

# Index

~CommandLineHandler  
cli::CommandLineHandler, 31

addCommand  
    parsing::FileData, 39  
addEnvironmentVariable  
    parsing::FileData, 39  
addPathValue  
    parsing::FileData, 40  
application  
    parsing::FileData, 43  
askToContinue  
    utilities::Utils, 79  
assignApplication  
    parsing::JsonHandler, 55  
assignCommand  
    parsing::JsonHandler, 55  
assignEntries  
    parsing::JsonHandler, 56  
assignEnvironmentVariable  
    parsing::JsonHandler, 57  
assignHideShell  
    parsing::JsonHandler, 57  
assignOutputFile  
    parsing::JsonHandler, 57  
assignPathValue  
    parsing::JsonHandler, 58

AUTHORS  
    config, 18

BatchCreator, 23  
    BatchCreator, 24  
    createBatch, 24  
    dataStream, 29  
    fileData, 29  
    getDataStream, 25  
    writeApp, 26  
    writeCommands, 26  
    writeEnd, 26  
    writeEnvVariables, 27  
    writeHideShell, 27  
    writePathVariables, 28  
    writeStart, 28

checkConfigFile  
    utilities::Utils, 80

checkDirectory  
    utilities::Utils, 80

cli, 17  
    options, 18

cli::CommandLineHandler, 29  
    ~CommandLineHandler, 31  
    CommandLineHandler, 31  
    parseArguments, 31  
    printCredits, 32  
    printHelp, 33  
    printVersion, 33

CommandLineHandler  
    cli::CommandLineHandler, 31

commands  
    parsing::FileData, 43

config, 18  
    AUTHORS, 18  
    DESCRIPTION, 18  
    EXECUTABLE\_NAME, 18  
    HOMEPAGE\_URL, 19  
    LOG\_CONFIG, 19  
    MAJOR\_VERSION, 19  
    MINOR\_VERSION, 19  
    PATCH\_VERSION, 19  
    PROJECT\_NAME, 19

createBatch  
    BatchCreator, 24

createFileData  
    parsing::JsonHandler, 58

data  
    parsing::JsonHandler, 61

dataStream  
    BatchCreator, 29

DESCRIPTION  
    config, 18

environmentVariables  
    parsing::FileData, 43

exceptions, 20

exceptions::CustomException, 34  
    what, 36

exceptions::FailedToOpenFileException, 36  
    FailedToOpenFileException, 37  
    message, 38  
    what, 38

exceptions::FileExistsException, 44  
    file, 46  
    FileExistsException, 45  
    message, 46  
    what, 46

exceptions::InvalidKeyException, 46  
    InvalidKeyException, 48  
    message, 48

- what, 48
- exceptions::InvalidTypeException, 48
  - InvalidTypeException, 50
  - message, 50
  - type, 50
  - what, 50
- exceptions::InvalidValueException, 51
  - InvalidValueException, 52
  - key, 53
  - message, 53
  - what, 52
- exceptions::MissingKeyException, 68
  - key, 70
  - message, 70
  - MissingKeyException, 70
  - type, 70
  - what, 70
- exceptions::MissingTypeException, 71
  - message, 72
  - MissingTypeException, 72
  - what, 72
- exceptions::NoSuchDirException, 73
  - message, 74
  - NoSuchDirException, 74
  - what, 74
- exceptions::ParsingException, 75
  - file, 77
  - message, 77
  - ParsingException, 76
  - what, 77
- exceptions::UnreachableCodeException, 77
  - message, 79
  - UnreachableCodeException, 78
  - what, 79
- EXECUTABLE\_NAME
  - config, 18
- FailedToOpenFileException
  - exceptions::FailedToOpenFileException, 37
- file
  - exceptions::FileExistsException, 46
  - exceptions::ParsingException, 77
- fileData
  - BatchCreator, 29
- FileExistsException
  - exceptions::FileExistsException, 45
- getApplication
  - parsing::FileData, 40
- getCommands
  - parsing::FileData, 40
- getDataStream
  - BatchCreator, 25
- getEnvironmentVariables
  - parsing::FileData, 41
- getFileData
  - parsing::JsonHandler, 59
- getHideShell
  - parsing::FileData, 41
- getInstance
  - parsing::KeyValidator, 63
- getOutputFile
  - parsing::FileData, 41
- getPathValues
  - parsing::FileData, 41
- getUnknownKeyLine
  - parsing::KeyValidator, 63
- getWrongKeys
  - parsing::KeyValidator, 64
- handleParseException
  - utilities::Utils, 81
- hideShell
  - parsing::FileData, 43
- Homepage\_URL
  - config, 19
- InvalidKeyException
  - exceptions::InvalidKeyException, 48
- InvalidTypeException
  - exceptions::InvalidTypeException, 50
- InvalidValueException
  - exceptions::InvalidValueException, 52
- JSON2Batch, 1
- JsonHandler
  - parsing::JsonHandler, 54
- key
  - exceptions::InvalidValueException, 53
  - exceptions::MissingKeyException, 70
- LOG\_CONFIG
  - config, 19
- main
  - main.cpp, 102
- main.cpp
  - main, 102
  - parseAndValidateArgs, 103
  - parseFile, 104
  - validateFiles, 105
- MAJOR\_VERSION
  - config, 19
- message
  - exceptions::FailedToOpenFileException, 38
  - exceptions::FileExistsException, 46
  - exceptions::InvalidKeyException, 48
  - exceptions::InvalidTypeException, 50
  - exceptions::InvalidValueException, 53
  - exceptions::MissingKeyException, 70
  - exceptions::MissingTypeException, 72
  - exceptions::NoSuchDirException, 74
  - exceptions::ParsingException, 77
  - exceptions::UnreachableCodeException, 79
- MINOR\_VERSION
  - config, 19
- MissingKeyException
  - exceptions::MissingKeyException, 70

- MissingTypeException
  - exceptions::MissingTypeException, 72
- NoSuchDirException
  - exceptions::NoSuchDirException, 74
- options, 75
  - cli, 18
- outputfile
  - parsing::FileData, 43
- parseAndValidateArgs
  - main.cpp, 103
- parseArguments
  - cli::CommandLineHandler, 31
- parseFile
  - main.cpp, 104
  - parsing::JsonHandler, 60
- parsing, 20
- parsing::FileData, 38
  - addCommand, 39
  - addEnvironmentVariable, 39
  - addPathValue, 40
  - application, 43
  - commands, 43
  - environmentVariables, 43
  - getApplication, 40
  - getCommands, 40
  - getEnvironmentVariables, 41
  - getHideShell, 41
  - getOutputFile, 41
  - getPathValues, 41
  - hideShell, 43
  - outputfile, 43
  - pathValues, 44
  - setApplication, 42
  - setHideShell, 42
  - setOutputFile, 42
- parsing::JsonHandler, 53
  - assignApplication, 55
  - assignCommand, 55
  - assignEntries, 56
  - assignEnvironmentVariable, 57
  - assignHideShell, 57
  - assignOutputFile, 57
  - assignPathValue, 58
  - createFileData, 58
  - data, 61
  - getFileData, 59
  - JsonHandler, 54
  - parseFile, 60
  - root, 61
- parsing::KeyValidator, 62
  - getInstance, 63
  - getUnknownKeyLine, 63
  - getWrongKeys, 64
  - typeToKeys, 67
  - validateEntries, 65
  - validateKeys, 66
  - validateTypes, 66
  - validEntryKeys, 67
  - validKeys, 68
- ParsingException
  - exceptions::ParsingException, 76
- PATCH\_VERSION
  - config, 19
- pathValues
  - parsing::FileData, 44
- printCredits
  - cli::CommandLineHandler, 32
- printHelp
  - cli::CommandLineHandler, 33
- printVersion
  - cli::CommandLineHandler, 33
- PROJECT\_NAME
  - config, 19
- README.md, 83
- root
  - parsing::JsonHandler, 61
- setApplication
  - parsing::FileData, 42
- setHideShell
  - parsing::FileData, 42
- setOutputFile
  - parsing::FileData, 42
- setupEasyLogging
  - utilities::Utils, 81
- src/include/BatchCreator.hpp, 83, 85
- src/include/CommandLineHandler.hpp, 85, 87
- src/include/config.hpp, 87, 89
- src/include/Exceptions.hpp, 90, 91
- src/include/FileData.hpp, 93, 95
- src/include/JsonHandler.hpp, 96, 97
- src/include/KeyValidator.hpp, 98, 99
- src/include/Utils.hpp, 100, 101
- src/main.cpp, 101, 106
- src/sources/BatchCreator.cpp, 107, 108
- src/sources/CommandLineHandler.cpp, 109, 110
- src/sources/FileData.cpp, 112, 113
- src/sources/JsonHandler.cpp, 114, 115
- src/sources/KeyValidator.cpp, 116, 117
- src/sources/Utils.cpp, 118, 120
- StyleHelpers, 15
- Todo List, 3
- type
  - exceptions::InvalidTypeException, 50
  - exceptions::MissingKeyException, 70
- typeToKeys
  - parsing::KeyValidator, 67
- UnreachableCodeException
  - exceptions::UnreachableCodeException, 78
- utilities, 21
  - utilities::Utils, 79
  - askToContinue, 79

- checkConfigFile, [80](#)
- checkDirectory, [80](#)
- handleParseException, [81](#)
- setupEasyLogging, [81](#)
- validateEntries
  - parsing::KeyValidator, [65](#)
- validateFiles
  - main.cpp, [105](#)
- validateKeys
  - parsing::KeyValidator, [66](#)
- validateTypes
  - parsing::KeyValidator, [66](#)
- validEntryKeys
  - parsing::KeyValidator, [67](#)
- validKeys
  - parsing::KeyValidator, [68](#)
- what
  - exceptions::CustomException, [36](#)
  - exceptions::FailedToOpenFileException, [38](#)
  - exceptions::FileExistsException, [46](#)
  - exceptions::InvalidKeyException, [48](#)
  - exceptions::InvalidTypeException, [50](#)
  - exceptions::InvalidValueException, [52](#)
  - exceptions::MissingKeyException, [70](#)
  - exceptions::MissingTypeException, [72](#)
  - exceptions::NoSuchDirException, [74](#)
  - exceptions::ParsingException, [77](#)
  - exceptions::UnreachableCodeException, [79](#)
- writeApp
  - BatchCreator, [26](#)
- writeCommands
  - BatchCreator, [26](#)
- writeEnd
  - BatchCreator, [26](#)
- writeEnvVariables
  - BatchCreator, [27](#)
- writeHideShell
  - BatchCreator, [27](#)
- writePathVariables
  - BatchCreator, [28](#)
- writeStart
  - BatchCreator, [28](#)