

JSON2Batch

0.2.1

Generated on Thu Apr 25 2024 12:19:31 for JSON2Batch by Doxygen 1.9.8

Thu Apr 25 2024 12:19:31

1 JSON2Batch	1
1.1 JSON2Batch	1
2 Todo List	3
3 Topic Index	5
3.1 Topics	5
4 Namespace Index	7
4.1 Namespace List	7
5 Hierarchical Index	9
5.1 Class Hierarchy	9
6 Class Index	11
6.1 Class List	11
7 File Index	13
7.1 File List	13
8 Topic Documentation	15
8.1 StyleHelpers	15
9 Namespace Documentation	17
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 exceptions Namespace Reference	18
9.2.1 Detailed Description	18
9.3 parsing Namespace Reference	19
9.3.1 Detailed Description	19
9.4 utilities Namespace Reference	19
9.4.1 Detailed Description	19
10 Class Documentation	21
10.1 BatchCreator Class Reference	21
10.1.1 Detailed Description	22
10.1.2 Constructor & Destructor Documentation	22
10.1.2.1 BatchCreator()	22
10.1.3 Member Function Documentation	22
10.1.3.1 createBatch()	22
10.1.3.2 getDataStream()	23
10.1.3.3 writeApp()	24
10.1.3.4 writeCommands()	24
10.1.3.5 writeEnd()	25

10.1.3.6 writeEnvVariables()	25
10.1.3.7 writeHideShell()	26
10.1.3.8 writePathVariables()	26
10.1.3.9 writeStart()	27
10.1.4 Member Data Documentation	27
10.1.4.1 dataStream	27
10.1.4.2 fileData	27
10.2 cli::CommandLineHandler Class Reference	27
10.2.1 Detailed Description	28
10.2.2 Constructor & Destructor Documentation	29
10.2.2.1 CommandLineHandler()	29
10.2.2.2 ~CommandLineHandler()	29
10.2.3 Member Function Documentation	29
10.2.3.1 parseArguments()	29
10.2.3.2 printCredits()	30
10.2.3.3 printHelp()	31
10.2.3.4 printVersion()	32
10.3 exceptions::CustomException Class Reference	32
10.3.1 Detailed Description	33
10.3.2 Member Function Documentation	34
10.3.2.1 what()	34
10.4 exceptions::FailedToOpenFileException Class Reference	34
10.4.1 Detailed Description	35
10.4.2 Constructor & Destructor Documentation	35
10.4.2.1 FailedToOpenFileException()	35
10.4.3 Member Function Documentation	36
10.4.3.1 what()	36
10.4.4 Member Data Documentation	36
10.4.4.1 message	36
10.5 parsing::FileData Class Reference	36
10.5.1 Detailed Description	37
10.5.2 Member Function Documentation	37
10.5.2.1 addCommand()	37
10.5.2.2 addEnvironmentVariable()	37
10.5.2.3 addPathValue()	38
10.5.2.4 getApplication()	38
10.5.2.5 getCommands()	39
10.5.2.6 getEnvironmentVariables()	39
10.5.2.7 getHideShell()	39
10.5.2.8 getOutputFile()	39
10.5.2.9 getPathValues()	40
10.5.2.10 setApplication()	40

10.5.2.11 setHideShell()	40
10.5.2.12 setOutputFile()	40
10.5.3 Member Data Documentation	41
10.5.3.1 application	41
10.5.3.2 commands	41
10.5.3.3 environmentVariables	41
10.5.3.4 hideShell	41
10.5.3.5 outputfile	42
10.5.3.6 pathValues	42
10.6 exceptions::FileExistsException Class Reference	42
10.6.1 Detailed Description	43
10.6.2 Constructor & Destructor Documentation	43
10.6.2.1 FileExistsException()	43
10.6.3 Member Function Documentation	44
10.6.3.1 what()	44
10.6.4 Member Data Documentation	44
10.6.4.1 file	44
10.6.4.2 message	44
10.7 exceptions::InvalidKeyException Class Reference	44
10.7.1 Detailed Description	45
10.7.2 Constructor & Destructor Documentation	46
10.7.2.1 InvalidKeyException()	46
10.7.3 Member Function Documentation	46
10.7.3.1 what()	46
10.7.4 Member Data Documentation	46
10.7.4.1 message	46
10.8 exceptions::InvalidTypeException Class Reference	46
10.8.1 Detailed Description	48
10.8.2 Constructor & Destructor Documentation	48
10.8.2.1 InvalidTypeException()	48
10.8.3 Member Function Documentation	48
10.8.3.1 what()	48
10.8.4 Member Data Documentation	48
10.8.4.1 message	48
10.8.4.2 type	49
10.9 exceptions::InvalidValueException Class Reference	49
10.9.1 Detailed Description	50
10.9.2 Constructor & Destructor Documentation	50
10.9.2.1 InvalidValueException()	50
10.9.3 Member Function Documentation	50
10.9.3.1 what()	50
10.9.4 Member Data Documentation	51

10.9.4.1 key	51
10.9.4.2 message	51
10.10 parsing::JsonHandler Class Reference	51
10.10.1 Detailed Description	52
10.10.2 Constructor & Destructor Documentation	52
10.10.2.1 JsonHandler() [1/2]	52
10.10.2.2 JsonHandler() [2/2]	52
10.10.3 Member Function Documentation	53
10.10.3.1 assignApplication()	53
10.10.3.2 assignCommand()	53
10.10.3.3 assignEntries()	54
10.10.3.4 assignEnvironmentVariable()	55
10.10.3.5 assignHideShell()	55
10.10.3.6 assignOutputFile()	56
10.10.3.7 assignPathValue()	56
10.10.3.8 createFileData()	57
10.10.3.9 getFileData()	57
10.10.3.10 parseFile()	58
10.10.4 Member Data Documentation	59
10.10.4.1 data	59
10.10.4.2 root	60
10.11 parsing::KeyValidator Class Reference	60
10.11.1 Detailed Description	61
10.11.2 Member Function Documentation	61
10.11.2.1 getInstance()	61
10.11.2.2 getUnknownKeyLine()	61
10.11.2.3 getWrongKeys()	62
10.11.2.4 validateEntries()	63
10.11.2.5 validateKeys()	64
10.11.2.6 validateTypes()	65
10.11.3 Member Data Documentation	65
10.11.3.1 validEntryKeys	65
10.11.3.2 validKeys	66
10.12 exceptions::MissingKeyException Class Reference	66
10.12.1 Detailed Description	67
10.12.2 Constructor & Destructor Documentation	68
10.12.2.1 MissingKeyException()	68
10.12.3 Member Function Documentation	68
10.12.3.1 what()	68
10.12.4 Member Data Documentation	68
10.12.4.1 key	68
10.12.4.2 message	68

10.12.4.3 type	68
10.13 exceptions::MissingTypeException Class Reference	69
10.13.1 Detailed Description	70
10.13.2 Constructor & Destructor Documentation	70
10.13.2.1 MissingTypeException()	70
10.13.3 Member Function Documentation	70
10.13.3.1 what()	70
10.13.4 Member Data Documentation	70
10.13.4.1 message	70
10.14 exceptions::NoSuchDirException Class Reference	71
10.14.1 Detailed Description	72
10.14.2 Constructor & Destructor Documentation	72
10.14.2.1 NoSuchDirException()	72
10.14.3 Member Function Documentation	72
10.14.3.1 what()	72
10.14.4 Member Data Documentation	72
10.14.4.1 message	72
10.15 options Struct Reference	73
10.15.1 Detailed Description	73
10.16 exceptions::ParsingException Class Reference	73
10.16.1 Detailed Description	74
10.16.2 Constructor & Destructor Documentation	74
10.16.2.1 ParsingException()	74
10.16.3 Member Function Documentation	75
10.16.3.1 what()	75
10.16.4 Member Data Documentation	75
10.16.4.1 file	75
10.16.4.2 message	75
10.17 exceptions::UnreachableCodeException Class Reference	75
10.17.1 Detailed Description	76
10.17.2 Constructor & Destructor Documentation	76
10.17.2.1 UnreachableCodeException()	76
10.17.3 Member Function Documentation	77
10.17.3.1 what()	77
10.17.4 Member Data Documentation	77
10.17.4.1 message	77
10.18 utilities::Utils Class Reference	77
10.18.1 Detailed Description	77
10.18.2 Member Function Documentation	77
10.18.2.1 askToContinue()	77
10.18.2.2 checkDirectory()	78
10.18.2.3 checkFileEnding()	78

10.18.2.4 checkIfFileExists()	79
10.18.2.5 setupEasyLogging()	80
11 File Documentation	81
11.1 README.md File Reference	81
11.2 src/include/BatchCreator.hpp File Reference	81
11.2.1 Detailed Description	82
11.3 BatchCreator.hpp	83
11.4 src/include/CommandLineHandler.hpp File Reference	83
11.4.1 Detailed Description	84
11.5 CommandLineHandler.hpp	85
11.6 src/include/config.hpp File Reference	85
11.6.1 Detailed Description	86
11.6.2 Macro Definition Documentation	87
11.6.2.1 AUTHORS	87
11.6.2.2 DESCRIPTION	87
11.6.2.3 EXECUTABLE_NAME	87
11.6.2.4 HOMEPAGE_URL	87
11.6.2.5 LOG_CONFIG	87
11.6.2.6 MAJOR_VERSION	87
11.6.2.7 MINOR_VERSION	87
11.6.2.8 PATCH_VERSION	88
11.6.2.9 PROJECT_NAME	88
11.7 config.hpp	88
11.8 src/include/Exceptions.hpp File Reference	88
11.8.1 Detailed Description	89
11.9 Exceptions.hpp	90
11.10 src/include/FileData.hpp File Reference	92
11.10.1 Detailed Description	93
11.11 FileData.hpp	93
11.12 src/include/JsonHandler.hpp File Reference	94
11.12.1 Detailed Description	95
11.13 JsonHandler.hpp	96
11.14 src/include/KeyValidator.hpp File Reference	96
11.14.1 Detailed Description	97
11.15 KeyValidator.hpp	98
11.16 src/include/Utils.hpp File Reference	98
11.17 Utils.hpp	99
11.18 src/main.cpp File Reference	99
11.18.1 Detailed Description	100
11.18.2 Function Documentation	101
11.18.2.1 main()	101

11.18.2.2 parseFiles()	101
11.18.2.3 validateFiles()	102
11.19 main.cpp	103
11.20 src/sources/BatchCreator.cpp File Reference	105
11.21 BatchCreator.cpp	105
11.22 src/sources/CommandLineHandler.cpp File Reference	107
11.22.1 Detailed Description	107
11.23 CommandLineHandler.cpp	108
11.24 src/sources/FileData.cpp File Reference	109
11.24.1 Detailed Description	110
11.25 FileData.cpp	110
11.26 src/sources/JsonHandler.cpp File Reference	111
11.26.1 Detailed Description	112
11.27 JsonHandler.cpp	112
11.28 src/sources/KeyValidator.cpp File Reference	114
11.28.1 Detailed Description	114
11.29 KeyValidator.cpp	115
11.30 src/sources/Utils.cpp File Reference	117
11.30.1 Detailed Description	117
11.31 Utils.cpp	118
Index	119

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: 0.2.1

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

Documentation: <https://definitelynotsimon13.github.io/ProjectJsonToBat>

Aktueller Plan:

- Verantwortlichkeiten zugewiesen
- "Sprint" bis ?

Verantwortlichkeiten:

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

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

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](#)

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

Documentation

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

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:

cli	Includes everything regarding the CLI	17
exceptions	Namespace used for customized exceptions	18
parsing	The namespace containing everything relevant to parsing	19
utilities	Includes all utilities	19

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

BatchCreator	21
cli::CommandLineHandler	27
std::exception	
exceptions::CustomException	32
exceptions::FailedToOpenFileException	34
exceptions::FileExistsException	42
exceptions::InvalidKeyException	44
exceptions::InvalidTypeException	46
exceptions::InvalidValueException	49
exceptions::MissingKeyException	66
exceptions::MissingTypeException	69
exceptions::NoSuchDirException	71
exceptions::ParsingException	73
exceptions::UnreachableCodeException	75
parsing::FileData	36
parsing::JsonHandler	51
parsing::KeyValidator	60
options	73
utilities::Utils	77

Chapter 6

Class Index

6.1 Class List

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

BatchCreator	
Erstellt Batch Datei	21
cli::CommandLineHandler	
Responsible for the Command Line Interface	27
exceptions::CustomException	
Base class for all custom exceptions	32
exceptions::FailedToOpenFileException	
.	34
parsing::FileData	
This class contains all data from the json file	36
exceptions::FileExistsException	
Exception for an already existing outputfile	42
exceptions::InvalidKeyException	
Exception for invalid keys	44
exceptions::InvalidTypeException	
Exception for invalid types	46
exceptions::InvalidValueException	
Exception for an invalid (usually empty) value field	49
parsing::JsonHandler	
This file reads all data from the json file	51
parsing::KeyValidator	
Validates keys of a Json::Value object	60
exceptions::MissingKeyException	
Exception for missing keys within entries	66
exceptions::MissingTypeException	
Exception for missing types of entries	69
exceptions::NoSuchDirException	
.	71
options	
The struct containing all possible options	73
exceptions::ParsingException	
Exception for syntax errors within the json file	73
exceptions::UnreachableCodeException	
Exception for when the application reaches code it shouldn't reach	75
utilities::Utils	
Responsible for utility function	77

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	99
src/include/BatchCreator.hpp	
Creates batch file	81
src/include/CommandLineHandler.hpp	
Responsible for the Command Line Interface	83
src/include/config.hpp	
Configures general project information	85
src/include/Exceptions.hpp	
Contains all the custom exceptions used in the project	88
src/include/FileData.hpp	
This file contains the FileData class	92
src/include/JsonHandler.hpp	
This file contains the JsonHandler class	94
src/include/KeyValidator.hpp	
This file contains the KeyValidator class	96
src/include/Utils.hpp	
.	98
src/sources/BatchCreator.cpp	
.	105
src/sources/CommandLineHandler.cpp	
Implementation for the Command Line Interface	107
src/sources/FileData.cpp	
.	109
src/sources/JsonHandler.cpp	
.	111
src/sources/KeyValidator.cpp	
.	114
src/sources/Utils.cpp	
Implementation for the Utils class	117

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 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.

9.2.1 Detailed Description

Namespace used for customized exceptions.

9.3 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.

9.3.1 Detailed Description

The namespace containing everything relevant to parsing.

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

See also

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

9.4 utilities Namespace Reference

Includes all utilities.

Classes

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

9.4.1 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](#) ()

Private Member Functions

- void [createBatch](#) ()
Setzt batch Datei zusammen.
- void [writeStart](#) ()
Anfang der Batch Datei.
- void [writeHideShell](#) ()
Sichtbarkeit Konsole.
- void [writeCommands](#) ()
Befehle ausführen.
- void [writeEnvVariables](#) ()
Umgebungsvariablen setzen.
- void [writePathVariables](#) ()
Pfade setzen.
- void [writeApp](#) ()
Öffnet Anwendung falls gewünscht.
- void [writeEnd](#) ()
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 25 of file [BatchCreator.hpp](#).

10.1.2 Constructor & Destructor Documentation

10.1.2.1 BatchCreator()

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

Initialisiert [BatchCreator](#).

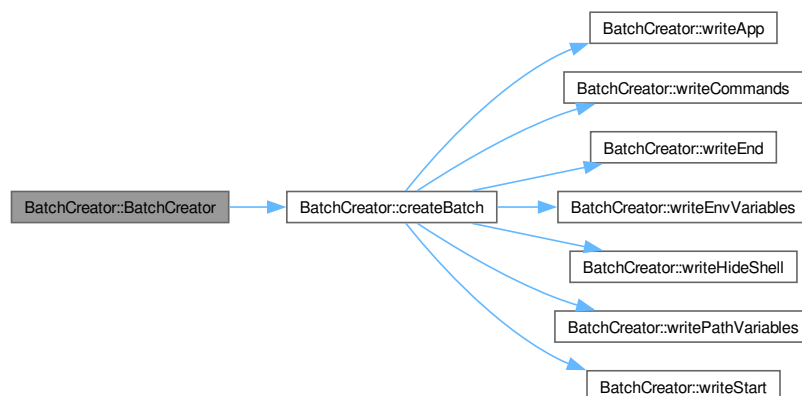
Parameters

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

Definition at line 16 of file [BatchCreator.cpp](#).

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

Here is the call graph for this function:



10.1.3 Member Function Documentation

10.1.3.1 createBatch()

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

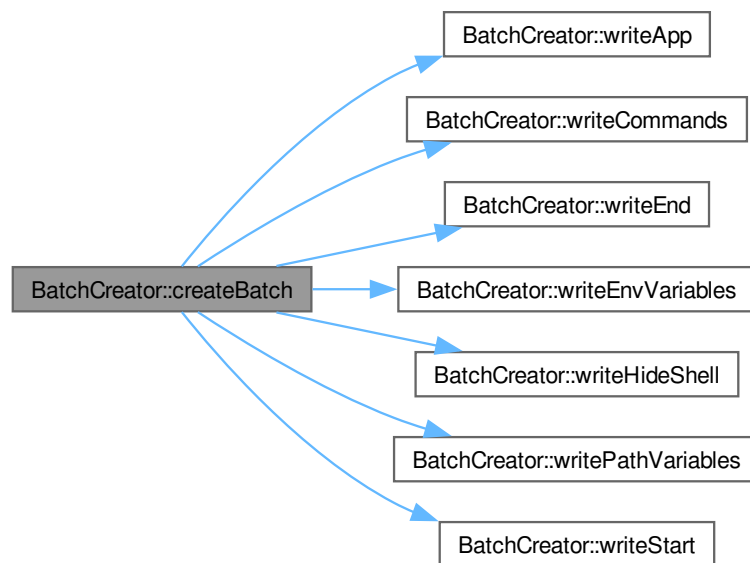

Setzt batch Datei zusammen.

Beinhaltet Aufrufe der einzelnen Komponenten der batch Datei

Definition at line 23 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 ( ) [inline]
```

Todo Documentation

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

References [dataStream](#).

Here is the caller graph for this function:



10.1.3.3 writeApp()

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

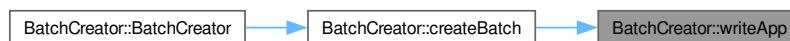
Öffnet Anwendung falls gewünscht.

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 ( ) [private]
```

Befehle ausführen.

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

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

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

Here is the caller graph for this function:



10.1.3.5 writeEnd()

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

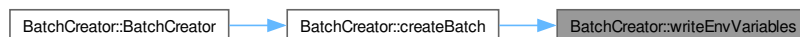
Umgebungsvariablen setzten.

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

Definition at line 59 of file [BatchCreator.cpp](#).

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

Here is the caller graph for this function:



10.1.3.7 writeHideShell()

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

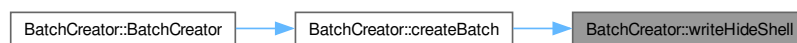
Sichtbarkeit Konsole.

Zeigt bzw. versteckt Konsolenausgabe

Definition at line 40 of file [BatchCreator.cpp](#).

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

Here is the caller graph for this function:



10.1.3.8 writePathVariables()

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

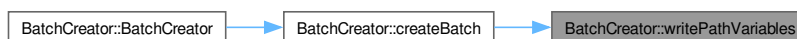
Pfade setzen.

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 ( ) [private]
```

Anfang der Batch Datei.

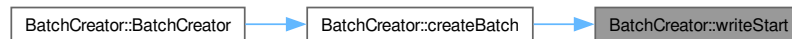
Schreibt den Teil der Batch Datei der immer gleich ist.

- setzt ECHO off
- startet cmd.exe

Definition at line 35 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 42 of file [BatchCreator.hpp](#).

10.1.4.2 fileData

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

Definition at line 44 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

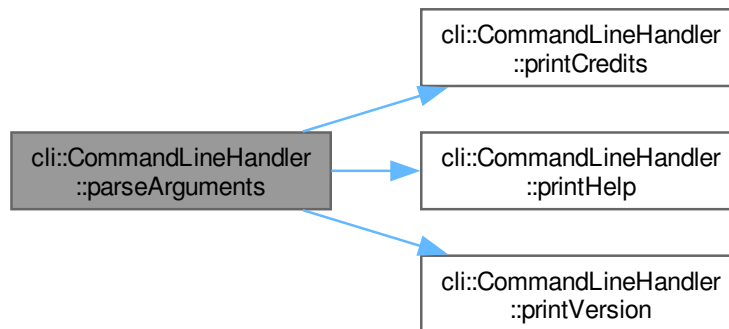
Returns a vector of strings containing all filenames.

Todo Update documentation

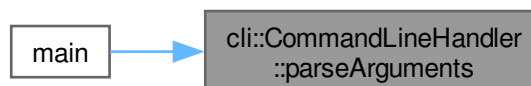
Definition at line 67 of file [CommandLineHandler.cpp](#).

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.

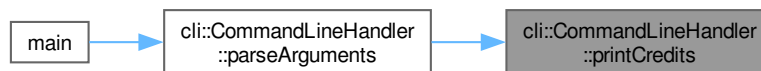
Note

This function ends the application.

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

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

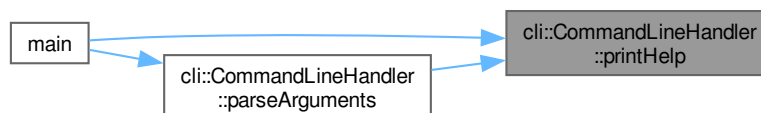
Note

This function ends the application.

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

References [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.

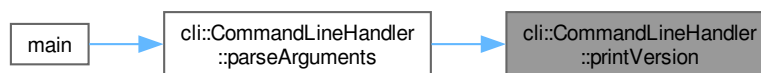
Note

This function ends the application.

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

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

Here is the caller graph for this function:



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

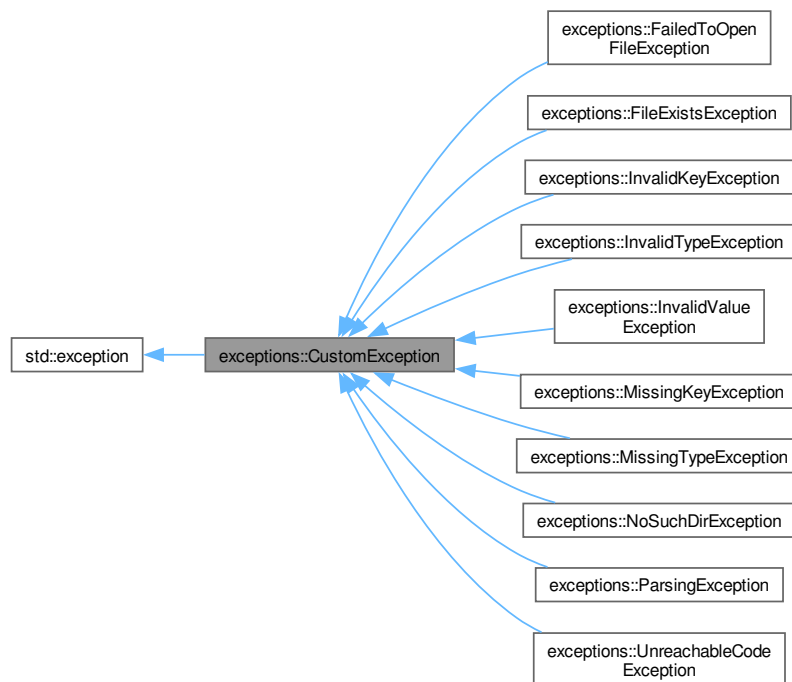
- [src/include/CommandLineHandler.hpp](#)
- [src/sources/CommandLineHandler.cpp](#)

10.3 exceptions::CustomException Class Reference

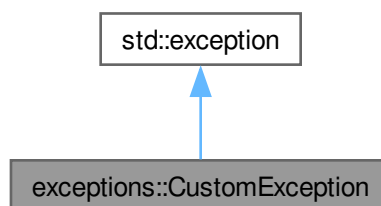
Base class for all custom exceptions.

```
#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](#).

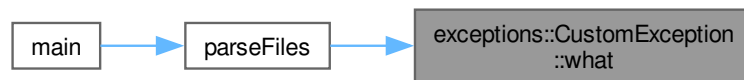
10.3.2 Member Function Documentation

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:



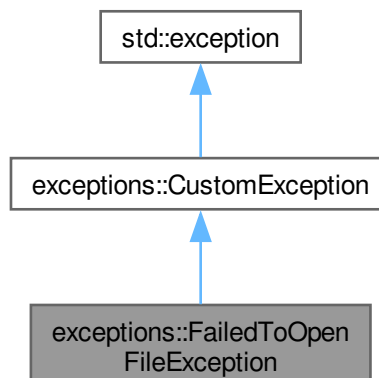
The documentation for this class was generated from the following file:

- `src/include/Exceptions.hpp`

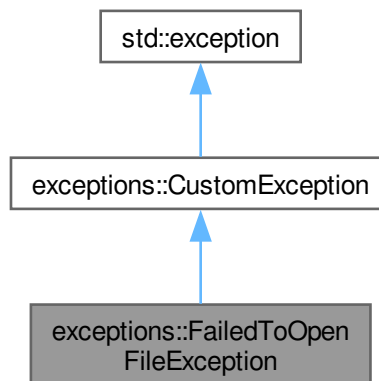
10.4 `exceptions::FailedToOpenFileException` Class Reference

```
#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

exceptions::InvalidValueException	
---	--

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

exceptions::InvalidValueException	
---	--

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

exceptions::InvalidValueException	
---	--

Definition at line 83 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 120 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 128 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 137 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 112 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 104 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 145 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 44 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

exceptions::InvalidValueException	
---	--

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 152 of file [FileData.hpp](#).

10.5.3.2 commands

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

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

10.5.3.3 environmentVariables

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

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

10.5.3.4 hideShell

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

Definition at line 151 of file [FileData.hpp](#).

10.5.3.5 outputfile

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

Definition at line 150 of file [FileData.hpp](#).

10.5.3.6 pathValues

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

Definition at line 155 of file [FileData.hpp](#).

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

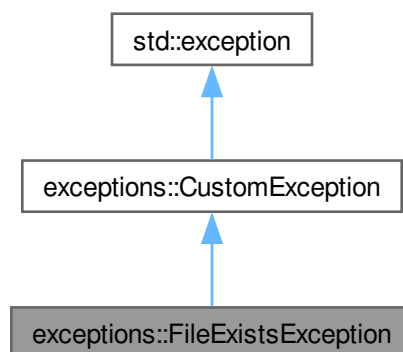
- [src/include/FileData.hpp](#)
- [src/sources/FileData.cpp](#)

10.6 exceptions::FileExistsException Class Reference

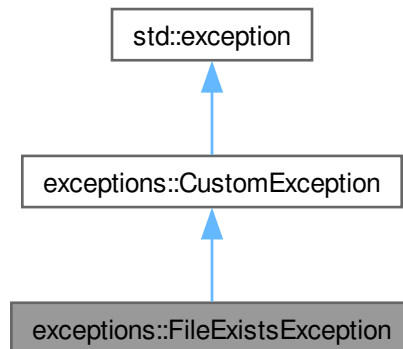
Exception for an already exisiting outputfile.

```
#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:

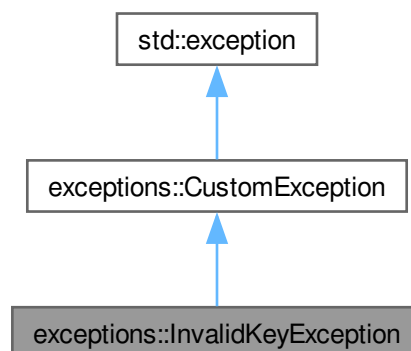
- [src/include/Exceptions.hpp](#)

10.7 exceptions::InvalidKeyException Class Reference

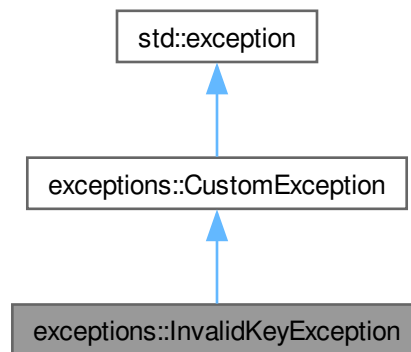
Exception for invalid keys.

```
#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:

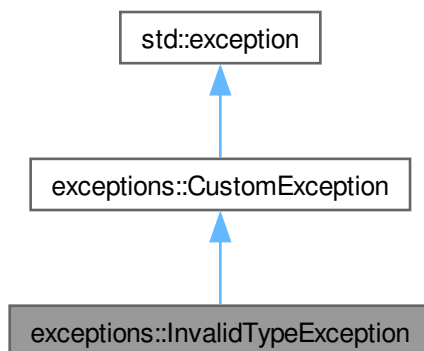
- [src/include/Exceptions.hpp](#)

10.8 exceptions::InvalidTypeException Class Reference

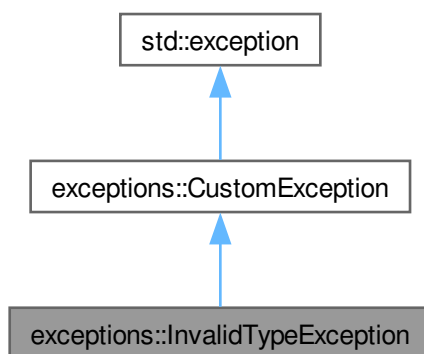
Exception for invalid types.

```
#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:

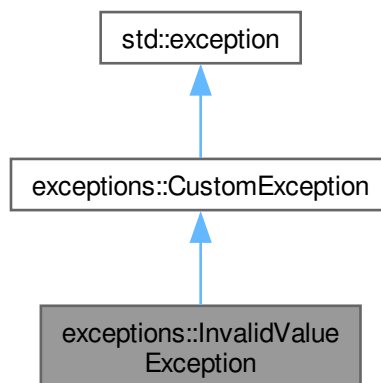
- [src/include/Exceptions.hpp](#)

10.9 exceptions::InvalidValueException Class Reference

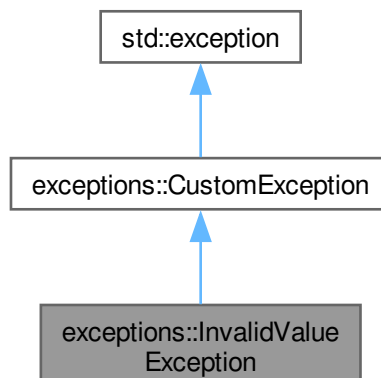
Exception for an ivalid (usually empty) value field.

```
#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.

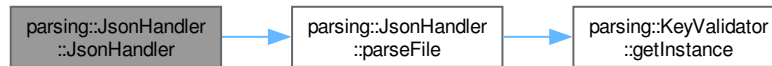
Parameters

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

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

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

Here is the call graph for this function:



10.10.3 Member Function Documentation

10.10.3.1 assignApplication()

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

Assigns application to this->data.

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

Definition at line 77 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.

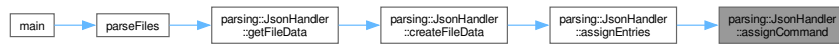
Parameters

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

Definition at line 107 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
--------------	--

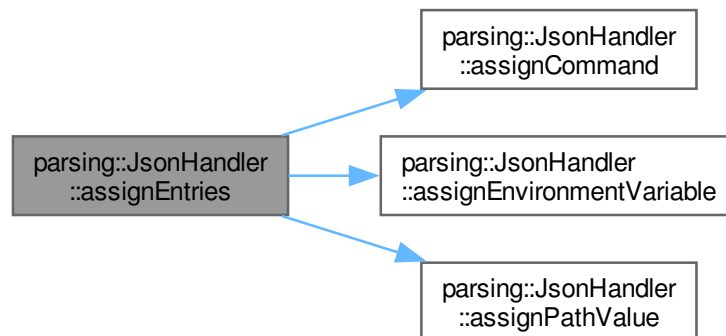
Exceptions

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

Definition at line 83 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 113 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 70 of file [JsonHandler.cpp](#).

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

Here is the caller graph for this function:



10.10.3.6 assignOutputFile()

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

Assigns the outputfile to this->data.

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

Exceptions

exceptions::FileExistsException

Definition at line 63 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.

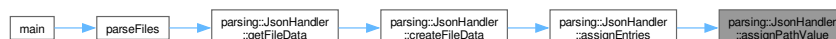
Parameters

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

Definition at line 120 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.

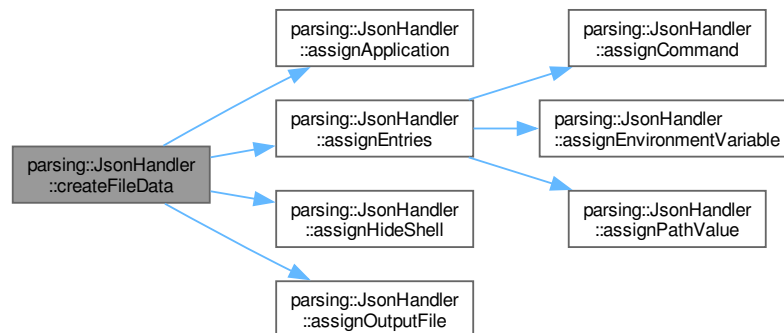
Returns

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

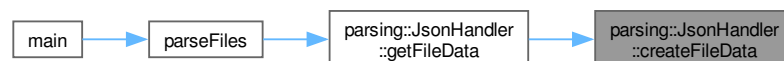
Definition at line 53 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



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.

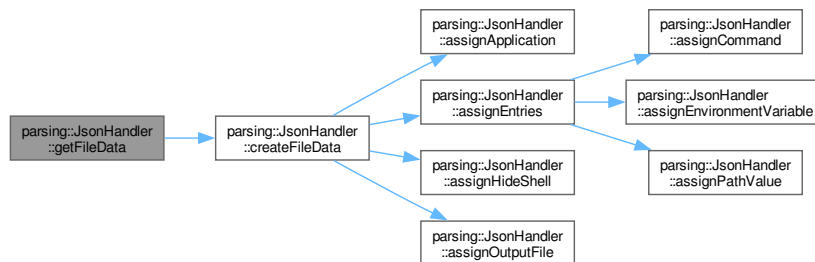
Returns

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

Definition at line 48 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

exceptions::ParsingException	
exceptions::InvalidKeyException	

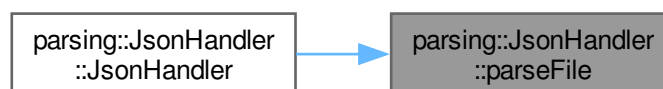
Definition at line 25 of file [JsonHandler.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



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)
Retrieve the wrong keys from a `Json::Value` object.
- `std::vector< std::tuple< int, std::string > >` [validateEntries](#) (const `std::string` &filename, const `std::vector< std::string >` &entryKeys)
Validates that an entries 'type' key is valid.

Static Private Member Functions

- static void [validateTypes](#) (const `std::string` &filename, const `Json::Value` &entry, `std::vector< std::string >` &entryKeys)
Validates types from the entries array.
- static `std::optional< int >` [getUnknownKeyLine](#) (const `std::string` &filename, const `std::string` &wrongKey)

Private Attributes

- `std::vector< std::string >` [validKeys](#)
- `std::vector< std::string >` [validEntryKeys](#)

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 26 of file [KeyValidator.hpp](#).

10.11.2 Member Function Documentation

10.11.2.1 getInstance()

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

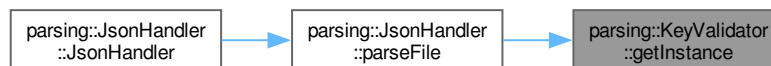
Get the instance of this class.

Returns

Reference to the instance of this class

Definition at line 19 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

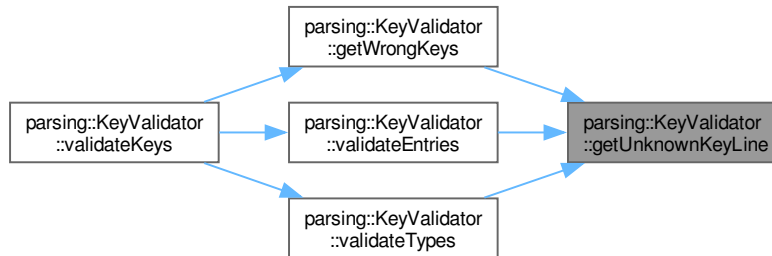
<i>filename</i>	
<i>wrongKey</i>	

Returns

Todo Documentation

Definition at line 151 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 ) [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.

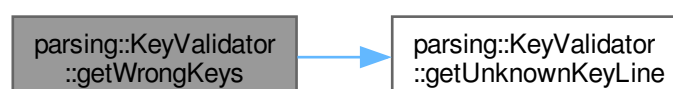
Returns

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

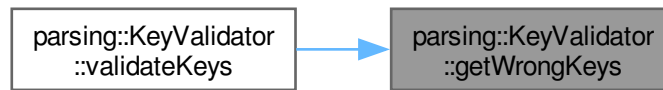
Definition at line 53 of file [KeyValidator.cpp](#).

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::vector< std::string > & entryKeys ) [private]
```

Validates that an entries 'type' key is valid.

Parameters

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

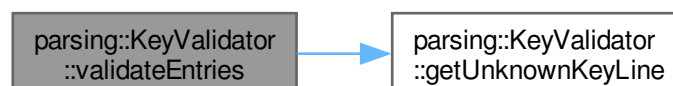
Returns

Todo Documentation

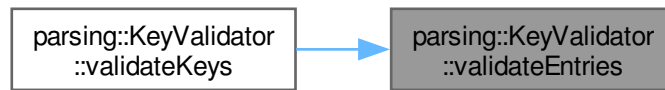
Definition at line 80 of file [KeyValidator.cpp](#).

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 `validKeys` attribute. It calls the necessary 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.

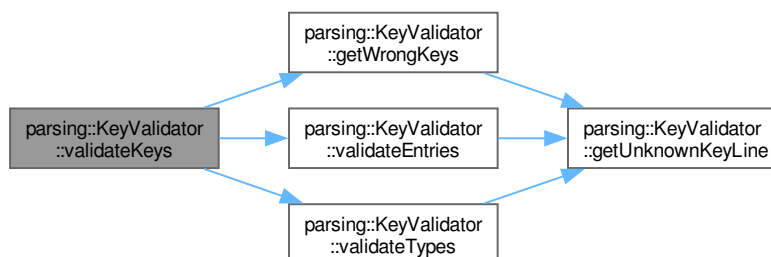
Returns

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

Definition at line 25 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,
    std::vector< std::string > & entryKeys ) [static], [private]
```

Validates types from the entries array.

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

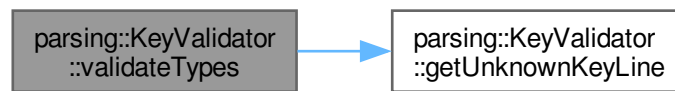
Parameters

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

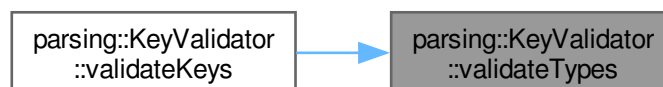
Definition at line 106 of file [KeyValidator.cpp](#).

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

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 validEntryKeys

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

Initial value:

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

Definition at line 111 of file [KeyValidator.hpp](#).

10.11.3.2 validKeys

```
std::vector<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:

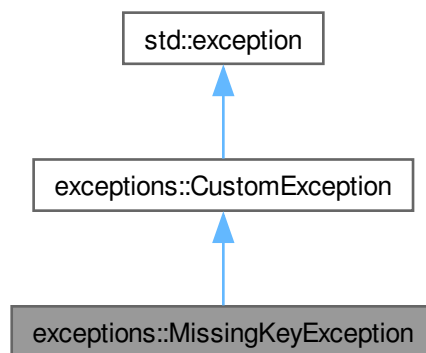
- [src/include/KeyValidator.hpp](#)
- [src/sources/KeyValidator.cpp](#)

10.12 exceptions::MissingKeyException Class Reference

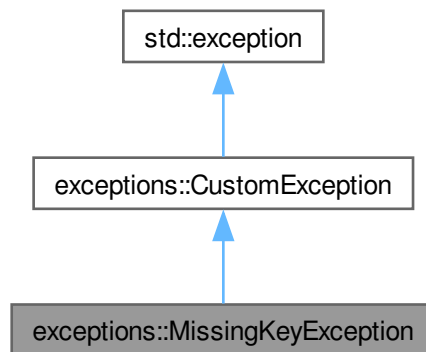
Exception for missing keys within entries.

```
#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:

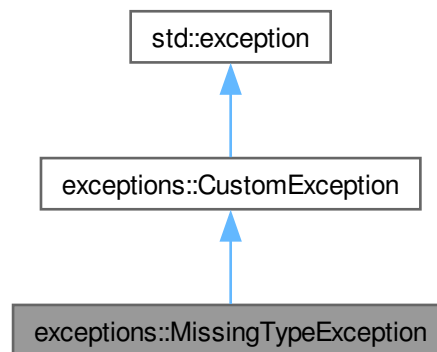
- [src/include/Exceptions.hpp](#)

10.13 exceptions::MissingTypeException Class Reference

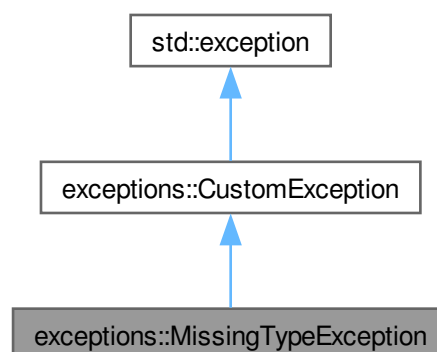
Exception for missing types of entries.

```
#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](#).

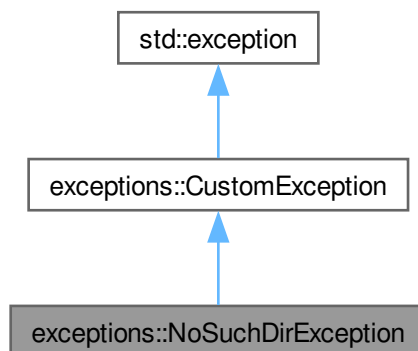
The documentation for this class was generated from the following file:

- `src/include/Exceptions.hpp`

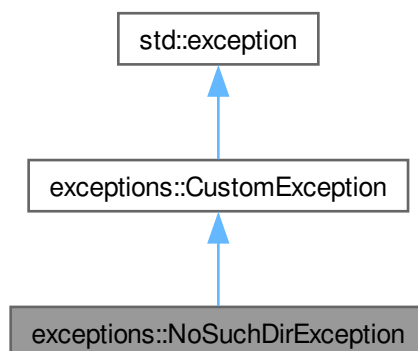
10.14 exceptions::NoSuchDirException Class Reference

```
#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:

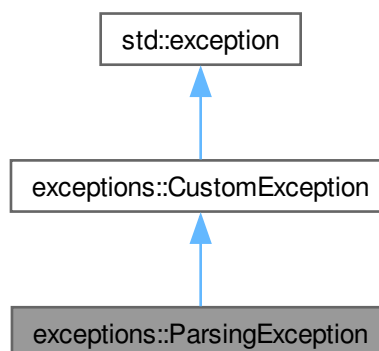
- [src/include/CommandLineHandler.hpp](#)

10.16 exceptions::ParsingException Class Reference

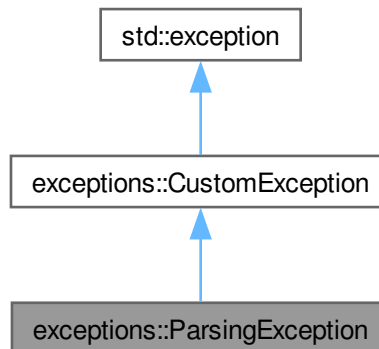
Exception for syntax errors within the json file.

```
#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:

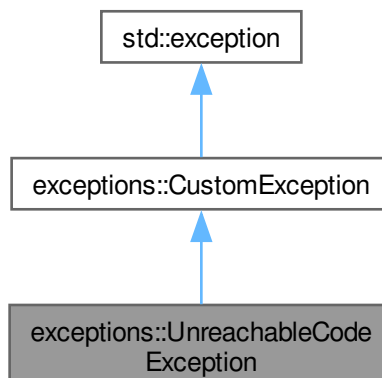
- [src/include/Exceptions.hpp](#)

10.17 exceptions::UnreachableCodeException Class Reference

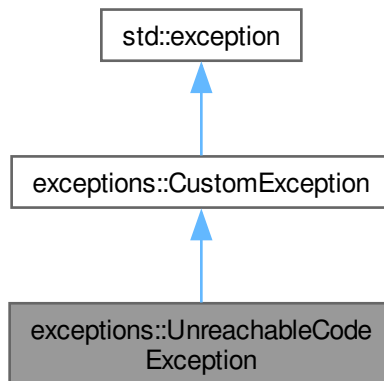
Exception for when the application reaches code it shouldn't reach.

```
#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 [checkIfFileExists](#) (const std::string &fileName)
Check if a file exists.
- static bool [checkFileEnding](#) (const std::string_view &fileName)
Checks if the file ending is ".json".
- static bool [askToContinue](#) (const std::string &prompt="Do you want to continue? (Y/N)\n")
Asks if the user wants to continue.
- 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 39 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

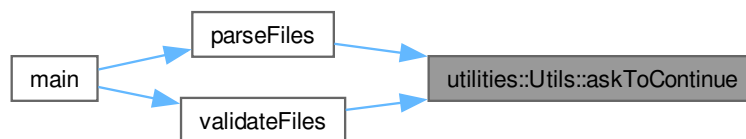
<i>prompt</i>	(Optional) A custom prompt to be used.
---------------	--

Returns

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

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

Here is the caller graph for this function:

**10.18.2.2 checkDirectory()**

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

Todo documentation

Definition at line 63 of file [Utils.cpp](#).

Here is the caller graph for this function:

**10.18.2.3 checkFileEnding()**

```
bool utilities::Utils::checkFileEnding (  
    const std::string_view & fileName ) [static]
```

Checks if the file ending is ".json".

This function checks if the given file ends with ".json".

Parameters

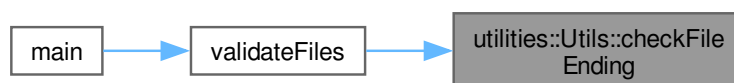
<i>fileName</i>	The file which should be checked.
-----------------	-----------------------------------

Returns

Returns true if the file ends with ".json" and false otherwise.

Definition at line 39 of file [Utils.cpp](#).

Here is the caller graph for this function:



10.18.2.4 checkIfFileExists()

```
bool utilities::Utils::checkIfFileExists (
    const std::string & fileName ) [static]
```

Check if a file exists.

This function checks if a file exists by trying to open it using `fstream`.

Parameters

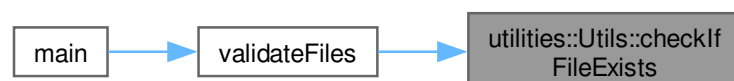
<i>fileName</i>	The file which should be checked.
-----------------	-----------------------------------

Returns

Returns true if the file exists and false otherwise

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

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.

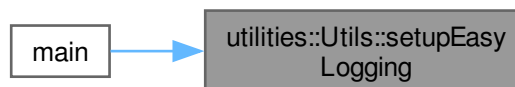
Parameters

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

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

References [HOMEPAGE_URL](#), [MAJOR_VERSION](#), [MINOR_VERSION](#), [PATCH_VERSION](#), and [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

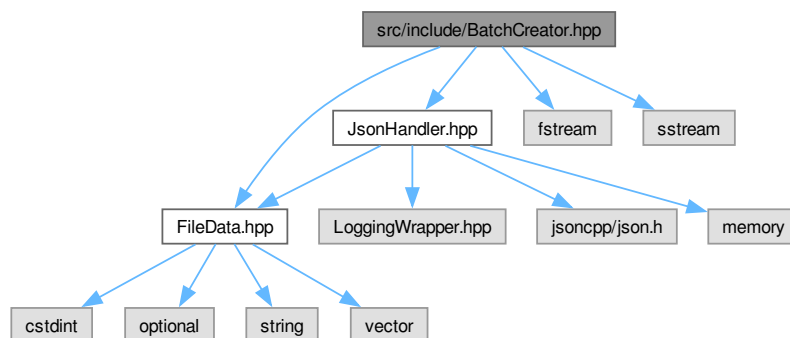
11.1 README.md File Reference

11.2 src/include/BatchCreator.hpp File Reference

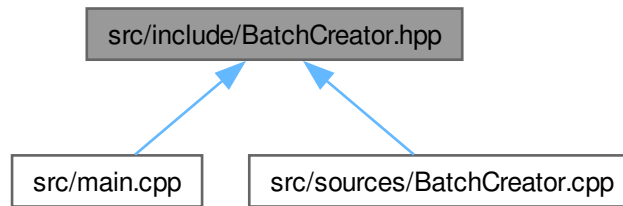
Creates batch file.

```
#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

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
00012 #include "FileData.hpp"
00013 #include "JsonHandler.hpp"
00014 #include <fstream>
00015 #include <sstream>
00016
00025 class BatchCreator {
00026 public:
00034     BatchCreator(std::shared_ptr<parsing::FileData> fileData);
00035
00037     std::shared_ptr<std::stringstream> getDataStream() {
00038         return dataStream;
00039     }
00040
00041 private:
00042     std::shared_ptr<std::stringstream> dataStream;
00043
00044     std::shared_ptr<parsing::FileData> fileData;
00045
00051     void createBatch();
00052
00060     void writeStart();
00061
00067     void writeHideShell();
00068
00075     void writeCommands();
00076
00083     void writeEnvVariables();
00084
00091     void writePathVariables();
00092
00099     void writeApp();
00100
00107     void writeEnd();
00108 };

```

11.4 src/include/CommandLineHandler.hpp File Reference

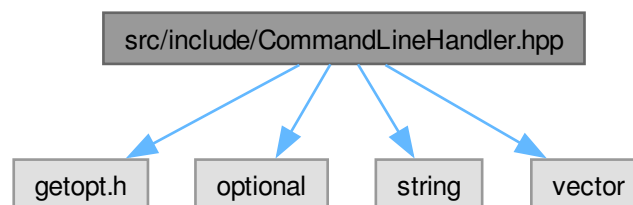
Responsible for the Command Line Interface.

```

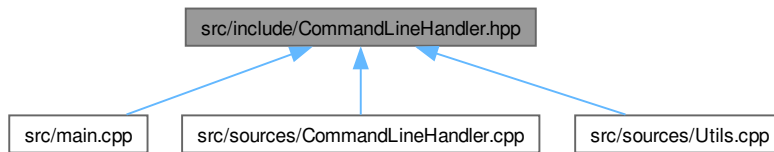
#include <getopt.h>
#include <optional>
#include <string>
#include <vector>

```

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.](#)

```

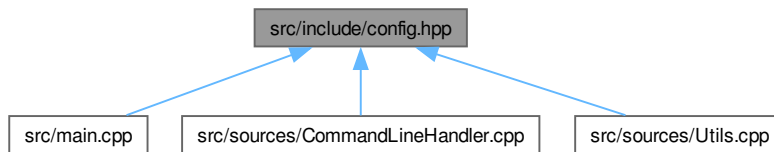
00001
00019 #ifndef COMMANDLINEHANDLER_HPP
00020 #define COMMANDLINEHANDLER_HPP
00021
00022 #include <getopt.h>
00023 #include <optional>
00024 #include <string>
00025 #include <vector>
00026
00039 namespace cli {
00040
00053 class CommandLineHandler {
00054 public:
00062     [[noreturn]] static void 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.

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



Macros

- `#define LOG_CONFIG` `"/home/simon/1_Coding/cpp/JsonToBat/build/Debug/config/easylogging.conf"`
- `#define EXECUTABLE_NAME` `"json2batch"`
- `#define MAJOR_VERSION` `"0"`
- `#define MINOR_VERSION` `"2"`
- `#define PATCH_VERSION` `"1"`
- `#define DESCRIPTION` `"A simple tool to convert json to batch."`
- `#define PROJECT_NAME` `"JSON2Batch"`
- `#define AUTHORS` `"Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci"`
- `#define HOMEPAGE_URL` `"https://definitelynotsimon13.github.io/ProjectJsonToBat"`

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](#).

11.6.2 Macro Definition Documentation

11.6.2.1 AUTHORS

```
#define AUTHORS "Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci"
```

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

11.6.2.2 DESCRIPTION

```
#define DESCRIPTION "A simple tool to convert json to batch."
```

Definition at line 25 of file [config.hpp](#).

11.6.2.3 EXECUTABLE_NAME

```
#define EXECUTABLE_NAME "json2batch"
```

Definition at line 21 of file [config.hpp](#).

11.6.2.4 HOMEPAGE_URL

```
#define HOMEPAGE_URL "https://definitelynotsimon13.github.io/ProjectJsonToBat"
```

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

11.6.2.5 LOG_CONFIG

```
#define LOG_CONFIG "/home/simon/1_Coding/cpp/JsonToBat/build/Debug/config/easylogging.conf"
```

Definition at line 20 of file [config.hpp](#).

11.6.2.6 MAJOR_VERSION

```
#define MAJOR_VERSION "0"
```

Definition at line 22 of file [config.hpp](#).

11.6.2.7 MINOR_VERSION

```
#define MINOR_VERSION "2"
```

Definition at line 23 of file [config.hpp](#).

11.6.2.8 PATCH_VERSION

```
#define PATCH_VERSION "1"
```

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

11.6.2.9 PROJECT_NAME

```
#define PROJECT_NAME "JSON2Batch"
```

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

11.7 config.hpp

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

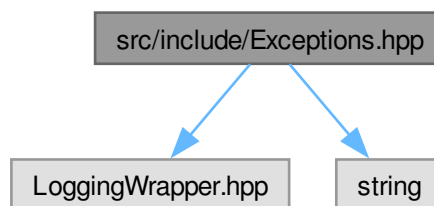
```
00001
00016 // This file is autogenerated. Changes will be overwritten
00017
00018 #ifndef CONFIG_HPP
00019 #define CONFIG_HPP
00020 #define LOG_CONFIG "/home/simon/1_Coding/cpp/JsonToBat/build/Debug/config/easylogging.conf"
00021 #define EXECUTABLE_NAME "json2batch"
00022 #define MAJOR_VERSION "0"
00023 #define MINOR_VERSION "2"
00024 #define PATCH_VERSION "1"
00025 #define DESCRIPTION "A simple tool to convert json to batch."
00026 #define PROJECT_NAME "JSON2Batch"
00027 #define AUTHORS "Elena Schwarzbach, Max Rodler, Simon Blum, Sonia Sinaci"
00028 #define HOMEPAGE_URL "https://definitelynotsimon13.github.io/ProjectJsonToBat"
00029 #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 };
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) {
00163         std::stringstream ss;
00164         ss << "Invalid type found at line " << line << ": \"" << type << "\"";
00165         this->message = ss.str();
00166         LOG_INFO << "InvalidTypeException: " << message;
00167     }
00168     [[nodiscard]] const char *what() const noexcept override {
00169         return message.c_str();
00170     }
00171 };
00172
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) {
00194         std::stringstream ss;
00195         ss << "Missing key \"" << key << "\" for type \"" << type << "\"!";
00196         this->message = ss.str();
00197         LOG_INFO << "MissingKeyException: " << message;
00198     }
00199     [[nodiscard]] const char *what() const noexcept override {
00200         return message.c_str();
00201     }
00202 };
00203
00215 class MissingTypeException : public CustomException {
00216 private:
00217     std::string message = "Missing \"type\" key for at least one entry!";
00218
00219 public:
00220     MissingTypeException() {
00221         LOG_INFO << "MissingTypeException: " << message;
00222     }
00223     [[nodiscard]] const char *what() const noexcept override {
00224         return message.c_str();
00225     }
00226 };
00227
00232 class UnreachableCodeException : public CustomException {
00233 private:
00234     std::string message;
00235
00236 public:
00237     explicit UnreachableCodeException(const std::string &message)
00238         : message(message) {
00239         LOG_INFO << "UnreachableCodeException: " << message;
00240     }
00241     [[nodiscard]] const char *what() const noexcept override {
00242         return message.c_str();
00243     }
00244 };
00245
00246 class FailedToOpenFileException : public CustomException {
00247 private:
00248     std::string message;
00249
00251 public:
00252     explicit FailedToOpenFileException(const std::string &file) {
00253         message = "Failed to open file: " + file;
00254         LOG_INFO << "FailedToOpenFileException: " << message;
00255     }
00256     [[nodiscard]] const char *what() const noexcept override {
00257         return message.c_str();
00258     }
00259 };
00260
00261 class NoSuchDirException : public CustomException {
00262 private:
00263     std::string message;
00264

```

```

00266 public:
00267     explicit NoSuchDirException(const std::string &dir) {
00268         message = "No such directory: " + dir;
00269         LOG_INFO << "NoSuchDirException: " << message;
00270     }
00271     [[nodiscard]] const char *what() const noexcept override {
00272         return message.c_str();
00273     }
00274 };
00275
00276 } // namespace exceptions
00277
00278 #endif

```

11.10 src/include/FileData.hpp File Reference

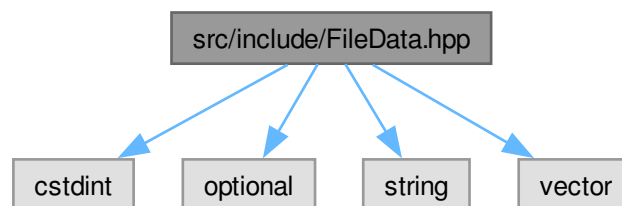
This file contains the FileData class.

```

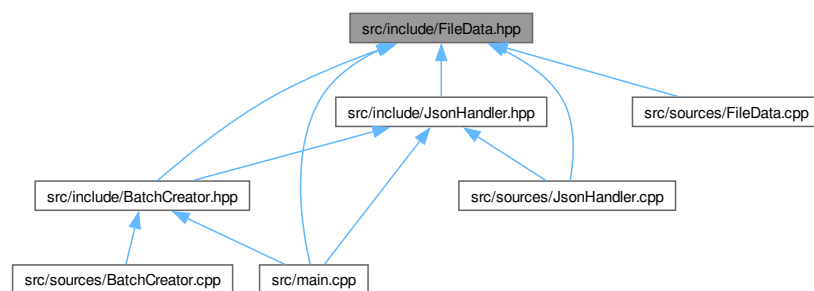
#include <cstdint>
#include <optional>
#include <string>
#include <vector>

```

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) {
00049         this->hideShell = newHideShell;
00050     }
00051
00060     void setApplication(const std::string &newApplication);
00061
00072     void addCommand(const std::string &command);
00073
00085     void addEnvironmentVariable(const std::string &name,
00086                                const std::string &value);
```

```

00087
00098 void addPathValue(const std::string &pathValue);
00099
00104 [[nodiscard]] const std::string &getOutputFile() const {
00105     return outputfile;
00106 }
00107
00112 [[nodiscard]] bool getHideShell() const {
00113     return hideShell;
00114 }
00115
00120 [[nodiscard]] const std::optional<std::string> &getApplication() const {
00121     return application;
00122 }
00123
00128 [[nodiscard]] const std::vector<std::string> &getCommands() const {
00129     return commands;
00130 }
00131
00136 [[nodiscard]] const std::vector<std::tuple<std::string, std::string> &
00137 getEnvironmentVariables() const {
00138     return environmentVariables;
00139 }
00140
00145 [[nodiscard]] const std::vector<std::string> &getPathValues() const {
00146     return pathValues;
00147 }
00148
00149 private:
00150     std::string outputfile;
00151     bool hideShell;
00152     std::optional<std::string> application;
00153     std::vector<std::string> commands;
00154     std::vector<std::tuple<std::string, std::string> > environmentVariables;
00155     std::vector<std::string> pathValues;
00156 };
00157 } // namespace parsing
00158
00159 #endif // FILEDATA_HPP

```

11.12 src/include/JsonHandler.hpp File Reference

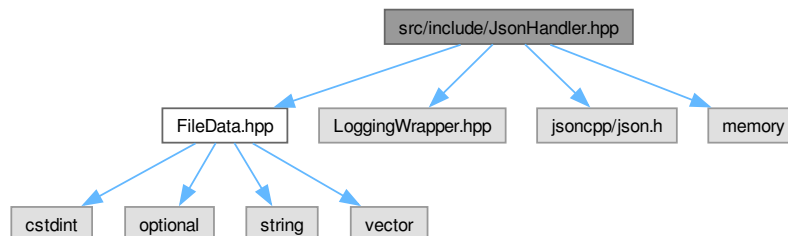
This file contains the JsonHandler class.

```

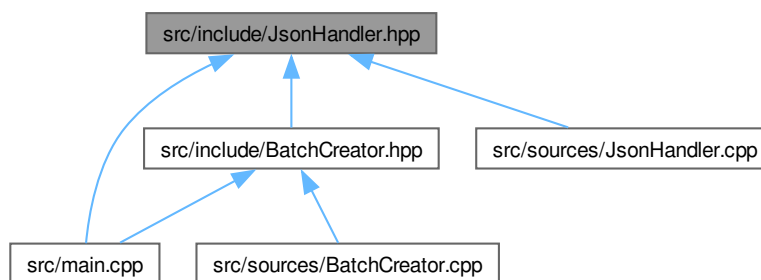
#include "FileData.hpp"
#include "LoggingWrapper.hpp"
#include <jsoncpp/json.h>
#include <memory>

```

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

```

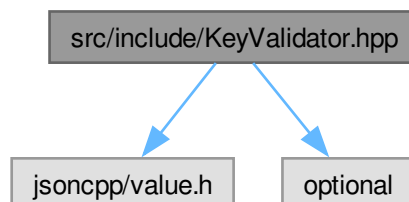
11.14 src/include/KeyValidator.hpp File Reference

This file contains the KeyValidator class.

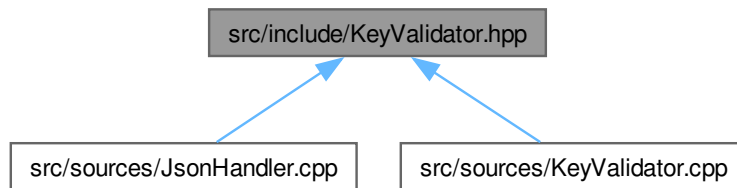
```
#include "jsoncpp/value.h"
```

```
#include <optional>
```

Include 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.](#)

```

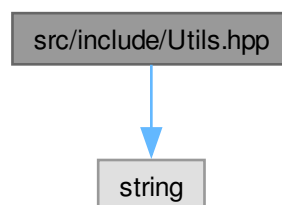
00001
00012 #ifndef KEYVALIDATOR_HPP
00013 #define KEYVALIDATOR_HPP
00014
00015 #include "jsoncpp/value.h"
00016 #include <optional>
00017 namespace parsing {
00026 class KeyValidator {
00027 public:
00033     static KeyValidator &getInstance();
00034
00048     std::vector<std::tuple<int, std::string>>
00049     validateKeys(const Json::Value &root, const std::string &filename);
00050
00051 private:
00064     std::vector<std::tuple<int, std::string>>
00065     getWrongKeys(const Json::Value& root, const std::string &filename);
00066
00077     static void validateTypes(const std::string &filename,
00078                             const Json::Value &entry,
00079                             std::vector<std::string> &entryKeys);
00080
00091     std::vector<std::tuple<int, std::string>>
00092     validateEntries(const std::string &filename,
00093                   const std::vector<std::string> &entryKeys);
00094
00105     static std::optional<int> getUnknownKeyLine(const std::string &filename,
00106                                                const std::string &wrongKey);
00107
00108     std::vector<std::string> validKeys = {"outputfile", "hideshell", "entries",
00109                                         "application"};
00110
00111     std::vector<std::string> validEntryKeys = {"type", "key", "value", "path",
00112                                               "command"};
00113 };
00114 };
00115 } // namespace parsing
00116
00117 #endif

```

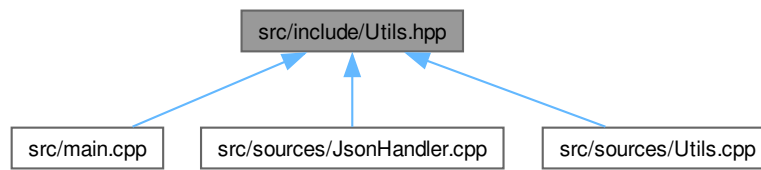
11.16 src/include/Utils.hpp File Reference

```
#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 <string>
00020
00030 namespace utilities {
00031
00039 class Utils {
00040 public:
00048     static void setupEasyLogging(const std::string &configFile);
00049
00057     static bool checkIfFileExists(const std::string &fileName);
00058
00066     static bool checkFileEnding(const std::string_view &fileName);
00067
00075     static bool
00076     askToContinue(const std::string &prompt = "Do you want to continue? (Y/N)\n");
00077
00079     static std::string &checkDirectory(std::string &directory);
00080 };
00081 } // namespace utilities
00082
00083 #endif // UTILITIES_HPP
  
```

11.18 src/main.cpp File Reference

Contains the main function.

```

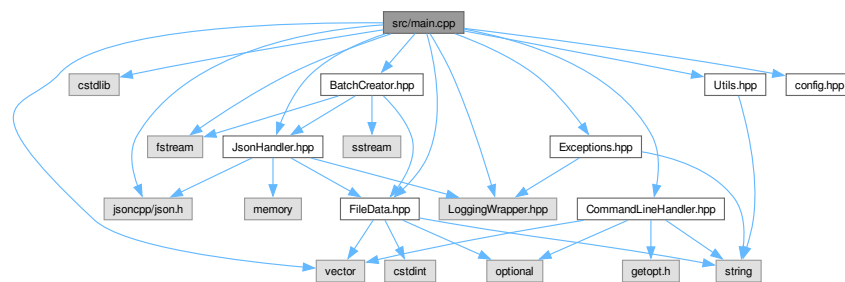
#include <LoggingWrapper.hpp>
#include <cstdlib>
  
```

```

#include <fstream>
#include <jsoncpp/json.h>
#include <vector>
#include "BatchCreator.hpp"
#include "CommandLineHandler.hpp"
#include "Exceptions.hpp"
#include "FileData.hpp"
#include "JsonHandler.hpp"
#include "Utils.hpp"
#include "config.hpp"

```

Include dependency graph for main.cpp:



Functions

- `INITIALIZE_EASYLOGGINGPP std::vector< std::string > validateFiles (std::vector< std::string > files)`
- `void parseFiles (std::vector< std::string > files, std::optional< std::string > outDir)`
- `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

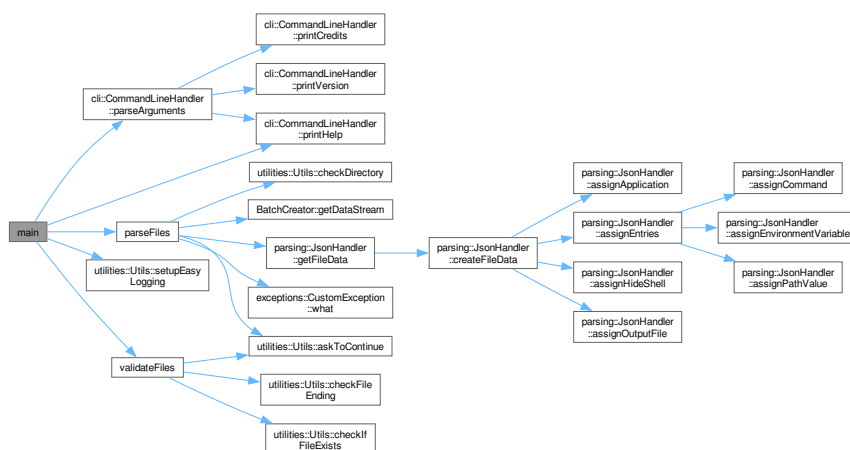
Todo Documentation

Refactoring

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

References [LOG_CONFIG](#), [cli::CommandLineHandler::parseArguments\(\)](#), [parseFiles\(\)](#), [cli::CommandLineHandler::printHelp\(\)](#), [utilities::Utils::setupEasyLogging\(\)](#), and [validateFiles\(\)](#).

Here is the call graph for this function:



11.18.2.2 parseFiles()

```
void parseFiles (
    std::vector< std::string > files,
    std::optional< std::string > outDir )
```

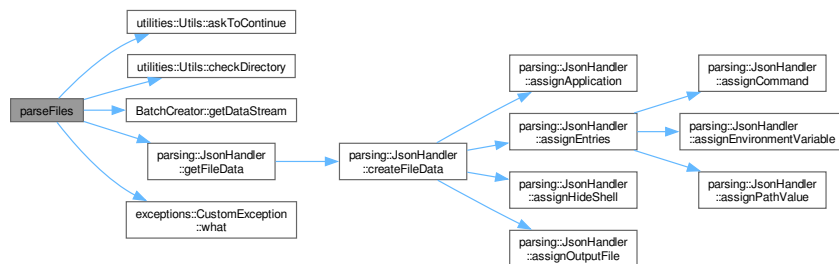
Parameters

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

Definition at line 143 of file [main.cpp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



11.18.2.3 validateFiles()

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

Parameters

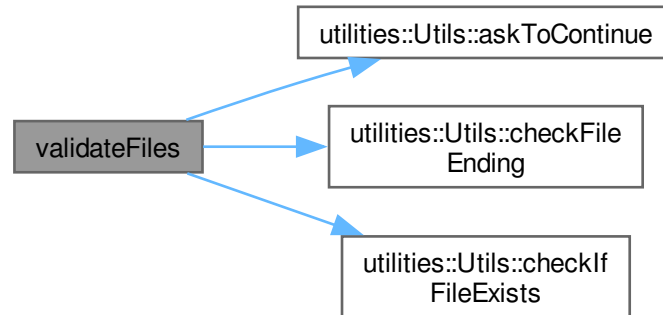
<i>files</i>	
--------------	--

Returns

Definition at line 105 of file [main.cpp](#).

References [utilities::Utils::askToContinue\(\)](#), [utilities::Utils::checkFileEnding\(\)](#), and [utilities::Utils::checkIfFileExists\(\)](#).

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 <fstream>
00016 #include <jsoncpp/json.h>
00017 #include <vector>
00018
00019 #include "BatchCreator.hpp"
00020 #include "CommandLineHandler.hpp"
00021 #include "Exceptions.hpp"
00022 #include "FileData.hpp"
00023 #include "JsonHandler.hpp"
00024 #include "Utils.hpp"
00025 #include "config.hpp"
00026
00027 INITIALIZE_EASYLOGGINGPP
00028
00036 std::vector<std::string> validateFiles(std::vector<std::string> files);
00037
00044 void parseFiles(std::vector<std::string> files,
00045                 std::optional<std::string> outDir);
00046
00062 int main(int argc, char *argv[]) {
00063     std::ifstream configFile(LOG_CONFIG);
00064     if (!configFile.good()) {
00065         std::cerr << cli::RED << cli::BOLD

```

```

00066         « "Fatal: Easylogging configuration file not found at:\n"
00067         « cli::RESET « cli::ITALIC « "\n\t\"" « LOG_CONFIG « "\"\n\n"
00068         « cli::RESET;
00069
00070         std::cout « "Aborting...\n";
00071         return 1;
00072     }
00073
00074     utilities::Utils::setupEasyLogging(LOG_CONFIG);
00075
00076     // Check if any options/arguments were given
00077     if (argc < 2) {
00078         LOG_ERROR « "No options given!\n";
00079         cli::CommandLineHandler::printHelp();
00080     }
00081
00082     // Vector of all inputted file names
00083     auto arguments = cli::CommandLineHandler::parseArguments(argc, argv);
00084
00085     std::vector<std::string> files = std::get<1>(arguments);
00086     std::optional<std::string> outDir = std::get<0>(arguments);
00087
00088     if (files.empty()) {
00089         LOG_ERROR « "No files were given as arguments!\n";
00090         return 1;
00091     }
00092     OUTPUT « cli::BOLD « "Parsing the following files:\n" « cli::RESET;
00093     for (const auto &file : files) {
00094         OUTPUT « "\t - " « file « "\n";
00095     }
00096
00097     // Replace the original files vector with the validFiles vector
00098     files = std::move(validateFiles(files));
00099     parseFiles(files, outDir);
00100
00101     LOG_INFO « "Exiting...";
00102     return 0;
00103 }
00104
00105 std::vector<std::string> validateFiles(std::vector<std::string> files) {
00106     std::vector<std::string> validFiles;
00107
00108     for (const auto &file : files) {
00109         if (!utilities::Utils::checkIfFileExists(file)) {
00110             LOG_ERROR « "The file \"" « file « "\" does not exist!\n";
00111
00112             if (files.size() != 1 &&
00113                 !utilities::Utils::askToContinue("Do you want to continue with the "
00114                 "remaining files? (y/n) ")) {
00115                 // Exit if it's the only file or the user does not want to
00116                 // continue
00117                 OUTPUT « "Aborting...\n";
00118                 LOG_INFO « "Application ended by user Input";
00119                 exit(1);
00120             }
00121
00122             continue;
00123         }
00124
00125         if (!utilities::Utils::checkFileEnding(file)) {
00126             LOG_WARNING « "The file \"" « file « "\" does not end in \".json\"\n";
00127             OUTPUT « "If the file is not in JSON Format, continuing may "
00128                 "result in\nunexpected behaviour!\n";
00129
00130             if (!utilities::Utils::askToContinue()) {
00131                 OUTPUT « "Aborting...\n";
00132                 LOG_INFO « "Application ended by user Input";
00133                 exit(1);
00134             }
00135         }
00136
00137         validFiles.push_back(file);
00138     }
00139
00140     return validFiles;
00141 }
00142
00143 void parseFiles(std::vector<std::string> files,
00144                 std::optional<std::string> outDir) {
00145
00146     for (auto file = files.begin(); file != files.end(); ++file) {
00147         OUTPUT « cli::ITALIC « "\nParsing file: " « *file « "... \n"
00148             « cli::RESET;
00149
00150         std::shared_ptr<parsing::FileData> fileData;
00151         try {
00152             parsing::JsonHandler jsonHandler(*file);

```

```

00153         fileData = jsonHandler.getFileData();
00154         BatchCreator batchCreator = BatchCreator(fileData);
00155         std::shared_ptr<std::stringstream> dataStream =
00156             batchCreator.getDataStream();
00157         std::ofstream outFile;
00158         std::string outValue;
00159         if (outDir.has_value()) {
00160             outValue = utilities::Utils::checkDirectory(outDir.value());
00161         }
00162         std::string fileName = outDir.has_value()
00163             ? outDir.value() + fileData->getOutputFile()
00164             : fileData->getOutputFile();
00165         outFile.open(fileName);
00166         if (!outFile.good()) {
00167             throw exceptions::FailedToOpenFileException(fileName);
00168         }
00169         outFile << dataStream->str();
00170         outFile.close();
00171     } catch (const exceptions::CustomException &e) {
00172         OUTPUT << "\nThere has been a error while trying to parse \"" << *file
00173             << ":\n";
00174         LOG_ERROR << e.what();
00175     }
00176     if (std::next(file) != files.end() &&
00177         !utilities::Utils::askToContinue(
00178             "Do you want to continue with the other files? (y/n) "
00179             "")) {
00180         OUTPUT << "Aborting...";
00181         LOG_INFO << "Application ended by user Input";
00182         exit(1);
00183     }
00184     std::cout << "\n";
00185     continue;
00186 }
00187 }
00188 }
00189 OUTPUT << "Done with files!\n";
00190 }

```

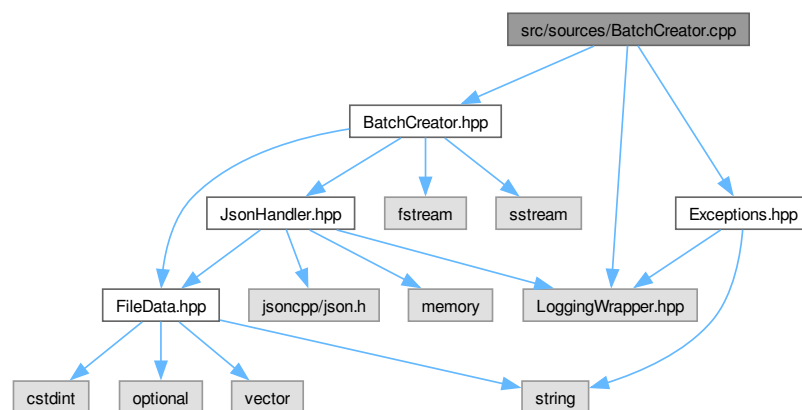
11.20 src/sources/BatchCreator.cpp File Reference

```

#include "BatchCreator.hpp"
#include "Exceptions.hpp"
#include "LoggingWrapper.hpp"

```

Include dependency graph for BatchCreator.cpp:



11.21 BatchCreator.cpp

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

```

00001
00012 #include "BatchCreator.hpp"
00013 #include "Exceptions.hpp"
00014 #include "LoggingWrapper.hpp"
00015
00016 BatchCreator::BatchCreator(std::shared_ptr< parsing::FileData> fileData) {
00017     LOG_INFO << "Initializing BatchCreator";
00018     this->fileData = fileData;
00019     this->dataStream = std::make_shared<std::stringstream>();
00020     this->createBatch();
00021 }
00022
00023 void BatchCreator::createBatch() {
00024     LOG_INFO << "Creating Batch file";
00025
00026     this->writeStart();
00027     this->writeHideShell();
00028     this->writeCommands();
00029     this->writeEnvVariables();
00030     this->writePathVariables();
00031     this->writeApp();
00032     this->writeEnd();
00033 }
00034
00035 void BatchCreator::writeStart() {
00036     LOG_INFO << "writing Start of Batch";
00037     *this->dataStream << "@ECHO OFF\r\nC:\\Windows\\System32\\cmd.exe ";
00038 }
00039
00040 void BatchCreator::writeHideShell() {
00041     if (this->fileData->getHideShell()) {
00042         LOG_INFO << "writing hide Shell";
00043         *this->dataStream << "/c ";
00044     } else {
00045         LOG_INFO << "writing show Shell";
00046         *this->dataStream << "/k ";
00047     }
00048 }
00049
00050
00051 void BatchCreator::writeCommands() {
00052     LOG_INFO << "writing Commands";
00053     *this->dataStream << "\n";
00054     for (const std::string &command : this->fileData->getCommands()) {
00055         *this->dataStream << command << " && ";
00056     }
00057 }
00058
00059 void BatchCreator::writeEnvVariables() {
00060     LOG_INFO << "writing Environment Variables";
00061     for (const std::tuple env : this->fileData->getEnvironmentVariables()) {
00062         *this->dataStream << "set " << std::get<0>(env) << "=" << std::get<1>(env)
00063             << " && ";
00064     }
00065 }
00066
00067 void BatchCreator::writePathVariables() {
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() {
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() {
00090     *this->dataStream << "@ECHO ON";
00091 }

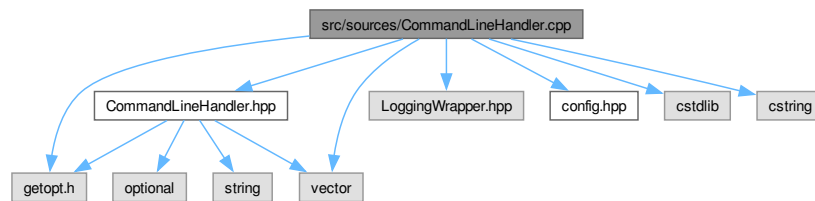
```

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.

11.22.1 Detailed Description

Implementation for the Command Line Interface.

Author

Simon Blum

Date

2024-04-18

Version

0.1.5

See also

`src/include/utility/CommandLineHandler.hpp`

Copyright

See LICENSE file

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

11.23 CommandLineHandler.cpp

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

```

00001
00013 #include "CommandLineHandler.hpp"
00014 #include "LoggingWrapper.hpp"
00015 #include "config.hpp"
00016 #include <cstdlib>
00017 #include <cstring>
00018 #include <getopt.h>
00019 #include <vector>
00020
00021 namespace cli {
00022 void CommandLineHandler::printHelp() {
00023     LOG_INFO << "Printing help message...";
00024     OUTPUT << BOLD << "Usage:\n"
00025         << RESET << "-----\n"
00026         << EXECUTABLE_NAME << " [options] [filenames]\n"
00027         << "\n"
00028         << BOLD << "Options:\n"
00029         << RESET << "-----\n"
00030         << "-o, --outdir\t [path]\t\tOutput the batch file to the given "
00031         << "dir\n"
00032         << "-h, --help\t\t\tPrint this help message\n"
00033         << "-v, --version\t\t\tPrint the version number\n"
00034         << "-c, --credits\t\t\tPrint the credits\n"
00035         << "    --verbose\t\t\tStart the application in verbose mode\n"
00036         << ITALIC
00037         << "        \t\t\tNote: Verbose flag should be passed first!\n\n"
00038         << RESET << BOLD << "Filenames:\n"
00039         << RESET << "-----\n"
00040         << "The json files to be processed into batch files.\n"
00041         << "Multiple files should be separated by spaces!\n\n";
00042     exit(0);
00043 }
00044 void CommandLineHandler::printVersion() {
00045     LOG_INFO << "Printing version number...";
00046     OUTPUT << PROJECT_NAME << " v" << MAJOR_VERSION << "." << MINOR_VERSION << "."
00047         << 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 << PROJECT_NAME << RESET << " v" << MAJOR_VERSION
00055         << "." << MINOR_VERSION << "." << PATCH_VERSION << "\n"
00056         << "\n"
00057         << DESCRIPTION << "\n"
00058         << "\n"
00059         << GREEN << "Authors: " << RESET << ITALIC << AUTHORS << RESET << "\n"
00060         << GREEN << "Documentation: " << RESET << ITALIC << HOMEPAGE_URL
00061         << RESET << GREEN << "\nContact: " << RESET << ITALIC
00062         << "simon21.blum@gmail.com" << "\n";
00063     exit(0);
00064 }
00065
00066 std::tuple<std::optional<std::string>, std::vector<std::string>>
00067 CommandLineHandler::parseArguments(int argc, char *argv[]) {
00068     LOG_INFO << "Parsing arguments...";
00069
00070     std::vector<std::string> files;
00071     std::optional<std::string> outDir;
00072
00073     while (true) {
00074         int optIndex = -1;
00075         struct option longOption = {};
00076         auto result = getopt_long(argc, argv, "hvco:", options, &optIndex);
00077
00078         if (result == -1) {
00079             LOG_INFO << "End of options reached";
00080             break;
00081         }
00082
00083         switch (result) {
00084             case '?':
00085                 LOG_ERROR << "Invalid Option (argument)\n";
00086                 CommandLineHandler::printHelp();
00087
00088             case 'h':
00089                 LOG_INFO << "Help option detected";
00090                 CommandLineHandler::printHelp();
00091
00092             case 'v':
00093                 LOG_INFO << "Version option detected";

```

```

00094         CommandLineHandler::printVersion();
00095
00096     case 'c':
00097         LOG_INFO << "Credit option detected";
00098         CommandLineHandler::printCredits();
00099
00100     case 'o':
00101         LOG_INFO << "Output option detected";
00102         outDir = optarg;
00103         break;
00104
00105     case 0:
00106         LOG_INFO << "Long option without short version detected";
00107         longOption = options[optIndex];
00108         LOG_INFO << "Option: " << longOption.name << " given";
00109
00110         if (longOption.has_arg) {
00111             LOG_INFO << " Argument: " << optarg;
00112         }
00113
00114         if (strcmp(longOption.name, "verbose") == 0) {
00115             logging::setVerboseMode(true);
00116             LOG_INFO << "Verbose mode activated";
00117         }
00118
00119         break;
00120
00121     default:
00122         LOG_ERROR << "Default case for options reached!";
00123         break;
00124     }
00125 }
00126
00127 LOG_INFO << "Options have been parsed";
00128 LOG_INFO << "Checking for arguments...";
00129
00130 while (optind < argc) {
00131     LOG_INFO << "Adding file: " << argv[optind];
00132     files.emplace_back(argv[optind++]);
00133 }
00134 LOG_DEBUG << files.size();
00135
00136 LOG_INFO << "Arguments and options have been parsed";
00137 return std::make_tuple(outDir, files);
00138 }
00139 } // namespace cli

```

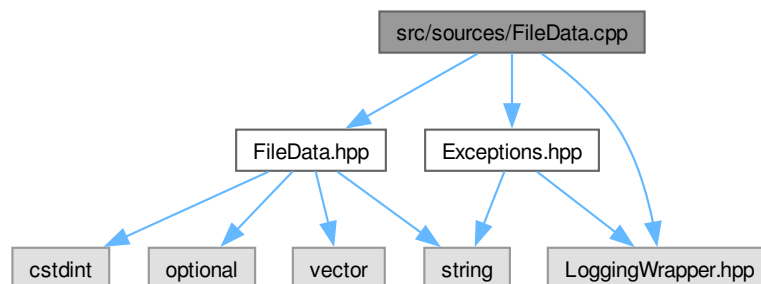
11.24 src/sources/FileData.cpp File Reference

```

#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 {
00019     LOG_INFO « "Setting outputfile to...";
00020
00021     // If no value for key "outputfile"
00022     if (newOutputfile.empty()) {
00023         LOG_INFO « "Escalating error to ErrorHandler::invalidValue!";
00024         throw exceptions::InvalidValueException("outputfile", "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", "Outputfile is already set!");
00031     }
00032
00033     // If outputfile does not end with ".bat"
00034     if (!newOutputfile.ends_with(".bat")) {
00035         newOutputfile += ".bat";
00036         LOG_WARNING « "Outputfile does not end with \".bat\", adding it now: "
00037                     « newOutputfile;
00038     }
00039
00040     this->outputfile = newOutputfile;
00041     LOG_INFO « "Outputfile set to: " « this->outputfile « "\n";
00042 }
00043
00044 void FileData::setApplication(const std::string &newApplication)
00045 {
```



```

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 {
00057     if (command.empty()) {
00058         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00059         throw exceptions::InvalidValueException("command", "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 {
00069     if (name.empty()) {
00070         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00071         throw exceptions::InvalidValueException("name", "Name value is empty!");
00072     }
00073
00074     if (value.empty()) {
00075         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00076         throw exceptions::InvalidValueException("key", "Key value is empty!");
00077     }
00078
00079     LOG_INFO << "Adding environment variable: " << name << "=" << value << "\n";
00080     this->environmentVariables.emplace_back(name, value);
00081 }
00082
00083 void FileData::addPathValue(const std::string &pathValue)
00084 {
00085     if (pathValue.empty()) {
00086         LOG_INFO << "Escalating error to ErrorHandler::invalidValue!";
00087         throw exceptions::InvalidValueException("path", "Path value is empty");
00088     }
00089
00090     LOG_INFO << "Adding path value: " << pathValue << "\n";
00091     this->pathValues.push_back(pathValue);
00092 }
00093 } // namespace parsing

```

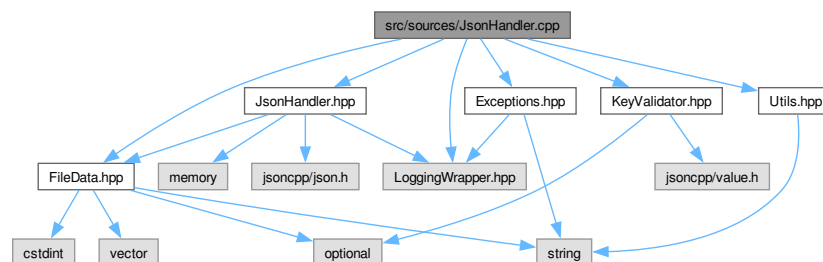
11.26 src/sources/JsonHandler.cpp File Reference

```

#include "JsonHandler.hpp"
#include "Exceptions.hpp"
#include "FileData.hpp"
#include "KeyValidator.hpp"
#include "LoggingWrapper.hpp"
#include "Utils.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 #include "Utils.hpp"
00018
00019 namespace parsing {
00020 JsonHandler::JsonHandler(const std::string &filename) {
00021     LOG_INFO << "Initializing JSONHandler with filename: " << filename << "\n";
00022     this->root = parseFile(filename);
00023 }
00024
00025 std::shared_ptr<Json::Value> JsonHandler::parseFile(const std::string &filename)
00026 {
00027     LOG_INFO << "Parsing file: " << filename << "\n";
00028     std::ifstream file(filename);
00029     Json::Value newRoot;
00030
00031     // Json::Reader.parse() returns false if parsing fails
00032     if (Json::Reader reader; !reader.parse(file, newRoot)) {
00033         throw exceptions::ParsingException(filename);
00034     }
00035
00036     // Validate keys
00037     // Check for errors
00038     if (auto errors = KeyValidator::getInstance().validateKeys(newRoot, filename);
00039         !errors.empty()) {
00040         throw exceptions::InvalidKeyException(errors);
00041     }
00042
00043     LOG_INFO << "File \"" << filename << "\" has been parsed\n";
00044     return std::make_shared<Json::Value>(newRoot);
00045 }
```

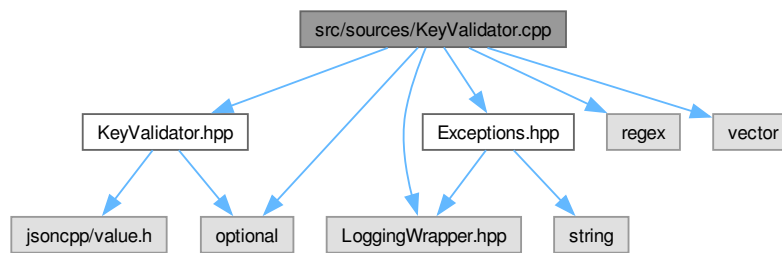
```

00046 }
00047
00048 std::shared_ptr<FileData> JsonHandler::getFileData() {
00049     LOG_INFO << "Creating FileData object for return...\n";
00050     return this->createFileData();
00051 }
00052
00053 std::shared_ptr<FileData> JsonHandler::createFileData() {
00054     LOG_INFO << "Creating FileData object...\n";
00055     this->data = std::make_shared<FileData>();
00056     this->assignOutputFile();
00057     this->assignHideShell();
00058     this->assignApplication();
00059     this->assignEntries();
00060     return this->data;
00061 }
00062
00063 void JsonHandler::assignOutputFile() const {
00064     LOG_INFO << "Assigning outputfile...\n";
00065     std::string outputFile = this->root->get("outputfile", "").asString();
00066
00067     this->data->setOutputFile(outputFile);
00068 }
00069
00070 void JsonHandler::assignHideShell() const {
00071     LOG_INFO << "Assigning hide shell...\n";
00072     // If the 'hideshell' key is not given, it defaults to false
00073     bool hideShell = this->root->get("hideshell", false).asBool();
00074     this->data->setHideShell(hideShell);
00075 }
00076
00077 void JsonHandler::assignApplication() const {
00078     LOG_INFO << "Assigning application...\n";
00079     std::string application = this->root->get("application", "").asString();
00080     this->data->setApplication(application);
00081 }
00082
00083 void JsonHandler::assignEntries() const {
00084     LOG_INFO << "Assigning entries...\n";
00085
00086     for (const auto &entry : this->root->get("entries", "")) {
00087         std::string entryType = entry.get("type", "").asString();
00088
00089         if (entryType == "EXE") {
00090             LOG_INFO << "Calling function to assign command...\n";
00091             this->assignCommand(entry);
00092         } else if (entryType == "ENV") {
00093             LOG_INFO << "Calling function to assign environment variable...\n";
00094             this->assignEnvironmentVariable(entry);
00095         } else if (entryType == "PATH") {
00096             LOG_INFO << "Calling function to assign path value...\n";
00097             this->assignPathValue(entry);
00098         } else {
00099             // Due to validation beforehand - this should never be reached!
00100             throw exceptions::UnreachableCodeException(
00101                 "Unknown entries should be caught by KeyValidator!\nPlease report "
00102                 "this bug!");
00103         }
00104     }
00105 }
00106
00107 void JsonHandler::assignCommand(const Json::Value &entry) const {
00108     LOG_INFO << "Assigning command...\n";
00109     std::string command = entry.get("command", "").asString();
00110     this->data->addCommand(command);
00111 }
00112
00113 void JsonHandler::assignEnvironmentVariable(const Json::Value &entry) const {
00114     LOG_INFO << "Assigning environment variable...\n";
00115     std::string key = entry.get("key", "").asString();
00116     std::string value = entry.get("value", "").asString();
00117     this->data->addEnvironmentVariable(key, value);
00118 }
00119
00120 void JsonHandler::assignPathValue(const Json::Value &entry) const {
00121     LOG_INFO << "Assigning path value...\n";
00122     std::string pathValue = entry.get("path", "").asString();
00123     this->data->addPathValue(pathValue);
00124 }
00125 } // 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
00018 namespace parsing {
00019 KeyValidator &KeyValidator::getInstance() {
00020     static KeyValidator keyValidator;
00021     LOG_INFO << "Returning KeyValidator instance!";
00022     return keyValidator;
00023 }
00024 std::vector<std::tuple<int, std::string>
00025 KeyValidator::validateKeys(const Json::Value &root,
00026                             const std::string &filename) {
00027
00028     // Initiate vector , with wrong keys at top leve
00029     std::vector<std::tuple<int, std::string> wrongKeys =
00030         getWrongKeys(root, filename);
00031
00032     // Go through the entry keys
00033     for (Json::Value entries = root.get("entries", "");
00034          const auto &entry : entries) {
00035
00036         // Retrieve all EntryKeys
00037         std::vector<std::string> entryKeys = entry.getMemberNames();
00038
00039         // Add all invalid entries to an array
00040         auto wrongEntries = validateEntries(filename, entryKeys);
00041
00042         // Append the invalid entries to the invalid keys
00043         wrongKeys.insert(wrongKeys.end(), wrongEntries.begin(), wrongEntries.end());
00044
00045         // Validate that each entry has it's necessary keys
00046         validateTypes(filename, entry, entryKeys);
00047     }
00048
00049     return wrongKeys;
00050 }
00051
00052 std::vector<std::tuple<int, std::string>
00053 KeyValidator::getWrongKeys(const Json::Value &root,
00054                             const std::string &filename) {
00055     std::vector<std::tuple<int, std::string> wrongKeys = {};
00056
00057     // Go through each key at top level
00058     for (std::vector<std::string> keys = root.getMemberNames();
00059          const auto &key : keys) {
00060         // Iterator tries to find the key within the valid keys
00061         auto keyIterator = std::ranges::find(validKeys, key);
00062
00063         // If the valid key isn't found, the iterator will be at the end
00064         if (keyIterator == validKeys.end()) {
00065             auto error = getUnknownKeyLine(filename, key);
00066
00067             if (!error.has_value()) {
00068                 LOG_ERROR << "Unable to find line of wrong key!";
00069                 continue;
00070             }
00071
00072             // Add the wrong key to the array
00073             wrongKeys.emplace_back(error.value_or(-1), key);
00074         }
00075     }
00076     return wrongKeys;
00077 }
00078
00079 std::vector<std::tuple<int, std::string>
00080 KeyValidator::validateEntries(const std::string &filename,
00081                             const std::vector<std::string> &entryKeys) {
00082     std::vector<std::tuple<int, std::string> wrongKeys = {};
00083
00084     // Go through each key within the entries
00085     for (const auto &key : entryKeys) {
00086         // try to find the key within the valid entry keys
00087         auto keyIterator = std::ranges::find(validEntryKeys, key);
00088
00089         // if the key isn't found, the iterator will be at the end
00090         if (keyIterator == validEntryKeys.end()) {
00091             auto error = getUnknownKeyLine(filename, key);

```

```

00092
00093         if (!error.has_value()) {
00094             LOG_ERROR « "Unable to find line of wrong key!";
00095             continue;
00096         }
00097
00098         // Add the wrong key to the array
00099         wrongKeys.emplace_back(error.value(), key);
00100     }
00101 }
00102
00103 return wrongKeys;
00104 }
00105
00106 void KeyValidator::validateTypes(const std::string &filename,
00107                                 const Json::Value &entry,
00108                                 std::vector<std::string> &entryKeys) {
00109     // Retrieve the type of the entry - ERROR if it can't be found
00110     std::string type = entry.get("type", "ERROR").asString();
00111
00112     if (type == "EXE") {
00113         // Try to find the "command" key
00114         if (auto commandIterator = std::ranges::find(entryKeys, "command");
00115             commandIterator == entryKeys.end()) {
00116             throw exceptions::MissingKeyException("command", "EXE");
00117         }
00118     } else if (type == "PATH") {
00119         // Try to find the "path" key
00120         if (auto pathIterator = std::ranges::find(entryKeys, "path");
00121             pathIterator == entryKeys.end()) {
00122             throw exceptions::MissingKeyException("path", "PATH");
00123         }
00124     } else if (type == "ENV") {
00125         // Try to find the "key" key
00126         if (auto keyIterator = std::ranges::find(entryKeys, "key");
00127             keyIterator == entryKeys.end()) {
00128             throw exceptions::MissingKeyException("key", "ENV");
00129         }
00130         // Try to find the "value" key
00131         if (auto valueIterator = std::ranges::find(entryKeys, "value");
00132             valueIterator == entryKeys.end()) {
00133             throw exceptions::MissingKeyException("value", "ENV");
00134         }
00135     } else if (type == "ERROR") {
00136         // If the "type" key wasn't found, throw an error
00137         throw exceptions::MissingTypeException();
00138     } else {
00139         // If the type wasn't any of the above, it's invalid
00140         std::optional<int> line = getUnknownKeyLine(filename, type);
00141
00142         if (!line.has_value()) {
00143             LOG_INFO « "Unable to find line of wrong type!";
00144         }
00145
00146         throw exceptions::InvalidTypeException(type, line.value());
00147     }
00148 }
00149
00150 std::optional<int>
00151 KeyValidator::getUnknownKeyLine(const std::string &filename,
00152                                 const std::string &wrongKey) {
00153     std::ifstream file(filename);
00154
00155     if (!file.is_open()) {
00156         LOG_ERROR « "File not open!";
00157     }
00158
00159     int lineNumber = 1;
00160     std::string errorLine;
00161     std::regex wrongKeyPattern("\\b" + wrongKey + "\\b");
00162
00163     // Go through each line of the file and search for the wrong key
00164     for (std::string line; std::getline(file, line);) {
00165         if (std::regex_search(line, wrongKeyPattern)) {
00166             errorLine = line;
00167             break;
00168         }
00169         ++lineNumber;
00170     }
00171
00172     if (errorLine.empty()) {
00173         return std::nullopt;
00174     }
00175
00176     return lineNumber;
00177 }
00178 }

```

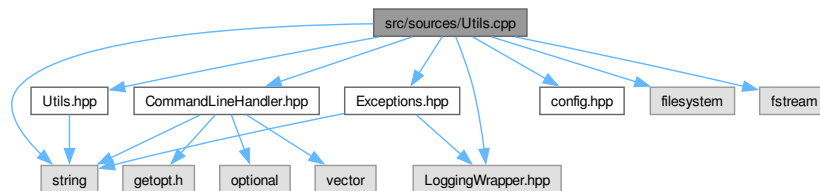
```
00179
00180 } // namespace parsing
```

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 <fstream>
#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 <fstream>
00023 #include <string>
00024
00025 namespace utilities {
00026 void Utils::setupEasyLogging(const std::string &configFile) {
00027     el::Configurations conf(configFile);
00028     el::Loggers::reconfigureAllLoggers(conf);
00029     LOG_INFO << "Running " << PROJECT_NAME << " v" << MAJOR_VERSION << "."
00030             << MINOR_VERSION << "." << PATCH_VERSION;
00031     LOG_INFO << "For more Information checkout " << HOMEPAGE_URL;
00032     LOG_INFO << "EasyLogging has been setup!";
00033 }
00034 bool Utils::checkIfFileExists(const std::string &fileName) {
00035     LOG_INFO << "Checking if file \"" << fileName << "\"exists...";
00036     std::ifstream file(fileName);
00037     return file.good();
00038 }
00039 bool Utils::checkFileEnding(const std::string_view &fileName) {
00040     return fileName.size() >= 5 && fileName.ends_with(".json");
00041 }
00042 bool Utils::askToContinue(const std::string &prompt) {
00043     std::string userInput;
00044     LOG_INFO << "Asking for user Confirmation to continue...";
00045     OUTPUT << cli::BOLD << prompt << cli::RESET;
00046
00047     do {
00048         std::cin >> userInput;
00049         std::ranges::transform(userInput, userInput.begin(), ::tolower);
00050
00051         if (userInput != "y" && userInput != "yes" && userInput != "n" &&
00052             userInput != "no") {
00053             LOG_INFO << "Wrong user input!";
00054             OUTPUT << cli::ITALIC << "Please enter Y/Yes or N/No!\n" << cli::RESET;
00055             continue;
00056         }
00057
00058         break;
00059     } while (true);
00060
00061     return userInput == "y" || userInput == "yes";
00062 }
00063 std::string &Utils::checkDirectory(std::string &directory) {
00064     if (!directory.empty() && directory.back() != '/' &&
00065         directory.back() != '\\') {
00066         directory += '/';
00067     }
00068
00069     if (!std::filesystem::exists(directory)) {
00070         throw exceptions::NoSuchDirException(directory);
00071     }
00072     return directory;
00073 }
00074 } // namespace utilities

```


Index

~CommandLineHandler
cli::CommandLineHandler, 29

addCommand
 parsing::FileData, 37
addEnvironmentVariable
 parsing::FileData, 37
addPathValue
 parsing::FileData, 38
application
 parsing::FileData, 41
askToContinue
 utilities::Utils, 77
assignApplication
 parsing::JsonHandler, 53
assignCommand
 parsing::JsonHandler, 53
assignEntries
 parsing::JsonHandler, 54
assignEnvironmentVariable
 parsing::JsonHandler, 55
assignHideShell
 parsing::JsonHandler, 55
assignOutputFile
 parsing::JsonHandler, 55
assignPathValue
 parsing::JsonHandler, 56
AUTHORS
 config.hpp, 87

BatchCreator, 21
 BatchCreator, 22
 createBatch, 22
 dataStream, 27
 fileData, 27
 getDataStream, 23
 writeApp, 24
 writeCommands, 24
 writeEnd, 24
 writeEnvVariables, 25
 writeHideShell, 25
 writePathVariables, 26
 writeStart, 26

checkDirectory
 utilities::Utils, 78
checkFileEnding
 utilities::Utils, 78
checkIfFileExists
 utilities::Utils, 79

cli, 17
 options, 18
cli::CommandLineHandler, 27
 ~CommandLineHandler, 29
 CommandLineHandler, 29
 parseArguments, 29
 printCredits, 30
 printHelp, 31
 printVersion, 31
CommandLineHandler
 cli::CommandLineHandler, 29
commands
 parsing::FileData, 41
config.hpp
 AUTHORS, 87
 DESCRIPTION, 87
 EXECUTABLE_NAME, 87
 HOMEPAGE_URL, 87
 LOG_CONFIG, 87
 MAJOR_VERSION, 87
 MINOR_VERSION, 87
 PATCH_VERSION, 87
 PROJECT_NAME, 88
createBatch
 BatchCreator, 22
createFileData
 parsing::JsonHandler, 56

data
 parsing::JsonHandler, 59
dataStream
 BatchCreator, 27
DESCRIPTION
 config.hpp, 87

environmentVariables
 parsing::FileData, 41
exceptions, 18
exceptions::CustomException, 32
 what, 34
exceptions::FailedToOpenFileException, 34
 FailedToOpenFileException, 35
 message, 36
 what, 36
exceptions::FileExistsException, 42
 file, 44
 FileExistsException, 43
 message, 44
 what, 44
exceptions::InvalidKeyException, 44

- InvalidKeyException, 46
 - message, 46
 - what, 46
- exceptions::InvalidTypeException, 46
 - InvalidTypeException, 48
 - message, 48
 - type, 48
 - what, 48
- exceptions::InvalidValueException, 49
 - InvalidValueException, 50
 - key, 51
 - message, 51
 - what, 50
- exceptions::MissingKeyException, 66
 - key, 68
 - message, 68
 - MissingKeyException, 68
 - type, 68
 - what, 68
- exceptions::MissingTypeException, 69
 - message, 70
 - MissingTypeException, 70
 - what, 70
- exceptions::NoSuchDirException, 71
 - message, 72
 - NoSuchDirException, 72
 - what, 72
- exceptions::ParsingException, 73
 - file, 75
 - message, 75
 - ParsingException, 74
 - what, 75
- exceptions::UnreachableCodeException, 75
 - message, 77
 - UnreachableCodeException, 76
 - what, 77
- EXECUTABLE_NAME
 - config.hpp, 87
- FailedToOpenFileException
 - exceptions::FailedToOpenFileException, 35
- file
 - exceptions::FileExistsException, 44
 - exceptions::ParsingException, 75
- fileData
 - BatchCreator, 27
- FileExistsException
 - exceptions::FileExistsException, 43
- getApplication
 - parsing::FileData, 38
- getCommands
 - parsing::FileData, 38
- getDataStream
 - BatchCreator, 23
- getEnvironmentVariables
 - parsing::FileData, 39
- getFileData
 - parsing::JsonHandler, 57
- getHideShell
 - parsing::FileData, 39
- getInstance
 - parsing::KeyValidator, 61
- getOutputFile
 - parsing::FileData, 39
- getPathValues
 - parsing::FileData, 39
- getUnknownKeyLine
 - parsing::KeyValidator, 61
- getWrongKeys
 - parsing::KeyValidator, 62
- hideShell
 - parsing::FileData, 41
- Homepage_URL
 - config.hpp, 87
- InvalidKeyException
 - exceptions::InvalidKeyException, 46
- InvalidTypeException
 - exceptions::InvalidTypeException, 48
- InvalidValueException
 - exceptions::InvalidValueException, 50
- JSON2Batch, 1
- JsonHandler
 - parsing::JsonHandler, 52
- key
 - exceptions::InvalidValueException, 51
 - exceptions::MissingKeyException, 68
- LOG_CONFIG
 - config.hpp, 87
- main
 - main.cpp, 101
- main.cpp
 - main, 101
 - parseFiles, 101
 - validateFiles, 102
- MAJOR_VERSION
 - config.hpp, 87
- message
 - exceptions::FailedToOpenFileException, 36
 - exceptions::FileExistsException, 44
 - exceptions::InvalidKeyException, 46
 - exceptions::InvalidTypeException, 48
 - exceptions::InvalidValueException, 51
 - exceptions::MissingKeyException, 68
 - exceptions::MissingTypeException, 70
 - exceptions::NoSuchDirException, 72
 - exceptions::ParsingException, 75
 - exceptions::UnreachableCodeException, 77
- MINOR_VERSION
 - config.hpp, 87
- MissingKeyException
 - exceptions::MissingKeyException, 68
- MissingTypeException

- exceptions::MissingTypeException, 70
- NoSuchDirException
 - exceptions::NoSuchDirException, 72
- options, 73
 - cli, 18
- outputfile
 - parsing::FileData, 41
- parseArguments
 - cli::CommandLineHandler, 29
- parseFile
 - parsing::JsonHandler, 58
- parseFiles
 - main.cpp, 101
- parsing, 19
- parsing::FileData, 36
 - addCommand, 37
 - addEnvironmentVariable, 37
 - addPathValue, 38
 - application, 41
 - commands, 41
 - environmentVariables, 41
 - getApplication, 38
 - getCommands, 38
 - getEnvironmentVariables, 39
 - getHideShell, 39
 - getOutputFile, 39
 - getPathValues, 39
 - hideShell, 41
 - outputfile, 41
 - pathValues, 42
 - setApplication, 40
 - setHideShell, 40
 - setOutputFile, 40
- parsing::JsonHandler, 51
 - assignApplication, 53
 - assignCommand, 53
 - assignEntries, 54
 - assignEnvironmentVariable, 55
 - assignHideShell, 55
 - assignOutputFile, 55
 - assignPathValue, 56
 - createFileData, 56
 - data, 59
 - getFileData, 57
 - JsonHandler, 52
 - parseFile, 58
 - root, 59
- parsing::KeyValidator, 60
 - getInstance, 61
 - getUnknownKeyLine, 61
 - getWrongKeys, 62
 - validateEntries, 63
 - validateKeys, 64
 - validateTypes, 64
 - validEntryKeys, 65
 - validKeys, 65
- ParsingException
 - exceptions::ParsingException, 74
- PATCH_VERSION
 - config.hpp, 87
- pathValues
 - parsing::FileData, 42
- printCredits
 - cli::CommandLineHandler, 30
- printHelp
 - cli::CommandLineHandler, 31
- printVersion
 - cli::CommandLineHandler, 31
- PROJECT_NAME
 - config.hpp, 88
- README.md, 81
- root
 - parsing::JsonHandler, 59
- setApplication
 - parsing::FileData, 40
- setHideShell
 - parsing::FileData, 40
- setOutputFile
 - parsing::FileData, 40
- setupEasyLogging
 - utilities::Utils, 79
- src/include/BatchCreator.hpp, 81, 83
- src/include/CommandLineHandler.hpp, 83, 85
- src/include/config.hpp, 85, 88
- src/include/Exceptions.hpp, 88, 90
- src/include/FileData.hpp, 92, 93
- src/include/JsonHandler.hpp, 94, 96
- src/include/KeyValidator.hpp, 96, 98
- src/include/Utils.hpp, 98, 99
- src/main.cpp, 99, 103
- src/sources/BatchCreator.cpp, 105
- src/sources/CommandLineHandler.cpp, 107, 108
- src/sources/FileData.cpp, 109, 110
- src/sources/JsonHandler.cpp, 111, 112
- src/sources/KeyValidator.cpp, 114, 115
- src/sources/Utils.cpp, 117, 118
- StyleHelpers, 15
- Todo List, 3
- type
 - exceptions::InvalidTypeException, 48
 - exceptions::MissingKeyException, 68
- UnreachableCodeException
 - exceptions::UnreachableCodeException, 76
- utilities, 19
- utilities::Utils, 77
 - askToContinue, 77
 - checkDirectory, 78
 - checkFileEnding, 78
 - checkIfFileExists, 79
 - setupEasyLogging, 79
- validateEntries

- `validateFiles`
 - `main.cpp`, 102
 - `validateKeys`
 - `parsing::KeyValidator`, 64
 - `validateTypes`
 - `parsing::KeyValidator`, 64
 - `validEntryKeys`
 - `parsing::KeyValidator`, 65
 - `validKeys`
 - `parsing::KeyValidator`, 65
- `what`
 - `exceptions::CustomException`, 34
 - `exceptions::FailedToOpenFileException`, 36
 - `exceptions::FileExistsException`, 44
 - `exceptions::InvalidKeyException`, 46
 - `exceptions::InvalidTypeException`, 48
 - `exceptions::InvalidValueException`, 50
 - `exceptions::MissingKeyException`, 68
 - `exceptions::MissingTypeException`, 70
 - `exceptions::NoSuchDirException`, 72
 - `exceptions::ParsingException`, 75
 - `exceptions::UnreachableCodeException`, 77
- `writeApp`
 - `BatchCreator`, 24
- `writeCommands`
 - `BatchCreator`, 24
- `writeEnd`
 - `BatchCreator`, 24
- `writeEnvVariables`
 - `BatchCreator`, 25
- `writeHideShell`
 - `BatchCreator`, 25
- `writePathVariables`
 - `BatchCreator`, 26
- `writeStart`
 - `BatchCreator`, 26