customCommands

1.0.0

Generated by Doxygen 1.13.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Command Class Reference

Represents a command in the parser framework.

```
#include <Command.h>
```

Inheritance diagram for Command:

classCommand-eps-converted-to.pdf

Public Member Functions

- Command (const std::string &name, const std::vector< std::string > &aliases, size_t nbOfArguments, const std::string &description, bool isMandatory, bool activateImmediately)

  Constructs a new Command object.
- virtual ∼Command ()=default

  Virtual destructor for Command.
- virtual void setArguments (const std::vector< std::string > &args)=0

  Sets the arguments parsed by the parser.
- virtual void execute ()=0

  Executes the command's main logic.
- const std::string & name () const

  Retrieves the name of the command.
- std::string description () const

  Retrieves a formatted description of the command.
- const std::vector< std::string > & aliases () const

  Retrieves the aliases of the command.
- bool isMandatoryCommand () const

  Checks if the command is mandatory.
- bool executesNow () const

  Checks if the command is set to execute immediately after parsing.
- std::size_t nbArguments () const

  Retrieves the number of arguments required by the command.

Protected Attributes

- std::string c_name

  The primary name of the command.
- std::vector< std::string > c_aliases

  A list of aliases for the command.
- size_t c_nbOfArguments

  The number of arguments the command expects.
- std::string c_description

  A short description of the command, displayed in the help menu.
- bool c_isMandatory

  Indicates if the command is mandatory.
- bool c_activateImmediately

  Determines if the command can be executed immediately after being parsed.

4.1.1 Detailed Description

Represents a command in the parser framework.

This abstract class serves as a base for creating commands that can be parsed, validated, and executed by the parser. Each command can have aliases, a description, arguments, and specific execution behavior.

Authors

- Paul Caillé
- Oregan Hardy

Version

1.0.0

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Command()

```
Command::Command (
            const std::string & name,
            const std::vector< std::string > & aliases,
            size_t nbOfArguments,
            const std::string & description,
            bool isMandatory,
            bool activateImmediately)
```

Constructs a new Command object.

Parameters

name	The primary name of the command.
aliases	A vector of aliases for the command.
nbOfArguments	The number of arguments required by the command.
description	A description of the command's functionality.
isMandatory	Whether the command is mandatory.
activateImmediately	Whether the command can execute directly upon parsing.

4.1.2.2 ∼Command()

```
virtual Command::∼Command ()  [virtual], [default]
```

Virtual destructor for Command.

The `Parsing` class deletes commands when it is destroyed.

4.1.3 Member Function Documentation

4.1.3.1 aliases()

```
const std::vector< std::string > & Command::aliases () const
```

Retrieves the aliases of the command.

Returns

A constant reference to the vector of aliases.

4.1.3.2 description()

```
std::string Command::description () const
```

Retrieves a formatted description of the command.

Combines the name, aliases, and description into a single string.

Returns

A string containing the command description.

4.1.3.3 execute()

```
virtual void Command::execute ()  [pure virtual]
```

Executes the command's main logic.

This is invoked by the parser after validation.

Implemented in HelpCommand.

4.1.3.4 executesNow()

```
bool Command::executesNow () const
```

Checks if the command is set to execute immediately after parsing.

Returns

`true` if the command executes immediately, `false` otherwise.

4.1.3.5 isMandatoryCommand()

```
bool Command::isMandatoryCommand () const
```

Checks if the command is mandatory.

Returns

`true` if the command is mandatory, `false` otherwise.

4.1.3.6 name()

```
const std::string & Command::name () const
```

Retrieves the name of the command.

Returns

The name of the command as a string.

4.1.3.7 nbArguments()

```
std::size_t Command::nbArguments () const
```

Retrieves the number of arguments required by the command.

Returns

The number of arguments as a `size_t`.

4.1.3.8 setArguments()

```
virtual void Command::setArguments (
             const std::vector< std::string > & args)  [pure virtual]
```

Sets the arguments parsed by the parser.

Parameters

args	A vector of arguments to set.

Implemented in HelpCommand.

4.1.4 Member Data Documentation

4.1.4.1 c_activateImmediately

```
bool Command::c_activateImmediately  [protected]
```

Determines if the command can be executed immediately after being parsed.

4.1.4.2 c_aliases

```
std::vector<std::string> Command::c_aliases  [protected]
```

A list of aliases for the command.

Aliases must begin with "-" for short names;

4.1.4.3 c_description

```
std::string Command::c_description  [protected]
```

A short description of the command, displayed in the help menu.

4.1.4.4 c_isMandatory

```
bool Command::c_isMandatory  [protected]
```

Indicates if the command is mandatory.

4.1.4.5 c_name

```
std::string Command::c_name  [protected]
```

The primary name of the command.

4.1.4.6 c_nbOfArguments

```
size_t Command::c_nbOfArguments  [protected]
```

The number of arguments the command expects.

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

- commands/Command.h
- commands/Command.cpp

4.2 HelpCommand Class Reference

A concrete implementation of the `Command` class for displaying help information.

```
#include <Command.h>
```

Inheritance diagram for HelpCommand:

classHelpCommand-eps-converted-to.pdf

Public Member Functions

- HelpCommand (const Parsing &parser)

  Constructs a `HelpCommand` object.
- void setArguments (const std::vector< std::string > &args) override

  Sets arguments for the command.
- void execute () override

  Executes the `HelpCommand`.

Public Member Functions inherited from Command

- Command (const std::string &name, const std::vector< std::string > &aliases, size_t nbOfArguments, const std::string &description, bool isMandatory, bool activateImmediately)

  Constructs a new Command object.
- virtual ∼Command ()=default

  Virtual destructor for Command.
- const std::string & name () const

  Retrieves the name of the command.
- std::string description () const

  Retrieves a formatted description of the command.
- const std::vector< std::string > & aliases () const

  Retrieves the aliases of the command.
- bool isMandatoryCommand () const

  Checks if the command is mandatory.
- bool executesNow () const

  Checks if the command is set to execute immediately after parsing.
- std::size_t nbArguments () const

  Retrieves the number of arguments required by the command.

Additional Inherited Members

Protected Attributes inherited from Command

- std::string c_name

  The primary name of the command.
- std::vector< std::string > c_aliases

  A list of aliases for the command.
- size_t c_nbOfArguments

  The number of arguments the command expects.
- std::string c_description

  A short description of the command, displayed in the help menu.
- bool c_isMandatory

  Indicates if the command is mandatory.
- bool c_activateImmediately

  Determines if the command can be executed immediately after being parsed.

4.2.1 Detailed Description

A concrete implementation of the `Command` class for displaying help information.

The `HelpCommand` displays usage instructions and a list of available commands within the parser.

Authors

- Paul Caillé
- Oregan Hardy

Version

1.0.0

4.2.2 Constructor & Destructor Documentation

4.2.2.1 HelpCommand()

```
HelpCommand::HelpCommand (
            const Parsing & parser)  [explicit]
```

Constructs a `HelpCommand` object.

Parameters

parser	A reference to the parser containing the commands.

4.2.3 Member Function Documentation

4.2.3.1 execute()

```
void HelpCommand::execute ()  [override], [virtual]
```

Executes the `HelpCommand`.

Displays the help message with usage and descriptions of all commands in the parser.

Implements Command.

4.2.3.2 setArguments()

```
void HelpCommand::setArguments (
            const std::vector< std::string > & args)  [override], [virtual]
```

Sets arguments for the command.

Overrides the base class method. Throws an exception if arguments are provided, as the `HelpCommand` does not accept any arguments.

Parameters

args	A vector of arguments.

Exceptions

std::runtime_error	If the vector of arguments is not empty.

Implements Command.

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

- commands/Command.h
- commands/Command.cpp

4.3 Parsing Class Reference

Class responsible for parsing command-line input, managing commands, and handling targets.

```
#include <Parsing.h>
```

Public Member Functions

- Parsing (Targets &targets)

  Constructs a Parsing object.
- void addCommand (Command *command)

  Adds a command to the parser.
- void parseInput (int argc, const char *argv[ ]) const

  Parses input arguments and processes commands and targets.
- std::vector< std::string > allCommandDescriptions () const

  Retrieves descriptions of all commands added to the parser.
- std::string generateUsage () const

  Generates the usage string for the parser.
- bool hasCommand (const std::string &name) const

  Checks if a specific command exists in the parser.
- std::string executableName () const

  Retrieves the name of the executable.
- const Targets & targets () const

  Retrieves the Targets object storing parsed targets.
- ∼Parsing ()

  Destructor for Parsing.

4.3.1 Detailed Description

Class responsible for parsing command-line input, managing commands, and handling targets.

This class represents a command-line parser that supports adding commands, parsing input arguments, and managing execution flow. It also validates mandatory commands and organizes the parsed targets.

Authors

- Paul Caillé
- Oregan Hardy

Version

1.0.0

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Parsing()

```
Parsing::Parsing (
            Targets & targets)  [explicit]
```

Constructs a Parsing object.

Parameters

targets	A reference to a Targets object where parsed targets will be stored.

4.3.2.2 ∼Parsing()

```
Parsing::∼Parsing ()
```

Destructor for Parsing.

Frees the memory allocated for the commands stored in the parser.

4.3.3 Member Function Documentation

4.3.3.1 addCommand()

```
void Parsing::addCommand (
            Command * command)
```

Adds a command to the parser.

Parameters

command	A pointer to the Command to add. Must not be nullptr.

4.3.3.2 allCommandDescriptions()

```
std::vector< std::string > Parsing::allCommandDescriptions () const
```

Retrieves descriptions of all commands added to the parser.

Generates a vector of command descriptions, including aliases and their purpose.

Returns

A vector of strings, each containing a command description.

4.3.3.3 executableName()

```
std::string Parsing::executableName () const
```

Retrieves the name of the executable.

This is typically the first element in the argument vector.

Returns

A string containing the executable's name.

4.3.3.4 generateUsage()

```
std::string Parsing::generateUsage () const
```

Generates the usage string for the parser.

This string details the expected input format, including commands, arguments, and targets.

Returns

A string representing the usage format of the program.

4.3.3.5 hasCommand()

```
bool Parsing::hasCommand (
             const std::string & name) const
```

Checks if a specific command exists in the parser.

Parameters

name	The name or alias of the command to check.

Returns

`true` if the command exists, `false` otherwise.

4.3.3.6 parseInput()

```
void Parsing::parseInput (
            int argc,
            const char * argv[ ]) const
```

Parses input arguments and processes commands and targets.

This is the main method for parsing command-line input. It organizes commands, validates arguments, and stores targets as needed.

Parameters

argc	The argument count (from the command line).
argv	The argument vector (from the command line).

Exceptions

std::runtime_error	if parsing errors occur (e.g., unrecognized commands, missing arguments).

4.3.3.7 targets()

```
const Targets & Parsing::targets () const
```

Retrieves the Targets object storing parsed targets.

Returns

A constant reference to the Targets object.

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

- parsing/Parsing.h
- parsing/Parsing.cpp

4.4 Targets Class Reference

Represents the collection of targets for the program, such as file paths.

```
#include <Targets.h>
```

Public Types

- using const_iterator = std::vector<std::string>::const_iterator

  Type alias for a constant iterator over the targets vector.

Public Member Functions

- Targets (bool canBeEmpty, const std::string &description)

  Constructs a Targets object.
- bool canBeEmpty () const

  Checks if the targets list can be empty.
- const std::vector< std::string > & targets () const

  Retrieves the list of targets.
- void addTarget (const std::string &targ)

  Adds a new target to the list of targets.
- bool empty () const

  Checks if the targets list is empty.
- const_iterator begin () const

  Returns an iterator pointing to the beginning of the targets list.
- const_iterator end () const

  Returns an iterator pointing to the end of the targets list.

Friends

- std::ostream & operator<< (std::ostream &os, const Targets &targets)

  Overloads the << operator to print the targets.

4.4.1 Detailed Description

Represents the collection of targets for the program, such as file paths.

This class manages a list of targets and provides functionality to add, iterate, and check properties of the targets. It also includes metadata such as a description and whether the list can be empty.

Authors

- Paul Caillé
- Oregan Hardy

Version

1.0.0

4.4.2 Member Typedef Documentation

4.4.2.1 const_iterator

```
using Targets::const_iterator = std::vector<std::string>::const_iterator
```

Type alias for a constant iterator over the targets vector.

4.4.3 Constructor & Destructor Documentation

4.4.3.1 Targets()

```
Targets::Targets (
            bool canBeEmpty,
            const std::string & description)
```

Constructs a Targets object.

Parameters

canBeEmpty	Specifies if the targets list can be empty.
description	A description of the targets.

4.4.4 Member Function Documentation

4.4.4.1 addTarget()

```
void Targets::addTarget (
            const std::string & targ)
```

Adds a new target to the list of targets.

Parameters

targ	The target to add.

4.4.4.2 begin()

```
Targets::const_iterator Targets::begin () const
```

Returns an iterator pointing to the beginning of the targets list.

Returns

A constant iterator to the start of the vector.

4.4.4.3 canBeEmpty()

```
bool Targets::canBeEmpty () const
```

Checks if the targets list can be empty.

Returns

`true` if the targets list can be empty, `false` otherwise.

4.4.4.4 empty()

```
bool Targets::empty () const
```

Checks if the targets list is empty.

Returns

`true` if the list is empty, `false` otherwise.

4.4.4.5 end()

```
Targets::const_iterator Targets::end () const
```

Returns an iterator pointing to the end of the targets list.

Returns

A constant iterator to the end of the vector.

4.4.4.6 targets()

```
const std::vector< std::string > & Targets::targets () const
```

Retrieves the list of targets.

Returns

A constant reference to the vector of targets.

4.4.5 Friends And Related Symbol Documentation

4.4.5.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Targets & targets) [friend]
```

Overloads the << operator to print the targets.

Outputs the description and the list of targets to the given output stream.

Parameters

os	The output stream.
targets	The Targets object to print.

Returns

A reference to the output stream.

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

- target/Targets.h
- target/Targets.cpp

Chapter 5

File Documentation

5.1 commands/Command.cpp File Reference

```
#include "Command.h"
#include "../parsing/Parsing.h"
#include <iostream>
```

5.2 commands/Command.h File Reference

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

Classes

- class Command

  Represents a command in the parser framework.
- class HelpCommand

  A concrete implementation of the `Command` class for displaying help information.

5.3 Command.h

Go to the documentation of this file.

```
//
// Created on 29/11/2024.
// CAILLE / HARDY
// PAUL / OREGAN
// M1 - CL
//

#ifndef COMMAND_H
#define COMMAND_H

#include <string>
#include <vector>

class Parsing;

class Command {
protected:
    std::string c_name;

    std::vector<std::string> c_aliases;

    size_t c_nbOfArguments;

    std::string c_description;

    bool c_isMandatory;

    bool c_activateImmediately;

public:
    Command(const std::string &name, const std::vector<std::string> &aliases, size_t nbOfArguments,
            const std::string &description, bool isMandatory, bool activateImmediately);

    virtual ~Command() = default;

    virtual void setArguments(const std::vector<std::string> &args) = 0;

    virtual void execute() = 0;

    const std::string &name() const;

    std::string description() const;

    const std::vector<std::string> &aliases() const;

    bool isMandatoryCommand() const;

    bool executesNow() const;

    std::size_t nbArguments() const;
};

class HelpCommand final : public Command {
    const Parsing &parser;

public:
    explicit HelpCommand(const Parsing &parser);

    void setArguments(const std::vector<std::string> &args) override;

    void execute() override;
};

#endif // COMMAND_H
```

5.4 parsing/Parsing.cpp File Reference

```
#include "Parsing.h"
#include <iostream>
#include <ostream>
```

5.5 parsing/Parsing.h File Reference

```
#include <vector>
#include "../commands/Command.h"
#include "../target/Targets.h"
```

Classes

- class Parsing

  Class responsible for parsing command-line input, managing commands, and handling targets.

5.6 Parsing.h

Go to the documentation of this file.

```
//
// Created on 29/11/2024.
// CAILLE / HARDY
// PAUL / OREGAN
// M1 - CL
//

#ifndef PARSING_H
#define PARSING_H
#include <vector>

#include "../commands/Command.h"
#include "../target/Targets.h"

class Parsing {
    Targets &p_targets;

    std::vector<Command *> p_commandsToParse;

    mutable std::string p_exename;

    Command *findCommand(const std::string &name) const;

    void executeAll() const;

    bool checkMissingMandatory(const std::vector<std::string> &inputParts) const;

public:

    explicit Parsing(Targets &targets);

    void addCommand(Command *command);

    void parseInput(int argc, const char *argv[]) const;

    std::vector<std::string> allCommandDescriptions() const;

    std::string generateUsage() const;

    bool hasCommand(const std::string &name) const;

    std::string executableName() const;

    const Targets &targets() const;

    ~Parsing();
};

#endif //PARSING_H
```

5.7 target/Targets.cpp File Reference

```
#include <ostream>
#include "Targets.h"
```

Functions

- std::ostream & operator<< (std::ostream &os, const Targets &targets)

5.7.1 Function Documentation

5.7.1.1 operator<<()

```
std::ostream & operator<< (
              std::ostream & os,
              const Targets & targets)
```

Outputs the description and the list of targets to the given output stream.

Parameters

os	The output stream.
targets	The Targets object to print.

Returns

A reference to the output stream.

5.8 target/Targets.h File Reference

```
#include <string>
#include <vector>
#include <ostream>
```

Classes

- class Targets

  Represents the collection of targets for the program, such as file paths.

5.9 Targets.h

Go to the documentation of this file.

```
//
// Created on 29/11/2024.
// CAILLE / HARDY
// PAUL / OREGAN
// M1 - CL
//

#ifndef TARGETS_H
#define TARGETS_H

#include <string>
#include <vector>
#include <ostream>

class Targets {
    std::string t_description;

    std::vector<std::string> t_targs;

    bool t_canBeEmpty;

public:
    using const_iterator = std::vector<std::string>::const_iterator;

    Targets(bool canBeEmpty, const std::string &description);

    bool canBeEmpty() const;

    const std::vector<std::string> &targets() const;

    void addTarget(const std::string &targ);

    bool empty() const;

    const_iterator begin() const;

    const_iterator end() const;

    friend std::ostream &operator«(std::ostream &os, const Targets &targets);
};

#endif // TARGETS_H
```