

x++todo

Generated by Doxygen 1.9.1

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 ArgumentReader Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Function Documentation	7
4.1.2.1 parseArguments()	7
4.1.2.2 readConfig()	8
4.1.2.3 saveConfig()	8
4.2 ChangeTask Class Reference	9
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	9
4.2.2.1 ChangeTask()	9
4.2.3 Member Function Documentation	10
4.2.3.1 task()	10
4.3 Date Struct Reference	10
4.3.1 Detailed Description	11
4.3.2 Constructor & Destructor Documentation	11
4.3.2.1 Date()	11
4.3.3 Member Function Documentation	11
4.3.3.1 isEmpty()	11
4.4 editwindow Class Reference	12
4.4.1 Detailed Description	12
4.4.2 Constructor & Destructor Documentation	13
4.4.2.1 editwindow()	13
4.5 Exception Class Reference	13
4.5.1 Detailed Description	14
4.5.2 Member Function Documentation	14
4.5.2.1 what()	14
4.6 FileProcessingException Class Reference	14
4.6.1 Detailed Description	15
4.6.2 Constructor & Destructor Documentation	15
4.6.2.1 FileProcessingException()	15
4.7 findwindow Class Reference	15
4.7.1 Detailed Description	16
4.7.2 Constructor & Destructor Documentation	16

4.7.2.1 findwindow()	16
4.8 Tasks::iterator Class Reference	16
4.8.1 Detailed Description	17
4.8.2 Constructor & Destructor Documentation	17
4.8.2.1 iterator()	17
4.8.3 Member Function Documentation	18
4.8.3.1 operator!=(())	18
4.8.3.2 operator*()	18
4.8.3.3 operator++()	18
4.9 MainWindow Class Reference	19
4.9.1 Detailed Description	20
4.9.2 Constructor & Destructor Documentation	20
4.9.2.1 MainWindow()	20
4.9.3 Member Function Documentation	21
4.9.3.1 addPart()	21
4.9.3.2 changeColor()	21
4.9.3.3 selectedItem()	21
4.10 MyItem Class Reference	22
4.10.1 Detailed Description	22
4.10.2 Constructor & Destructor Documentation	22
4.10.2.1 MyItem()	22
4.10.3 Member Function Documentation	23
4.10.3.1 getIndex()	23
4.11 NonExistingItemException Class Reference	23
4.11.1 Detailed Description	24
4.11.2 Constructor & Destructor Documentation	24
4.11.2.1 NonExistingItemException()	24
4.12 NonGivenSetting Class Reference	24
4.12.1 Detailed Description	25
4.12.2 Constructor & Destructor Documentation	25
4.12.2.1 NonGivenSetting()	25
4.13 predTask Struct Reference	25
4.13.1 Detailed Description	25
4.13.2 Member Function Documentation	25
4.13.2.1 operator()(())	26
4.14 Reader Class Reference	26
4.14.1 Detailed Description	26
4.14.2 Member Function Documentation	26
4.14.2.1 readFile()	26
4.14.2.2 readFiles()	27
4.14.2.3 saveFile()	27
4.15 Settings Struct Reference	27

4.15.1 Detailed Description	28
4.16 Task Class Reference	28
4.16.1 Detailed Description	29
4.16.2 Constructor & Destructor Documentation	29
4.16.2.1 Task()	29
4.16.3 Member Function Documentation	29
4.16.3.1 context()	30
4.16.3.2 getCompletionDate()	30
4.16.3.3 getContext()	30
4.16.3.4 getCreationDate()	30
4.16.3.5 getPriority()	31
4.16.3.6 getProject()	31
4.16.3.7 getText()	31
4.16.3.8 isComplete()	31
4.16.3.9 markedForDeletion()	32
4.16.3.10 match()	32
4.16.3.11 project()	33
4.16.3.12 setCompletion()	33
4.16.3.13 setCompletionDate() [1/2]	33
4.16.3.14 setCompletionDate() [2/2]	34
4.16.3.15 setCreationDate() [1/2]	34
4.16.3.16 setCreationDate() [2/2]	34
4.16.3.17 setDeletion()	34
4.16.3.18 setPriority()	35
4.16.3.19 text()	35
4.17 Tasks Class Reference	35
4.17.1 Detailed Description	36
4.17.2 Member Function Documentation	36
4.17.2.1 addEmpty()	36
4.17.2.2 addTask()	36
4.17.2.3 at()	37
4.17.2.4 begin()	37
4.17.2.5 end()	37
4.17.2.6 operator[]()	37
4.17.2.7 print()	38
4.17.2.8 printAllTasks()	38
4.17.2.9 printTasks()	38
4.17.2.10 redo()	39
4.17.2.11 size()	39
4.17.2.12 undo()	39
5 File Documentation	41

5.1 exception.cpp File Reference	41
5.1.1 Detailed Description	41
5.2 exception.hpp File Reference	41
5.2.1 Detailed Description	42
5.3 main.cpp File Reference	42
5.3.1 Detailed Description	42
5.3.2 Function Documentation	42
5.3.2.1 main()	42
5.4 qui.hpp File Reference	43
5.4.1 Detailed Description	43
5.5 reader.cpp File Reference	43
5.5.1 Detailed Description	44
5.6 reader.hpp File Reference	44
5.6.1 Detailed Description	44
5.7 task.hpp File Reference	44
5.7.1 Detailed Description	45
5.7.2 Function Documentation	45
5.7.2.1 bindStrings()	45
5.7.2.2 convertDate()	46
5.7.2.3 operator<() [1/2]	46
5.7.2.4 operator<() [2/2]	46
5.7.2.5 operator<<() [1/2]	47
5.7.2.6 operator<<() [2/2]	47
5.7.2.7 splitString()	48
5.8 terminal.cpp File Reference	48
5.8.1 Detailed Description	48
5.8.2 Function Documentation	48
5.8.2.1 edit()	48
5.8.2.2 printEditDate()	49
5.8.2.3 printEditPart()	49
5.8.2.4 terminalRun()	49
5.9 terminal.hpp File Reference	50
5.9.1 Detailed Description	50
5.9.2 Function Documentation	50
5.9.2.1 edit()	50
5.9.2.2 printEditDate()	51
5.9.2.3 printEditPart()	51
5.9.2.4 terminalRun()	51
Index	53

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

ArgumentReader	7
ChangeTask	9
Date	10
std::exception	
Exception	13
FileProcessingException	14
NonExistingItemException	23
NonGivenSetting	24
Tasks::iterator	16
predTask	25
QDialog	
editwindow	12
findwindow	15
QListWidgetItem	
MyItem	22
QMainWindow	
MainWindow	19
Reader	26
Settings	27
Task	28
Tasks	35

Chapter 2

Class Index

2.1 Class List

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

ArgumentReader	Read config file and arguments, also save config if needed	7
ChangeTask	When changing the task make this class to preserve the old task for undo and redo purposes .	9
Date	Struct for holding data about given date	10
editwindow	Window for editing and adding tasks	12
Exception	Default class for exceptions	13
FileProcessingException	When the givven file couldn't be accesed	14
findwindow	Window for editing and adding tasks	15
Tasks::iterator	Class for iterator in Tasks	16
MainWindow	Main window for showing tasks	19
MyItem	Derivative of list widget item to store index of the task	22
NonExistingItemException	When someone is trying to reach nonexistent task	23
NonGivenSetting	When there is missing settings to proper start the application	24
predTask	Functor for pointer to task to sort vector with task pointer	25
Reader	Reading and saving files with todo.txt syntax in it	26
Settings	What is the settings of current program	27
Task	Class for holding all properties of task	28
Tasks	Class holding all tasks and to work with them	35

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

exception.cpp	Source file for handling all exceptions	41
exception.hpp	Header file for handling all exceptions	41
main.cpp	Main entrypoint	42
qui.hpp	Handling graphical interface	43
reader.cpp	Cpp file for implementation of reader.hpp	43
reader.hpp	Header file for reading files, arguments and configuration file	44
task.hpp	Source file for classes Task , Tasks and everything related to them	44
terminal.cpp	Source file for running application in terminal	48
terminal.hpp	Header file for running application in terminal	50

Chapter 4

Class Documentation

4.1 ArgumentReader Class Reference

Read config file and arguments, also save config if needed.

```
#include <reader.hpp>
```

Public Member Functions

- void [readConfig](#) ([Settings](#) &settings, std::vector< std::string > &files, std::string &ofile)
- bool [parseArguments](#) (const std::vector< std::string > &args, [Settings](#) &settings, std::vector< std::string > &files, std::string &ofile)

Private Member Functions

- void [saveConfig](#) (const [Settings](#) &settings, const std::vector< std::string > &files, const std::string ofile)

Private Attributes

- const std::string [config](#) = ".config"
Filepath to config file.

4.1.1 Detailed Description

Read config file and arguments, also save config if needed.

4.1.2 Member Function Documentation

4.1.2.1 [parseArguments\(\)](#)

```
bool ArgumentReader::parseArguments (
    const std::vector< std::string > & args,
    Settings & settings,
    std::vector< std::string > & files,
    std::string & ofile )
```

Parse input arguments.

Parameters

<i>args</i>	Vector of arguments.
<i>settings</i>	Current settings.
<i>files</i>	Vector of given files.
<i>ofile</i>	Output file.

Returns

If help was found, then do not run application and print helpline.

4.1.2.2 readConfig()

```
void ArgumentReader::readConfig (
    Settings & settings,
    std::vector< std::string > & files,
    std::string & ofile )
```

Read config file.

Parameters

<i>settings</i>	Which setting to change or load.
<i>files</i>	Which are the files that are being read from.
<i>ofile</i>	What is the output file for storing changes.

4.1.2.3 saveConfig()

```
void ArgumentReader::saveConfig (
    const Settings & settings,
    const std::vector< std::string > & files,
    const std::string ofile ) [private]
```

Save config file with current settings.

Parameters

<i>settings</i>	Current stettings.
<i>files</i>	Vector of given files.
<i>ofile</i>	Output file.

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

- [reader.hpp](#)
- [reader.cpp](#)

4.2 ChangeTask Class Reference

When changing the task make this class to preserve the old task for undo and redo purposes.

```
#include <task.hpp>
```

Public Member Functions

- [ChangeTask](#) ([Task](#) **task*)
- void [undo](#) ()
Undo all changes made to the task.
- [Task](#) * [task](#) ()

Private Attributes

- [Task](#) * [task_](#)
Pointer to the changed task.
- bool [markedForDeletion_](#) = false
If it was marked for deletion.
- std::string [text_](#)
What was the old content.
- size_t [priority_](#)
What was its priority.
- bool [completion_](#)
If it was set for completion or not.
- [Date](#) [completion_date_](#)
What was its completion date.
- [Date](#) [creation_date_](#)
What was its creation date.
- std::string [project_tag_](#)
What was its project tag.
- std::string [context_tag_](#)
What was its context tag.

4.2.1 Detailed Description

When changing the task make this class to preserve the old task for undo and redo purposes.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ChangeTask()

```
ChangeTask::ChangeTask (  
    Task * task )
```

Default constructor.

Parameters

<i>task</i>	Pointer to the changed task.
-------------	------------------------------

4.2.3 Member Function Documentation

4.2.3.1 task()

```
Task* ChangeTask::task ( ) [inline]
```

Getter for the pointer to the changed task.

Returns

Pointer to the task.

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

- [task.hpp](#)
- [task.cpp](#)

4.3 Date Struct Reference

Struct for holding data about given date.

```
#include <task.hpp>
```

Public Member Functions

- [Date](#) (int y=0, int m=0, int d=0)
- bool [isEmpty](#) () const

Public Attributes

- int [day](#)
Number of day.
- int [month](#)
Number of month.
- int [year](#)
Number of year.

4.3.1 Detailed Description

Struct for holding data about given date.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Date()

```
Date::Date (
    int y = 0,
    int m = 0,
    int d = 0 )
```

Default constructor.

Parameters

<i>y</i>	Year.
<i>m</i>	Month.
<i>d</i>	Day.

4.3.3 Member Function Documentation

4.3.3.1 isEmpty()

```
bool Date::isEmpty ( ) const
```

Whether the date is undefined (all values are equal to 0).

Returns

True if it is undefined.

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

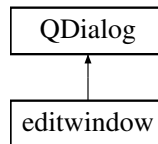
- [task.hpp](#)
- [task.cpp](#)

4.4 editwindow Class Reference

Window for editing and adding tasks.

```
#include <qui.hpp>
```

Inheritance diagram for editwindow:



Public Slots

- void [mysave](#) ()
Slot for pressing save button.

Public Member Functions

- [editwindow](#) ([Task](#) *task, [MainWindow](#) *mainParent, bool emptyText=false, bool emptyDate=false, QWidget *parent=nullptr)
- [~editwindow](#) ()
Default destructor.
- void [write](#) ()
Write everything about the task into boxes.
- void [showError](#) ()
If there was mistake show error when loading new window.

Private Attributes

- [Ui::editwindow](#) * [ui](#)
Qt ui.
- [Task](#) * [task_](#)
Which task it is handling.
- [MainWindow](#) * [parent_](#)
Which mainwindow called this window. For reloading the parent.
- bool [emptyText_](#)
When date is not defined and should be.
- bool [emptyDate_](#)
When there was an empty text before.

4.4.1 Detailed Description

Window for editing and adding tasks.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 editwindow()

```
editwindow::editwindow (
    Task * task,
    MainWindow * mainParent,
    bool emptyText = false,
    bool emptyDate = false,
    QWidget * parent = nullptr ) [explicit]
```

Default constructor.

Parameters

<i>task</i>	With which task it is operating.
<i>mainParent</i>	Which mainWindow called this editwindow.
<i>*parent</i>	qt constructor
<i>emptyText</i>	If the last editwindow ended with emptytext.
<i>emptyDate</i>	If the last editwindow ended with wrong dates.

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

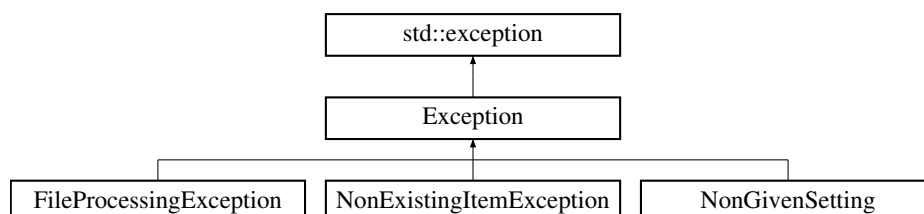
- [qui.hpp](#)
- [qui.cpp](#)

4.5 Exception Class Reference

Default class for exceptions.

```
#include <exception.hpp>
```

Inheritance diagram for Exception:



Public Member Functions

- `std::string & what ()`

Public Attributes

- `std::string message_`
Error message.

4.5.1 Detailed Description

Default class for exceptions.

4.5.2 Member Function Documentation

4.5.2.1 what()

```
std::string& Exception::what ( ) [inline]
```

Get the error message.

Returns

Reference to the error message.

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

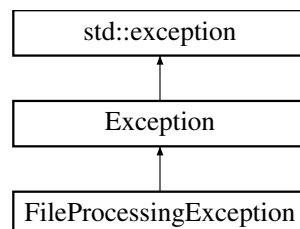
- [exception.hpp](#)

4.6 FileProcessingException Class Reference

When the given file couldn't be accesed.

```
#include <exception.hpp>
```

Inheritance diagram for FileProcessingException:



Public Member Functions

- [FileProcessingException](#) (const std::string &filename, FILETYPE type)

Additional Inherited Members

4.6.1 Detailed Description

When the given file couldn't be accesed.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 FileProcessingException()

```
FileProcessingException::FileProcessingException (
    const std::string & filename,
    FILETYPE type )
```

Default constructor.

Parameters

<i>filename</i>	What is the filename / filepath.
<i>type</i>	If it is output or input.

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

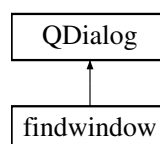
- [exception.hpp](#)
- [exception.cpp](#)

4.7 findwindow Class Reference

Window for editing and adding tasks.

```
#include <qui.hpp>
```

Inheritance diagram for findwindow:



Public Slots

- void [find](#) ()
Slot for pressing ok button.

Public Member Functions

- [findwindow](#) ([MainWindow](#) *mainParent, QWidget *parent=nullptr)
- [~findwindow](#) ()

Default destructor.

Private Attributes

- Ui::findwindow * [ui](#)

Qt ui.

- [MainWindow](#) * [parent_](#)

Which mainWindow called this window. For reloading the parent.

4.7.1 Detailed Description

Window for editing and adding tasks.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 findwindow()

```
findwindow::findwindow (
    MainWindow * mainParent,
    QWidget * parent = nullptr ) [explicit]
```

Default constructor.

Parameters

<i>mainParent</i>	Which mainWindow called this editwindow.
<i>*parent</i>	qt constructor

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

- [qui.hpp](#)
- [qui.cpp](#)

4.8 Tasks::iterator Class Reference

Class for iterator in [Tasks](#).

```
#include <task.hpp>
```

Public Types

- using `iterator_category` = `std::forward_iterator_tag`
What category is this iterator.
- using `difference_type` = `std::ptrdiff_t`
How to solve difference of these iterators.
- using `value_type` = `Task`
What is the type it is iterating.
- using `pointer` = `Task *`
What is the pointer.
- using `reference` = `Task &`
What is the reference.

Public Member Functions

- `iterator` (`Tasks *tasks`, `std::size_t position`)
- `Task & operator*` () const
- bool `operator!=` (const `iterator` &other) const
- `iterator & operator++` ()

Private Attributes

- `Tasks * tasks_`
Pointer to parent `Tasks`.
- `size_t position_`
At which position is the iterator looking.

Friends

- class `Tasks`

4.8.1 Detailed Description

Class for iterator in `Tasks`.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 `iterator()`

```
Tasks::iterator::iterator (  
    Tasks * tasks,  
    std::size_t position )
```

Basic constructor.

Parameters

<i>tasks</i>	Which tasks it is bounded to.
<i>position</i>	At which position it is looking.

4.8.3 Member Function Documentation

4.8.3.1 operator"!=()

```
bool Tasks::iterator::operator!= (
    const iterator & other ) const
```

How to compare two iterators.

Parameters

<i>other</i>	Second iterator.
--------------	------------------

Returns

If they are different.

4.8.3.2 operator*()

```
Task & Tasks::iterator::operator* ( ) const
```

To use * operator with the iterator.

Returns

reference to that [Task](#).

4.8.3.3 operator++()

```
Tasks::iterator & Tasks::iterator::operator++ ( )
```

Default operator for incrementation.

Returns

Reference to changed iterator.

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

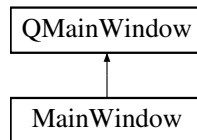
- [task.hpp](#)
- [task.cpp](#)

4.9 MainWindow Class Reference

Main window for showing tasks.

```
#include <qui.hpp>
```

Inheritance diagram for MainWindow:



Public Slots

- void [undo](#) ()
Slot for pressing undo button.
- void [redo](#) ()
Slot for pressing redo button.
- void [edit](#) ()
Slot for pressing edit button.
- void [add](#) ()
Slot for pressing add button.
- void [done](#) ()
Slot for pressing done button.
- void [myDelete](#) ()
Slot for pressing delete button.
- void [sort](#) ()
Slot for sorting tasks.
- void [save](#) ()
Slot for saving document.
- void [open](#) ()
Opening new file.
- void [import](#) ()
Importing from another file.
- void [checkDel](#) ()
Show or not show deleted items.
- void [checkDone](#) ()
Show or not show done items.
- void [openFind](#) ()
Open find window.
- void [reload](#) ()
Reload file.

Public Member Functions

- [MainWindow](#) ([Tasks](#) *tasks, [Reader](#) *reader, std::string *ofile, QWidget *parent=nullptr)
- [~MainWindow](#) ()
Default destructor.
- void [refresh](#) ()
Reload list with tasks.
- void [refreshMatch](#) (const std::string &match)
Reload list with matching ones.
- void [addPart](#) (QString text)
- void [changeColor](#) ([Task](#) *task, size_t index)
- size_t [selectedItem](#) (QListWidgetItem *item)
- void [showFound](#) (std::string match)
For showing found tasks.

Private Attributes

- Ui::MainWindow * [ui](#)
Qt ui.
- [Tasks](#) * [tasks_](#)
Which tasks it is handling.
- [Reader](#) * [reader_](#)
Which reader to use when writing to files or reading them.
- std::string * [ofile_](#)
What is the path to output file.
- bool [done_](#) = false
Whether to show done items.
- bool [deleted_](#) = false
Whether to show deleted items.

4.9.1 Detailed Description

Main window for showing tasks.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 MainWindow()

```
MainWindow::MainWindow (  
    Tasks * tasks,  
    Reader * reader,  
    std::string * ofile,  
    QWidget * parent = nullptr )
```

Default constructor.

Parameters

<i>tasks</i>	What tasks it is operating with.
<i>reader</i>	The reader that is used.
<i>ofile</i>	What is the filepath to the output file.
<i>*parent</i>	qt constructor.

4.9.3 Member Function Documentation

4.9.3.1 addPart()

```
void MainWindow::addPart (
    QString text )
```

Add new line to list with taks.

Parameters

<i>text</i>	What is to be inserted.
-------------	-------------------------

4.9.3.2 changeColor()

```
void MainWindow::changeColor (
    Task * task,
    size_t index )
```

Change color of item based on its priority or if it is incorrect.

Parameters

<i>task</i>	Which task.
<i>index</i>	What is the index in list widget.

4.9.3.3 selectedItem()

```
size_t MainWindow::selectedItem (
    QListWidgetItem * item )
```

Return index of the task that is selected.

Returns

Its index.

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

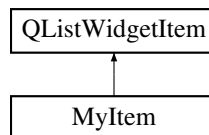
- [qui.hpp](#)
- [qui.cpp](#)

4.10 MyItem Class Reference

Derivative of list widget item to store index of the task.

```
#include <qui.hpp>
```

Inheritance diagram for MyItem:



Public Member Functions

- [MyItem](#) (size_t index)
- size_t [getIndex](#) ()

Private Attributes

- size_t [task_index_](#)
Tasks index.

4.10.1 Detailed Description

Derivative of list widget item to store index of the task.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 MyItem()

```
MyItem::MyItem (  
    size_t index )
```

Default constructor.

Parameters

<i>index</i>	Index of the task.
--------------	--------------------

4.10.3 Member Function Documentation

4.10.3.1 getIndex()

```
size_t MyItem::getIndex ( )
```

Get the index of the task.

Returns

Its index.

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

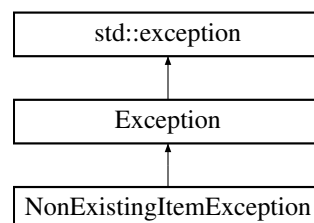
- [qui.hpp](#)
- [qui.cpp](#)

4.11 NonExistingItemException Class Reference

When someone is trying to reach nonexistent task.

```
#include <exception.hpp>
```

Inheritance diagram for NonExistingItemException:



Public Member Functions

- [NonExistingItemException](#) (size_t given, size_t max)

Additional Inherited Members

4.11.1 Detailed Description

When someone is trying to reach nonexistent task.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 NonExistingItemException()

```
NonExistingItemException::NonExistingItemException (
    size_t given,
    size_t max )
```

Default constructor.

Parameters

<i>given</i>	Which task is nonreachable.
<i>max</i>	What is the maximal position that cold be reached.

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

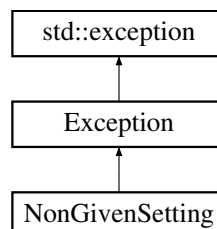
- [exception.hpp](#)
- [exception.cpp](#)

4.12 NonGivenSetting Class Reference

When there is missing settings to proper start the application.

```
#include <exception.hpp>
```

Inheritance diagram for NonGivenSetting:



Public Member Functions

- [NonGivenSetting](#) (`SETTINGTYPE` type)

Additional Inherited Members

4.12.1 Detailed Description

When there is missing settings to proper start the application.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 NonGivenSetting()

```
NonGivenSetting::NonGivenSetting (
    SETTINGTYPE type )
```

Default constructor.

Parameters

<i>type</i>	What settings is missing.
-------------	---------------------------

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

- [exception.hpp](#)
- [exception.cpp](#)

4.13 predTask Struct Reference

Functor for pointer to task to sort vector with task pointer.

```
#include <task.hpp>
```

Public Member Functions

- bool [operator\(\)](#) ([Task](#) *task1, [Task](#) *task2) const

4.13.1 Detailed Description

Functor for pointer to task to sort vector with task pointer.

4.13.2 Member Function Documentation

4.13.2.1 operator()

```
bool predTask::operator() (
    Task * task1,
    Task * task2 ) const [inline]
```

Operator () of the functor.

Parameters

<i>task1</i>	Pointer to first task.
<i>task2</i>	Pointer to second task.

Returns

If the first task has bigger priority.

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

- [task.hpp](#)

4.14 Reader Class Reference

Reading and saving files with todo.txt syntax in it.

```
#include <reader.hpp>
```

Public Member Functions

- void [readFiles](#) (const std::vector< std::string > &files, [Tasks](#) &tasks)
- void [readFile](#) (const std::string &file, [Tasks](#) &tasks)
- void [saveFile](#) (const [Tasks](#) &tasks, const std::string &ofile)

4.14.1 Detailed Description

Reading and saving files with todo.txt syntax in it.

4.14.2 Member Function Documentation

4.14.2.1 readFile()

```
void Reader::readFile (
    const std::string & file,
    Tasks & tasks )
```

read only one file.

Parameters

<i>file</i>	Whhich file to read.
<i>tasks</i>	Where to store all tasks.

4.14.2.2 readFiles()

```
void Reader::readFiles (
    const std::vector< std::string > & files,
    Tasks & tasks )
```

Read all files in given vector and add all tasks.

Parameters

<i>files</i>	Vector of given files with tasks.
<i>tasks</i>	Where to store all tasks.

4.14.2.3 saveFile()

```
void Reader::saveFile (
    const Tasks & tasks,
    const std::string & ofile )
```

Save current state of tasks to output file.

Parameters

<i>tasks</i>	From where to get tasks.
<i>ofile</i>	Which file to use as an output file.

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

- [reader.hpp](#)
- [reader.cpp](#)

4.15 Settings Struct Reference

What is the settings of current program.

```
#include <reader.hpp>
```

Public Attributes

- `CLIENT client` = NONE
Chosen client.
- `bool save` = false
Whether to save this settings to config file.
- `bool useConfig` = true
Whether to read config file or not.

4.15.1 Detailed Description

What is the settings of current program.

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

- [reader.hpp](#)

4.16 Task Class Reference

Class for holding all properties of task.

```
#include <task.hpp>
```

Public Member Functions

- `Task` (bool done=false, char priority='0')
- `std::string & text` ()
- `const std::string & getText` () const
- `std::string & project` ()
- `const std::string & getProject` () const
- `std::string & context` ()
- `const std::string & getContext` () const
- `void setDeletion` (bool del)
- `bool markedForDeletion` () const
- `void switchDeletion` ()
Switch deletion to the negation of the current state.
- `void setCompletion` (bool completion)
- `bool isComplete` () const
- `void switchCompletion` ()
Switch completion to its negation of the current state.
- `void setCompletionDate` (std::string date)
- `void setCompletionDate` (`Date` date)
- `const Date & getCompletionDate` () const
- `void setCreationDate` (std::string date)
- `void setCreationDate` (`Date` date)
- `const Date & getCreationDate` () const
- `size_t getPriority` () const
- `void setPriority` (char pr)
- `bool match` (const std::string &match)

Private Attributes

- bool `markedForDeletion_` = false
If the task is set to be deleted (not saved).
- std::string `text_`
What is the content of the task.
- size_t `priority_`
What priority does the task have (from A to Z, but in size_t format).
- bool `completion_`
If the task is marked as completed or not.
- Date `completion_date_`
When the task is set to be done.
- Date `creation_date_`
When the task was created.
- std::string `project_tag_`
What is the project tag.
- std::string `context_tag_`
What is the context tag.

4.16.1 Detailed Description

Class for holding all properties of task.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Task()

```
Task::Task (
    bool done = false,
    char priority = '0' )
```

Default constructor.

Parameters

<i>done</i>	If the task was marked as done or not.
<i>priority</i>	If the task has any given priority.

4.16.3 Member Function Documentation

4.16.3.1 context()

```
std::string& Task::context ( ) [inline]
```

Get the reference to the context tag.

Returns

Reference to the context tag.

4.16.3.2 getCompletionDate()

```
const Date& Task::getCompletionDate ( ) const [inline]
```

Getter for constant reference to the completion date.

Returns

Const reference to the completion date.

4.16.3.3 getContext()

```
const std::string& Task::getContext ( ) const [inline]
```

Getter for constant reference to the context tag.

Returns

Const reference to the context tag.

4.16.3.4 getCreationDate()

```
const Date& Task::getCreationDate ( ) const [inline]
```

Getter for constant reference to the completion date.

Returns

Const reference to the completion date.

4.16.3.5 getPriority()

```
size_t Task::getPriority ( ) const [inline]
```

Getter for priority in size_t format.

Returns

Priority.

4.16.3.6 getProject()

```
const std::string& Task::getProject ( ) const [inline]
```

Getter for constant reference to the project tag.

Returns

Const reference to the project tag.

4.16.3.7 getText()

```
const std::string& Task::getText ( ) const [inline]
```

Getter for constant reference to the text.

Returns

Const reference to the text.

4.16.3.8 isComplete()

```
bool Task::isComplete ( ) const [inline]
```

Getter for completion flag.

Returns

If the task is complete or not.

4.16.3.9 markedForDeletion()

```
bool Task::markedForDeletion ( ) const [inline]
```

Getter for deletion flag.

Returns

If the task is set to be deleted.

4.16.3.10 match()

```
bool Task::match (
    const std::string & match )
```

If the task is matching with given string.

Parameters

<i>match</i>	Matching string.
--------------	------------------

Returns

If it matches.

4.16.3.11 project()

```
std::string& Task::project ( ) [inline]
```

Get the reference to the project tag.

Returns

Reference to the project tag.

4.16.3.12 setCompletion()

```
void Task::setCompletion (
    bool completion ) [inline]
```

Set the completion of the task.

Parameters

<i>completion</i>	Whether the task is going to be completed or not.
-------------------	---

4.16.3.13 setCompletionDate() [1/2]

```
void Task::setCompletionDate (
    Date date ) [inline]
```

Set completion date with already existing date.

Parameters

<i>date</i>	Given date.
-------------	-------------

4.16.3.14 setCompletionDate() [2/2]

```
void Task::setCompletionDate (
    std::string date )
```

Set completion date with date in string format.

Parameters

<i>date</i>	The string with writte date.
-------------	------------------------------

4.16.3.15 setCreationDate() [1/2]

```
void Task::setCreationDate (
    Date date ) [inline]
```

Set comletion date with already existing date.

Parameters

<i>date</i>	Given date.
-------------	-------------

4.16.3.16 setCreationDate() [2/2]

```
void Task::setCreationDate (
    std::string date )
```

Set creation date with date in string format.

Parameters

<i>date</i>	The string with writte date.
-------------	------------------------------

4.16.3.17 setDeletion()

```
void Task::setDeletion (
    bool del ) [inline]
```

Set the deletion for the task.

Parameters

<i>del</i>	If the task is to be deleted or not.
------------	--------------------------------------

4.16.3.18 setPriority()

```
void Task::setPriority (
    char pr )
```

Setter for priority with given character.

Parameters

<i>pr</i>	Priority in char format (A-Z).
-----------	--------------------------------

4.16.3.19 text()

```
std::string& Task::text ( ) [inline]
```

Get reference to the text.

Returns

Reference to the text.

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

- [task.hpp](#)
- [task.cpp](#)

4.17 Tasks Class Reference

Class holding all tasks and to work with them.

```
#include <task.hpp>
```

Classes

- class [iterator](#)
Class for iterator in [Tasks](#).

Public Member Functions

- void [addTask](#) (const std::string &line)
- void [printTasks](#) (std::ostream &os=std::cout)
- void [printAllTasks](#) (std::ostream &os=std::cout)
- void [print](#) (std::ostream &os) const
- void [sort](#) ()

Sort all tasks based on their priority.

- [Task](#) & [at](#) (size_t position)
- [Task](#) & [addEmpty](#) ()
- void [undo](#) (bool [print](#)=false)
- void [redo](#) (bool [print](#)=false)
- size_t [size](#) () const
- [Task](#) & [operator\[\]](#) (size_t index)
- [Tasks::iterator](#) [begin](#) ()
- [Tasks::iterator](#) [end](#) ()
- [~Tasks](#) ()

Default destructor for destructing all tasks.

- void [clear](#) ()

Destroy all tasks and change tasks.

Private Attributes

- std::vector< [Task](#) * > [tasks_](#)

Where all task are stored. As pointers to easily make new pointers to them.

- std::stack< [ChangeTask](#) > [undo_](#)

Stack of last changes.

- std::stack< [ChangeTask](#) > [redo_](#)

Stack of last undos.

4.17.1 Detailed Description

Class holding all tasks and to work with them.

4.17.2 Member Function Documentation

4.17.2.1 addEmpty()

```
Task & Tasks::addEmpty ( )
```

Adding new (empty) task.

Returns

Reference to the newly constructed task.

4.17.2.2 addTask()

```
void Tasks::addTask (
    const std::string & line )
```

Adding task in string format.

Parameters

<i>line</i>	Task in string format.
-------------	--

4.17.2.3 at()

```
Task & Tasks::at (
    size_t position )
```

Getter for reference of the task at the given index, also make copy for undo.

Parameters

<i>position</i>	What is the index of the task.
-----------------	--------------------------------

Returns

Reference to the task.

4.17.2.4 begin()

```
Tasks::iterator Tasks::begin ( )
```

Beginning iterator.

Returns

Newly constructed iterator pointing to the beginning.

4.17.2.5 end()

```
Tasks::iterator Tasks::end ( )
```

Ending iterator.

Returns

Newly constructed iterator pointing to the end.

4.17.2.6 operator[]()

```
Task & Tasks::operator[] (
    size_t index )
```

Get [Task](#) on the position through brackets.

Parameters

<i>index</i>	which position.
--------------	-----------------

Returns

Reference to thatt task if it exists.

4.17.2.7 print()

```
void Tasks::print (
    std::ostream & os ) const
```

Print all tasks in its base string format.

Parameters

<i>os</i>	Which stream to use.
-----------	----------------------

4.17.2.8 printAllTasks()

```
void Tasks::printAllTasks (
    std::ostream & os = std::cout )
```

Print all tasks with ther index.

Parameters

<i>os</i>	Which stream to use.
-----------	----------------------

4.17.2.9 printTasks()

```
void Tasks::printTasks (
    std::ostream & os = std::cout )
```

Print tasks that are not done and not marked for deletion with their index.

Parameters

<i>os</i>	Which stream to use.
-----------	----------------------

4.17.2.10 redo()

```
void Tasks::redo (
    bool print = false )
```

Redo last undo.

Parameters

<i>print</i>	Whether to print the change to cout or not.
--------------	---

4.17.2.11 size()

```
size_t Tasks::size ( ) const [inline]
```

How many tasks it has.

Returns

The size.

4.17.2.12 undo()

```
void Tasks::undo (
    bool print = false )
```

Undo last change.

Parameters

<i>print</i>	Wheter to print the change to cout or not.
--------------	--

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

- [task.hpp](#)
- [task.cpp](#)

Chapter 5

File Documentation

5.1 exception.cpp File Reference

Source file for handling all exceptions.

```
#include "exception.hpp"
```

5.1.1 Detailed Description

Source file for handling all exceptions.

5.2 exception.hpp File Reference

Header file for handling all exceptions.

```
#include <string>
#include <sstream>
#include <exception>
```

Classes

- class [Exception](#)
Default class for exceptions.
- class [NonExistingItemException](#)
When someone is trying to reach nonexistent task.
- class [FileProcessingException](#)
When the givven file couldn't be accesed.
- class [NonGivenSetting](#)
When there is missing settings to proper start the application.

Enumerations

- enum FILETYPE { OUTPUT , INPUT }
if the file is for output or input.
- enum SETTINGTYPE { OFILE , IFILE , INTERFACE }
What settings is missing, output file, input file or user interface.

5.2.1 Detailed Description

Header file for handling all exceptions.

5.3 main.cpp File Reference

is the main entrypoint.

```
#include <iostream>
#include <string>
#include <vector>
#include "reader.hpp"
#include "task.hpp"
#include "terminal.hpp"
#include "exception.hpp"
#include <QApplication>
#include "qui.hpp"
```

Functions

- void printHelpArgs ()
Print all possible arguments when calling program.
- int main (int argc, char **argv)

5.3.1 Detailed Description

is the main entrypoint.

5.3.2 Function Documentation

5.3.2.1 main()

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

Main function of the application.

Parameters

<i>argc</i>	Argument count.
<i>argv</i>	List of arguments.

Returns

Application exit code.

5.4 qui.hpp File Reference

Handling graphical interface.

```
#include <QDialog>
#include <QMainWindow>
#include <QErrorMessage>
#include <QFileDialog>
#include <QListWidgetItem>
#include <string>
#include <typeinfo>
#include "task.hpp"
#include "reader.hpp"
```

Classes

- class [MainWindow](#)
Main window for showing tasks.
- class [editwindow](#)
Window for editing and adding tasks.
- class [MyItem](#)
Derivative of list widget item to store index of the task.
- class [findwindow](#)
Window for editing and adding tasks.

5.4.1 Detailed Description

Handling graphical interface.

Header file for handling GUI.

5.5 reader.cpp File Reference

Cpp file for implementation of [reader.hpp](#).

```
#include "reader.hpp"
#include "task.hpp"
```

5.5.1 Detailed Description

Cpp file for implementation of [reader.hpp](#).

5.6 reader.hpp File Reference

Header file for reading files, arguments and configuration file.

```
#include <string>
#include <set>
#include <fstream>
#include <vector>
#include "task.hpp"
#include "exception.hpp"
```

Classes

- struct [Settings](#)
What is the settings of current program.
- class [ArgumentReader](#)
Read config file and arguments, also save config if needed.
- class [Reader](#)
Reading and saving files with todo.txt syntax in it.

Enumerations

- enum [CLIENT](#) { [GUI](#) , [CLI](#) , [NONE](#) }
User interface.

5.6.1 Detailed Description

Header file for reading files, arguments and configuration file.

5.7 task.hpp File Reference

Source file for classes [Task](#), [Tasks](#) and everything related to them.

```
#include <string>
#include <sstream>
#include <iostream>
#include <vector>
#include <algorithm>
#include <stack>
#include "exception.hpp"
```

Classes

- struct [Date](#)
Struct for holding data about given date.
- class [Task](#)
Class for holding all properties of task.
- class [ChangeTask](#)
When changing the task make this class to preserve the old task for undo and redo purposes.
- struct [predTask](#)
Functor for pointer to task to sort vector with task pointer.
- class [Tasks](#)
Class holding all tasks and to work with them.
- class [Tasks::iterator](#)
Class for iterator in [Tasks](#).

Functions

- void [bindStrings](#) (const std::vector< std::string > &parts, size_t start, size_t end, std::string &bind)
- void [splitString](#) (const std::string &line, const char splitter, std::vector< std::string > &parts)
- std::ostream & [operator<<](#) (std::ostream &os, const [Date](#) &date)
- bool [operator<](#) (const [Date](#) &date1, const [Date](#) &date2)
- [Date](#) [convertDate](#) (const std::string &writtenDate)
- std::ostream & [operator<<](#) (std::ostream &os, const [Task](#) &task)
- bool [operator<](#) (const [Task](#) &task1, const [Task](#) &task2)

5.7.1 Detailed Description

Source file for classes [Task](#), [Tasks](#) and everything related to them.

Header file for classes [Task](#), [Tasks](#) and everything related to them.

5.7.2 Function Documentation

5.7.2.1 bindStrings()

```
void bindStrings (
    const std::vector< std::string > & parts,
    size_t start,
    size_t end,
    std::string & bind )
```

Put together strings from vector (start to end) to one string.

Parameters

<i>parts</i>	String parts in vector.
<i>start</i>	First index of string from vector that will be used.
<i>end</i>	First index of string that won't be used.
<i>bind</i>	New constructed string.

5.7.2.2 convertDate()

```
Date convertDate (
    const std::string & writtenDate )
```

Convert date from string in format y-m-d.

Parameters

<i>writtenDate</i>	String with this format.
--------------------	--------------------------

Returns

Newly constructed [Date](#).

5.7.2.3 operator<() [1/2]

```
bool operator< (
    const Date & date1,
    const Date & date2 )
```

To compare dates between each other.

Parameters

<i>date1</i>	First date.
<i>date2</i>	Second date.

Returns

If the first was earlier.

5.7.2.4 operator<() [2/2]

```
bool operator< (
    const Task & task1,
    const Task & task2 )
```

To compare tasks between eachother.

Parameters

<i>task1</i>	First tasks.
<i>task2</i>	Second task.

Returns

If the first task has less "priority" (deletion - done - priority - completion date - creation date).

5.7.2.5 operator<<() [1/2]

```
std::ostream& operator<< (
    std::ostream & os,
    const Date & date )
```

To use default << operator to print date.

Parameters

<i>os</i>	Which stream to use.
<i>date</i>	Which date will be putted to the stream.

Returns

Reference to the stream.

5.7.2.6 operator<<() [2/2]

```
std::ostream& operator<< (
    std::ostream & os,
    const Task & task )
```

<< operator for task.

Parameters

<i>os</i>	Reference to the used stream.
<i>task</i>	Which task will be used.

Returns

Reference to the stream.

5.7.2.7 splitString()

```
void splitString (
    const std::string & line,
    const char splitter,
    std::vector< std::string > & parts )
```

Split string based on given splitter.

Parameters

<i>line</i>	Which line is going to be split.
<i>splitter</i>	By which character I am going to split.
<i>parts</i>	Where to put all the parts.

5.8 terminal.cpp File Reference

Source file for running application in terminal.

```
#include "terminal.hpp"
```

Functions

- void [printEditPart](#) (const std::string &text, std::string &before)
- void [printEditDate](#) (const std::string &text, const [Date](#) &date)
- void [edit](#) ([Task](#) &task, std::string &text)
- void [printHelp](#) ()
 - print all possible calls in terminal app.*
- void [terminalRun](#) ([Tasks](#) &tasks, [Reader](#) &reader, const std::string &ofile)

5.8.1 Detailed Description

Source file for running application in terminal.

5.8.2 Function Documentation

5.8.2.1 edit()

```
void edit (
    Task & task,
    std::string & text )
```

Edit or add task dialog.

Parameters

<i>task</i>	What is the task.
<i>text</i>	What to write after.

5.8.2.2 printEditDate()

```
void printEditDate (
    const std::string & text,
    const Date & date )
```

When editing print date part of a dialog.

Parameters

<i>text</i>	What is the text to be shown on the beggining.
<i>date</i>	Which was the date beforehand.

5.8.2.3 printEditPart()

```
void printEditPart (
    const std::string & text,
    std::string & before )
```

When editing print one part of a dialog.

Parameters

<i>text</i>	What is the text to be shown on the beggining.
<i>before</i>	What is the old text that is to be edited.

5.8.2.4 terminalRun()

```
void terminalRun (
    Tasks & tasks,
    Reader & reader,
    const std::string & ofile )
```

Main loop for running terminal application.

Parameters

<i>tasks</i>	What are the tasks.
<i>reader</i>	Which reader is to be used. For saving files.
<i>ofile</i>	Where will the tasks be saved in an output file.

5.9 terminal.hpp File Reference

Header file for running application in terminal.

```
#include "task.hpp"
#include "reader.hpp"
#include "exception.hpp"
#include <iostream>
```

Functions

- void [printEditPart](#) (const std::string &text, std::string &before)
- void [printEditDate](#) (const std::string &text, const [Date](#) &date)
- void [terminalRun](#) ([Tasks](#) &tasks, [Reader](#) &reader, const std::string &ofile)
- void [edit](#) ([Task](#) &task, std::string &text)
- void [printHelp](#) ()

print all possible calls in terminal app.

5.9.1 Detailed Description

Header file for running application in terminal.

5.9.2 Function Documentation

5.9.2.1 edit()

```
void edit (
    Task & task,
    std::string & text )
```

Edit or add task dialog.

Parameters

<i>task</i>	What is the task.
<i>text</i>	What to write after.

5.9.2.2 printEditDate()

```
void printEditDate (
    const std::string & text,
    const Date & date )
```

When editing print date part of a dialog.

Parameters

<i>text</i>	What is the text to be shown on the beggining.
<i>date</i>	Which was the date beforehand.

5.9.2.3 printEditPart()

```
void printEditPart (
    const std::string & text,
    std::string & before )
```

When editing print one part of a dialog.

Parameters

<i>text</i>	What is the text to be shown on the beggining.
<i>before</i>	What is the old text that is to be edited.

5.9.2.4 terminalRun()

```
void terminalRun (
    Tasks & tasks,
    Reader & reader,
    const std::string & ofile )
```

Main loop for running terminal application.

Parameters

<i>tasks</i>	What are the tasks.
<i>reader</i>	Which reader is to be used. For saving files.
<i>ofile</i>	Where will the tasks be saved in an output file.

Index

- addEmpty
 - Tasks, [36](#)
- addPart
 - MainWindow, [21](#)
- addTask
 - Tasks, [36](#)
- ArgumentReader, [7](#)
 - parseArguments, [7](#)
 - readConfig, [8](#)
 - saveConfig, [8](#)
- at
 - Tasks, [37](#)
- begin
 - Tasks, [37](#)
- bindStrings
 - task.hpp, [45](#)
- changeColor
 - MainWindow, [21](#)
- ChangeTask, [9](#)
 - ChangeTask, [9](#)
 - task, [10](#)
- context
 - Task, [29](#)
- convertDate
 - task.hpp, [46](#)
- Date, [10](#)
 - Date, [11](#)
 - isEmpty, [11](#)
- edit
 - terminal.cpp, [48](#)
 - terminal.hpp, [50](#)
- editwindow, [12](#)
 - editwindow, [13](#)
- end
 - Tasks, [37](#)
- Exception, [13](#)
 - what, [14](#)
- exception.cpp, [41](#)
- exception.hpp, [41](#)
- FileProcessingException, [14](#)
 - FileProcessingException, [15](#)
- findwindow, [15](#)
 - findwindow, [16](#)
- getCompletionDate
 - Task, [30](#)
- getContext
 - Task, [30](#)
- getCreationDate
 - Task, [30](#)
- getIndex
 - MylItem, [23](#)
- getPriority
 - Task, [30](#)
- getProject
 - Task, [31](#)
- getText
 - Task, [31](#)
- isComplete
 - Task, [31](#)
- isEmpty
 - Date, [11](#)
- iterator
 - Tasks::iterator, [17](#)
- main
 - main.cpp, [42](#)
- main.cpp, [42](#)
 - main, [42](#)
- MainWindow, [19](#)
 - addPart, [21](#)
 - changeColor, [21](#)
 - MainWindow, [20](#)
 - selectedItem, [21](#)
- markedForDeletion
 - Task, [31](#)
- match
 - Task, [32](#)
- MylItem, [22](#)
 - getIndex, [23](#)
 - MylItem, [22](#)
- NonExistingItemException, [23](#)
 - NonExistingItemException, [24](#)
- NonGivenSetting, [24](#)
 - NonGivenSetting, [25](#)
- operator!=
 - Tasks::iterator, [18](#)
- operator<
 - task.hpp, [46](#)
- operator<<
 - task.hpp, [47](#)
- operator*
 - Tasks::iterator, [18](#)

- operator()
 - predTask, 25
- operator++
 - Tasks::iterator, 18
- operator[]
 - Tasks, 37
- parseArguments
 - ArgumentReader, 7
- predTask, 25
 - operator(), 25
- print
 - Tasks, 38
- printAllTasks
 - Tasks, 38
- printEditDate
 - terminal.cpp, 49
 - terminal.hpp, 51
- printEditPart
 - terminal.cpp, 49
 - terminal.hpp, 51
- printTasks
 - Tasks, 38
- project
 - Task, 33
- qui.hpp, 43
- readConfig
 - ArgumentReader, 8
- Reader, 26
 - readFile, 26
 - readFiles, 27
 - saveFile, 27
- reader.cpp, 43
- reader.hpp, 44
- readFile
 - Reader, 26
- readFiles
 - Reader, 27
- redo
 - Tasks, 39
- saveConfig
 - ArgumentReader, 8
- saveFile
 - Reader, 27
- selectedItem
 - MainWindow, 21
- setCompletion
 - Task, 33
- setCompletionDate
 - Task, 33
- setCreationDate
 - Task, 34
- setDeletion
 - Task, 34
- setPriority
 - Task, 35
- Settings, 27
- size
 - Tasks, 39
- splitString
 - task.hpp, 47
- Task, 28
 - context, 29
 - getCompletionDate, 30
 - getContext, 30
 - getCreationDate, 30
 - getPriority, 30
 - getProject, 31
 - getText, 31
 - isComplete, 31
 - markedForDeletion, 31
 - match, 32
 - project, 33
 - setCompletion, 33
 - setCompletionDate, 33
 - setCreationDate, 34
 - setDeletion, 34
 - setPriority, 35
 - Task, 29
 - text, 35
- task
 - ChangeTask, 10
- task.hpp, 44
 - bindStrings, 45
 - convertDate, 46
 - operator<, 46
 - operator<<, 47
 - splitString, 47
- Tasks, 35
 - addEmpty, 36
 - addTask, 36
 - at, 37
 - begin, 37
 - end, 37
 - operator[], 37
 - print, 38
 - printAllTasks, 38
 - printTasks, 38
 - redo, 39
 - size, 39
 - undo, 39
- Tasks::iterator, 16
 - iterator, 17
 - operator!=, 18
 - operator*, 18
 - operator++, 18
- terminal.cpp, 48
 - edit, 48
 - printEditDate, 49
 - printEditPart, 49
 - terminalRun, 49
- terminal.hpp, 50
 - edit, 50
 - printEditDate, 51

- printEditPart, [51](#)
 - terminalRun, [51](#)
- terminalRun
 - terminal.cpp, [49](#)
 - terminal.hpp, [51](#)
- text
 - Task, [35](#)
- undo
 - Tasks, [39](#)
- what
 - Exception, [14](#)