KText Editor

Generated by Doxygen 1.9.6

1 Evaluation

To conclude, the program I have made is decent. It is still lacking in many areas and is still very ameaturish, but it has all the main objectives that we were looking for initially.

The program is very fast and effiecient with sub millisecond speed.

Improvements that we could have made to the already exsisting code could be to link SFML with Opengl and use them together not just a stand alone SFML library. This should be changed next time becuase SFML by itself is certainly powerful but for something more general and not game oriented it can be limiting on certain features that can only be changed by either messing with the SFML code so it works or by bringing opengl into it and expanding on the current functionality. The limiting features where Text handling, specifically for this project the colouring of text.

2 Kamil Editor

A Text Editor for kamil

2.1 Analysis

2.1.1 Background and Identifying the problem

The Project I will be developing will be in answer to the challenge set out by the end user and friend of mine, Kamil. He challenged me to make a lightweight editor that he can use in his day-to-day life and when doing Python projects and general day-to-day use.

The challenge started when he commented on my use of neovim and how it would be better if I used an actual IDE. I told him that I've used IDEs in the past and overall prefer the look and feel of a customized neovim. I then suggested that he learn Vim himself and that he wouldn't regret it, but he declined. Kamil then told me that I should create something easier for him to use and that could potentially change his use of IDEs.

Upon being issued with this challenge I created a few starter questions that I would research around for my NEA.

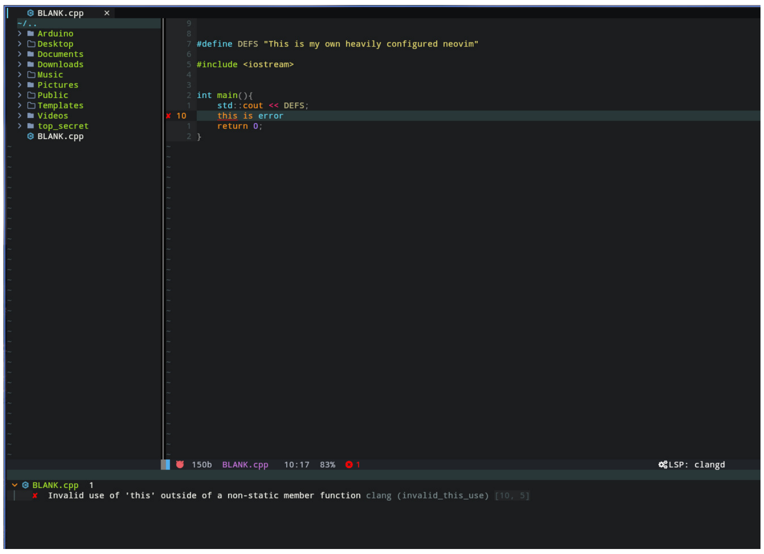
1) What is a text editor and how does it differ from an IDE? 2) How do I make a text editor for Kamil 3) How do I make it efficient enough to meet his standards?

To kick things along I began to research Text editors and IDEs and found out that the difference between them isn't limited to Operating System platforms or by how much better one is at a specific task but by the features each can do. Text Editors, as the name suggests are specifically designed for manipulating any form of text that it can open. While an IDE (Integrated Development Environment) is specifically designed for software development and comes with a multitude of features that engineers can make use of to streamline their workflow.

A table of pros and cons:

	Pros	cons
Text Editor	Light weight,	Limited in capability
	Fast,	
	Resource efficient	
	Very Modular	
IDE	Has everything out the box	Slow
		Not very Resource efficient
	can view memory	Too many menus
		Limited in compatability

Here are pictures of some text editors and IDE's:

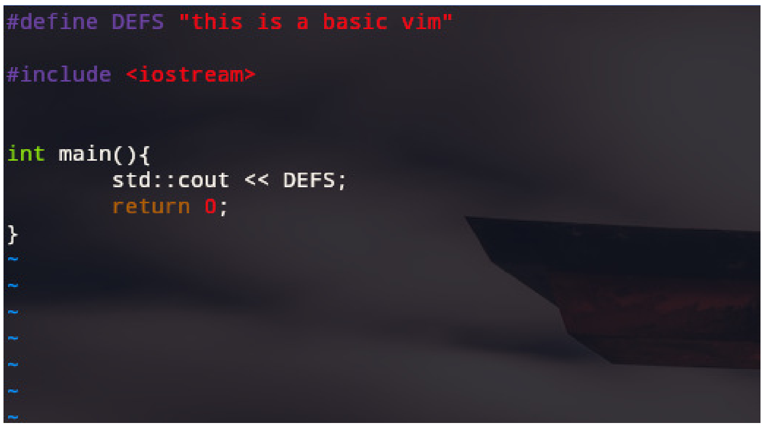


```
BLANK.cpp ×
~/..
> Arduino
> Desktop
> Documents
> Downloads
> Music
> Pictures
> Public
> Templates
> Videos
> top_secret
  BLANK.cpp
```

```
#define DEFS "This is my own heavily configured neovim"

#include <iostream>


int main(){
    std::cout << DEFS;
    this is error
    return 0;
}
```

```
150b  BLANK.cpp  10:17  83%  1                                LSP: clangd

BLANK.cpp  1
  Invalid use of 'this' outside of a non-static member function clang (invalid_this_use) [10, 5]
```

Figure 1 My Neovim

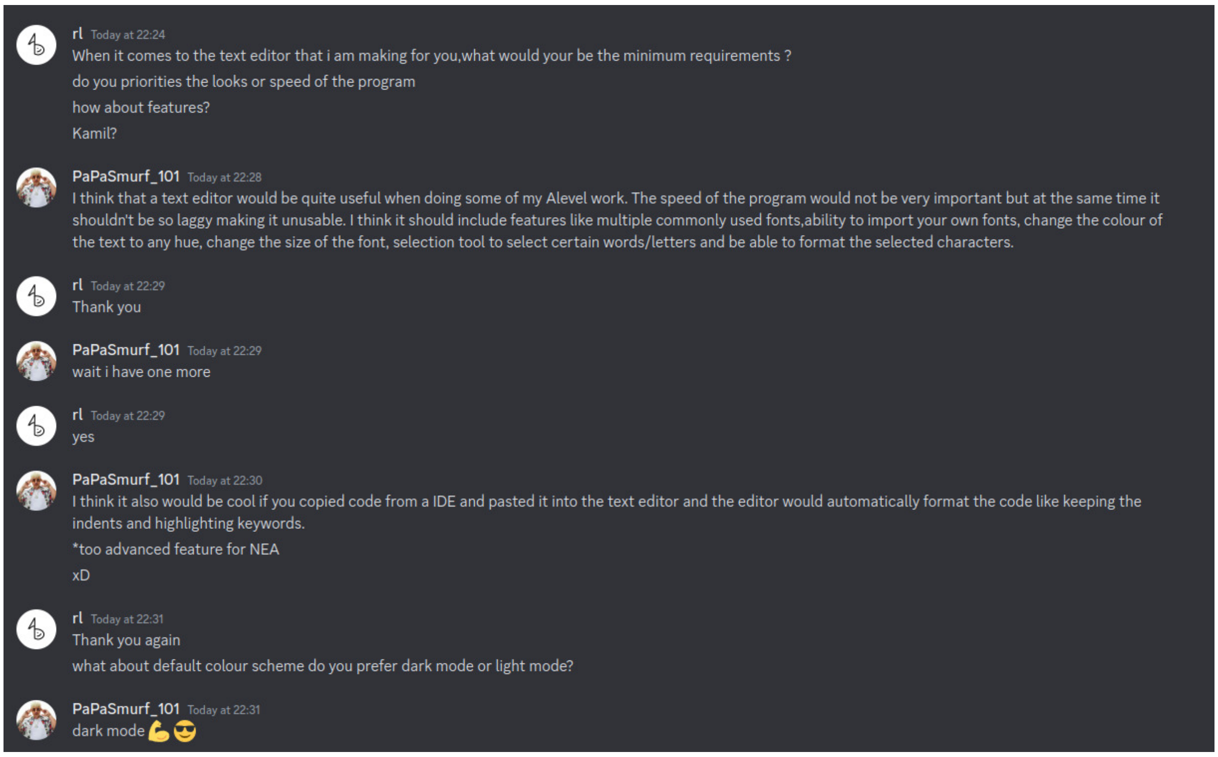


```
#define DEFS "this is a basic vim"

#include <iostream>


int main(){
        std::cout << DEFS;
        return 0;
}
~
~
~
~
~
~
~
```

Figure 2 My Vim

2.1.2 End User needs

When talking to Kamil about his needs it was apparent that he wanted something modular in the sense that it comes with what he needs so it is not a hassle to work with and it works with multiple different file types.



rl Today at 22:24
When it comes to the text editor that i am making for you,what would your be the minimum requirements ?
do you priorities the looks or speed of the program
how about features?
Kamil?

PaPaSmurf_101 Today at 22:28
I think that a text editor would be quite useful when doing some of my Alevel work. The speed of the program would not be very important but at the same time it shouldn't be so laggy making it unusable. I think it should include features like multiple commonly used fonts,ability to import your own fonts, change the colour of the text to any hue, change the size of the font, selection tool to select certain words/letters and be able to format the selected characters.

rl Today at 22:29
Thank you

PaPaSmurf_101 Today at 22:29
wait i have one more

rl Today at 22:29
yes

PaPaSmurf_101 Today at 22:30
I think it also would be cool if you copied code from a IDE and pasted it into the text editor and the editor would automatically format the code like keeping the indents and highlighting keywords.
*too advanced feature for NEA
xD

rl Today at 22:31
Thank you again
what about default colour scheme do you prefer dark mode or light mode?

PaPaSmurf_101 Today at 22:31
dark mode 💪😎

Since this is a project that could quickly grow in scale due to all the different parts of the editor, text and documents. The overall time complexity of the different algorithms coming together could increase the latency between the different commands. To ensure that the project meets the needs of Kamil, I have a few objectives.

2.1.2.1 Objective 1 Optimised:

Ensure that the program is properly optimised and not laggy. To ensure that the program is not laggy and has minimal delay we can get the time taken between function calls and their effect like the time taken for a key press to happen and it being executed.

Furthermore, In the code youll see optimisation techniques such as passing classes by pointer which essentially allows me to directly access the class in memory instead of accessing a copy of the class.

2.1.2.2 Objective 2 User configurable:

Allow it to be user configurable. We need the ability to load and use fonts specified by the user as well as change the theme of the program based on what the user wants.

We can check this by successfully storing font files and changing them as well as having the user use a theme through a colourscheme and successfully have the program use them.

2.1.2.3 Objective 3 Run Python:

Run Python programs. Since Kamil mainly codes in Python having some python compatability would help him greatly. With that said we would need to be able to write, save and run python code and have it be accurate.

This is easily measurable since a fully working python Implementation would be able to run and save our own files.

2.1.2.4 **Objective 4** File I/O:

Allow for user data to be stored and recalled locally in a file system. For the Text editor to be a text editor we need the ability to successfully access the file tree system this allows for further I/O like file saving and writing.

To test if this feature works we need to successfully save and load a file.

2.1.2.5 **Objective 5** Cross-platform:

Allow for cross platform support. Have the program successfully run on at least 2 of the 4 different Operating systems tehe four being, Windows, Mac, Linux, OpenBSD.

Upon completion the program would be completely cross-platform.

To summarise the objectives, the program must be:

- Cross-platform
- User-configurable
- file I/O
- Run Python files
- Not Laggy

2.1.3 Limitations

The Limitations of my program are what give it a general architecure to work with. The Limits my program will face are:

2.1.3.1 **Limitations 1** Programming Language:

When it comes to the Programming Language I wrote my project in C++ (Cpp, Cxx, cc) with access to the C++17 language standard. I chose this language becauase I am most familiar with it and prefer it over python for larger projects like this. It is a fast and performant language with good cross compatability which would make it good for the project.

2.1.3.2 **Limitations 2** Formatting standard:

This limitation is for the reusability and modularbility of the program when different users are making changes or when you come back to the code after a while. It is to keep everything orderly so you can focus entierly on the problem solving without having to worry about spaces and comments.

When it comes to writing good clean code, formatting the code is a big necessity that can help the overall functionality of the code and user experience.

An example being:

```
\code{.cpp}

//without a formatting standard

int printAninttoOutput
(int val) {return val;}


int setintTooutPut
(int val){
    reutrn val
}

// with a formatting standard

int Print_Int_To_Out(int val){
    return val;
}

int Set_Int_To_Out(int val){
    return val
}
```

From the examples shown above its clear that with the formatting the code is easier to read without any weird (but legal) C++ syntax, it also allows programmers to see a pattern and predict what the function they want to call is called without checking documentation.

2.1.3.3 Limitations 3 Operating System:

The operating System is a default limiter and denotes how everything comes together. By default I use Linux. This has the benefit of having more support for C++ coding and development in general with the caveat of programs not being very portable to other devices like windows machines. This means that I will either need to cross-compile my program or convert Kamil, who is a windows user, over to Linux.

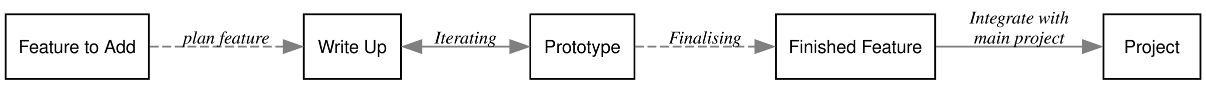
2.1.3.4 Limitations 4 Libraries:

Similar with the Operating system the functionality of a program is limited by the libraries that it calls and uses. For this project I need to use libraries that are cross platform and efficient.

2.1.4 Design

Throught the creation of the project I utilised a mix between modular and iterative design procedure where I would have a seperate test project (not one - one) that had all the basic functionality I needed and I would make the feature I am trying to add and test it on this dummy code, using techniques like integration and white-box testing. I would then iterate on any bugs found and fix them before moving the code to the main project.

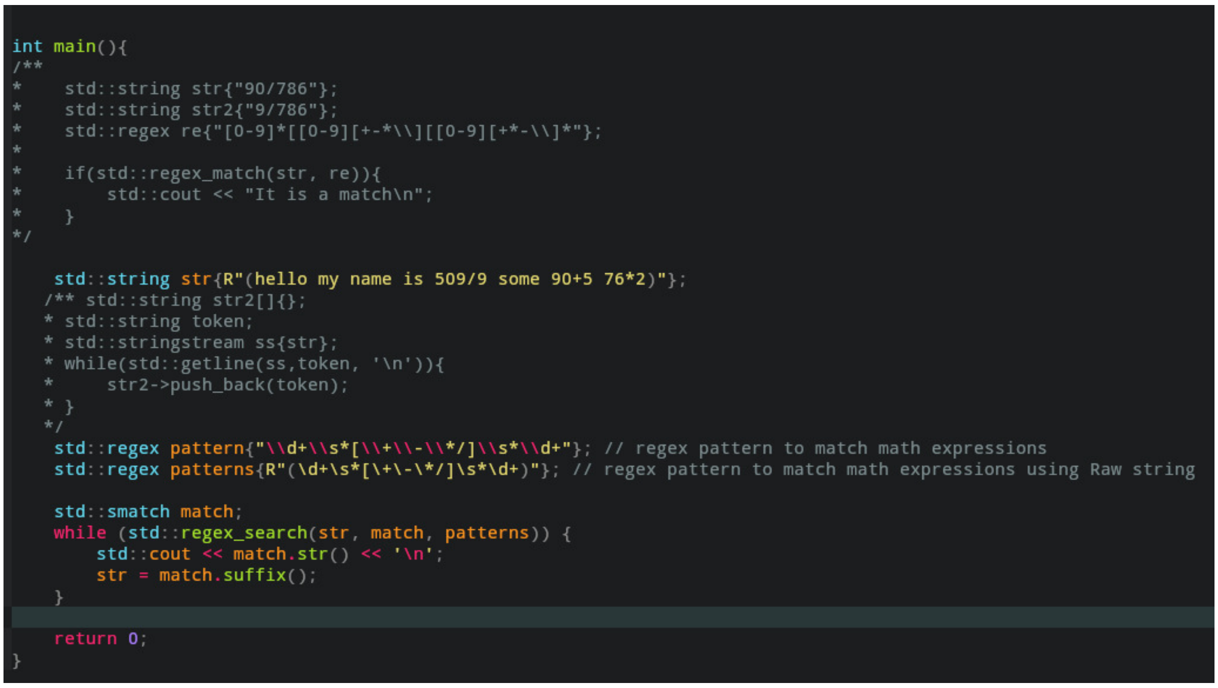
My workflow is as:



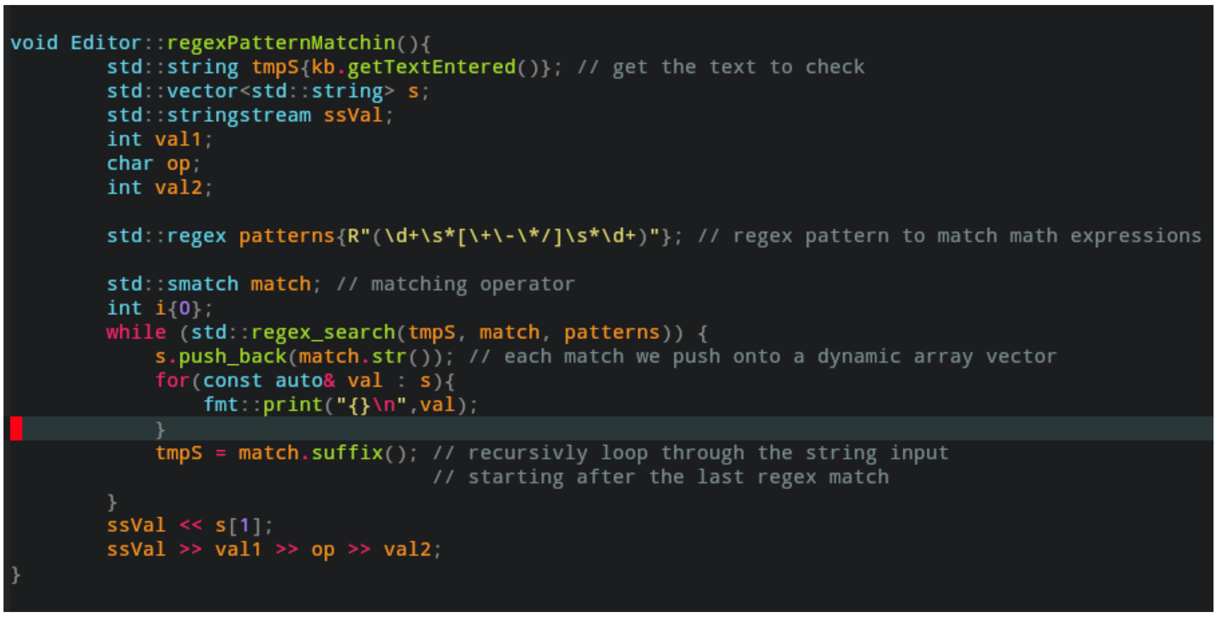
- reg -

Figure 3 workflow

Example regex:



```
int main(){
/**
*    std::string str{"90/786"};
*    std::string str2{"9/786"};
*    std::regex re{"[0-9]*[[0-9][+-*\\][[0-9][+*-\\]*"};
*
*    if(std::regex_match(str, re)){
*        std::cout << "It is a match\n";
*    }
*/

    std::string str{R"(hello my name is 509/9 some 90+5 76*2)"};
   /** std::string str2[]{};
   * std::string token;
   * std::stringstream ss{str};
   * while(std::getline(ss,token, '\n')){
   *     str2->push_back(token);
   * }
   */
    std::regex pattern{"\\d+\\s*[\\+\\-\\*/]\\s*\\d+"}; // regex pattern to match math expressions
    std::regex patterns{R"(\d+\s*[\+\-\*/]\s*\d+)"}; // regex pattern to match math expressions using Raw string

    std::smatch match;
    while (std::regex_search(str, match, patterns)) {
        std::cout << match.str() << '\n';
        str = match.suffix();
    }

    return 0;
}
```



```
void Editor::regexPatternMatchin(){
        std::string tmpS{kb.getTextEntered()}; // get the text to check
        std::vector<std::string> s;
        std::stringstream ssVal;
        int val1;
        char op;
        int val2;

        std::regex patterns{R"(\d+\s*[\+\-\*/]\s*\d+)"}; // regex pattern to match math expressions

        std::smatch match; // matching operator
        int i{0};
        while (std::regex_search(tmpS, match, patterns)) {
            s.push_back(match.str()); // each match we push onto a dynamic array vector
            for(const auto& val : s){
                fmt::print("{}\n",val);
            }
            tmpS = match.suffix(); // recursivly loop through the string input
                                   // starting after the last regex match
        }
        ssVal << s[1];
        ssVal >> val1 >> op >> val2;
}
```

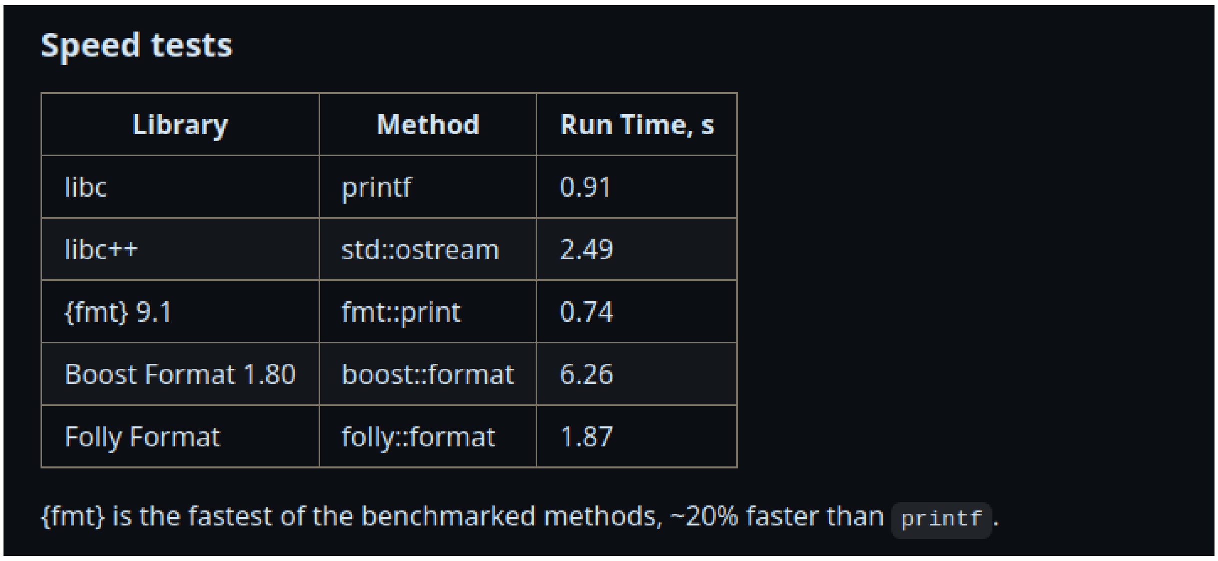
Figure 4 After testing

2.1.4.1 Design Choices general When developing the project I made a series of design choices that I thought would be best for the project.

For the Libraries I will be using I chose to go with SFML for graphics, fmtllib for formatted printing and tomlc++ for .toml I/O.

SFML is a graphics library that gives access to Opengl functions, it is also be used stand alone.

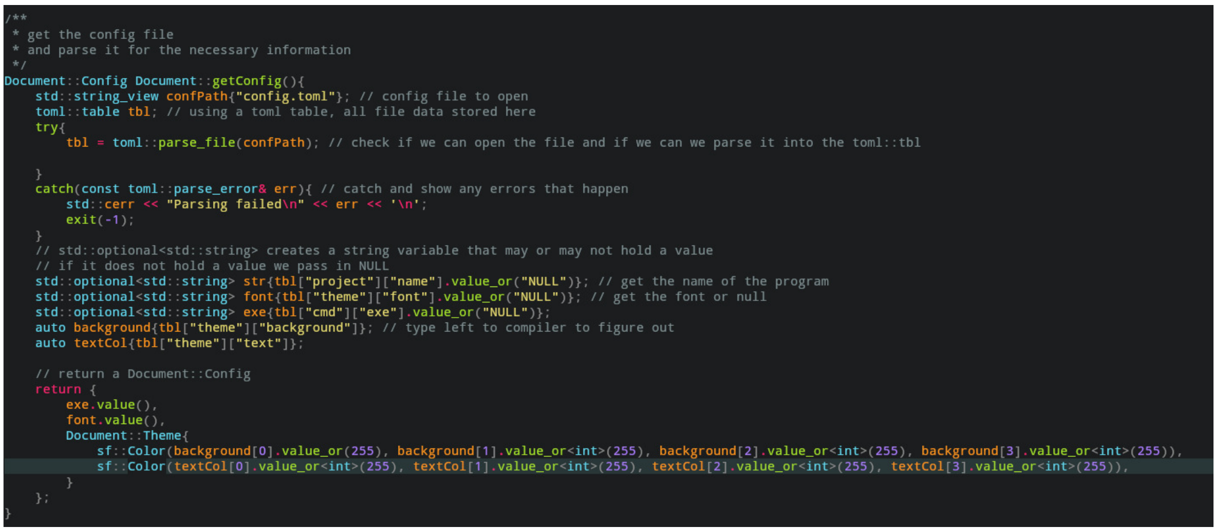
fmtlib is a formatting library for standard out printing, it gives rust and python style printing to Cpp whilst being fast and efficient. fmtlib is faster than normal cpp standard out functions and the c-style printf whilst being safer.


Speed tests

Library	Method	Run Time, s
libc	printf	0.91
libc++	std::ostream	2.49
{fmt} 9.1	fmt::print	0.74
Boost Format 1.80	boost::format	6.26
Folly Format	folly::format	1.87

{fmt} is the fastest of the benchmarked methods, ~20% faster than `printf`.

tomlc++ is a header only library that handles reading and writing to toml, json and yaml files. It handles everything in toml but can convert to json and yaml. It is very efficient and makes it easy for the user to configure the program


```cpp
/**
 * get the config file
 * and parse it for the necessary information
 */
Document::Config Document::getConfig(){
    std::string_view confPath{"config.toml"}; // config file to open
    toml::table tbl; // using a toml table, all file data stored here
    try{
        tbl = toml::parse_file(confPath); // check if we can open the file and if we can we parse it into the toml::tbl

    }
    catch(const toml::parse_error& err){ // catch and show any errors that happen
        std::cerr << "Parsing failed\n" << err << '\n';
        exit(-1);
    }
    // std::optional<std::string> creates a string variable that may or may not hold a value
    // if it does not hold a value we pass in NULL
    std::optional<std::string> str{tbl["project"]["name"].value_or("NULL")}; // get the name of the program
    std::optional<std::string> font{tbl["theme"]["font"].value_or("NULL")}; // get the font or null
    std::optional<std::string> exe{tbl["cmd"]["exe"].value_or("NULL")};
    auto background{tbl["theme"]["background"]}; // type left to compiler to figure out
    auto textCol{tbl["theme"]["text"]};

    // return a Document::Config
    return {
        exe.value(),
        font.value(),
        Document::Theme{
            sf::Color(background[0].value_or(255), background[1].value_or<int>(255), background[2].value_or<int>(255), background[3].value_or<int>(255)),
            sf::Color(textCol[0].value_or<int>(255), textCol[1].value_or<int>(255), textCol[2].value_or<int>(255), textCol[3].value_or<int>(255)),
        }
    };
}
```

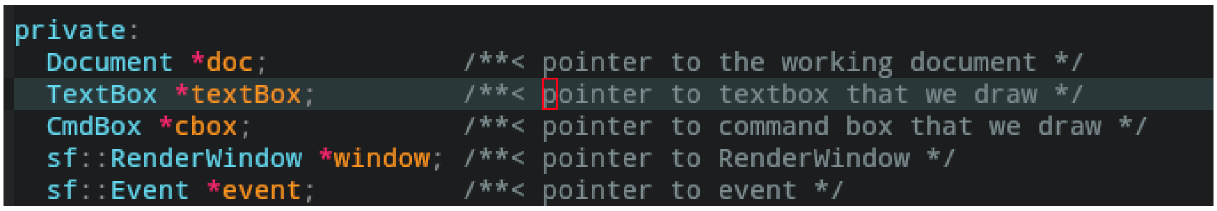
In SFML when writing text to a screen it takes a, const sf::String& string, which devolves into std::string types and char[] arrays. Due to this and a need to be efficient I made the choice to manipulate all text input and output in a dynamic one dimensional character array (std::string), By doing this any changes that can be made I just need to loop through the string checking each character for what im looking for. They take up minimal space since it is stored as a single string which is by default 24 bytes.

This also has the added benefit of always knowing its length and size as well as being able to convert into other types when needed.

Furthermore, I also made some optimisation decisions like, passing classes used through by pointer and dynamically allocating them on the heap when created. I did this because when they are passed through by pointer the program is only accessing one instance of it and not copying the class, manipulating it and then passing the values back to it when its done like what happens by default when passing a class through parameters. This choice speeds up the program since it doesnt need to copy and directly access the class.

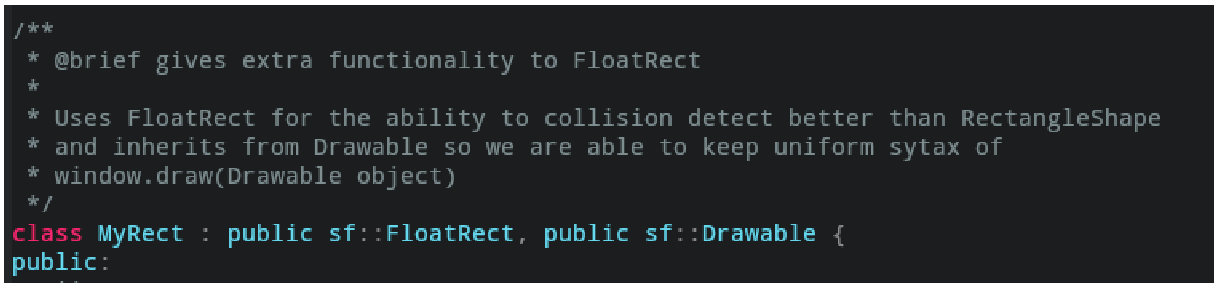
2.1.4.2 Design Choices in OOP When making the code I decided to use Object Oriented programming instead of Generic or Functional programming. This is because I wanted to maximise the modularbility of my code and make it easily scalable since all the related data would be grouped together and I can control how the data is modified behind the scenes leaving only the interface details to use.

I would use many different techniques such as, prefering composition over inheritance unless its necessary; polymorphic behaviour and class dynamic heap allocation.



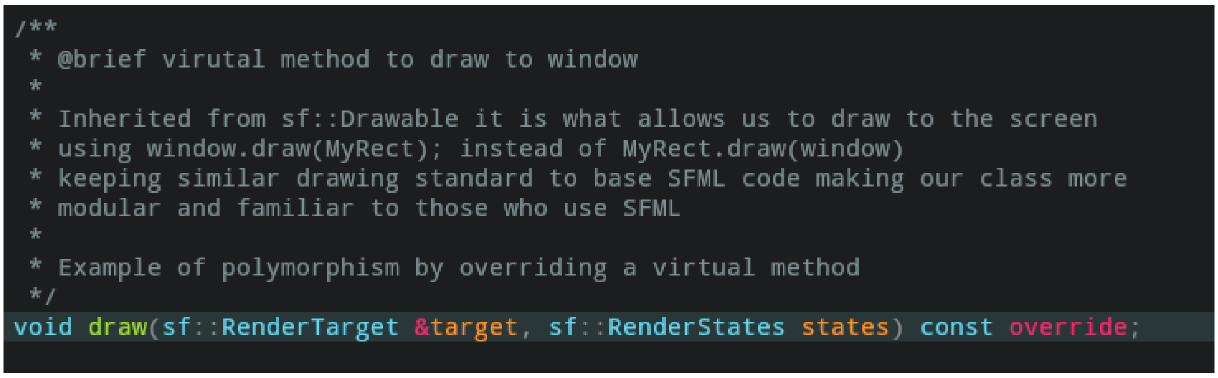
```
private:
  Document *doc;              /**< pointer to the working document */
  TextBox *textBox;           /**< pointer to textbox that we draw */
  CmdBox *cbox;               /**< pointer to command box that we draw */
  sf::RenderWindow *window;   /**< pointer to RenderWindow */
  sf::Event *event;           /**< pointer to event */
```

I would instantiate classes inside other classes so i can manipulate their behaviour without inherting it directly. I did this because if I mainly inherited the classes then when i change the values of the parent class through the child class and i want to pass those values to a different subroutine or class I would need to copy the data over or pass in the child class and use that to access the values in the parent class. But by having an instantiated class inside I can simply call that variable and not the whole child class.



```
/**
 * @brief gives extra functionality to FloatRect
 *
 * Uses FloatRect for the ability to collision detect better than RectangleShape
 * and inherits from Drawable so we are able to keep uniform sytax of
 * window.draw(Drawable object)
 */
class MyRect : public sf::FloatRect, public sf::Drawable {
public:
```

Inhertance is only done here so the class becomes recognised as Drawable by sf::RenderTargets i.e. Render←
Window. polymorphic behaviour is also shown here since when inheriting from sf::Drawable its mandatory to override the draw method.



```
/**
 * @brief virutal method to draw to window
 *
 * Inherited from sf::Drawable it is what allows us to draw to the screen
 * using window.draw(MyRect); instead of MyRect.draw(window)
 * keeping similar drawing standard to base SFML code making our class more
 * modular and familiar to those who use SFML
 *
 * Example of polymorphism by overriding a virtual method
 */
void draw(sf::RenderTarget &target, sf::RenderStates states) const override;
```

Finishing off this section is Dynamic class allocation. When instantiate classes inside others like TextBox inside Editor I would dynamically create them, add them to the heap and not stack, this is because the classes are decently large in memory and I would be changing the values held inside the classes fairly often and it would be more problematic to have it pushed onto the stack.

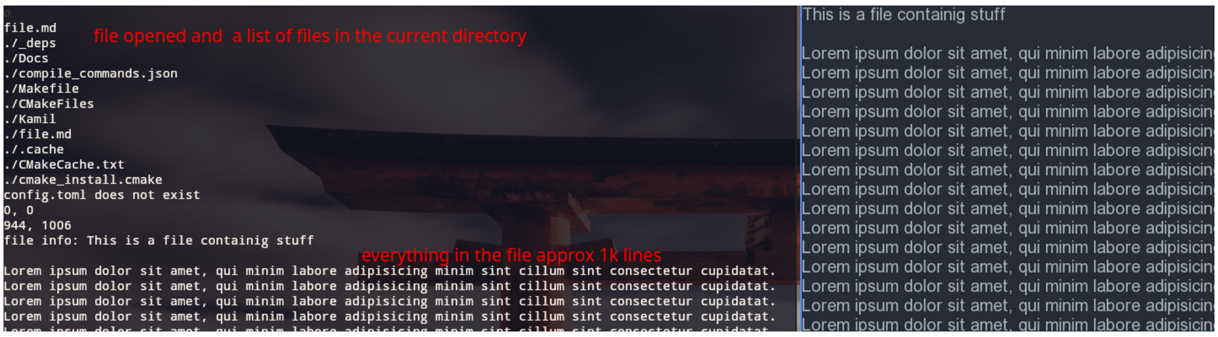
2.1.5 Testing the code

When it comes to testing the project I have 3 main points of focus:

1) The data being transferred around the project 2) The data is read from and saved to a file 3) Does it meet the objectives

Focussing on the first two points youll see that:

When it comes to testing the project when reading and saving to a file, I have made use of white-box testing. This form of testing is most apparent when the program is handling the file data and keyboard Inputs. Since white-box testing is the process of making the data flow apparent and that there are no broken loops through the code. I make sure that the data is printed out at each stage in plain text and in hex so that we can view it and see if any changes have happened that shouldnt have happened.


file opened and a list of files in the current directory
everything in the file approx 1k lines

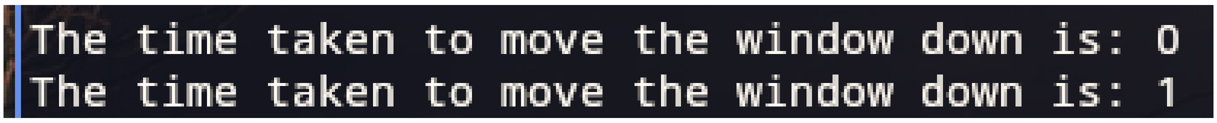
```
file.md
./_deps
./Docs
./compile_commands.json
./Makefile
./CMakeFiles
./Kamil
./file.md
./.cache
./CMakeCache.txt
./cmake_install.cmake
config.toml does not exist
0, 0
944, 1006
file info: This is a file containig stuff

Lorem ipsum dolor sit amet, qui minim labore adipisicing minim sint cillum sint consectetur cupidatat.
Lorem ipsum dolor sit amet, qui minim labore adipisicing minim sint cillum sint consectetur cupidatat.
Lorem ipsum dolor sit amet, qui minim labore adipisicing minim sint cillum sint consectetur cupidatat.
Lorem ipsum dolor sit amet, qui minim labore adipisicing minim sint cillum sint consectetur cupidatat.
```

```
This is a file containig stuff

Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
Lorem ipsum dolor sit amet, qui minim labore adipisicin
```

When checking how large the data I could send around the program was I made use of light integration testing by increasing the size of file and data the program transferes each time but I am yet to hit the maximum. It theoretically should be the same size as the computer allows since everything is trasnfered through a string which is dynamic ini size and we are most of the time accessing it directly in memory wich gets rid of any extra overhead.

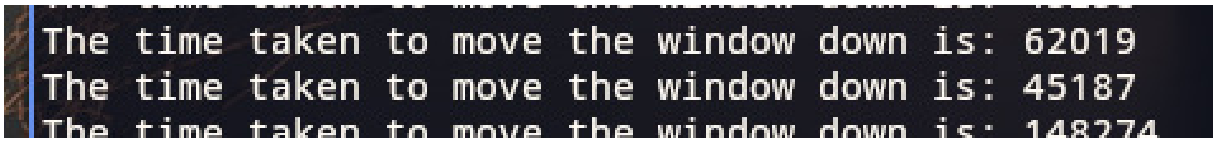
Onto the final point of it meeting the objectives.

The program is optimised and extremly quick opening approx 1k lines immediatly. key presses were routinely sub 1 millisecond.



```
The time taken to move the window down is: 0
The time taken to move the window down is: 1
```

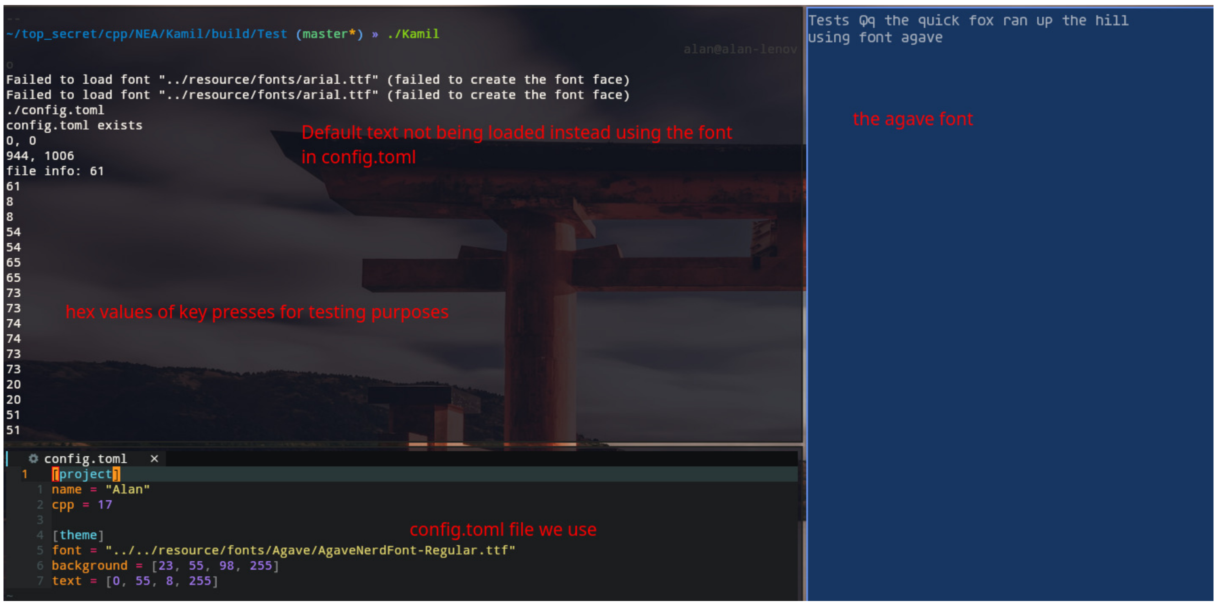
Figure 5 millisecond time



```
The time taken to move the window down is: 62019
The time taken to move the window down is: 45187
The time taken to move the window down is: 148274
```

Figure 6 nanosecond time

The project can be loaded from a config file called config.toml



```
~/top_secret/cpp/NEA/Kamil/build/Test (master*) » ./Kamil
                                                                alan@alan-lenov
Failed to load font "../resource/fonts/arial.ttf" (failed to create the font face)
Failed to load font "../resource/fonts/arial.ttf" (failed to create the font face)
./config.toml
config.toml exists
0, 0
944, 1006
file info: 61
61
8
8
54
54
65
65
73
73
74
74
73
73
20
20
51
51
```

```
config.toml
[project]
name = "Alan"
cpp = 17

[theme]
font = "../../resource/fonts/Agave/AgaveNerdFont-Regular.ttf"
background = [23, 55, 98, 255]
text = [0, 55, 8, 255]
```

```
Tests Qq the quick fox ran up the hill
using font agave
```

Default text not being loaded instead using the font in config.toml
hex values of key presses for testing purposes
config.toml file we use
the agave font

The project works is cross platform and works in windows

∗+(show work in linux)

3 Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

4 Hierarchical Index

4.1 Class Hierarchy

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

5 Class Index

5.1 Class List

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

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Namespace Documentation

7.1 KEYS Namespace Reference

An enum for Keyboard characters in hex form.

Enumerations

- enum {
  ESCAPE = 0x1B , ENTER = 0xD , BS = 0x8 , Shift_A = 0x41 ,
  CTRL = 0x11 , DELETE = 0x7f , CR = 0x13 , UP_ARROW = 0x48 ,
  DOWN_ARROW = 0x50 , RIGHT_ARROW = 0x4D , LEFT_ARROW = 0x4B }

7.1.1 Detailed Description

An enum for Keyboard characters in hex form.

Used for when we need to check if some characters are being entered into the textbox since not all characters can be directly drawn to the screen but SFML will try to. We can intercept them here and change or mute their behaviour.

7.1.2 Enumeration Type Documentation

Enumerator

7.1.2.1 anonymous enum `anonymous enum`

Enumerator

ESCAPE	
ENTER	
BS	
Shift_A	
CTRL	
DELETE	
CR	
UP_ARROW	
DOWN_ARROW	
RIGHT_ARROW	
LEFT_ARROW	

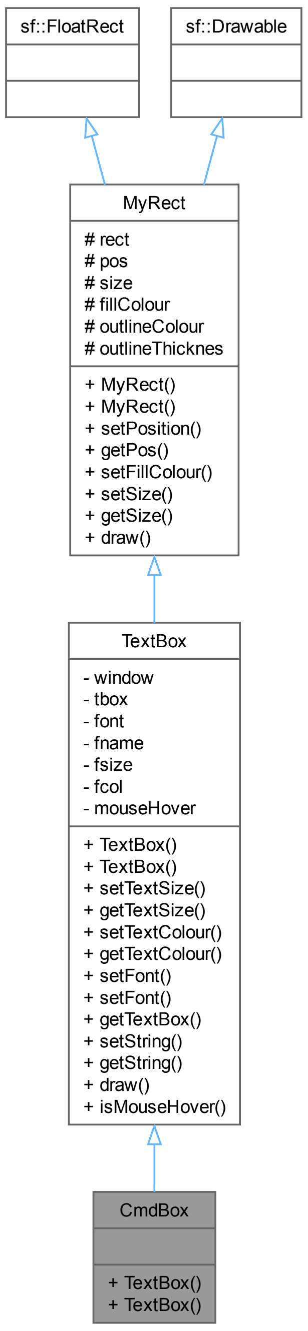
8 Class Documentation

8.1 CmdBox Class Reference

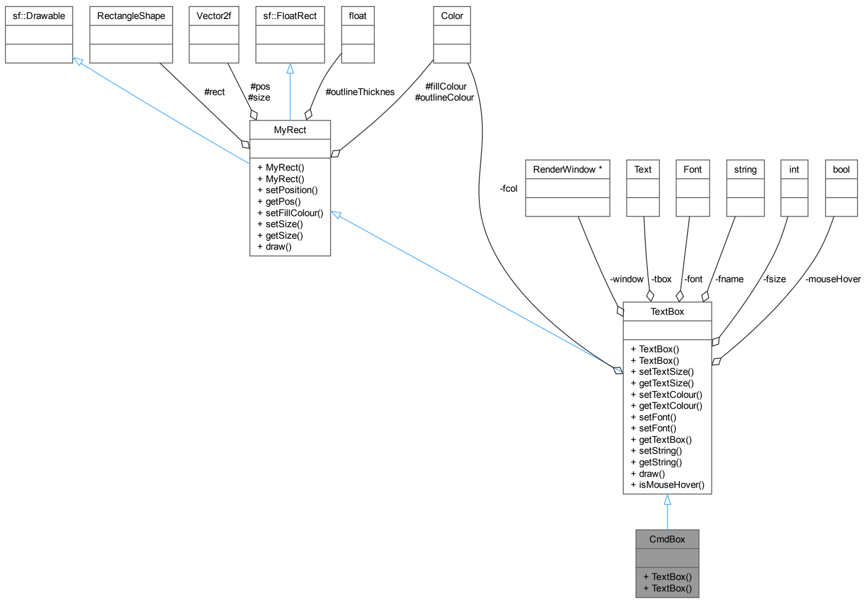
Class to handle the command TextBox.

```
#include <CmdBox.h>
```

Inheritance diagram for CmdBox:



Collaboration diagram for CmdBox:



Public Member Functions

- TextBox (sf::RenderWindow ∗win, sf::Vector2f pos, sf::Vector2f size, std::string sfont, int fsize, sf::Color fcol, sf::Color background, float thicc)

  Using the Parent class constructor.
- TextBox ()

  Using the Parent class constructor.

Public Member Functions inherited from TextBox

- TextBox (sf::RenderWindow ∗win, sf::Vector2f pos, sf::Vector2f size, std::string sfont, int fsize, sf::Color fcol, sf::Color background, float thicc)

  Constructor for TextBox.
- TextBox ()
- void setTextSize (int size)

  Set the size of the text.
- int getTextSize () const

  Get the size of the text.
- void setTextColour (sf::Color colour)

  Set the colour of the text.
- sf::Color getTextColour () const

  Get the colour of the text.
- void setFont (sf::Font &font)

  set what font you want to use
- void setFont (std::string font)

  set what font you use
- sf::Text getTextBox () const

Get sf::Text of the textbox.
- void setString (std::string nstring)

  Sets the string.
- std::string getString () const

  returns the text in tbox
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  used to draw to the screen virutal method inherited from MyRect which inherited it from sf::Drawable thats overrided here it is an example of polymorphism as we are changing the behaviour of a method in the child class. By inheriting from sf::Drawable it allows us to keep a similar syntax to other SFML shapes and drawable objects window.draw(my←_object). This allows our code to be more modular and easy for other people to use since they dont need to fumble around with my_object.draw(window)
- bool isMouseHover ()

  check if mouse is hovering over current textbox

Public Member Functions inherited from MyRect

- MyRect (sf::Vector2f pos, sf::Vector2f size, sf::Color fillColour, sf::Color outlineColour, float outlineThicknes)

  constructor for MyRect
- MyRect ()
- void setPosition (sf::Vector2f pos)

  sets the position of rect
- sf::Vector2f getPos () const

  get the position of rect
- void setFillColour (sf::Color colour)

  set the fill colour of the rect
- void setSize (sf::Vector2f size)

  set the size of the rect
- sf::Vector2f getSize () const

  get the size of the rect
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  virutal method to draw to window

Additional Inherited Members

Protected Attributes inherited from MyRect

- sf::RectangleShape rect
- sf::Vector2f pos
- sf::Vector2f size
- sf::Color fillColour
- sf::Color outlineColour
- float outlineThicknes

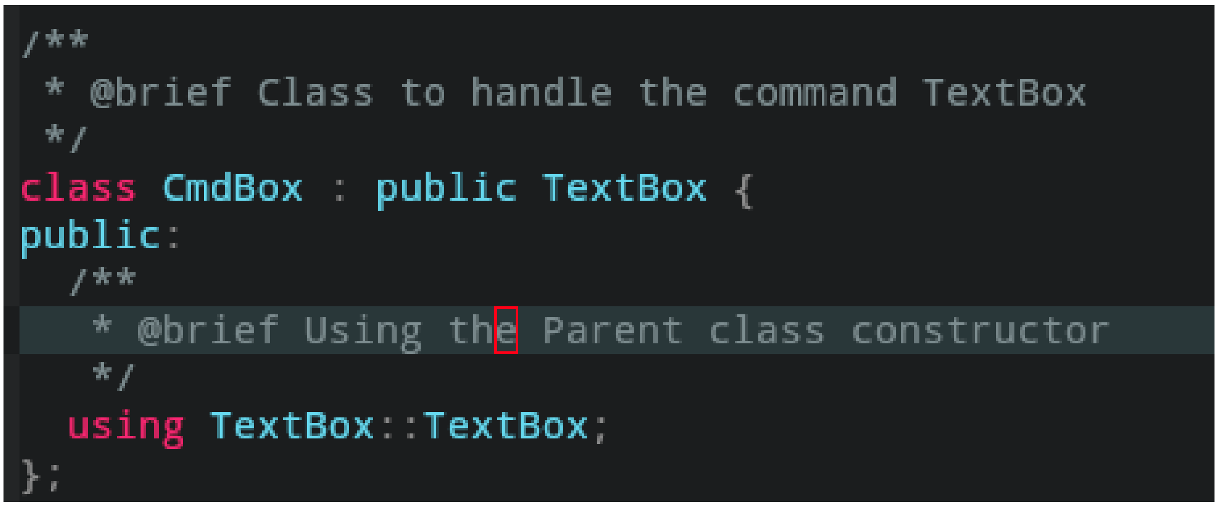
8.1.1 Detailed Description

Class to handle the command TextBox.

8.1.2 Member Function Documentation

8.1.2.1 TextBox() [1/2] `TextBox::TextBox ( )`

Using the Parent class constructor.

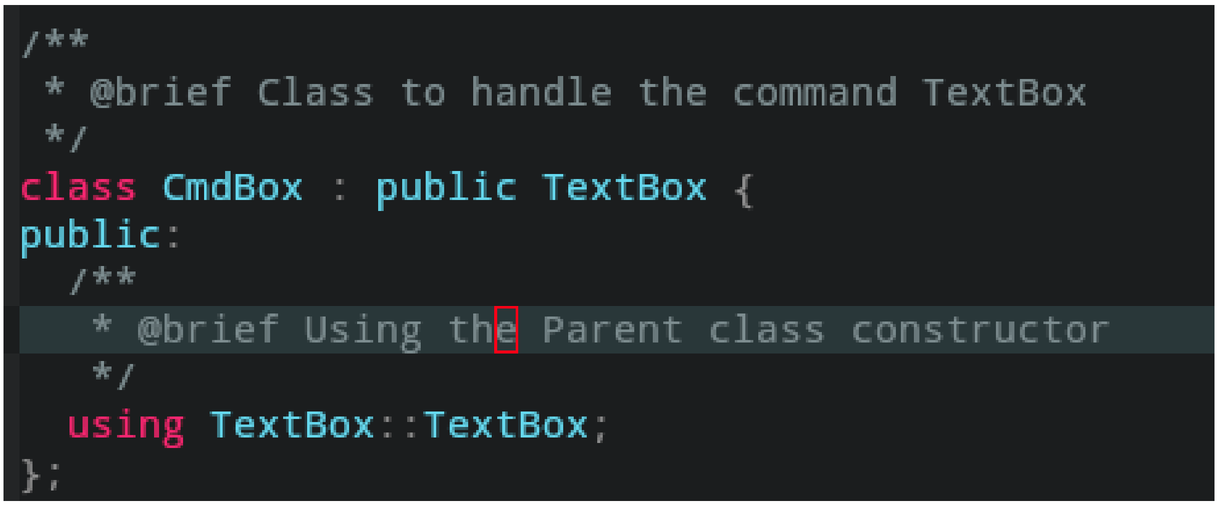


```
/**
 * @brief Class to handle the command TextBox
 */
class CmdBox : public TextBox {
public:
  /**
   * @brief Using the Parent class constructor
   */
  using TextBox::TextBox;
};
```

8.1.2.2 TextBox() [2/2]

```
TextBox::TextBox (
        sf::RenderWindow * win,
        sf::Vector2f pos,
        sf::Vector2f size,
        std::string sfont,
        int fsize,
        sf::Color fcol,
        sf::Color background,
        float thicc )
```

Using the Parent class constructor.



```
/**
 * @brief Class to handle the command TextBox
 */
class CmdBox : public TextBox {
public:
  /**
   * @brief Using the Parent class constructor
   */
  using TextBox::TextBox;
};
```

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

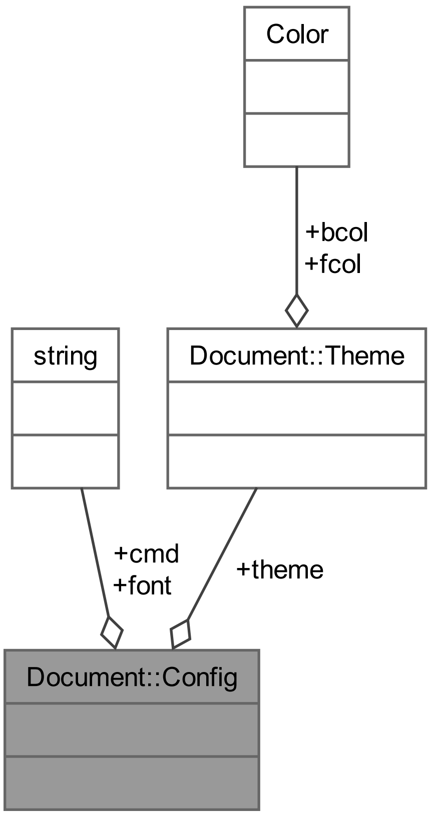
- include/Kamil/CmdBox.h

8.2 Document::Config Struct Reference

A struct for the configuration.

`#include <Document.h>`

Collaboration diagram for Document::Config:



Public Attributes

- std::string cmd
- std::string font
- Theme theme

8.2.1 Detailed Description

A struct for the configuration.

A struct containing all the necessary information for the configuration of the Text editor

Parameters

std::string	- The command to run the programs i.e. "python3" will execute python3
std::string	- The font
Theme	the struct containing theme information

8.2.2 Member Data Documentation

8.2.2.1 cmd `std::string Document::Config::cmd`

8.2.2.2 font `std::string Document::Config::font`

8.2.2.3 theme `Theme Document::Config::theme`

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

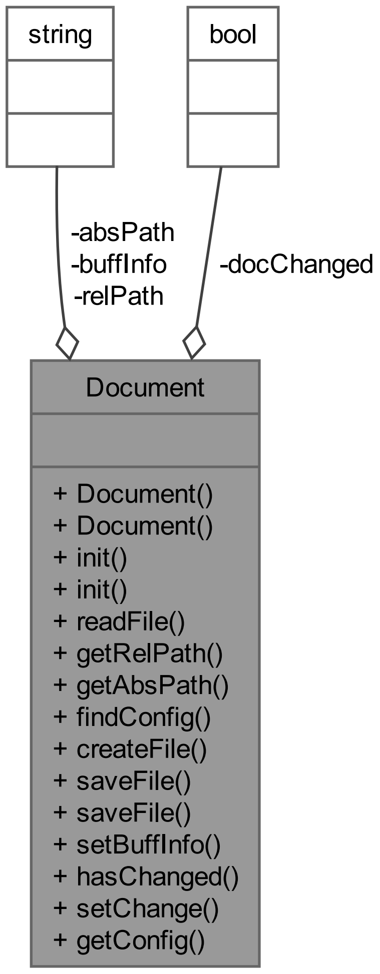
- include/Kamil/Document.h

8.3 Document Class Reference

Document class.

```
#include <Document.h>
```

Collaboration diagram for Document:



Classes

- struct Config

  A struct for the configuration.
- struct Theme

  a struct for the Theme

Public Member Functions

- Document ()

  Constructor for Document class.
- Document (std::string fileP)

  Constructor for Document class.
- void init ()

initialise the file
- void init (std::string inF)

  initialise the file
- std::string readFile ()

  read the file
- std::string getRelPath ()

  get the relative path
- std::string getAbsPath ()

  get the relative path
- bool findConfig ()

  check if the config.toml exist
- void createFile (std::string filename)

  create the file
- bool saveFile (const std::string &filename)

  save to a file
- bool saveFile ()

  save to a file
- void setBuffInfo (std::string info)

  save file infor to buffer
- bool hasChanged ()

  if the file has changed
- void setChange ()

  set file has changed
- Config getConfig ()

  retrieves information from config.toml

Private Attributes

- std::string relPath
- std::string absPath
- std::string buffInfo
- bool docChanged

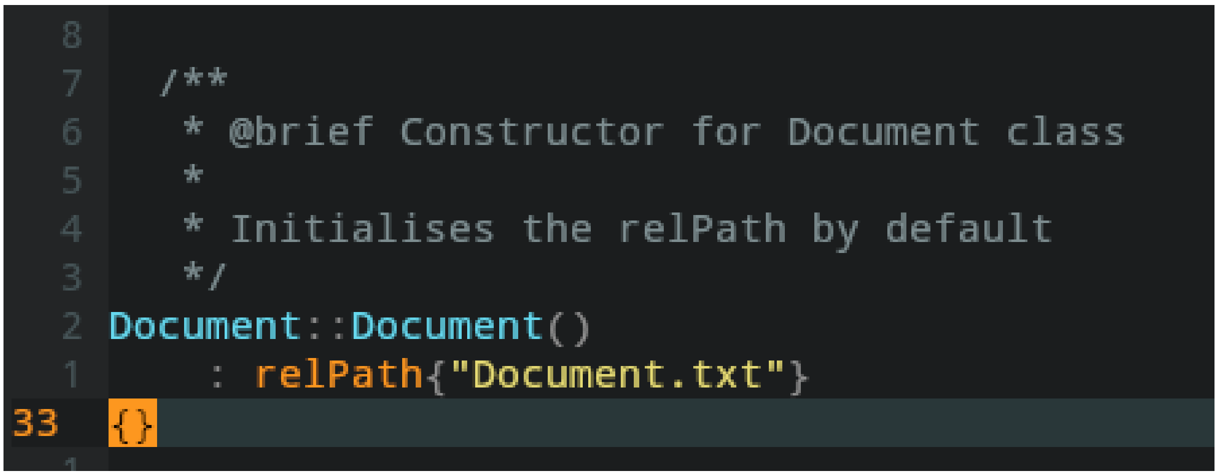
8.3.1 Detailed Description

Document class.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Document() [1/2] `Document::Document ( )`

Constructor for Document class.



```
  /**
   * @brief Constructor for Document class
   *
   * Initialises the relPath by default
   */
Document::Document()
    : relPath{"Document.txt"}
{}
```

Figure 7 Constructor for Document

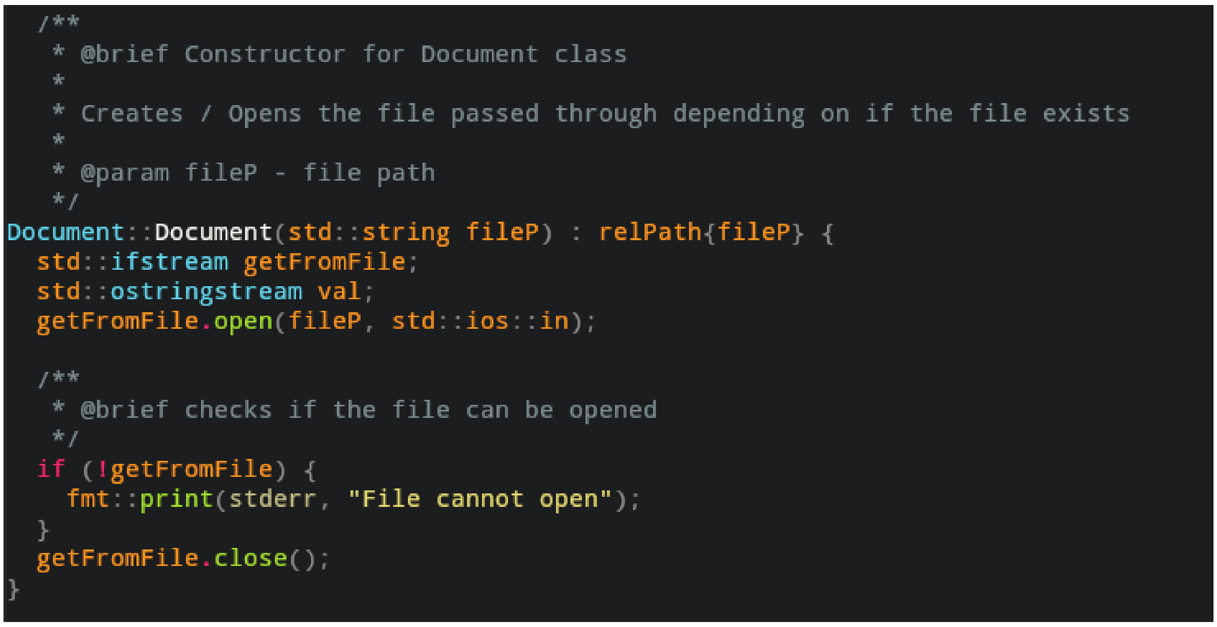
Initialises the relPath by default

8.3.2.2 Document() [2/2] `Document::Document ( std::string fileP )`

Constructor for Document class.

Parameters

fileP	- file path



```
  /**
   * @brief Constructor for Document class
   *
   * Creates / Opens the file passed through depending on if the file exists
   *
   * @param fileP - file path
   */
Document::Document(std::string fileP) : relPath{fileP} {
  std::ifstream getFromFile;
  std::ostringstream val;
  getFromFile.open(fileP, std::ios::in);

  /**
   * @brief checks if the file can be opened
   */
  if (!getFromFile) {
    fmt::print(stderr, "File cannot open");
  }
  getFromFile.close();
}
```

Figure 8 Constructor for Document

Creates / Opens the file passed through depending on if the file exists

Parameters

fileP	- file path

checks if the file can be opened

8.3.3 Member Function Documentation

8.3.3.1 createFile() `void Document::createFile ( std::string filename )`

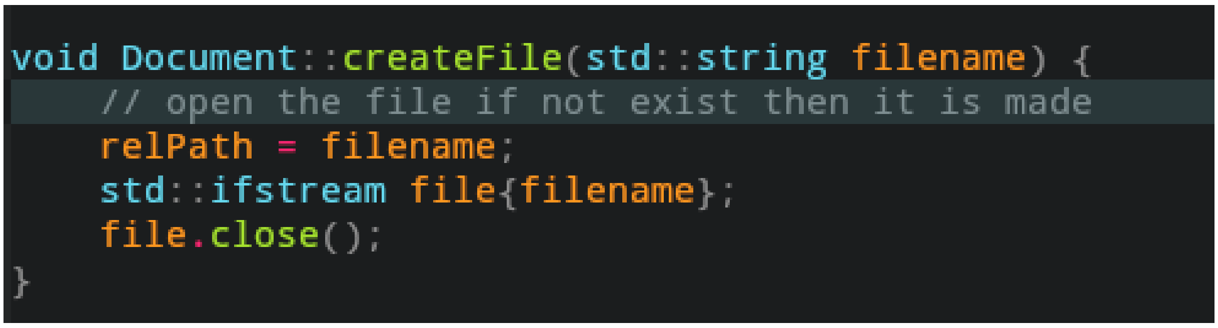
create the file

Parameters

std::string	- file name

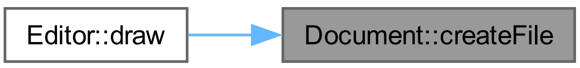
Returns

void



```
void Document::createFile(std::string filename) {
    // open the file if not exist then it is made
    relPath = filename;
    std::ifstream file{filename};
    file.close();
}
```

Here is the caller graph for this function:



8.3.3.2 findConfig() `bool Document::findConfig ( )`

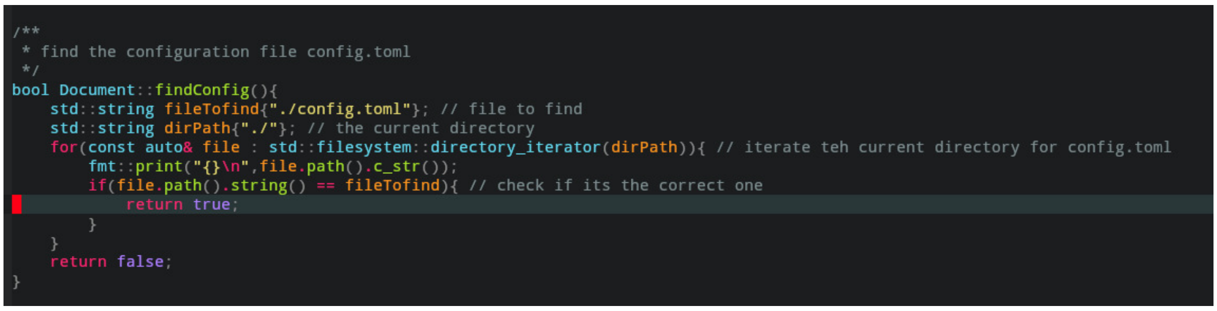
check if the config.toml exist

Parameters

void	

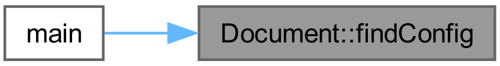
Returns

bool - true if config.toml exists



```
/**
 * find the configuration file config.toml
 */
bool Document::findConfig(){
    std::string fileTofind{"./config.toml"}; // file to find
    std::string dirPath{"./"}; // the current directory
    for(const auto& file : std::filesystem::directory_iterator(dirPath)){ // iterate teh current directory for config.toml
        fmt::print("{}\n",file.path().c_str());
        if(file.path().string() == fileTofind){ // check if its the correct one
            return true;
        }
    }
    return false;
}
```

find the configuration file config.toml Here is the caller graph for this function:



8.3.3.3 getAbsPath() `std::string Document::getAbsPath ( )`

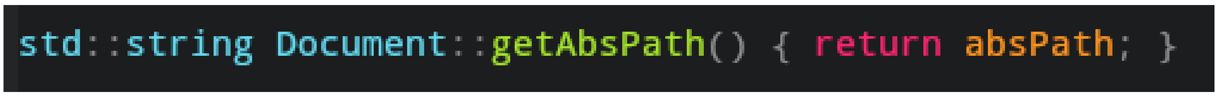
get the relative path

Parameters

void	

Returns

string for absolute path



```
std::string Document::getAbsPath() { return absPath; }
```

8.3.3.4 getConfig() `Document::Config Document::getConfig ( )`

retrieves information from config.toml

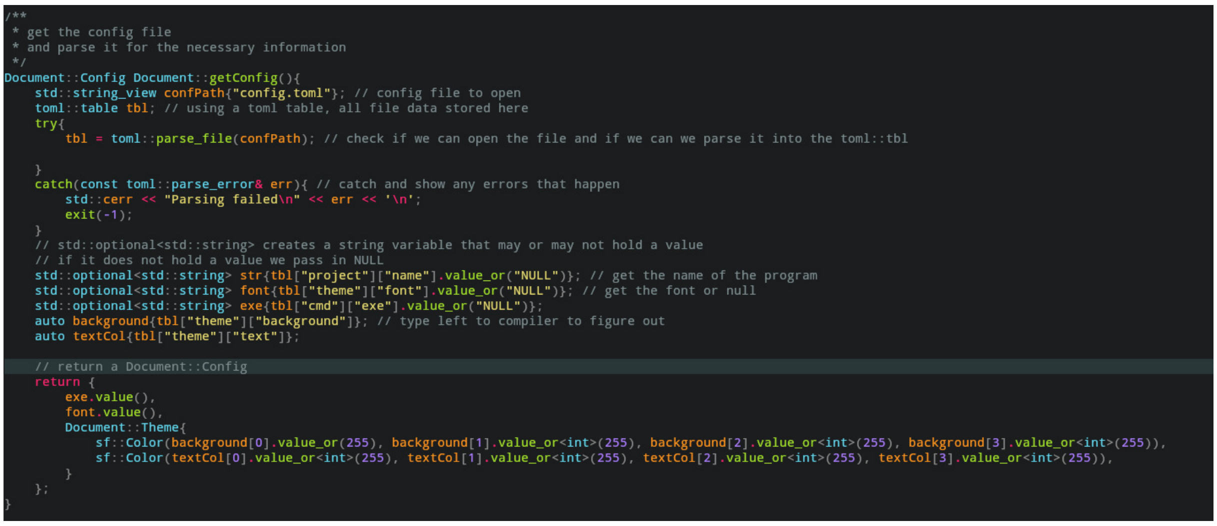
searches through the config.toml files for the necessary information and extracts it.

Parameters

void	

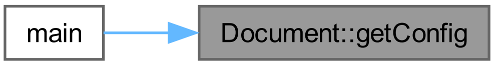
Returns

Config - config.toml information



```
/**
 * get the config file
 * and parse it for the necessary information
 */
Document::Config Document::getConfig(){
    std::string_view confPath{"config.toml"}; // config file to open
    toml::table tbl; // using a toml table, all file data stored here
    try{
        tbl = toml::parse_file(confPath); // check if we can open the file and if we can we parse it into the toml::tbl

    }
    catch(const toml::parse_error& err){ // catch and show any errors that happen
        std::cerr << "Parsing failed\n" << err << '\n';
        exit(-1);
    }
    // std::optional<std::string> creates a string variable that may or may not hold a value
    // if it does not hold a value we pass in NULL
    std::optional<std::string> str{tbl["project"]["name"].value_or("NULL")}; // get the name of the program
    std::optional<std::string> font{tbl["theme"]["font"].value_or("NULL")}; // get the font or null
    std::optional<std::string> exe{tbl["cmd"]["exe"].value_or("NULL")};
    auto background{tbl["theme"]["background"]}; // type left to compiler to figure out
    auto textCol{tbl["theme"]["text"]};

    // return a Document::Config
    return {
        exe.value(),
        font.value(),
        Document::Theme{
            sf::Color(background[0].value_or(255), background[1].value_or<int>(255), background[2].value_or<int>(255), background[3].value_or<int>(255)),
            sf::Color(textCol[0].value_or<int>(255), textCol[1].value_or<int>(255), textCol[2].value_or<int>(255), textCol[3].value_or<int>(255)),
        }
    };
}
```

get the config file and parse it for the necessary information Here is the caller graph for this function:



8.3.3.5 getRelPath() `std::string Document::getRelPath ( )`

get the relative path

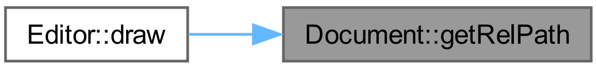
Parameters

void	

Returns

string for relative path

Here is the caller graph for this function:



8.3.3.6 hasChanged() `bool Document::hasChanged ( )`

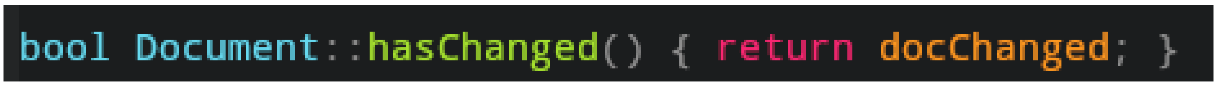
if the file has changed

Parameters

void	

Returns

bool - true if file has changed



```
bool Document::hasChanged() { return docChanged; }
```

8.3.3.7 init() `[1/2]` `void Document::init ( )`

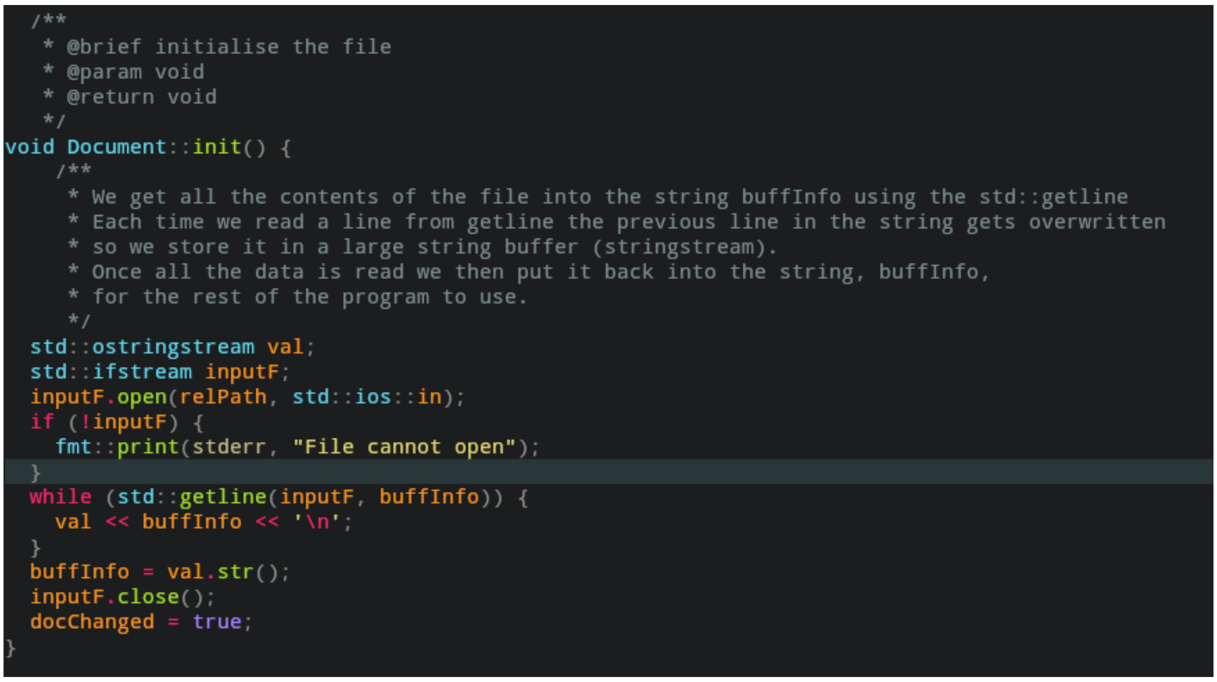
initialise the file

Parameters

void	

Returns

void



```
  /**
   * @brief initialise the file
   * @param void
   * @return void
   */
void Document::init() {
    /**
     * We get all the contents of the file into the string buffInfo using the std::getline
     * Each time we read a line from getline the previous line in the string gets overwritten
     * so we store it in a large string buffer (stringstream).
     * Once all the data is read we then put it back into the string, buffInfo,
     * for the rest of the program to use.
     */
  std::ostringstream val;
  std::ifstream inputF;
  inputF.open(relPath, std::ios::in);
  if (!inputF) {
    fmt::print(stderr, "File cannot open");
  }
  while (std::getline(inputF, buffInfo)) {
    val << buffInfo << '\n';
  }
  buffInfo = val.str();
  inputF.close();
  docChanged = true;
}
```

Figure 9 init method for Document

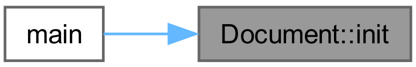
Parameters

void	

Returns

void

We get all the contents of the file into the string buffInfo using the std::getline Each time we read a line from getline the previous line in the string gets overwritten so we store it in a large string buffer (stringstream). Once all the data is read we then put it back into the string, buffInfo, for the rest of the program to use.Here is the caller graph for this function:



8.3.3.8 init() [2/2] `void Document::init ( std::string inF )`

initialise the file

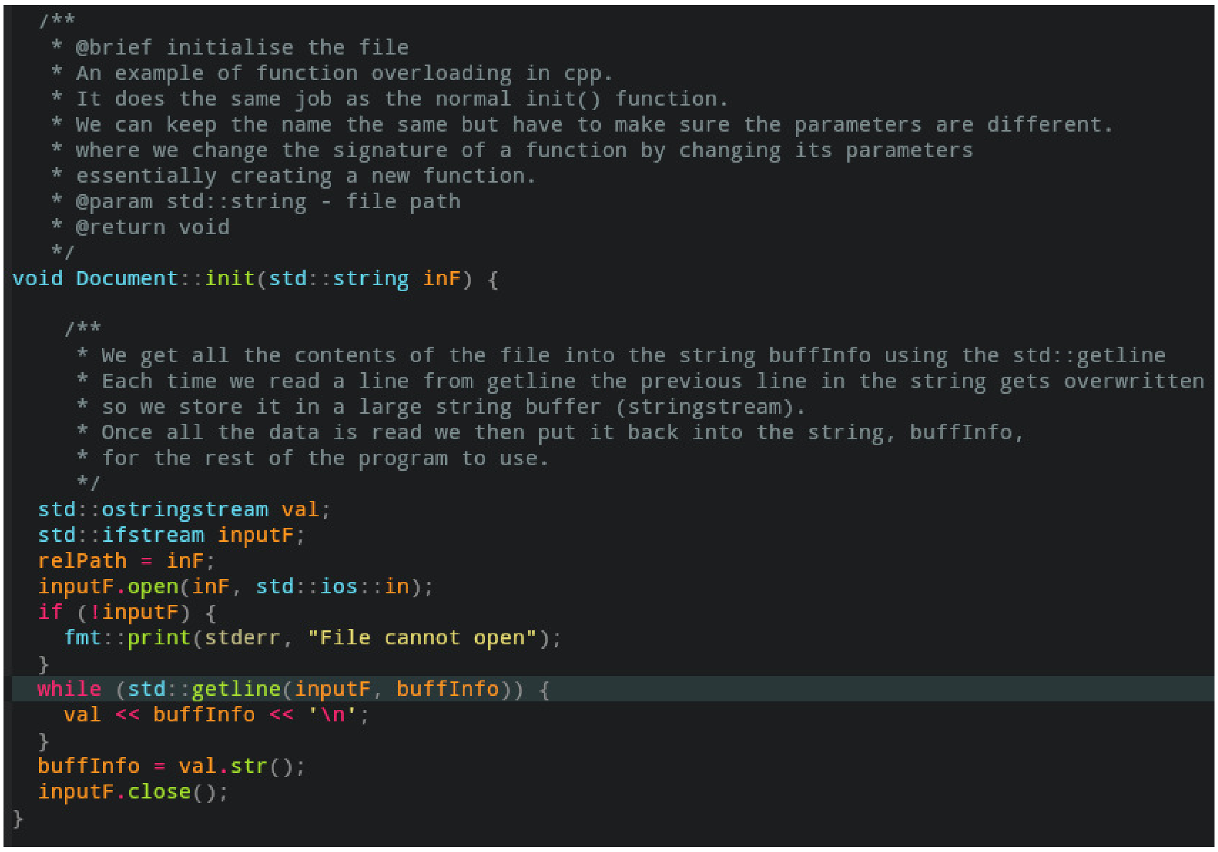
initialise the file An example of function overloading in cpp. It does the same job as the normal init() function. We can keep the name the same but have to make sure the parameters are different. where we change the signature of a function by changing its parameters essentially creating a new function.

Parameters

inF	- file location

Returns

void



```
  /**
   * @brief initialise the file
   * An example of function overloading in cpp.
   * It does the same job as the normal init() function.
   * We can keep the name the same but have to make sure the parameters are different.
   * where we change the signature of a function by changing its parameters
   * essentially creating a new function.
   * @param std::string - file path
   * @return void
   */
void Document::init(std::string inF) {

    /**
     * We get all the contents of the file into the string buffInfo using the std::getline
     * Each time we read a line from getline the previous line in the string gets overwritten
     * so we store it in a large string buffer (stringstream).
     * Once all the data is read we then put it back into the string, buffInfo,
     * for the rest of the program to use.
     */
  std::ostringstream val;
  std::ifstream inputF;
  relPath = inF;
  inputF.open(inF, std::ios::in);
  if (!inputF) {
    fmt::print(stderr, "File cannot open");
  }
  while (std::getline(inputF, buffInfo)) {
    val << buffInfo << '\n';
  }
  buffInfo = val.str();
  inputF.close();
}
```

Figure 10 init method for Document

Parameters

std::string	- file path

Returns

void

We get all the contents of the file into the string buffInfo using the std::getline Each time we read a line from getline the previous line in the string gets overwritten so we store it in a large string buffer (stringstream). Once all the data is read we then put it back into the string, buffInfo, for the rest of the program to use.

8.3.3.9 readFile() `std::string Document::readFile ( )`

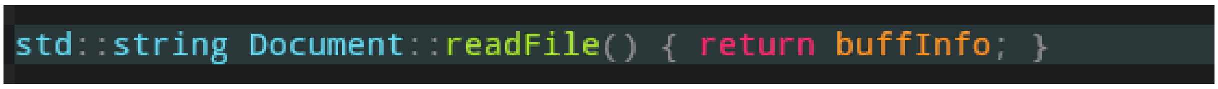
read the file

Parameters

void	

Returns

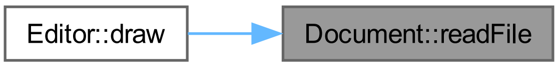
string containing the file info



```
std::string Document::readFile() { return buffInfo; }
```

Figure 11 readFile method for Document

Here is the caller graph for this function:



8.3.3.10 saveFile() `[1/2]` `bool Document::saveFile ( )`

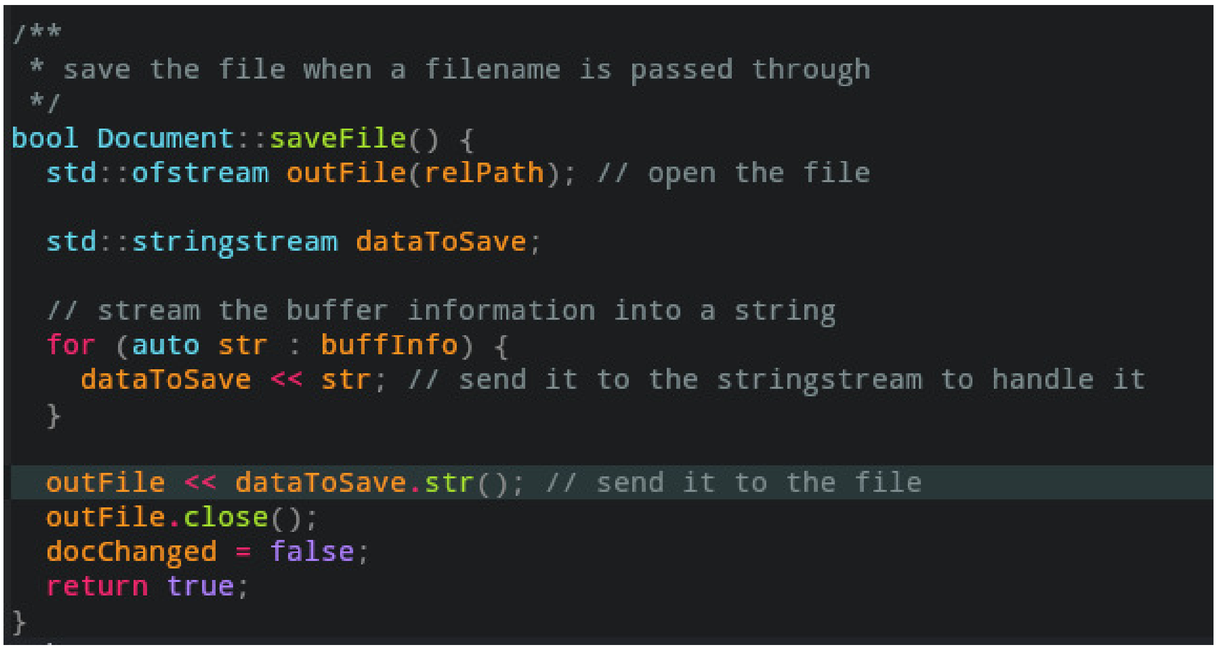
save to a file

Parameters

void	

Returns

bool - true if saved



```
/**
 * save the file when a filename is passed through
 */
bool Document::saveFile() {
  std::ofstream outFile(relPath); // open the file

  std::stringstream dataToSave;

  // stream the buffer information into a string
  for (auto str : buffInfo) {
    dataToSave << str; // send it to the stringstream to handle it
  }

  outFile << dataToSave.str(); // send it to the file
  outFile.close();
  docChanged = false;
  return true;
}
```

save the file when a filename is passed through

8.3.3.11 saveFile() [2/2] `bool Document::saveFile ( const std::string & filename )`

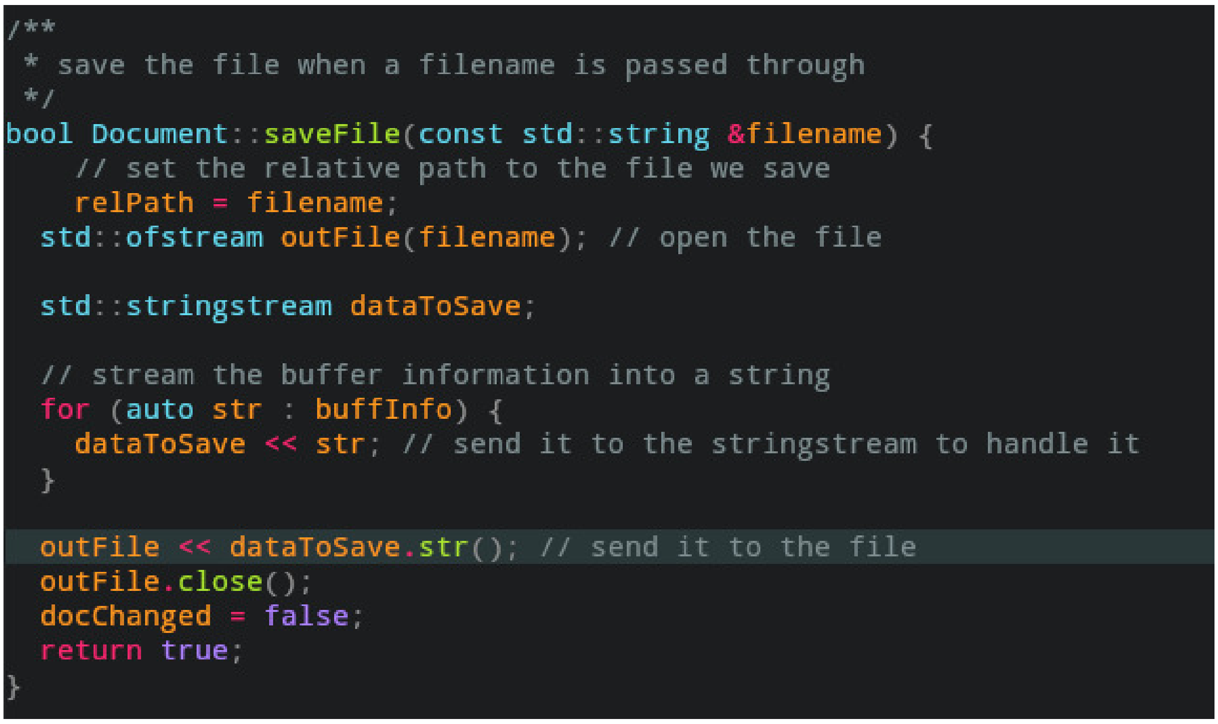
save to a file

Parameters

string	- filename to save to

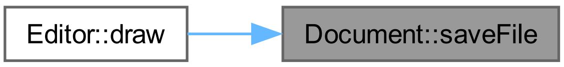
Returns

bool - true if saved



```
/**
 * save the file when a filename is passed through
 */
bool Document::saveFile(const std::string &filename) {
    // set the relative path to the file we save
    relPath = filename;
  std::ofstream outFile(filename); // open the file

  std::stringstream dataToSave;

  // stream the buffer information into a string
  for (auto str : buffInfo) {
    dataToSave << str; // send it to the stringstream to handle it
  }

  outFile << dataToSave.str(); // send it to the file
  outFile.close();
  docChanged = false;
  return true;
}
```

save the file when a filename is passed through Here is the caller graph for this function:



8.3.3.12 setBuffInfo() `void Document::setBuffInfo ( std::string info )`

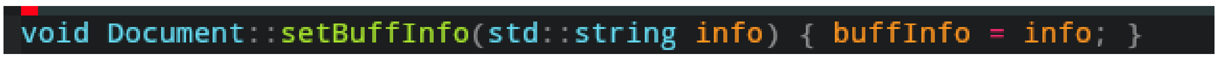
save file infor to buffer

Parameters

string	buffer info

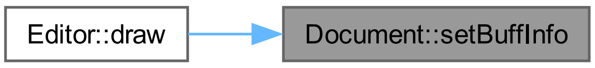
Returns

void



```
void Document::setBuffInfo(std::string info) { buffInfo = info; }
```

Here is the caller graph for this function:



8.3.3.13 setChange() `void Document::setChange ( )`

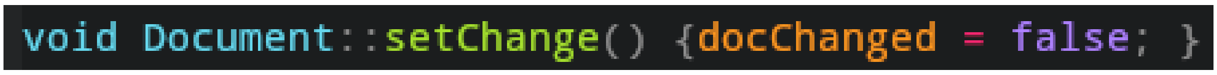
set file has changed

Parameters

void	

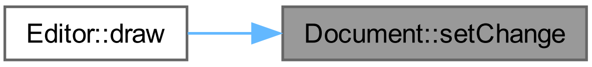
Returns

void



```
void Document::setChange() {docChanged = false; }
```

Here is the caller graph for this function:



8.3.4 Member Data Documentation

8.3.4.1 absPath `std::string Document::absPath [private]`

absolute path

8.3.4.2 buffInfo `std::string Document::buffInfo [private]`

8.3.4.3 docChanged `bool Document::docChanged [private]`

buffer information (the file text) if the file has changed

8.3.4.4 relPath `std::string Document::relPath [private]`

relative path

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

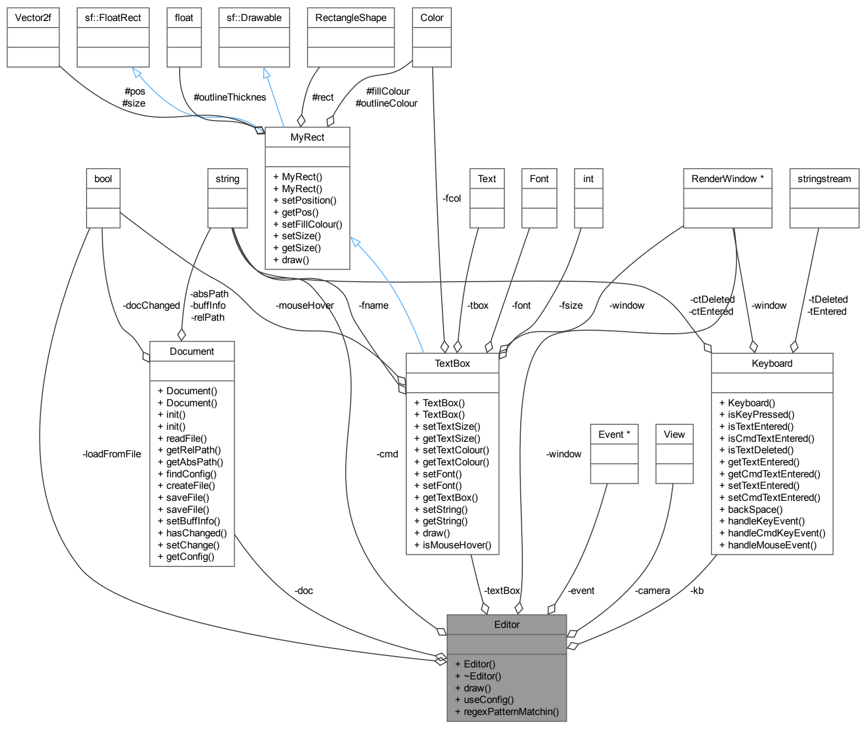
- include/Kamil/Document.h
- src/Document.cpp

8.4 Editor Class Reference

Class that is a centre for how the other classes interact with each other and draws everything in the Editor to the screen.

`#include <Editor.h>`

Collaboration diagram for Editor:



Public Member Functions

- Editor (sf::RenderWindow ∗window, sf::Event ∗event, Document ∗doc)

  Constructor for Editor.
- ∼Editor ()

  Destructor for Editor class.
- void draw ()

  function that draws everything to RenderWindow
- void useConfig (const Document::Config &conf)

  changes the editor looks and workings
- void regexPatternMatchin ()

  match patterns in the string

Private Attributes

- Document ∗ doc
- TextBox ∗ textBox
- sf::RenderWindow ∗ window
- sf::Event ∗ event
- sf::View camera
- Keyboard kb
- std::string cmd
- bool loadFromFile

8.4.1 Detailed Description

Class that is a centre for how the other classes interact with each other and draws everything in the Editor to the screen.

8.4.2 Constructor & Destructor Documentation

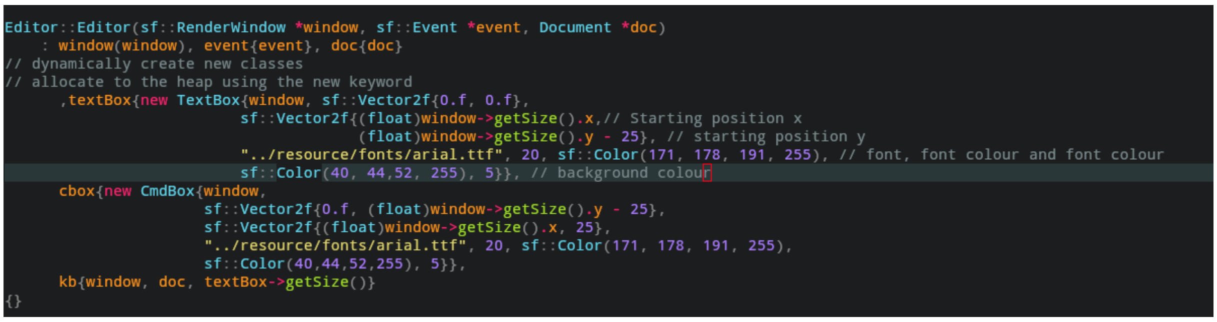
8.4.2.1 Editor()

```
Editor::Editor (
    sf::RenderWindow * window,
    sf::Event * event,
    Document * doc )
```

Constructor for Editor.

Parameters

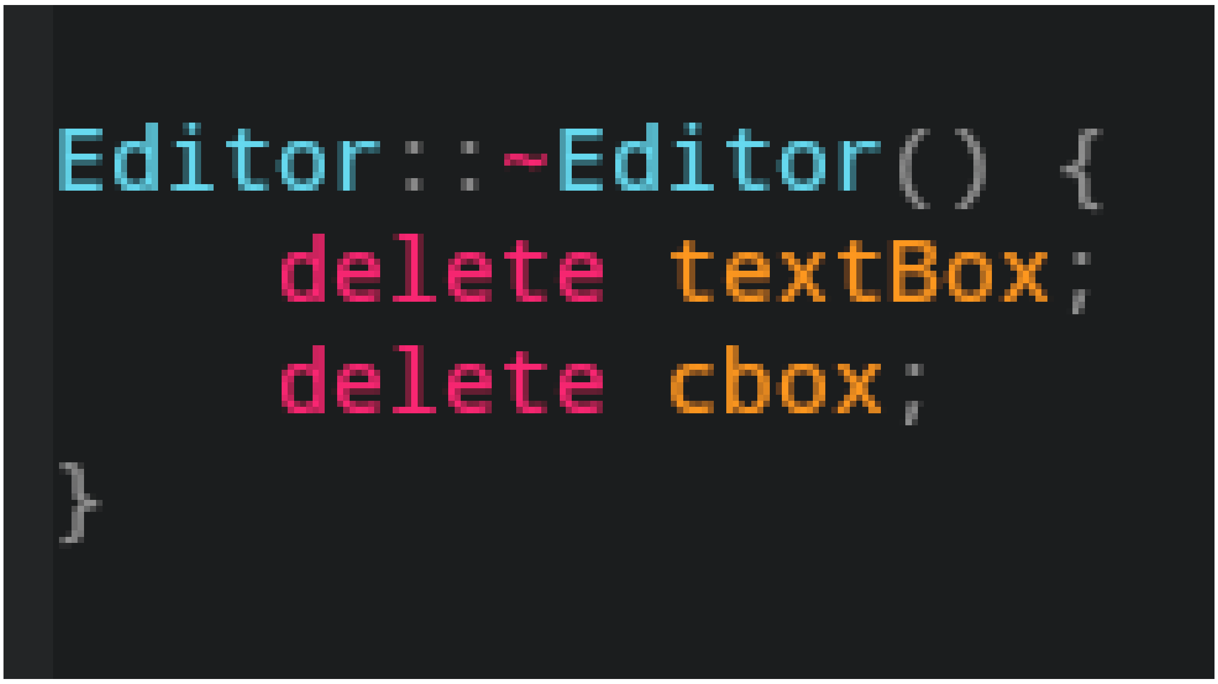
window	- pointer to main RenderWindow
event	- pointer to main event
doc	- pointer to document



```
Editor::Editor(sf::RenderWindow *window, sf::Event *event, Document *doc)
    : window(window), event{event}, doc{doc}
// dynamically create new classes
// allocate to the heap using the new keyword
      ,textBox{new TextBox{window, sf::Vector2f{0.f, 0.f},
                          sf::Vector2f{(float)window->getSize().x,// Starting position x
                                       (float)window->getSize().y - 25}, // starting position y
                          "../resource/fonts/arial.ttf", 20, sf::Color(171, 178, 191, 255), // font, font colour and font colour
                          sf::Color(40, 44,52, 255), 5}}, // background colour
      cbox{new CmdBox{window,
                      sf::Vector2f{0.f, (float)window->getSize().y - 25},
                      sf::Vector2f{(float)window->getSize().x, 25},
                      "../resource/fonts/arial.ttf", 20, sf::Color(171, 178, 191, 255),
                      sf::Color(40,44,52,255), 5}},
      kb{window, doc, textBox->getSize()}
{}
```

8.4.2.2 ∼Editor() `Editor::∼Editor ( )`

Destructor for Editor class.



```
Editor::~Editor() {
    delete textBox;
    delete cbox;
}
```

8.4.3 Member Function Documentation

8.4.3.1 draw() `void Editor::draw ( )`

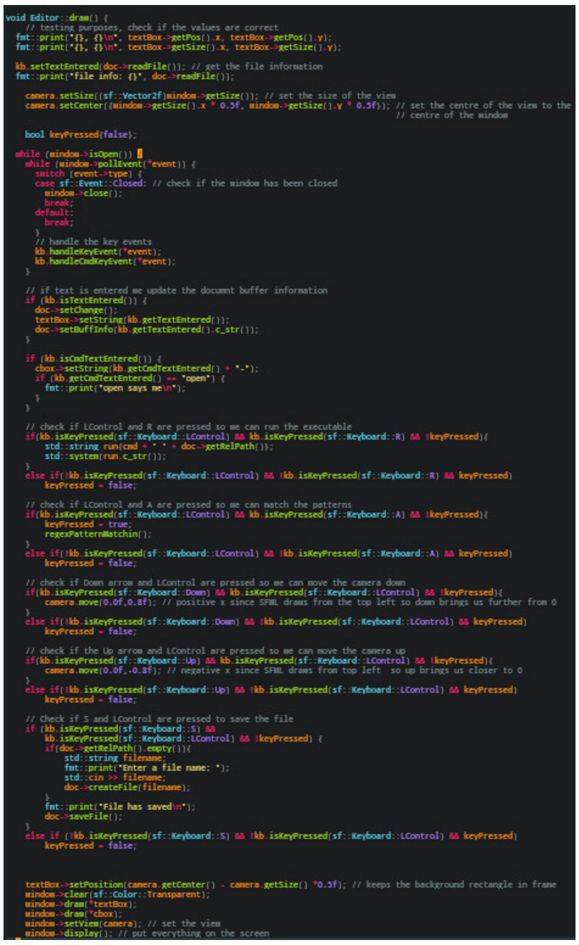
function that draws everything to RenderWindow

Parameters

void	

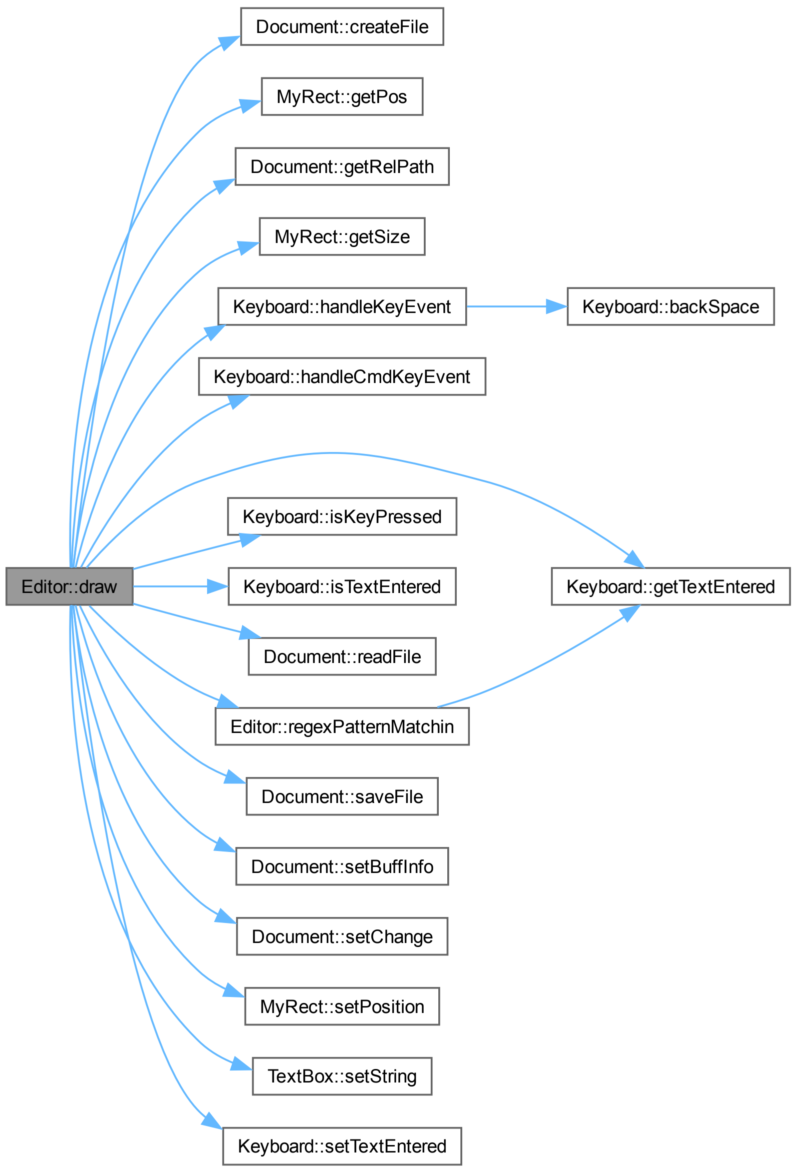
Returns

void



```
void Editor::draw() {
    // testing purposes, check if the values are correct
  fmt::print("{}, {}\n", textBox->getPos().x, textBox->getPos().y);
  fmt::print("{}, {}\n", textBox->getSize().x, textBox->getSize().y);

  kb.setTextEntered(doc->readFile()); // get the file information
  fmt::print("file info: {}", doc->readFile());

    camera.setSize((sf::Vector2f)window->getSize()); // set the size of the view
    camera.setCenter({window->getSize().x * 0.5f, window->getSize().y * 0.5f}); // set the centre of the view to the
                                                                                 // centre of the window

    bool keyPressed{false};

  while (window->isOpen()) {
    while (window->pollEvent(*event)) {
      switch (event->type) {
      case sf::Event::Closed: // check if the window has been closed
        window->close();
        break;
      default:
        break;
      }
      // handle the key events
      kb.handleKeyEvent(*event);
      kb.handleCmdKeyEvent(*event);
    }

    // if text is entered we update the documnt buffer information
    if (kb.isTextEntered()) {
      doc->setChange();
      textBox->setString(kb.getTextEntered());
      doc->setBuffInfo(kb.getTextEntered().c_str());
    }

    if (kb.isCmdTextEntered()) {
      cbox->setString(kb.getCmdTextEntered() + "-");
      if (kb.getCmdTextEntered() == "open") {
        fmt::print("open says me\n");
      }
    }

    // check if LControl and R are pressed so we can run the executable
    if(kb.isKeyPressed(sf::Keyboard::LControl) && kb.isKeyPressed(sf::Keyboard::R) && !keyPressed){
        std::string run{cmd + " " + doc->getRelPath()};
        std::system(run.c_str());
    }
    else if(!kb.isKeyPressed(sf::Keyboard::LControl) && !kb.isKeyPressed(sf::Keyboard::R) && keyPressed)
        keyPressed = false;

    // check if LControl and A are pressed so we can match the patterns
    if(kb.isKeyPressed(sf::Keyboard::LControl) && kb.isKeyPressed(sf::Keyboard::A) && !keyPressed){
        keyPressed = true;
        regexPatternMatchin();
    }
    else if(!kb.isKeyPressed(sf::Keyboard::LControl) && !kb.isKeyPressed(sf::Keyboard::A) && keyPressed)
        keyPressed = false;

    // check if Down arrow and LControl are pressed so we can move the camera down
    if(kb.isKeyPressed(sf::Keyboard::Down) && kb.isKeyPressed(sf::Keyboard::LControl) && !keyPressed){
        camera.move(0.0f,0.8f); // positive x since SFML draws from the top left so down brings us further from 0
    }
    else if(!kb.isKeyPressed(sf::Keyboard::Down) && !kb.isKeyPressed(sf::Keyboard::LControl) && keyPressed)
        keyPressed = false;

    // check if the Up arrow and LControl are pressed so we can move the camera up
    if(kb.isKeyPressed(sf::Keyboard::Up) && kb.isKeyPressed(sf::Keyboard::LControl) && !keyPressed){
        camera.move(0.0f,-0.8f); // negative x since SFML draws from top left  so up brings us closer to 0
    }
    else if(!kb.isKeyPressed(sf::Keyboard::Up) && !kb.isKeyPressed(sf::Keyboard::LControl) && keyPressed)
        keyPressed = false;

    // Check if S and LControl are pressed to save the file
    if (kb.isKeyPressed(sf::Keyboard::S) &&
        kb.isKeyPressed(sf::Keyboard::LControl) && !keyPressed) {
        if(doc->getRelPath().empty()){
            std::string filename;
            fmt::print("Enter a file name: ");
            std::cin >> filename;
            doc->createFile(filename);
        }
        fmt::print("File has saved\n");
        doc->saveFile();
    }
    else if (!kb.isKeyPressed(sf::Keyboard::S) && !kb.isKeyPressed(sf::Keyboard::LControl) && keyPressed)
        keyPressed = false;



    textBox->setPosition(camera.getCenter() - camera.getSize() *0.5f); // keeps the background rectangle in frame
    window->clear(sf::Color::Transparent);
    window->draw(*textBox);
    window->draw(*cbox);
    window->setView(camera); // set the view
    window->display(); // put everything on the screen
```

Here is the call graph for this function:



8.4.3.2 regexPatternMatchin() `void Editor::regexPatternMatchin ( )`

match patterns in the string

A method that is used to match and extract any math expressions in the text on the screen.

Kamil enjoys maths alot and is even doing further maths at A-level. However, he finds the small maths questions like 782∗7 tedius to type into the calculator. So to make sure he can focus only on the problem at hand and not any side questions I developed this regex method.

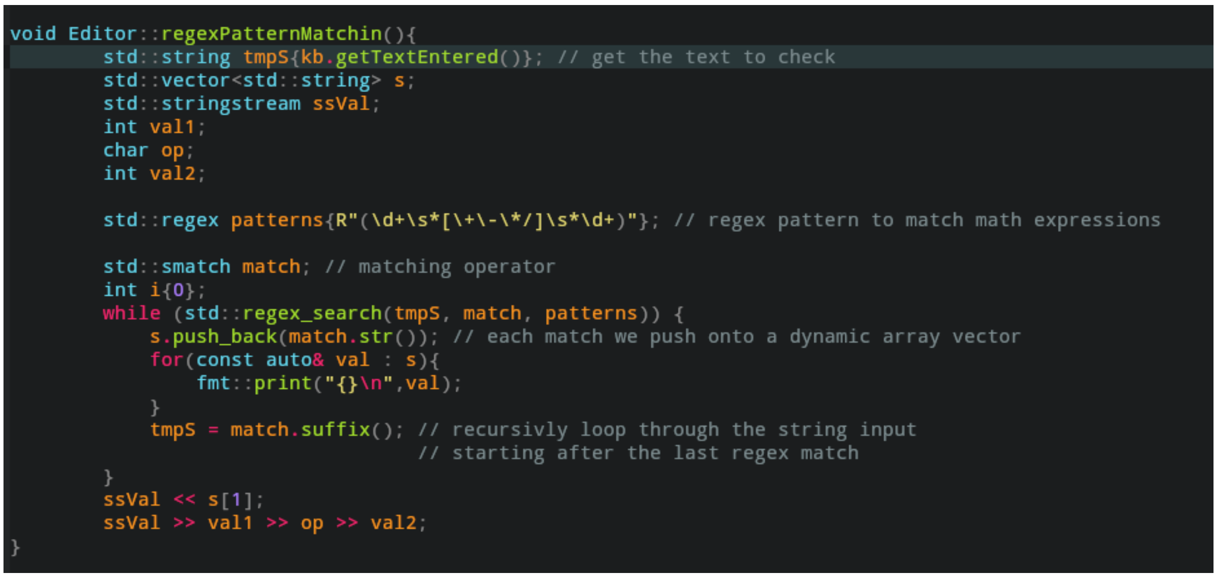
It extracts the text on the screen and stores it as a string. It then calls the regex_search() function passing in the string to search, an smatch type that contains the matchedsubstr and the regex to use. if the match is found then we extract it in the smatch type and preform the calculations on the substr. we then recursivly search the string by calling the smatch::suffix() method checking the rest of the string after the last match by doing this we ensure that any and all potential matches have been made

Parameters

void	

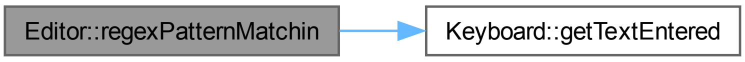
Returns

void

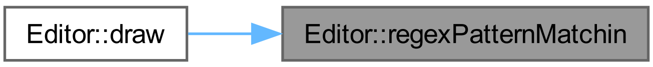


```
void Editor::regexPatternMatchin(){
        std::string tmpS{kb.getTextEntered()}; // get the text to check
        std::vector<std::string> s;
        std::stringstream ssVal;
        int val1;
        char op;
        int val2;

        std::regex patterns{R"(\d+\s*[\+\-\*/]\s*\d+)"}; // regex pattern to match math expressions

        std::smatch match; // matching operator
        int i{0};
        while (std::regex_search(tmpS, match, patterns)) {
            s.push_back(match.str()); // each match we push onto a dynamic array vector
            for(const auto& val : s){
                fmt::print("{}\n",val);
            }
            tmpS = match.suffix(); // recursivly loop through the string input
                                   // starting after the last regex match
        }
        ssVal << s[1];
        ssVal >> val1 >> op >> val2;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



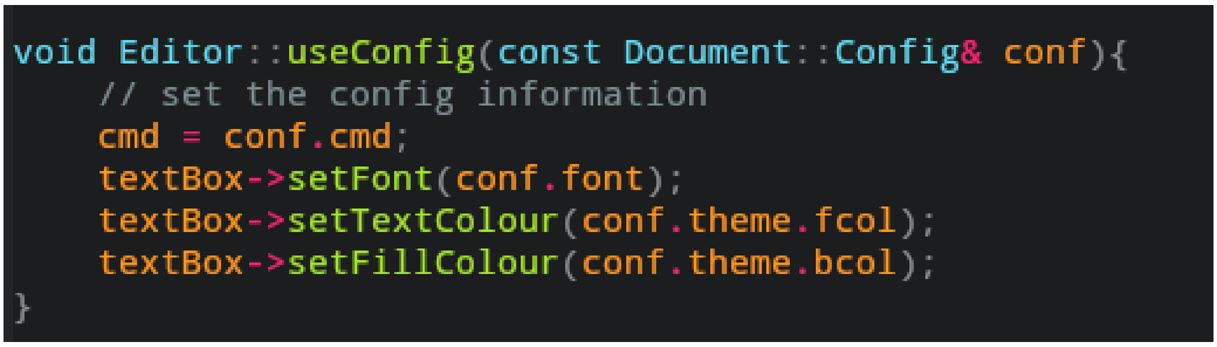
8.4.3.3 useConfig() `void Editor::useConfig ( const Document::Config & conf )`

changes the editor looks and workings

uses the information of the config.toml to change how the editor looks and how it can work

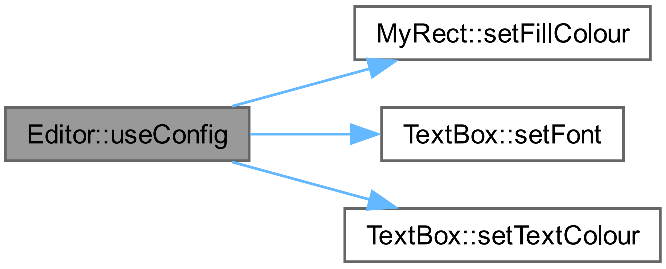
Parameters

const	Document::Config& - the config.toml information @reutrn void



```
void Editor::useConfig(const Document::Config& conf){
    // set the config information
    cmd = conf.cmd;
    textBox->setFont(conf.font);
    textBox->setTextColour(conf.theme.fcol);
    textBox->setFillColour(conf.theme.bcol);
}
```

Here is the call graph for this function:



8.4.4 Member Data Documentation

8.4.4.1 camera `sf::View Editor::camera [private]`

for the camera

8.4.4.2 cmd `std::string Editor::cmd [private]`

8.4.4.3 doc `Document* Editor::doc [private]`

pointer to the working document

8.4.4.4 event `sf::Event* Editor::event [private]`

pointer to event

8.4.4.5 kb `Keyboard Editor::kb [private]`

handles keyboard events

8.4.4.6 loadFromFile `bool Editor::loadFromFile [private]`

check if we are loading from file

8.4.4.7 textBox `TextBox* Editor::textBox [private]`

pointer to textbox that we draw

8.4.4.8 window `sf::RenderWindow* Editor::window [private]`

pointer to RenderWindow

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

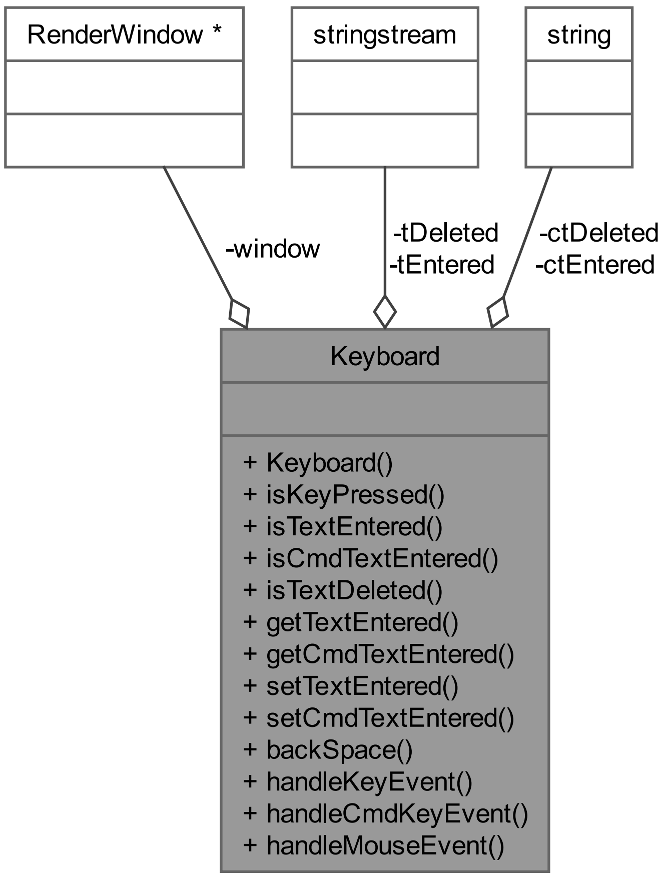
- include/Kamil/Editor.h
- src/Editor.cpp

8.5 Keyboard Class Reference

A class to handle Keyboard input.

```
#include <Keyboard.h>
```

Collaboration diagram for Keyboard:



Public Member Functions

- Keyboard (sf::RenderWindow ∗win, Document ∗doc, sf::Vector2f bounds)

  Constructor for Keyboard class.
- bool isKeyPressed (sf::Keyboard::Key)

  checks if a key is pressed
- bool isTextEntered ()

  checks if a text is entered to the text box
- bool isCmdTextEntered ()

  checks if text is entered to the command box
- bool isTextDeleted ()

  check if text is being deleted
- std::string getTextEntered ()

  returns text entered
- std::string getCmdTextEntered ()

  returns text entered from the command box
- void setTextEntered (std::string)

sets text
- void setCmdTextEntered (std::string)

  sets text A setter method that sets the new text
- void backSpace ()

  when we backspace on teh text
- void handleKeyEvent (sf::Event &event)

  handle keyboard events
- void handleCmdKeyEvent (sf::Event &event)

  handle keyboard events
- void handleMouseEvent (sf::Event &event)

  mouse keyboard events

Private Attributes

- sf::RenderWindow ∗ window
- std::stringstream tEntered
- std::stringstream tDeleted
- std::string ctEntered
- std::string ctDeleted

8.5.1 Detailed Description

A class to handle Keyboard input.

8.5.2 Constructor & Destructor Documentation

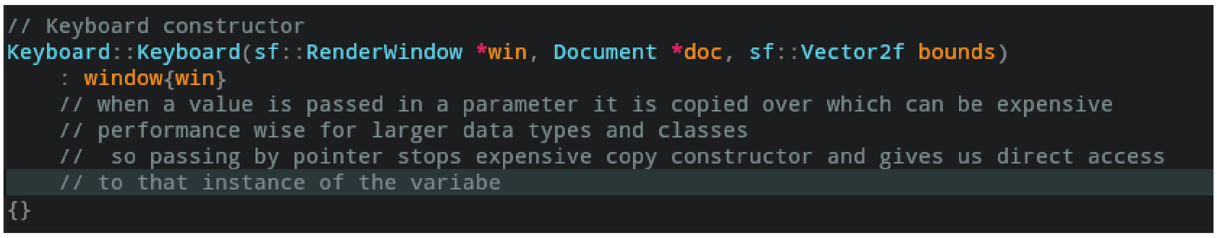
8.5.2.1 Keyboard()

```
Keyboard::Keyboard (
        sf::RenderWindow ∗ win,
        Document ∗ doc,
        sf::Vector2f bounds )
```

Constructor for Keyboard class.

Parameters

win	- reference to main window
bounds	- bounds of the window we are working in



```
// Keyboard constructor
Keyboard::Keyboard(sf::RenderWindow *win, Document *doc, sf::Vector2f bounds)
    : window{win}
    // when a value is passed in a parameter it is copied over which can be expensive
    // performance wise for larger data types and classes
    //  so passing by pointer stops expensive copy constructor and gives us direct access
    // to that instance of the variabe
{}
```

8.5.3 Member Function Documentation

8.5.3.1 backSpace() `void Keyboard::backSpace ( )`

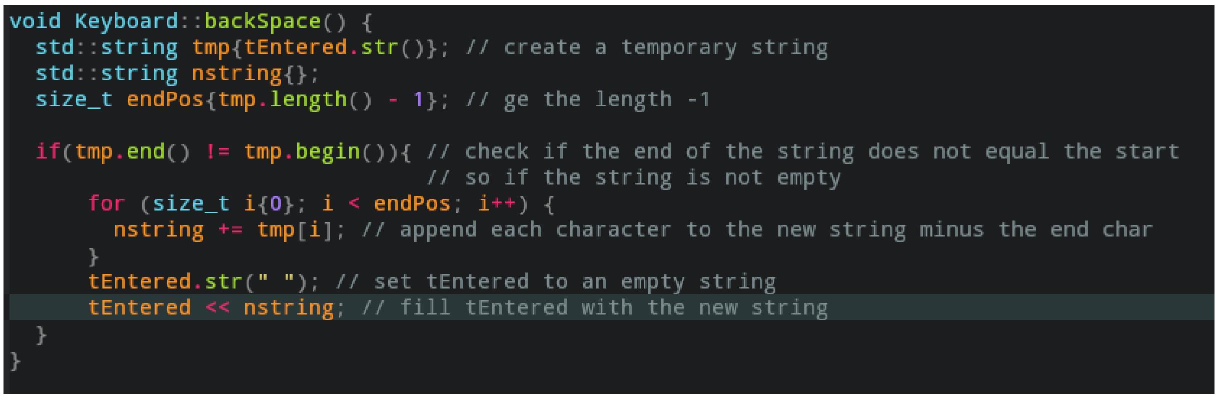
when we backspace on teh text

Parameters

void	

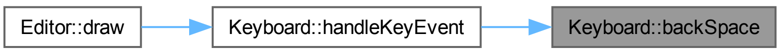
Returns

void



```
void Keyboard::backSpace() {
  std::string tmp{tEntered.str()}; // create a temporary string
  std::string nstring{};
  size_t endPos{tmp.length() - 1}; // ge the length -1

  if(tmp.end() != tmp.begin()){ // check if the end of the string does not equal the start
                                // so if the string is not empty
      for (size_t i{0}; i < endPos; i++) {
        nstring += tmp[i]; // append each character to the new string minus the end char
      }
      tEntered.str(" "); // set tEntered to an empty string
      tEntered << nstring; // fill tEntered with the new string
  }
}
```

Here is the caller graph for this function:



8.5.3.2 getCmdTextEntered() `std::string Keyboard::getCmdTextEntered ( )`

returns text entered from the command box

A getter method that returns the text entered to the command box

Parameters

void	

Returns

std::string text entered

8.5.3.3 getTextEntered() `std::string Keyboard::getTextEntered ( )`

returns text entered

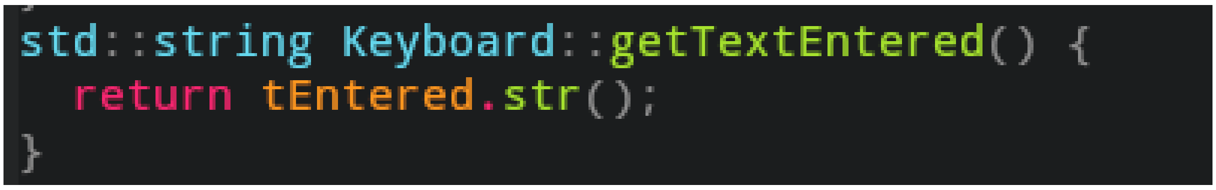
A getter method that returns the text entered

Parameters

void	

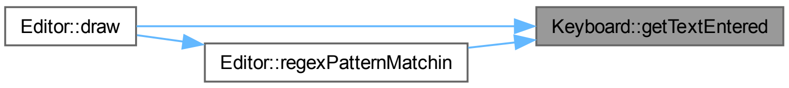
Returns

std::string text entered



```
std::string Keyboard::getTextEntered() {
  return tEntered.str();
}
```

Here is the caller graph for this function:



8.5.3.4 handleCmdKeyEvent() `void Keyboard::handleCmdKeyEvent ( sf::Event & event )`

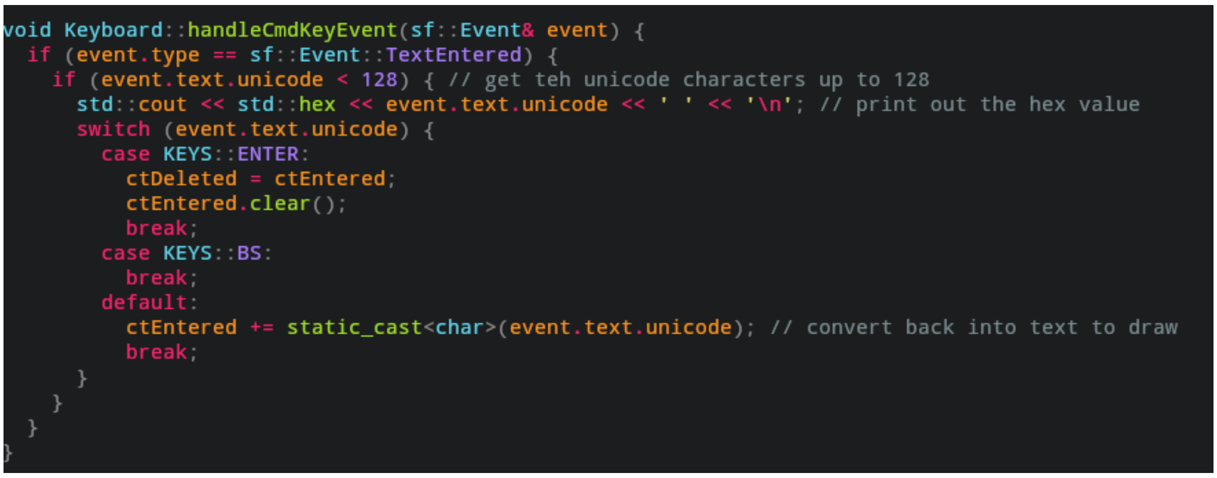
handle keyboard events

Parameters

event	- to get text entered from events

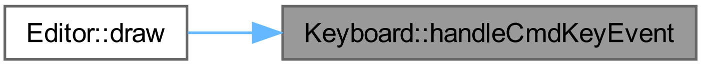
Returns

void



```
void Keyboard::handleCmdKeyEvent(sf::Event& event) {
  if (event.type == sf::Event::TextEntered) {
    if (event.text.unicode < 128) { // get teh unicode characters up to 128
      std::cout << std::hex << event.text.unicode << ' ' << '\n'; // print out the hex value
      switch (event.text.unicode) {
        case KEYS::ENTER:
          ctDeleted = ctEntered;
          ctEntered.clear();
          break;
        case KEYS::BS:
          break;
        default:
          ctEntered += static_cast<char>(event.text.unicode); // convert back into text to draw
          break;
      }
    }
  }
}
```

Here is the caller graph for this function:



8.5.3.5 handleKeyEvent()

```
void Keyboard::handleKeyEvent (
        sf::Event & event )
```

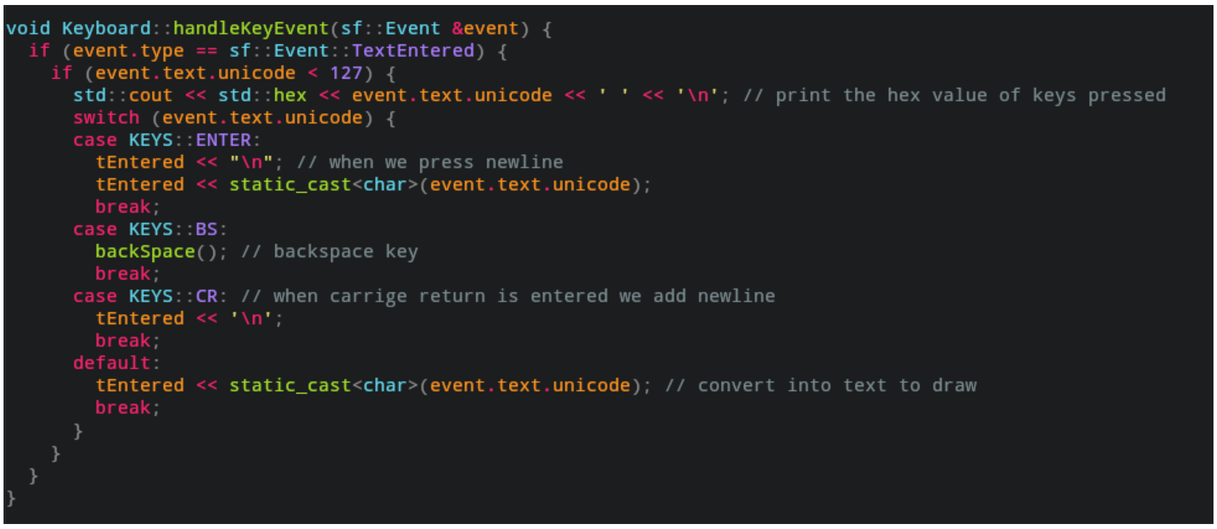
handle keyboard events

Parameters

event	- to get text entered from events

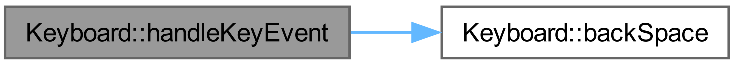
Returns

void

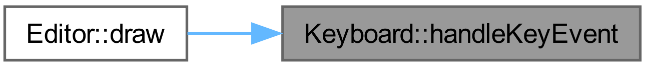


```
void Keyboard::handleKeyEvent(sf::Event &event) {
  if (event.type == sf::Event::TextEntered) {
    if (event.text.unicode < 127) {
      std::cout << std::hex << event.text.unicode << ' ' << '\n'; // print the hex value of keys pressed
      switch (event.text.unicode) {
      case KEYS::ENTER:
        tEntered << "\n"; // when we press newline
        tEntered << static_cast<char>(event.text.unicode);
        break;
      case KEYS::BS:
        backSpace(); // backspace key
        break;
      case KEYS::CR: // when carrige return is entered we add newline
        tEntered << '\n';
        break;
      default:
        tEntered << static_cast<char>(event.text.unicode); // convert into text to draw
        break;
      }
    }
  }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.5.3.6 handleMouseEvent() `void Keyboard::handleMouseEvent ( sf::Event & event )`

mouse keyboard events

Parameters

event	- to get text entered from events

Returns

void

8.5.3.7 isCmdTextEntered() `bool Keyboard::isCmdTextEntered ( )`

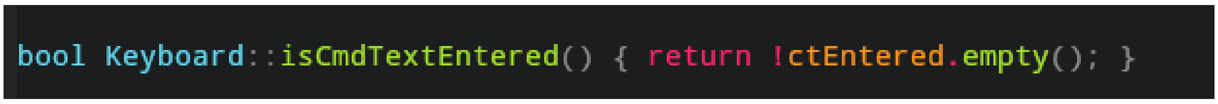
checks if text is entered to the command box

Parameters

void	

Returns

bool tru eif key is pressed false if not



```
bool Keyboard::isCmdTextEntered() { return !ctEntered.empty(); }
```

8.5.3.8 isKeyPressed() `bool Keyboard::isKeyPressed ( sf::Keyboard::Key key )`

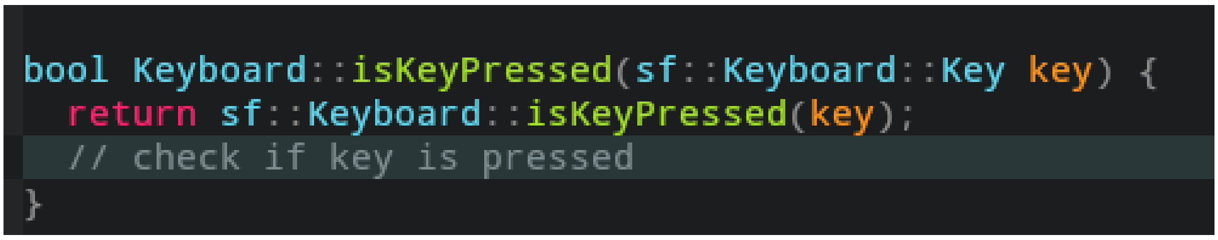
checks if a key is pressed

Parameters

sf::Keyboard::key	enum from SFML

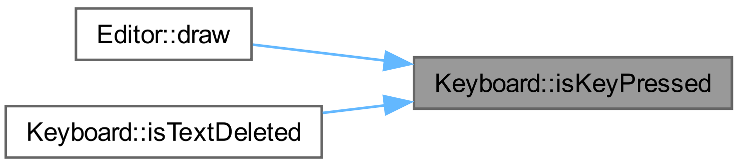
Returns

bool true if key is pressed false if not



```
bool Keyboard::isKeyPressed(sf::Keyboard::Key key) {
  return sf::Keyboard::isKeyPressed(key);
  // check if key is pressed
}
```

Here is the caller graph for this function:



8.5.3.9 isTextDeleted() `bool Keyboard::isTextDeleted ( )`

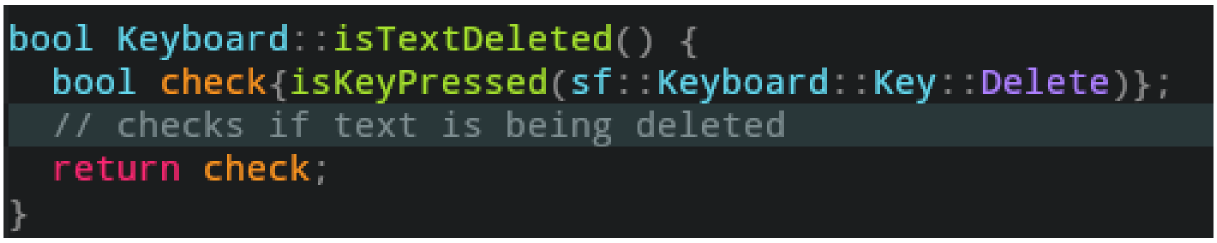
check if text is being deleted

Parameters

void	

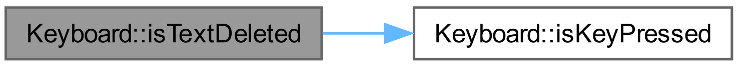
Returns

bool true if text is being deleted



```
bool Keyboard::isTextDeleted() {
  bool check{isKeyPressed(sf::Keyboard::Key::Delete)};
  // checks if text is being deleted
  return check;
}
```

Here is the call graph for this function:



8.5.3.10 isTextEntered() `bool Keyboard::isTextEntered ( )`

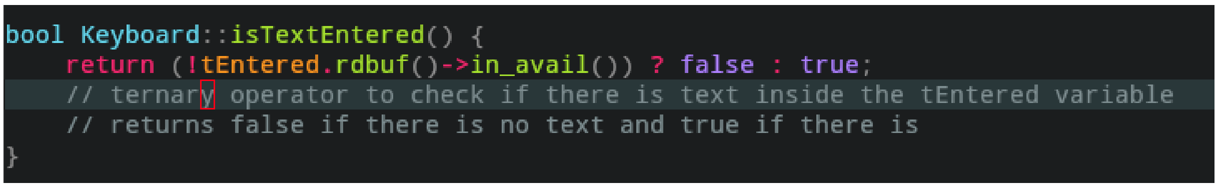
checks if a text is entered to the text box

Parameters

void	

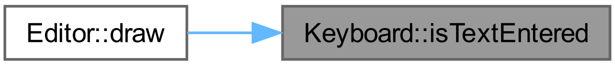
Returns

bool true if key is pressed false if not



```
bool Keyboard::isTextEntered() {
    return (!tEntered.rdbuf()->in_avail()) ? false : true;
    // ternary operator to check if there is text inside the tEntered variable
    // returns false if there is no text and true if there is
}
```

Here is the caller graph for this function:



8.5.3.11 setCmdTextEntered() `void Keyboard::setCmdTextEntered ( std::string nstring )`

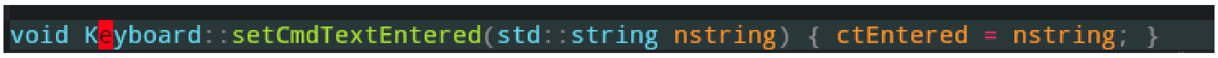
sets text A setter method that sets the new text

Parameters

std::string	- new string

Returns

void



```
void Keyboard::setCmdTextEntered(std::string nstring) { ctEntered = nstring; }
```

8.5.3.12 setTextEntered() `void Keyboard::setTextEntered ( std::string *nstring* )`

sets text

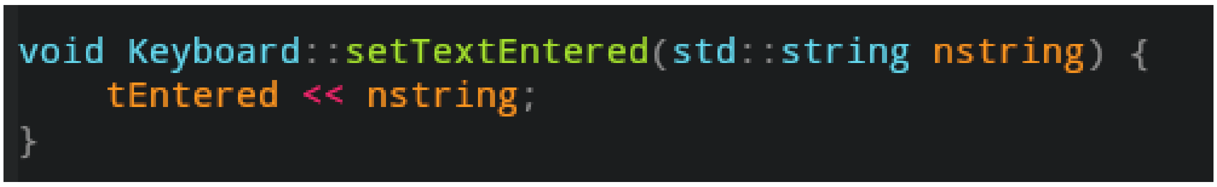
A setter method that sets the new text

Parameters

std::string	- new string

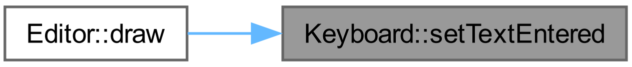
Returns

void



```
void Keyboard::setTextEntered(std::string nstring) {
    tEntered << nstring;
}
```

Here is the caller graph for this function:



8.5.4 Member Data Documentation

8.5.4.1 ctDeleted `std::string Keyboard::ctDeleted [private]`

temporary for text deleted to cmd not working

8.5.4.2 ctEntered `std::string Keyboard::ctEntered [private]`

temporary for text enterd to cmd not working

8.5.4.3 tDeleted `std::stringstream Keyboard::tDeleted [private]`

the text deleted from main box

8.5.4.4 tEntered `std::stringstream Keyboard::tEntered [private]`

the text entered to main box

8.5.4.5 window `sf::RenderWindow* Keyboard::window [private]`

refernce to window

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

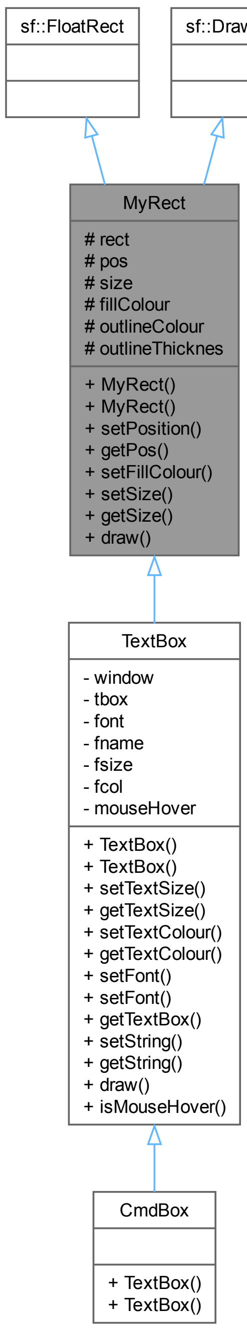
- include/Kamil/Keyboard.h
- src/Keyboard.cpp

8.6 MyRect Class Reference

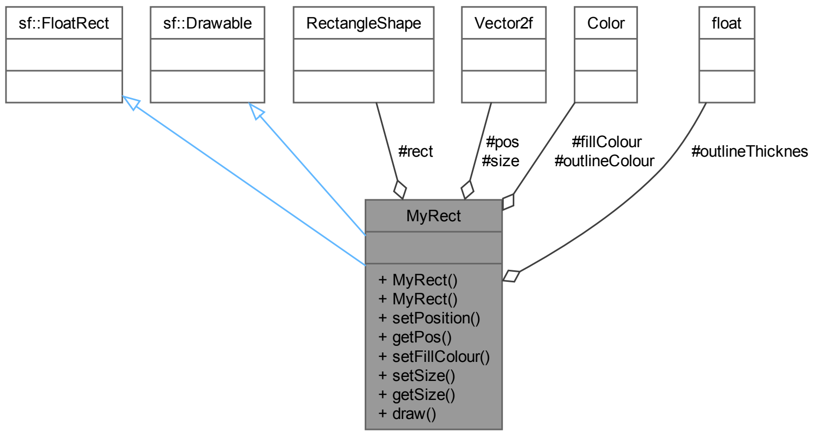
gives extra functionality to FloatRect

```
#include <MyRect.h>
```

Inheritance diagram for MyRect:



Collaboration diagram for MyRect:



Public Member Functions

- MyRect (sf::Vector2f pos, sf::Vector2f size, sf::Color fillColour, sf::Color outlineColour, float outlineThicknes)

  constructor for MyRect
- MyRect ()
- void setPosition (sf::Vector2f pos)

  sets the position of rect
- sf::Vector2f getPos () const

  get the position of rect
- void setFillColour (sf::Color colour)

  set the fill colour of the rect
- void setSize (sf::Vector2f size)

  set the size of the rect
- sf::Vector2f getSize () const

  get the size of the rect
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  virutal method to draw to window

Protected Attributes

- sf::RectangleShape rect
- sf::Vector2f pos
- sf::Vector2f size
- sf::Color fillColour
- sf::Color outlineColour
- float outlineThicknes

8.6.1 Detailed Description

gives extra functionality to FloatRect

Uses FloatRect for the ability to collision detect better than RectangleShape and inherits from Drawable so we are able to keep uniform sytax of window.draw(Drawable object)

8.6.2 Constructor & Destructor Documentation

8.6.2.1 MyRect() [1/2]

```
MyRect::MyRect (
        sf::Vector2f pos,
        sf::Vector2f size,
        sf::Color fillColour,
        sf::Color outlineColour,
        float outlineThicknes )
```

constructor for MyRect

Parameters

pos	- position of rect
size	- size of rect
fillColour	- fill colour of rect
outlineColour	- ouline colour of rect
outlineThicknes	- outline thickness of rect

8.6.2.2 MyRect() [2/2]

```
MyRect::MyRect ( )
```

8.6.3 Member Function Documentation

8.6.3.1 draw()

```
void MyRect::draw (
        sf::RenderTarget & target,
        sf::RenderStates states ) const [override]
```

virutal method to draw to window

Inherited from sf::Drawable it is what allows us to draw to the screen using window.draw(MyRect); instead of My↩ Rect.draw(window) keeping similar drawing standard to base SFML code making our class more modular and familiar to those who use SFML

Example of polymorphism by overriding a virtual method

8.6.3.2 getPos()

```
sf::Vector2f MyRect::getPos ( ) const
```

get the position of rect

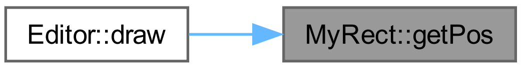
Parameters

void	

Returns

sf::Vector2f pos

Here is the caller graph for this function:



8.6.3.3 getSize() `sf::Vector2f MyRect::getSize ( ) const`

get the size of the rect

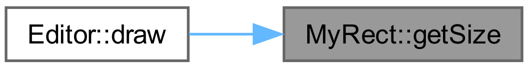
Parameters

void	

Returns

sf::Vector2f size

Here is the caller graph for this function:



8.6.3.4 setFillColour() `void MyRect::setFillColour ( sf::Color colour )`

set the fill colour of the rect

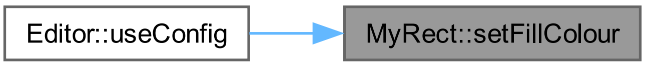
Parameters

sf::Color	colour

Returns

void

Here is the caller graph for this function:



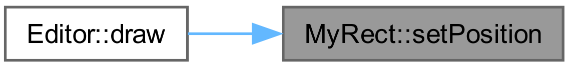
8.6.3.5 setPosition() `void MyRect::setPosition ( sf::Vector2f pos )`

sets the position of rect

Parameters

sf::Vector2f	pos

Here is the caller graph for this function:



8.6.3.6 setSize() `void MyRect::setSize ( sf::Vector2f size )`

set the size of the rect

Parameters

sf::Vector2f	size

Returns

void

8.6.4 Member Data Documentation

8.6.4.1 fillColour `sf::Color MyRect::fillColour [protected]`

colour of rect

8.6.4.2 outlineColour `sf::Color MyRect::outlineColour [protected]`

outline colour of rect

8.6.4.3 outlineThicknes `float MyRect::outlineThicknes [protected]`

outline thickness of rect

8.6.4.4 pos `sf::Vector2f MyRect::pos [protected]`

position of rect

8.6.4.5 rect `sf::RectangleShape MyRect::rect [protected]`

8.6.4.6 size `sf::Vector2f MyRect::size [protected]`

size of rect

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

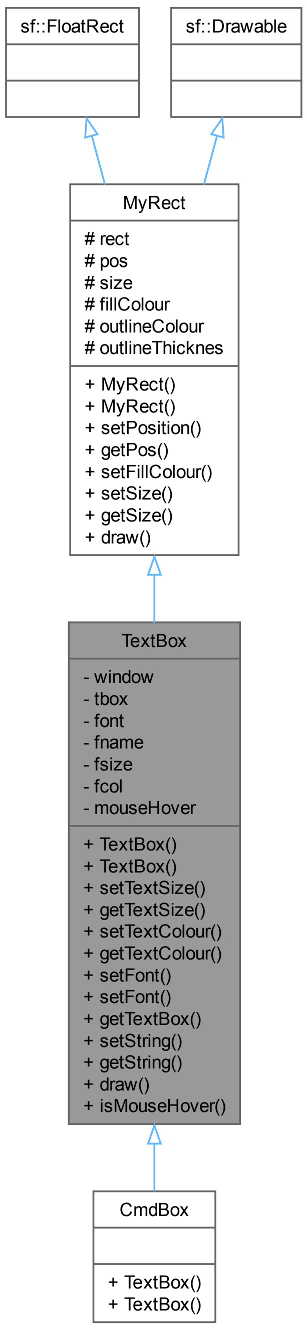
- include/Kamil/MyRect.h
- src/MyRect.cpp

8.7 TextBox Class Reference

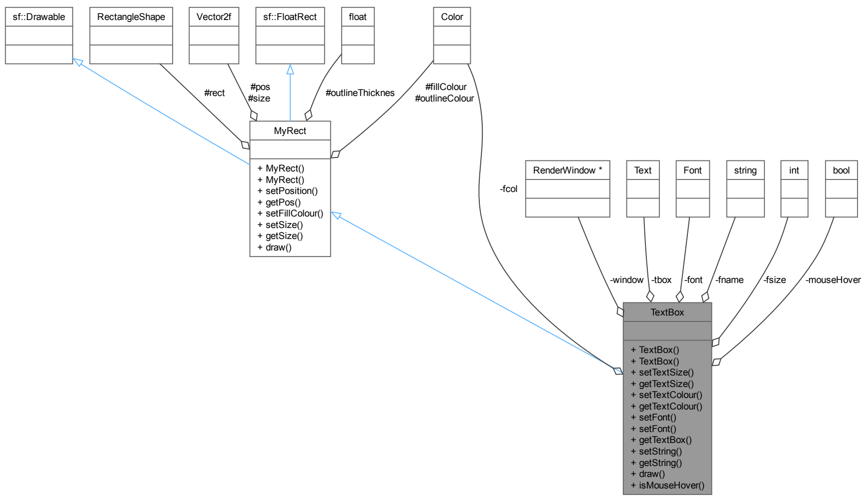
A class that makes a Textbox in SFML.

```
#include <TextBox.h>
```

Inheritance diagram for TextBox:



Collaboration diagram for TextBox:



Public Member Functions

- TextBox (sf::RenderWindow ∗win, sf::Vector2f pos, sf::Vector2f size, std::string sfont, int fsize, sf::Color fcol, sf::Color background, float thicc)

  Constructor for TextBox.
- TextBox ()
- void setTextSize (int size)

  Set the size of the text.
- int getTextSize () const

  Get the size of the text.
- void setTextColour (sf::Color colour)

  Set the colour of the text.
- sf::Color getTextColour () const

  Get the colour of the text.
- void setFont (sf::Font &font)

  set what font you want to use
- void setFont (std::string font)

  set what font you use
- sf::Text getTextBox () const

  Get sf::Text of the textbox.
- void setString (std::string nstring)

  Sets the string.
- std::string getString () const

  returns the text in tbox
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  used to draw to the screen virutal method inherited from MyRect which inherited it from sf::Drawable thats overrided here it is an example of polymorphism as we are changing the behaviour of a method in the child class. By inheriting from sf::Drawable it allows us to keep a similar syntax to other SFML shapes and drawable objects window.draw(my←↩ _object). This allows our code to be more modular and easy for other people to use since they dont need to fumble around with my_object.draw(window)
- bool isMouseHover ()

  check if mouse is hovering over current textbox

Public Member Functions inherited from MyRect

- MyRect (sf::Vector2f pos, sf::Vector2f size, sf::Color fillColour, sf::Color outlineColour, float outlineThicknes)

    constructor for MyRect
- MyRect ()
- void setPosition (sf::Vector2f pos)

    sets the position of rect
- sf::Vector2f getPos () const

    get the position of rect
- void setFillColour (sf::Color colour)

    set the fill colour of the rect
- void setSize (sf::Vector2f size)

    set the size of the rect
- sf::Vector2f getSize () const

    get the size of the rect
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

    virutal method to draw to window

Private Attributes

- sf::RenderWindow ∗ window
- sf::Text tbox {}
- sf::Font font {}
- std::string fname {}
- int fsize {}
- sf::Color fcol {}
- bool mouseHover

Additional Inherited Members

Protected Attributes inherited from MyRect

- sf::RectangleShape rect
- sf::Vector2f pos
- sf::Vector2f size
- sf::Color fillColour
- sf::Color outlineColour
- float outlineThicknes

8.7.1 Detailed Description

A class that makes a Textbox in SFML.

The class creates a textbox for inputting and handling text and Keyboard commands and allows the use of commands in the secondary textbox cmdbox

8.7.2 Constructor & Destructor Documentation

8.7.2.1 TextBox() [1/2]

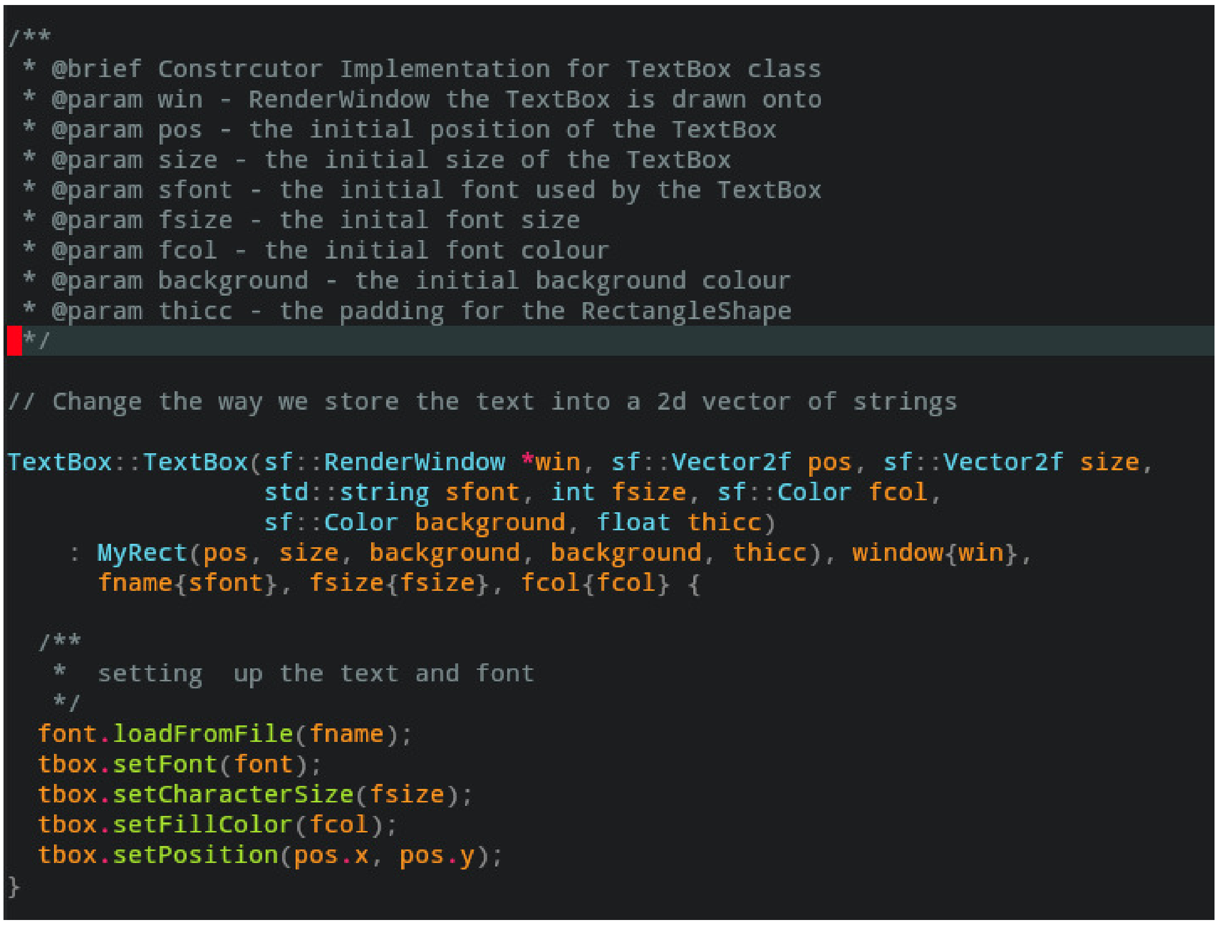
```
TextBox::TextBox (
        sf::RenderWindow * win,
        sf::Vector2f pos,
        sf::Vector2f size,
        std::string sfont,
        int fsize,
        sf::Color fcol,
        sf::Color background,
        float thicc )
```

Constructor for TextBox.

Constrcutor Implementation for TextBox class.

Parameters

win	- RenderWindow the TextBox is drawn onto
pos	- the initial position of the TextBox
size	- the initial size of the TextBox
sfont	- the initial font used by the TextBox
fsize	- the inital font size
fcol	- the initial font colour
background	- the initial background colour
thicc	- the padding for the RectangleShape



```
/**
 * @brief Constrcutor Implementation for TextBox class
 * @param win - RenderWindow the TextBox is drawn onto
 * @param pos - the initial position of the TextBox
 * @param size - the initial size of the TextBox
 * @param sfont - the initial font used by the TextBox
 * @param fsize - the inital font size
 * @param fcol - the initial font colour
 * @param background - the initial background colour
 * @param thicc - the padding for the RectangleShape
 */

// Change the way we store the text into a 2d vector of strings

TextBox::TextBox(sf::RenderWindow *win, sf::Vector2f pos, sf::Vector2f size,
                 std::string sfont, int fsize, sf::Color fcol,
                 sf::Color background, float thicc)
    : MyRect(pos, size, background, background, thicc), window{win},
      fname{sfont}, fsize{fsize}, fcol{fcol} {

  /**
   *  setting  up the text and font
   */
  font.loadFromFile(fname);
  tbox.setFont(font);
  tbox.setCharacterSize(fsize);
  tbox.setFillColor(fcol);
  tbox.setPosition(pos.x, pos.y);
}
```

Implementation of the TextBox class

Note

other structs or classes may be used here

Parameters

win	- RenderWindow the TextBox is drawn onto
pos	- the initial position of the TextBox
size	- the initial size of the TextBox
sfont	- the initial font used by the TextBox
fsize	- the inital font size
fcol	- the initial font colour
background	- the initial background colour
thicc	- the padding for the RectangleShape

setting up the text and font

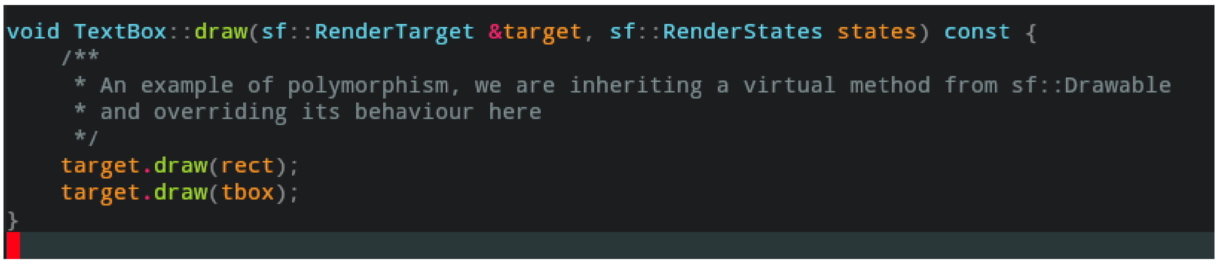
8.7.2.2 TextBox() [2/2] `TextBox::TextBox ( )`

8.7.3 Member Function Documentation

8.7.3.1 draw()

```
void TextBox::draw (
        sf::RenderTarget & target,
        sf::RenderStates states ) const  [override]
```

used to draw to the screen virutal method inherited from MyRect which inherited it from sf::Drawable thats overrided here it is an example of polymorphism as we are changing the behaviour of a method in the child class. By inheriting from sf::Drawable it allows us to keep a similar syntax to other SFML shapes and drawable objects window.↩ draw(my_object). This allows our code to be more modular and easy for other people to use since they dont need to fumble around with my_object.draw(window)



```
void TextBox::draw(sf::RenderTarget &target, sf::RenderStates states) const {
    /**
     * An example of polymorphism, we are inheriting a virtual method from sf::Drawable
     * and overriding its behaviour here
     */
    target.draw(rect);
    target.draw(tbox);
}
```

An example of polymorphism, we are inheriting a virtual method from sf::Drawable and overriding its behaviour here

8.7.3.2 getString() `std::string TextBox::getString ( ) const`

returns the text in tbox

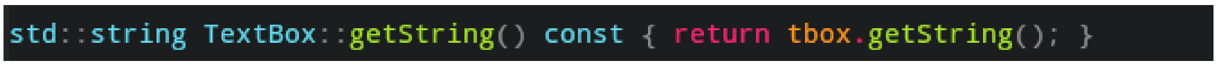
A getter method that returns the string thats displayed on the screen

Parameters

void	

Returns

type std::string



```
std::string TextBox::getString() const { return tbox.getString(); }
```

8.7.3.3 getTextBox() `sf::Text TextBox::getTextBox ( ) const`

Get sf::Text of the textbox.

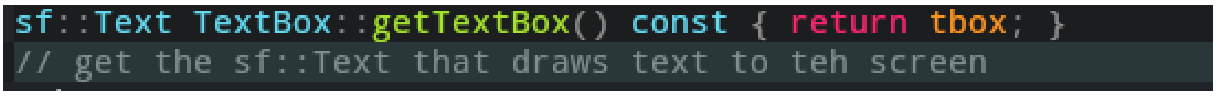
A getter method for getting the sf::Text part of the TextBox class this is the part responsible for displaying all the text

Parameters

void	

Returns

sf::Text - contains the part responsible for drawing text on the screen



```
sf::Text TextBox::getTextBox() const { return tbox; }
// get the sf::Text that draws text to teh screen
```

8.7.3.4 getTextColour() `sf::Color TextBox::getTextColour ( ) const`

Get the colour of the text.

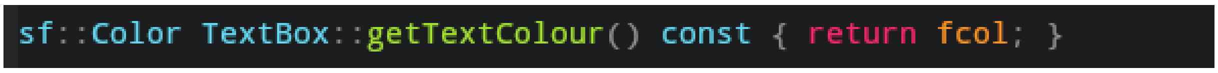
A getter method that returns the colour of the text

Parameters

void	

Returns

sf::Colour textColour



```
sf::Color TextBox::getTextColour() const { return fcol; }
```

8.7.3.5 getTextSize() `int TextBox::getTextSize ( ) const`

Get the size of the text.

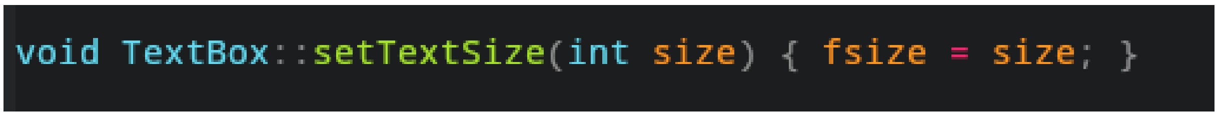
A getter method that returns the size of the text

Parameters

void	

Returns

an int of the text size



```
void TextBox::setTextSize(int size) { fsize = size; }
```

8.7.3.6 isMouseHover() `bool TextBox::isMouseHover ( )`

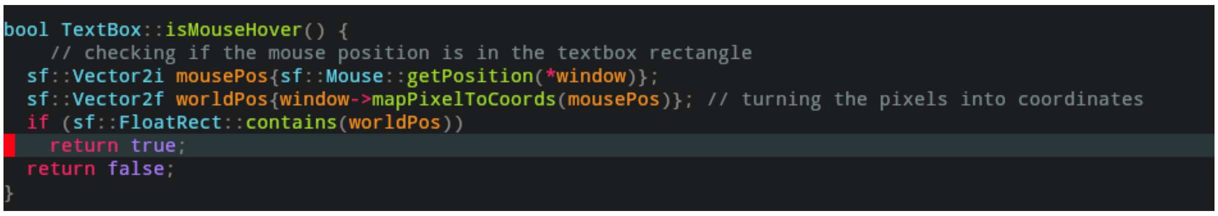
check if mouse is hovering over current textbox

Useful for when you want specific events to happen only when the mouse hovers over like text inputting.

We are able to check if the mouse is hovering through the extra collision functionality that FloatRect gives us. see MyRect

Returns

bool - yes if hovering over the text box



```
bool TextBox::isMouseHover() {
    // checking if the mouse position is in the textbox rectangle
  sf::Vector2i mousePos{sf::Mouse::getPosition(*window)};
  sf::Vector2f worldPos{window->mapPixelToCoords(mousePos)}; // turning the pixels into coordinates
  if (sf::FloatRect::contains(worldPos))
    return true;
  return false;
}
```

8.7.3.7 setFont() [1/2] `void TextBox::setFont ( sf::Font & font )`

set what font you want to use

A setter method overload of setFont function that sets the font using an object of type sf::Font

Parameters

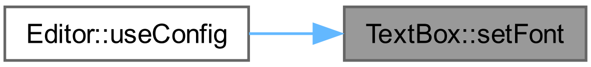
font	file dir of font

Returns

void

```
void TextBox::setFont(sf::Font &font) { font = font; }
```

Here is the caller graph for this function:



8.7.3.8 setFont() [2/2] `void TextBox::setFont ( std::string font )`

set what font you use

A setter method overload of setFont function that sets the font Allows the passing of strings instead of sf::Font types.

Parameters

font	file dir of font

Returns

void

```
void TextBox::setFont(std::string fnt) {
    // overload function for setFont
    try{
        font.loadFromFile(fnt); // see if we can load the file
    }
    catch(...){
        std::cerr << "Cannot load font from file\n"; // if we arent able to load it
    }
}
```

8.7.3.9 setString()

```
void TextBox::setString (
        std::string nstring )
```

Sets the string.

A setter method that sets the string that is displayed on the screen

Parameters

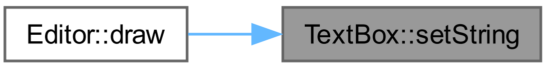
std::string	- new string placed on tbox

Returns

void

```
void TextBox::setString(std::string nstring) {
    sf::String val{nstring};
    tbox.setString(val.toWideString());  // allow wider characters like japanese characters
                                         // for future language expansion
}
```

Here is the caller graph for this function:



8.7.3.10 setTextColour()

```
void TextBox::setTextColour (
        sf::Color colour )
```

Set the colour of the text.

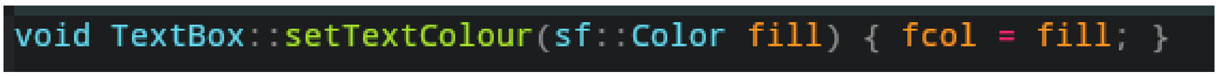
A setter mehod that sets the colour of the text

Parameters

fill	font colour

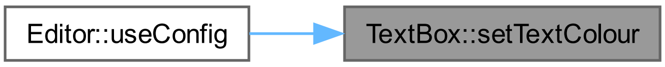
Returns

void



```
void TextBox::setTextColour(sf::Color fill) { fcol = fill; }
```

Here is the caller graph for this function:



8.7.3.11 setTextSize() `void TextBox::setTextSize ( int size )`

Set the size of the text.

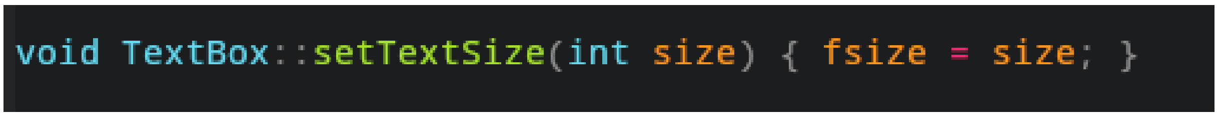
A setter method that sets the size of the text

Parameters

size	text size

Returns

void



```
void TextBox::setTextSize(int size) { fsize = size; }
```

8.7.4 Member Data Documentation

8.7.4.1 fcol `sf::Color TextBox::fcol {} [private]`

the font colour

8.7.4.2 fname `std::string TextBox::fname {} [private]`

the name of the font used

8.7.4.3 font `sf::Font TextBox::font {} [private]`

the font that the TextBox uses

8.7.4.4 fsize `int TextBox::fsize {} [private]`

the font size

8.7.4.5 mouseHover `bool TextBox::mouseHover [private]`

if the mouse is hovering over

8.7.4.6 tbox `sf::Text TextBox::tbox {} [private]`

the text that everything is written onto

8.7.4.7 window `sf::RenderWindow* TextBox::window [private]`

pointer to the main RenderWindow variable

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

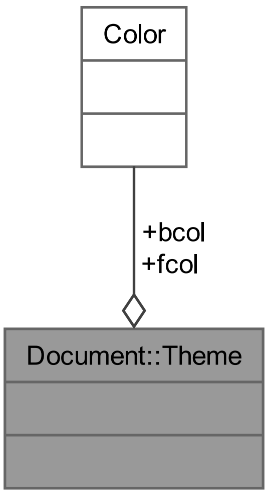
- include/Kamil/TextBox.h
- src/TextBox.cpp

8.8 Document::Theme Struct Reference

a struct for the Theme

`#include <Document.h>`

Collaboration diagram for Document::Theme:



Public Attributes

- sf::Color bcol
- sf::Color fcol

8.8.1 Detailed Description

a struct for the Theme

A struct containing all the information for the Theme of the Text editor

Parameters

sf::Color	- background colour
sf::Color	- font colour

8.8.2 Member Data Documentation

8.8.2.1 bcol `sf::Color Document::Theme::bcol`

8.8.2.2 fcol `sf::Color Document::Theme::fcol`

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

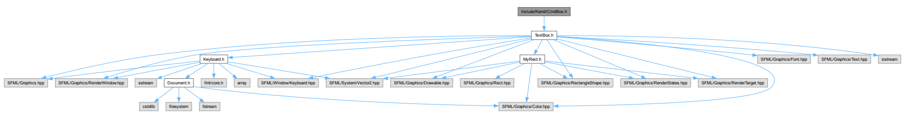
- include/Kamil/Document.h

9 File Documentation

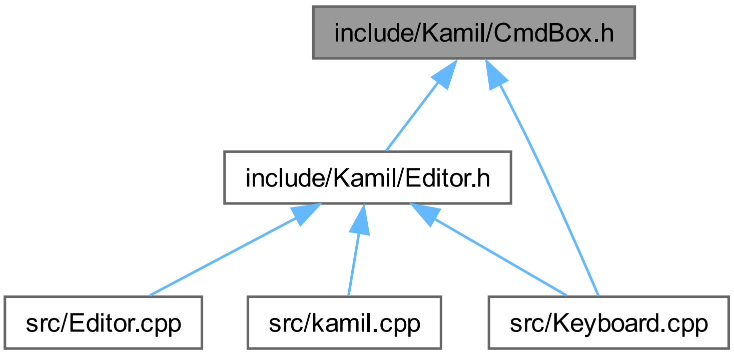
9.1 eval.md File Reference

9.2 include/Kamil/CmdBox.h File Reference

```
#include "TextBox.h"
```
Include dependency graph for CmdBox.h:



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



Classes

- class CmdBox

  Class to handle the command TextBox.

9.3 CmdBox.h

Go to the documentation of this file.

```
#ifndef KAMIL_CMDBOX_H
#define KAMIL_CMDBOX_H

#include "TextBox.h"

class CmdBox : public TextBox {
public:
  using TextBox::TextBox;
};
#endif // KAMIL_CMDBOX_H
```

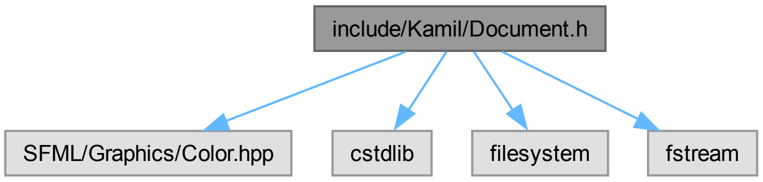
9.4 include/Kamil/Document.h File Reference

Interface file for the Document class.

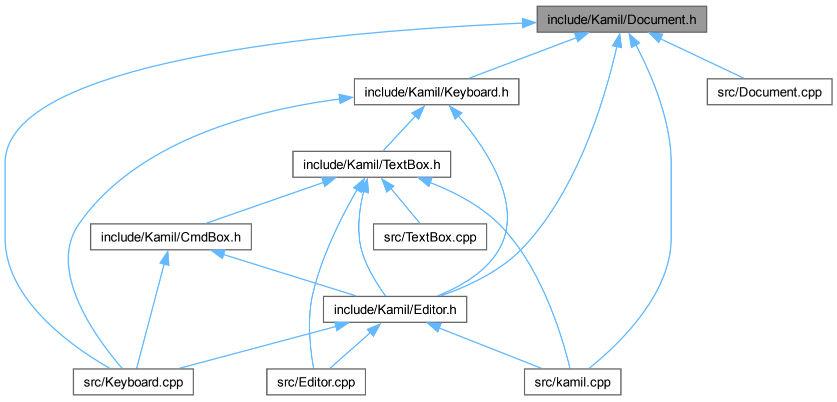
```
#include "SFML/Graphics/Color.hpp"
#include <cstdlib>
#include <filesystem>
```

```
#include <fstream>
```

Include dependency graph for Document.h:



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



Classes

- class Document

  Document class.
- struct Document::Theme

  a struct for the Theme
- struct Document::Config

  A struct for the configuration.

9.4.1 Detailed Description

Interface file for the Document class.

The Document.h file is responsible for all File I/O between the system and the program it can read and write files. It is also responsible for the configuration file in the 'config.toml' format

9.5 Document.h

Go to the documentation of this file.

```
#ifndef KAMIL_DOCUMENT_H
#define KAMIL_DOCUMENT_H

#include "SFML/Graphics/Color.hpp"
#include <cstdlib>
#include <filesystem>
#include <fstream>


class Document {
public:
    struct Theme{
        sf::Color bcol;
        sf::Color fcol;
    };

    struct Config{
        std::string cmd;
        std::string font;
        Theme theme;
    };
  Document();

  Document(std::string fileP);

  void init();

  void init(std::string inF);

  std::string readFile();

  std::string getRelPath();

  std::string getAbsPath();


  bool findConfig();


  void createFile(std::string filename);

  bool saveFile(const std::string &filename);

  bool saveFile();

  void setBuffInfo(std::string info);

  bool hasChanged();

  void setChange();

  Config getConfig();


private:
  std::string relPath;
  std::string absPath;
  std::string buffInfo;
  bool docChanged;
};
#endif // KAMIL_DOCUMENT_H
```

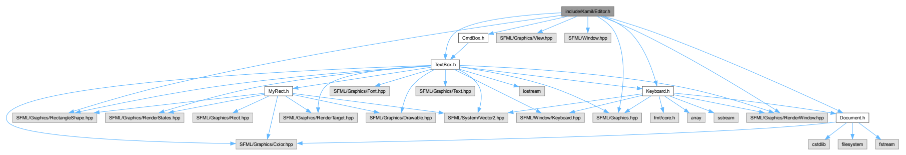
9.6 include/Kamil/Editor.h File Reference

Interface file for the Editor class.

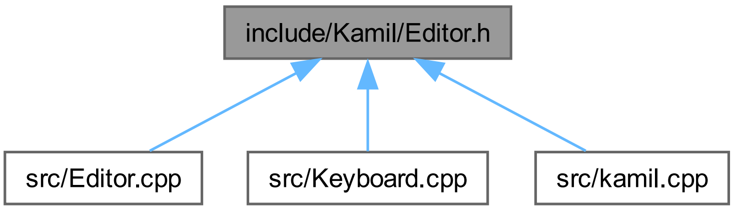
```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
#include <SFML/Window.hpp>
#include "CmdBox.h"
#include "Document.h"
```

```
#include "Keyboard.h"
#include "TextBox.h"
```

Include dependency graph for Editor.h:



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



Classes

- class Editor

  Class that is a centre for how the other classes interact with each other and draws everything in the Editor to the screen.

9.6.1 Detailed Description

Interface file for the Editor class.

The Editor class is responsible for the interaction between the different classes. All things outside the main while loop will be checked or initialise. Anything to do with the Editor Window will happen here

9.7 Editor.h

Go to the documentation of this file.

```
#ifndef KAMIL_EDITOR_WINDOW_HPP
#define KAMIL_EDITOR_WINDOW_HPP

#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
#include <SFML/Window.hpp>

#include "CmdBox.h"
#include "Document.h"
#include "Keyboard.h"
```

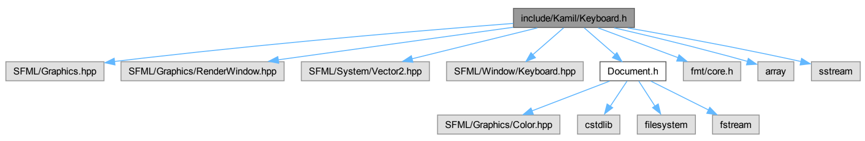
```
#include "TextBox.h"

class Editor {
public:
  Editor(sf::RenderWindow *window, sf::Event *event, Document *doc);

  ~Editor();

  void draw();


  void useConfig(const Document::Config& conf);

  void regexPatternMatchin();

private:
  Document *doc;
  TextBox *textBox;
//  CmdBox *cbox;              /**< pointer to command box that we draw */
  sf::RenderWindow *window;
  sf::Event *event;
  sf::View camera;
  Keyboard kb;
  std::string cmd;
  bool loadFromFile;
};

#endif // KAMIL_EDITOR_WINDOW_HPP
```

9.8 include/Kamil/Keyboard.h File Reference

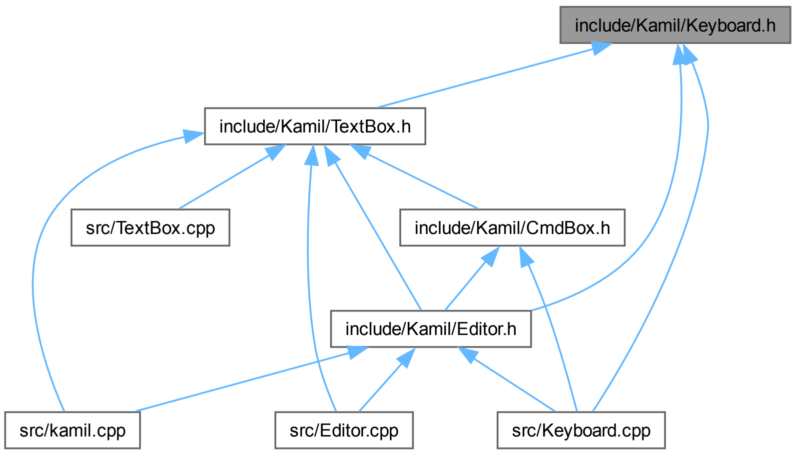
Interface file for Keyboard.h.

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>
#include "Document.h"
#include <fmt/core.h>
#include <array>
#include <sstream>
```

Include dependency graph for Keyboard.h:



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



Classes

- class Keyboard

  A class to handle Keyboard input.

Namespaces

- namespace KEYS

  An enum for Keyboard characters in hex form.

Enumerations

- enum {
  KEYS::ESCAPE = 0x1B , KEYS::ENTER = 0xD , KEYS::BS = 0x8 , KEYS::Shift_A = 0x41 ,
  KEYS::CTRL = 0x11 , KEYS::DELETE = 0x7f , KEYS::CR = 0x13 , KEYS::UP_ARROW = 0x48 ,
  KEYS::DOWN_ARROW = 0x50 , KEYS::RIGHT_ARROW = 0x4D , KEYS::LEFT_ARROW = 0x4B }

9.8.1 Detailed Description

Interface file for Keyboard.h.

A class that handles all keyboard and mouse events for the editor is responsible for manging input of keyboard data and their corresponding command

9.9 Keyboard.h

Go to the documentation of this file.

```
#ifndef KAMIL_KEYBOARD_H
#define KAMIL_KEYBOARD_H

#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>

#include "Document.h"
#include <fmt/core.h>

#include <array>
#include <sstream>

namespace KEYS {
enum {
  ESCAPE = 0x1B,
  ENTER = 0xD,
  BS = 0x8,
  Shift_A = 0x41,
  CTRL = 0x11,
  DELETE = 0x7f,
  CR = 0x13,
  UP_ARROW = 0x48,
  DOWN_ARROW = 0x50,
  RIGHT_ARROW = 0x4D,
  LEFT_ARROW = 0x4B,
};
}

class Keyboard {
public:
  Keyboard(sf::RenderWindow *win, Document *doc, sf::Vector2f bounds);


  bool isKeyPressed(sf::Keyboard::Key);

  bool isTextEntered();

  bool isCmdTextEntered();

  bool isTextDeleted();

  std::string getTextEntered();

  std::string getCmdTextEntered();

  void setTextEntered(std::string);

  void setCmdTextEntered(std::string);


  void backSpace();

  void handleKeyEvent(sf::Event &event);

  void handleCmdKeyEvent(sf::Event& event);

  void handleMouseEvent(sf::Event &event); // not implemented yet


private:
  sf::RenderWindow *window;
  std::stringstream tEntered;
  std::stringstream tDeleted;
  std::string ctEntered;
  std::string ctDeleted;
};
#endif // KAMIL_KEYBOARD_H
```

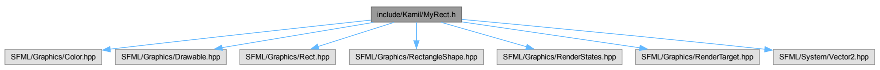
9.10 include/Kamil/MyRect.h File Reference

Interface file for the MyRect class.

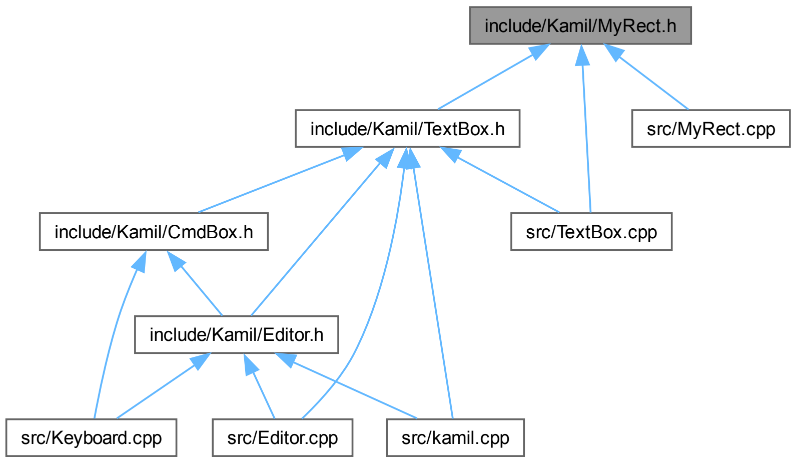
```
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Drawable.hpp>
```

```
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for MyRect.h:


include/Kamil/MyRect.h
SFML/Graphics/Color.hpp
SFML/Graphics/Drawable.hpp
SFML/Graphics/Rect.hpp
SFML/Graphics/RectangleShape.hpp
SFML/Graphics/RenderStates.hpp
SFML/Graphics/RenderTarget.hpp
SFML/System/Vector2.hpp

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



Classes

- class MyRect

    gives extra functionality to FloatRect

9.10.1 Detailed Description

Interface file for the MyRect class.

Inherits from sf::FloatRect and sf::Drawable. sf::FloatRect is a templated class of sf::Rect<float> and its primary use is for defining the border and creating a hollow rectangle, as such it only has methods for collision detection and intersections. The normal RectangleShape class creates a basic rectangle without the collision and intersections checking so we inherit this functionality from FloatRect and in effect add it to the instantiated RectangleShape in the MyRect class.

The sf::Drawable is only here to add a draw property to our class so when we draw to the RenderTarget, in this case RenderWindow, we can use the same code of window.draw(our_own_object) instead of the general our_own_↵
object.draw(window). This is done so when others use this code it makes it easier for them to follow a standard way of drawing to the RenderTarget and not having to worry about passing parameters into the objects.

9.11 MyRect.h

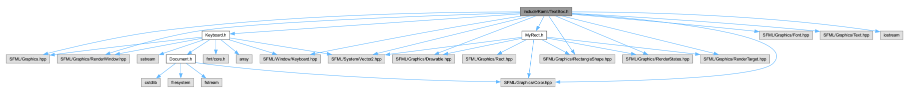
Go to the documentation of this file.

```
#ifndef KAMIL_MYRECT_H
#define KAMIL_MYRECT_H

#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Vector2.hpp>

class MyRect : public sf::FloatRect, public sf::Drawable {
public:
  MyRect(sf::Vector2f pos, sf::Vector2f size, sf::Color fillColour,
         sf::Color outlineColour, float outlineThicknes);
  MyRect();

  void setPosition(sf::Vector2f pos);

  sf::Vector2f getPos() const;

  void setFillColour(sf::Color colour);

  void setSize(sf::Vector2f size);

  sf::Vector2f getSize() const;

  void draw(sf::RenderTarget &target, sf::RenderStates states) const override;

protected:
  sf::RectangleShape rect;
  sf::Vector2f pos;
  sf::Vector2f size;
  sf::Color fillColour;
  sf::Color outlineColour;
  float outlineThicknes;
};

#endif // KAMIL_MYRECT_H
```

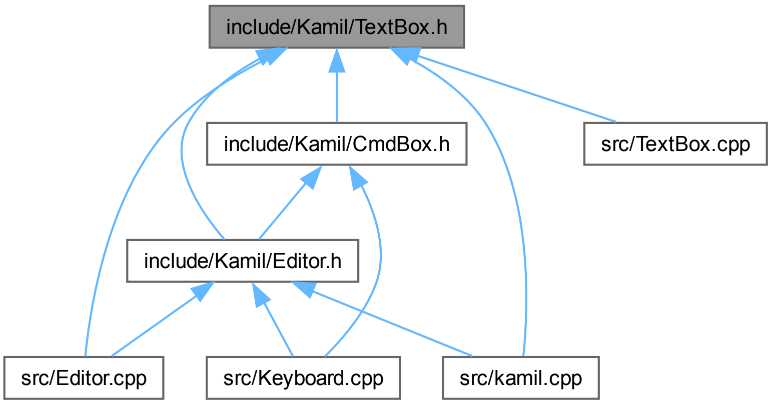
9.12 include/Kamil/TextBox.h File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Font.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Text.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <iostream>
#include "Keyboard.h"
#include "MyRect.h"
```

Include dependency graph for TextBox.h:



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



Classes

- class TextBox

  A class that makes a Textbox in SFML.

9.13 TextBox.h

Go to the documentation of this file.

```
#ifndef KAMIL_TEXTBOX_HPP
#define KAMIL_TEXTBOX_HPP

#include <SFML/Graphics.hpp>
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Font.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Text.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <iostream>

#include "Keyboard.h"
#include "MyRect.h"

class TextBox : public MyRect {
public:
  TextBox(sf::RenderWindow *win, sf::Vector2f pos, sf::Vector2f size,
          std::string sfont, int fsize, sf::Color fcol, sf::Color background,
          float thicc);
  TextBox();


  void setTextSize(int size);

  int getTextSize() const;

  void setTextColour(sf::Color colour);

  sf::Color getTextColour() const;

  void setFont(sf::Font &font);


  void setFont(std::string font);

  sf::Text getTextBox() const;

  void setString(std::string nstring);

  std::string getString() const;

  void draw(sf::RenderTarget &target, sf::RenderStates states) const override;

  bool isMouseHover();

private:
  sf::RenderWindow *window;
  sf::Text tbox{};
  sf::Font font{};
  std::string fname{};
  int fsize{};
  sf::Color fcol{};
  bool mouseHover;
};
#endif // KAMIL_TEXTBOX_HPP
```

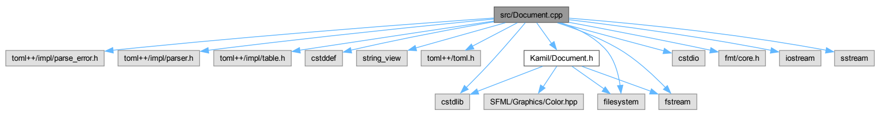
9.14 README.md File Reference

9.15 src/Document.cpp File Reference

The Implementation for Document.h.

```
#include "toml++/impl/parse_error.h"
#include "toml++/impl/parser.h"
#include "toml++/impl/table.h"
#include <cstddef>
#include <string_view>
#include <toml++/toml.h>
#include <Kamil/Document.h>
#include <cstdio>
#include <cstdlib>
#include <fmt/core.h>
#include <fstream>
#include <iostream>
#include <sstream>
#include <filesystem>
```

Include dependency graph for Document.cpp:



9.15.1 Detailed Description

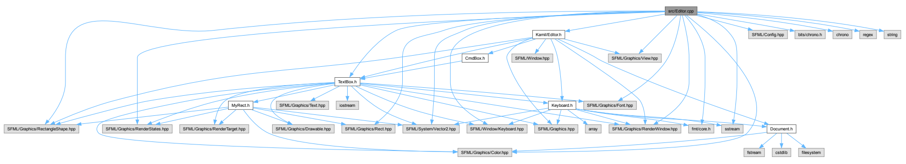
The Implementation for Document.h.

This is the Implementation code for the interface file Document.h It is where all the code for file I/O is handled.

9.16 src/Editor.cpp File Reference

```
#include <Kamil/Editor.h>
#include <Kamil/TextBox.h>
#include <SFML/Config.hpp>
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Font.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <bits/chrono.h>
#include <chrono>
#include <regex>
#include <fmt/core.h>
#include <sstream>
#include <string>
```

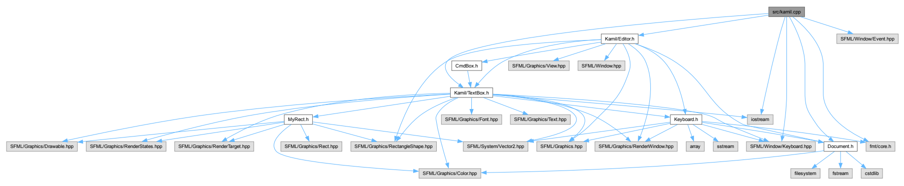
Include dependency graph for Editor.cpp:



9.17 src/kamil.cpp File Reference

```
#include <Kamil/TextBox.h>
#include <Kamil/Editor.h>
#include <SFML/Window/Event.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <fmt/core.h>
#include <iostream>
#include <Kamil/Document.h>
```

Include dependency graph for kamil.cpp:



Functions

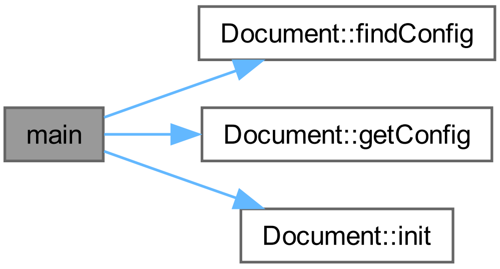
- int main (int argc, char ∗argv[ ])

9.17.1 Function Documentation

9.17.1.1 main()

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

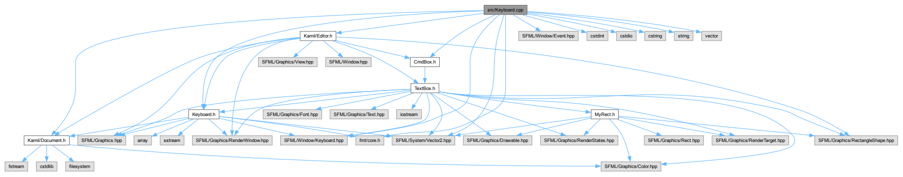
Here is the call graph for this function:



9.18 src/Keyboard.cpp File Reference

```
#include <Kamil/Document.h>
#include <Kamil/Editor.h>
#include <Kamil/Keyboard.h>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Event.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <Kamil/CmdBox.h>
#include <cstdint>
#include <cstdio>
#include <cstring>
#include <fmt/core.h>
#include <string>
#include <vector>
```

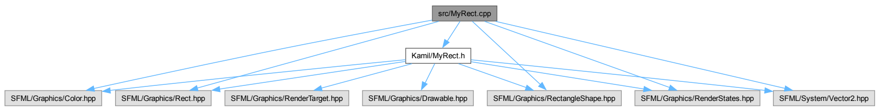
Include dependency graph for Keyboard.cpp:



9.19 src/MyRect.cpp File Reference

```
#include <Kamil/MyRect.h>
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/System/Vector2.hpp>
```

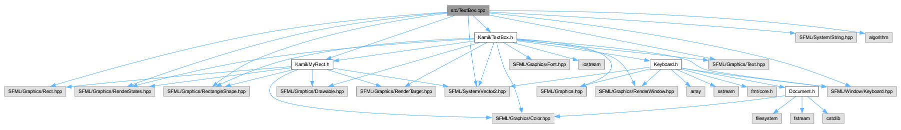
Include dependency graph for MyRect.cpp:


src/MyRect.cpp
Kamil/MyRect.h
SFML/Graphics/Color.hpp
SFML/Graphics/Rect.hpp
SFML/Graphics/RenderTarget.hpp
SFML/Graphics/Drawable.hpp
SFML/Graphics/RectangleShape.hpp
SFML/Graphics/RenderStates.hpp
SFML/System/Vector2.hpp

9.20 src/TextBox.cpp File Reference

```
#include <Kamil/MyRect.h>
#include <Kamil/TextBox.h>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/Text.hpp>
#include <SFML/System/String.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <algorithm>
```

Include dependency graph for TextBox.cpp:



Index