GRAPHICAL LIBRARY “IDISPLAYMODULE” DOCUMENTATION

`void IDisplayModule::reset()`

Reset the library.

`void IDisplayModule::open()`

Open and initialize the window.

`void IDisplayModule::close()`

Close and destroy the window.

`void IDisplayModule::reset()`

Reset the library.

`bool IDisplayModule::isOpen()`

Check if the window is open.

Return

True if the window is open
False if the window is close

HANDLE SWITCHING LIBS & GAMES

`bool IDisplayModule::switchToNextLib() const`

Check if the key **N** is pressed to switch to the next library.

Return

True if the key is pressed
False if the key is not pressed

`bool IDisplayModule::switchToPreviousLib() const`

Check if the key **B** is pressed to switch to the previous library.

Return

True if the key is pressed
False if the key is not pressed

`bool IDisplayModule::switchToNextGame() const`

Check if the key **P** is pressed to switch to the next game.

Return

True if the key is pressed
False if the key is not pressed

`bool IDisplayModule::switchToPreviousGame() const`

Check if the key **O** is pressed to switch to the next game.

Return

True if the key is pressed
False if the key is not pressed

`bool IDisplayModule::shouldBeRestarted() const`

Check if the key **R** is pressed to restart the library.

Return

True if the key is pressed
False if the key is not pressed

`bool IDisplayModule::shouldGoToMenu() const`

Check if the key **M** is pressed to go to the menu.

Return

True if the key is pressed
False if the key is not pressed

`bool IDisplayModule::shouldExit() const`

Check if the key **ESCAPE** is pressed to exit.

Return

True if the key is pressed
False if the key is not pressed

HANDLE INPUTS & EVENTS

```
bool IDisplayModule::isKeyPressed(IDisplayModule::Keys key) const
```

Check if the **key** is pressed to exit.

The **key** type is the following enumeration:

```
IDisplayModule::Keys {
        LEFT,
        RIGHT,
        UP,
        DOWN,
        Z,
        Q,
        S,
        D,
        A,
        E,
        W,
        X,
        SPACE,
        J,
        K,
        U,
        I,
        ENTER,
        BACKSPACE,
        KEYS_END
};
```

Parameters

key the key pressed

Return

True if the **key** is pressed
False if the **key** is not pressed

HANDLE LOOP

Your core (or games) should nonetheless call all of these functions in this specific order: clear -> update -> render

```
void IDisplayModule::clear() const
```

Clear the window.

```
void IDisplayModule::update()
```

Update the window.

```
void IDisplayModule::render() const
```

Render the window.

```
char IDisplayModule::getKeyCode() const
```

Get key pressed.

Return

The key pressed
returns \n if enter was pressed and \0 if nothing was pressed.

GETTERS

```
float IDisplayModule::getDelta() const
```

Get the number of frames that passed between two calls to this function.

Return

The number of frames that passed between two calls to this function

```
Std::string &IDisplayModule::getLibName() const
```

Get the name of the library.

Return

The name of the library

DISPLAY STUFF

Everything you display after this will have the selected color.

```
void IDisplayModule::setColor(IDisplayModule::Colors color)
```

Sets the color for all the following draw functions.

The color type is the following enumeration:

```
IDisplayModule::Colors {
        DEFAULT,
        BLACK,
        RED,
        GREEN,
        YELLOW,
        BLUE,
        MAGENTA,
        CYAN,
        LIGHT_GRAY,
        DARK_GRAY,
        LIGHT_RED,
        LIGHT_GREEN,
        LIGHT_YELLOW,
        LIGHT_BLUE,
        LIGHT_MAGENTA,
        LIGHT_CYAN,
        WHITE,
        COLORS_END
};
```

Parameters

color the color

```
void IDisplayModule::putPixel(float x, float y) const
```

Display a pixel.

Parameters

x the x position of the pixel
y the y position of the pixel

```
void IDisplayModule::putLine(float x1, float y1, float x2, float y2) const
```

Display a line.

Parameters

x1 the x position of the first point of the line
y1 the y position of the first point of the line
x2 the x position of the second point of the line
y2 the y position of the second point of the line

```
void IDisplayModule::putRect(float x, float y, float w, float h) const
```

Display an empty rectangle.

Parameters

- **x** the x position of the rectangle
- **y** the y position of the rectangle
- **w** the width of the rectangle
- **h** the height of the rectangle

```
void IDisplayModule::putFillRect(float x, float y, float w, float h) const
```

Display a fill rectangle.

Parameters

- **x** the x position of the rectangle
- **y** the y position of the rectangle
- **w** the width of the rectangle
- **h** the height of the rectangle

```
void IDisplayModule::putCircle(float x, float y, float rad) const
```

Display an empty circle.

Parameters

- **x** the x position of the circle
- **y** the y position of the circle
- **rad** the radian of the circle

```
void IDisplayModule::putFillCircle(float x, float y, float rad) const
```

Display a fill circle.

Parameters

- **x** the x position of the circle
- **y** the y position of the circle
- **rad** the radian of the circle

```
void IDisplayModule::putText(const std::string &text, unsigned int size,
float x, float, y) const
```

Display some text.

Parameters

- **text** the string to display
- **size** the size of text
- **x** the x position of the text
- **y** the y position of the text

OUR ENTRY POINT TO INSTANTIATE THE LIBRARY:
The symbol is “createLib”.

```
extern "C" std::unique_ptr<IDisplayModule> createLib(void)
{
    return std::make_unique<"libraryClass">();
}
```