DESIGN MODULE

The module documentation

A module for drawing components on screen

Author

Rômulo Peres de Moraes

Table of Contents

1. The purposes of this document

This document has the purpose of describe the module and its interfaces with the goal of improve the Developer Experience (DX) once the developers have to create new features or even for maintenance.

2. Design module

The design module is a component from the Rclock that draws all Rclock's components on screen. This module is not able to define size and positions of the elements, but when necessary, its routines will draw the digits, the colons and the date of the clock on screen.

3. The layout of the documentation and the module

The module is divided into two parts, the public code and the private code, they will be placed inside the directories public/ and private/ respectively. All private interface names shall begin with an underscore. The public interfaces may use the private interfaces, the private interfaces may use another private interfaces, however, a private interface can't use a public interface.

4. Interfaces

4.1. Public interfaces

void fillClockSegment(WINDOW *clockWindows[ ], unsigned char numberToDraw, unsigned char windowIndex);

Purposes:

Once called, this procedure will draw the given number on the given pair of Ncurses windows.

Preconditions:

The windows have to be generated by the **screen-manager** module before calling this subroutine.

The windows' placeholders have to be defined before calling this subroutine.

The windows have to be moved to its placeholders before calling this subroutine.

Postconditions:

After accomplishing its tasks, the specified segment given by the **windowIndex** argument will be drawn with the given number.

Special considerations:

This procedure uses the _**normalizeSegment( )** to split a number into two digits (if greater or equal than 10) or return the number with a zero (if less or equal than 9).

This procedure uses the **getDigitColor( )** from the **colors** module to fetch the digit's color ID.

This procedure uses the **getDigitShape( )** from the **shapes** module to fetch the given number's shape, to act like a map to draw the number succesfully.

This procedure uses the private procedure _**drawClockWindow( )** that acts as the brush.

void drawAllClockWindows(struct tm *timeStruct, struct DatetimeScreenManagerDesignerModules userArguments);

Purposes:

Once necessary, this procedure will fill all digits and colons components of the clock.

Examples:

This procedure can be useful when the terminal is resized and all components need to be resized and moved to a new position.

Preconditions:

The user arguments have to be collected before calling this procedure.

The windows have to be generated by the **screen-manager** module before calling this subroutine.

The windows' placeholders have to be defined before calling this subroutine.

The windows have to be moved to its placeholders before calling this subroutine.

The **struct tm** have to be initialized and optionally updated with the custom time given by the user using the **datetime** module before calling this subroutine.

Postconditions:

All Rclock's windows, except the date, will be drawn with the correct contents.

Special considerations:

This procedure checks if the seconds should be visible as it draw the components, the conditions for the seconds be invisible are: a command-line argument given by the user and the terminal be horizontally small enough to doesn't be able to show the seconds.

This procedure can verify if the user provided a flag that hides the seconds, but it also uses a function from the **screen-manager** module that tells us if the seconds should really be visible with the current terminal size.

This routine uses the _**fillClockColons( ),** that is a private procedure to draw the clock's colons.

void drawDate(struct tm *theTime, struct DatetimeModule datetimeArguments, struct ColorsModule colorArguments);

Purposes:

Once called, this procedure draws the date stored inside the **struct tm** to the date component on screen.

Preconditions:

The user arguments have to be collected before calling this procedure.

The windows have to be generated by the **screen-manager** module before calling this subroutine.

The date window placeholder have to be defined before calling this subroutine.

The **struct tm** have to be initialized and optionally updated with the custom time given by the user using the **datetime** module before calling this subroutine.

Postconditions:

After accomplishing its tasks, the date will be visible below the time, using the default format or the custom date format given by the user through command-line arguments.

Special considerations:

This procedure uses function from the **screen-manager, datetime and colors** modules to get the reference to the date's window, move it to its placeholder, stringify the date to be shown on screen and set the correct color defined to the component.

void drawProgramErrorCallback(void *arguments);

Purposes:

This procedure is a callback that is given to the **screen-manager** module as it is managing the error screen.

Postconditions:

Once this procedure has completed its tasks, the error window will show the given error message.

Special considerations:

The **screen-manager** module is the only one that can interact with sizes and positions of the components on screen, therefore, it isn't allowed to draw the components on its operations. This callback is given to the module for it be able to call it and draw the error message.

This callback needs arguments to accomplish its operations, the received argument is a void pointer that needs to be typecasted to the **struct ErrorUpdateCallbacksArguments**. The given struct contains the error message and the windows that are necessary to report a program error.

This procedure clears the whole screen and uses the private subroutine **_writeErrorMessageOnErrorWindow( )** to draw the error message.

4.2. Private interfaces

void _writeErrorMessageOnErrorWindow(char *msg, size_t windowWidth, WINDOW *errorWindow);

Purposes:
Once called, this procedure will draw the given error message on the given error window.

Preconditions:
The error window that this procedure works on needs to be generated by the **screen-manager** module before calling it.

Postconditions:
Once this procedure has completed its tasks, the error window will show the given error message.

Special considerations:
This subroutine draws the error string justified to the window, the last line that usually doesn't fit perfectly with the width is aligned to the center.

The error message is always red, and defined by the macro ERROR_MESSAGE_RED_ID

void _normalizeSegment(unsigned char number, Digit segmentDigits[2]);

Purposes:
Once called, this procedure will split the given number into two digits (if greater or equal than 10) or generate a zero before the number (if less or equal than 9) and copy them to the **segmentDigits.**

Examples:
This routine is useful because the clock's windows are independents and only one digit can be drawn on each window

Postconditions:
The given number will be available on **segmentDigits** as two separate digits.

void _drawClockWindow(WINDOW *targetWindow, ClockPixel (*shapeToBeDrawn)[3], ColorID digitColorID);

Purposes:

Once called, this procedure will draw the **shapeToBeDrawn** on the **targetWindow** using the color **digitColorID**.

Preconditions:

The window have to be generated by the **screen-manager** module before calling this procedure.

The window have to be moved to its placeholder before calling this procedure.

The colors feature needs to be initialized by the **colors** module.

Postconditions:

The given window will display the digit that references the **shapeToBeDrawn** once this procedure has finished its tasks.

Special considerations:

The **shapeToBeDrawn** is a 2D array that is given by the shapes module, it acts like a map to draw the digit on the window.

void _fillClockColons(struct DatetimeScreenManagerDesignerModules userArguments);

Purposes:

Once called, this procedure draws the clock’s colons on screen.

Preconditions:

The command-line user arguments have to be collected before calling this procedure.

The clock windows have to be generated by the **screen-manager** module before calling this procedure.

The windows have to be moved to their placeholders before calling this procedure.

The colors features have to be initialized by the **colors** module before calling this procedure.

Postconditions:

The clock colons will be visible on screen once this subroutine has completed its tasks.

Special considerations:

This procedure verifies if the seconds are visible (this state depends on the user arguments and the terminal size), if they are visible, the second colon that is placed between the seconds and the minutes is also visible, otherwise it is hidden.

This procedure uses functions from **shapes** and **colors** modules to fetch the colon shape and its color.

This procedure uses a function from the **screen-manager** module to verify if there's enough width to draw the second colon (if not disabled by the user).