SCREEN-MANAGER MODULE

The module documentation

A module for managing screen operations and states

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. Screen-manager module

The screen-manager module is a component from the Rclock that create windows, define placeholders, move/resize windows, set states and so on...This is the one of the most important modules of the software, any operations that interact with the screen use this module somehow.

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 setDateStringLength(size_t newLength);

Purposes:

Once called, this procedure will update a static variable with the length of the date string. This value is used to align the string to the center of the screen.

Postconditions:

The calculation to align the string to the center can be done after this procedure has completed its tasks.

void destroyRclockWindows(ProgramArguments arguments);

Purposes:

Once called, this procedure will destroy all components on screen.

Examples:

This is useful when the terminal is extremely small and an error message show up. All components are destroyed and recreated when the terminal has enough dimensions to render the clock.

Postconditions:

All components are destroyed and no operations can be performed on those window pointers.

The windows have to be regenerated to continue with the normal clock operations.

void setValuesForClockStates(ProgramArguments arguments);

Purposes:

Once called, this function set states related to the terminal size with the goal of hide/show the date and the seconds correctly.

Preconditions:

This subroutine have to be called after calling the **loadInitialTerminalSize( )**.

This procedure shall be called at the configuration stage of the program, the rest of the program requires the values generated by this subroutine.

Postconditions:

The Rclock will be able to hide the date and the seconds on the program’s startup if the terminal is small enough at the software launch.

struct ErrorWindows generateErrorWindows(float errorWindowWidthFraction, bool enableExitMessage);

Purposes:

Once called, this function will create the error window with the dimensions relative to the terminal size and return them inside the **struct ErrorWindows**. If exit message is enabled, an additional window will be created to show an additional message.

Examples:

If the exit message is enabled, a text will be shown below the error message with the contents like “Press enter to exit”, only when the error isn’t recoveriable.

Postconditions:

The generated windows can be used to display a program error.

void moveTimeWindowsToPlaceholders( );

Purposes:

Once called, this procedure will move the clock windows (digits and colons) to the placeholders defined by the subroutine **setPlaceHolders( )**.

Preconditions:

This procedure can only be called after generating the windows.

This procedure can only be called after setting the placeholders with the **setPlaceHolders( )** procedure.

Postconditions:

After this procedure has completed its tasks, the clock windows will be in the correct place on screen and ready to be drawn.

Special considerations:

This procedure has a debug mode, it is only enabled when the DEBUG macro is defined at the **debug.h** file.

Once the debug mode is enabled, the time windows will have borders, making it easier to identify where they are.

void moveDateWindowToPlaceholder( );

Purposes:

Once called, this procedure will calculate the center of the screen and move the date window to that positions.

Preconditions:

This procedure can only be called after generating the windows.

This procedure can only be called after setting the placeholders with the **setPlaceHolders( )** procedure.

Postconditions:

After this procedure has completed its tasks, the date window will be in the correct place on screen and ready to be drawn.

Special considerations:

The values generated by the **setPlaceHolders( )** is also used here, but only the Y axis position.

The X axis position is generated inside the procedure.

This procedure has a debug mode, it is only enabled when the DEBUG macro is defined at the **debug.h** file.

Once the debug mode is enabled, the date window will have borders, making it easier to identify where it is.

bool detectTerminalResizes( );

Purposes:

Once called, this function returns true If the terminal resized since last checking, otherwise it will return false.

Preconditions:

The subroutine **loadInitialTerminalSize( )** have to be called before calling this procedure.

Special considerations:

This function fetch the current dimensions of the terminal using the Ncurses library and compare with static variables inside the module.

void generateWindows(struct DatetimeScreenManagerDesignerModules userArguments);

Purposes:

Once called, this procedure will generate the clock and the date windows.

Preconditions:

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

Special considerations:

This routine doesn’t generate the date window if the user disabled it through command-line arguments.

void setPlaceHolders(ProgramArguments arguments);

Purposes:

Once called, this procedure will generate the placeholders for all windows created by the **generateWindows( )**.

Preconditions:

This procedure shall be called after calling the **generateWindows( )** and before calling any procedure that moves the windows to their placeholders.

Postconditions:

The windows can be moved to their placeholders after calling this subroutine.

Special considerations:

The window placeholders are stored in a static variable inside the module’s public file.

The placeholders are defined based on the count of windows, this calculation is necessary because the clock have to be aligned to the center, there’s difference if the seconds are hidden or not.

void refreshWindows( );

Purposes:

This procedure refresh all windows (clock and date windows).

This routine is necessary because all changes made by the Ncurses are buffered until a new refresh.

Preconditions:

The windows have to be generated before calling this procedure.

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

The terminal's states have to be defined using the **setValuesForClockStates( )** before calling this procedure.

The windows must have been drawn before calling this procedure.

Postconditions:

The windows will be updated with the new changes after this procedure has completed its tasks.

Special considerations:

This procedure has a debug mode, it is only enabled when the macro **DEBUG** is defined at the **debug.h** file.

Once the debug mode is enabled, all windows will be shown with borders, making it easier to identify their locations.

This procedure only refreshes the date and the seconds windows if they weren't hidden by the user through command-line arguments or by the current terminal size.

The windows are stored on a static variable inside the module's public file.

void updateErrorMessageFrames(struct ErrorWindows windows, float errorWindowWidthFraction, char *errorMessage, void (*drawProgramErrorCallback)(void *arguments), void *drawErrorArguments, bool (*errorVerificationCallback)(), bool enableExitMessage);

Purposes:

As soon an error is generated by the program, this procedure will update the frames (for responsiveness purposes) until the user exits the program or the error conditions become invalid.

Preconditions:

The error windows must be generated by the **generateErrorWindows( )**.

Postconditions:

For each loop iteration, the error message will be moved and resized if necessary.

Special considerations:

This procedure uses the **_getTerminalSize( )** and the **_calculateErrorWindowsMeasures( )** to set the correct position and the dimensions of the error windows.

This procedure receives a callback (***drawProgramErrorCallback**) that is responsible for draw the error on the error windows.

This procedure receives a callback (***errorVerificationCallback**) that is responsible for check if the error conditions still valid. This callback is only useful when the error is recoveriable. The recoveriable errors are: the terminal width is extremely small and the terminal height is extremely small.

The procedure’s loop sleeps periodically, there’s no necessity of a continuous execution loop.

4.2. Private interfaces

void _getTerminalSize(unsigned int *width, unsigned int *height);

Purposes:

Once called, this procedure will fetch the width and height of the terminal using the Ncurses library.

Preconditions:

The Ncurses library must be linked to the program.

Postconditions:

The width and the height of the terminal will be available through the pointers.

struct ErrorWindowsMeasures _calculateErrorWindowsMeasures(float errorWindowWidthFraction, struct WindowSize winSize);

Purposes:

Once called, this function will calculate the correct dimensions for the error windows based on the given terminal size.

Postconditions:

The returned struct contains all necessary values to move and resize the error windows correctly.