DATETIME MODULE

The module documentation

A module for date/time operations

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. Datetime module

The datetime module is a component from the Rclock software that handles everything about date and time. The use of this component is crucial for the correct operation of the entire program.

3. The layout of the documentation and the module

The module is divided into two parts, the public code and the private code, the 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 interface, however, a private interface can't use a public interface.

4. Interfaces

4.1. Public interface

struct tm* generateDateAndTime( );

Purposes:

It will generate and fill a **struct tm** pointer that will be used to hold the date/time.

Preconditions:

Should be called on the program's startup.

Postconditions:

The **struct tm** pointer returned from the interface may be used to draw the clock or set a custom date/time from the current value.

void setNewTime(struct tm *datetimeStruct, struct DatetimeModule dateTimeArguments, char *errorOutput);

Purposes:

It will update the **struct tm** pointer with the custom time (hours, minutes and seconds) provided by the user.

This routine will also save the initial time of the program if any custom time was given and enable a bool flag for identification.

Preconditions:

The **struct tm** pointer needs to be already filled at this point by the **generateDateAndTime( )** subroutine.

The user arguments must be collected before calling this procedure.

Postconditions:

The clock will show the new time set by the user, however, the new values must be verified for range errors by other subroutine.

Additional comments:

Any error that this subroutine may generate internally should be copy into the **errorOutput** pointer.

void setNewDate(struct tm *datetimeStruct, struct DatetimeModule datetimeArguments, char *errorOutput);

Purposes:

It will update the **struct tm** pointer with the custom date (days, month and year) provided by the user.

This routine will also save the initial time of the program if any custom time was given and enable a bool flag for identification.

Preconditions:

The **struct tm** pointer needs to be already filled at this point by the **generateDateAndTime( )** subroutine.

The user arguments must be collected before calling this procedure.

Postconditions:

The clock will be able to show the new date set by the user (if enabled), however, the new values must be verified for range errors by other subroutine

Additional comments:

Any error that this subroutine may generate internally should be copy into the **errorOutput** pointer.

char* generateDateString(struct tm datetimeStruct, struct DatetimeModule datetimeArguments, char *outputBuffer);

Purposes:

This function will generate the date that will be shown below the clock time.

If the user provided a custom date format for the string, the custom format will be used instead of the default one.

Preconditions:

The **struct tm** needs to be already filled at this point by the **generateDateAndTime( )** subroutine.

This function is always called after set the custom date/time value. The user arguments must be collected before calling this subroutine.

Postconditions:

The date string returned from the function, through either return value or the **outputBuffer** pointer, can be used to be drawn below the clock.

Additional comments:

The default date format used by this function is "%A, %b %d %Y", that follows the strftime( ) function syntax.

void incrementClockSecond(struct tm *datetimeStruct);

Purposes:

Given the **struct tm** pointer, this procedure will increment its value in one second. This is useful for updating the clock for every elapsed second.

Special considerations:

After incrementing, a normalize should be applied to the struct for updating the minutes, hours, days and so on automatically once the seconds be greater than 59.

Preconditions:

The **struct tm** needs to be already filled at this point by the **generateDateAndTime( )** subroutine.

If the user provided any custom date/time value, the **struct tm** must be updated with these values before calling this procedure.

Postconditions:

The correct clock segments must be updated after calling this subroutine.

void sleepClock(unsigned int milliseconds);

Purposes:

It will freeze a loop of the program for x milliseconds, useful for don’t let the program in a raw loop, that would be a unnecessary overhead.

Preconditions:

This subroutine should be called at the end of the loop, to give the program a chance to draw something on screen on the first loop iteration.

Limitations:

The software uses an alarm signal as strategy to update the time struct once every second. A signal has the trait of abort a sleep that this subroutine creates, for this reason, this procedure should sleep in a loop that only stops when the necessary amont of sleep time has elapsed.

void verifyForDateAndTimeErrors(struct tm *datetimeStruct, char *errorOutput);

Purposes:

This procedure will verify any date/time range errors.

Example:

This subroutine would generate a error if the user provided 27 as a custom hour to the program.

Preconditions:

This procedure should be called after the **struct tm** pointer be initialized by the **generateDateAndTime( )** function.

This procedure should be called after setting any custom date/time values given by the user.

Postconditions:

If the subroutine found a range error, the program must read it and show an error screen, otherwise, the program can continue without further distractions.

Additional comments:

Any error generated by this subroutine should be copied to the **errorOutput** pointer.

void tryToUpdateTheClock(struct tm **timeStruct);

Purposes:

This procedure will update the date/time of the clock, this is useful when the computer sleeps, freezing the program.

Preconditions:

The **setNewTime( )** and **setNewDate( )** must be called first, these procedures set flags and variables that will help this routine decide the best thing to do.

Postconditions:

If the clock's time is more than 5 seconds of difference from the time that it should have, the time will be fixed.

Special considerations:

Once this procedure is called, it will work based on the given user arguments, if the user provided a custom date or time, this procedure will work based on the current time that the program started to be executed and the custom time that the user provided, with some calculations the clock is able to update the correct amount of seconds that elapsed since the computer started to sleep. If the user didn't provide any custom time, the time struct receive the current real time.

Additional comments:

This function have to be called once every 5 seconds

4.2. Private interface

bool _checkIfDateOrTimesSegmentsAreDigits(char *customDateTime);

Purposes:

This function will verify if the segments of a date and a time are digits.

Special considerations:

This routine takes advantage by the similar structure of both values (DD/MM/YYYY and xx:xx:xx).

struct DateStruct _parseDate(struct DatetimeModule datetimeArguments);

Purposes:

This routine will parse a date string using the format DD/MM/YYYY and return the fetched values.

Preconditions:

The user arguments must be fetched before calling this function.

Special considerations:

This function uses strlen( ) and sscanf( ) to make sure that the given date is in a correct format. Any error is reported by a boolean value returned inside the struct.

struct TimeStruct _parseTime(struct DatetimeModule datetimeArguments);

Purposes:

This function will parse a time using the format xx:xx:xx and return the fetched values.

Preconditions:

The user arguments must be fetched before calling this function;

Special considerations:

This function uses strlen( ) and sscanf( ) to make sure that the given date is in a correct format.

Any error is reported by a boolean value returned inside the struct.

void _setCustomTime(struct tm *datetimeStruct, struct DatetimeModule dateTimeArguments, char *errorOutput);

Purposes:

This procedure checks if the user provided a custom time string and uses the **_parseTime( )** to read and set values inside the **struct tm** pointer.

Preconditions:

The **struct tm** pointer must be initialized by the **generateDateAndTime( )** before calling this procedure.

The user arguments must be fetched before calling this procedure.

Postconditions:

If no errors were reported, the **struct tm** has the new time provided by the user.

Special considerations:

The errors that the **_parseTime( )** may generate are handled here, the error message is copied to the **errorOutput** pointer.

void _setCustomHourMinuteAndSecond(struct tm *datetimeStruct, struct DatetimeModule dateTimeArguments);

Purposes:

This procedure will update the **struct tm** pointer with the given hour, minute and second arguments that user may provide.

Preconditions:

The **struct tm** pointer must be initialized by the **generateDateAndTime( )** before calling this procedure.

The user arguments must be fetched before calling this procedure.

Postconditions:

The **struct tm** is updated with the new hours, minutes or seconds individually.

void _setCustomDate(struct tm *datetimeStruct, struct DatetimeModule datetimeArguments, char *errorOutput);

Purposes:

This procedure checks if the user provided a custom date string and uses the **_parseDate( )** to read and set values inside the **struct tm** pointer.

Preconditions:

The **struct tm** pointer must be initialized by the **generateDateAndTime( )** before calling this procedure.

The user arguments must be fetched before calling this procedure.

Special considerations:

The errors that the **_parseDate( )** may generate are handled here, the error message is copied to the **errorOutput** pointer.

Postconditions:

If no errors were reported, the **struct tm** has the new date provided by the user.

void _setCustomDayMonthAndYear(struct tm *datetimeStruct, struct DatetimeModule datetimeArguments);

Purposes:

This procedure will update the **struct tm** pointer with the given day, month and year arguments that the user might provide.

Preconditions:

The **struct tm** pointer must be initialized by the **generateDateAndTime( )** before calling this procedure.

The user arguments must be fetched before calling this procedure.

Postconditions:

The **struct tm** is updated with the new day, month or year individually.

char *_createZerosPaddingForTheYear(int year, char *outputPadding);

Purposes:

This function will generate zeros for the year when the value be smaller than 1000, the purpose of this is always make the year hold 4 or more digits.

Special considerations:

The char pointer return value is the same pointer that is provided by arguments (I.e *outputPadding).

Postconditions:

The char pointer will be filled with additional zeros characters that will make the final year be greater or equal than 4 digits.