

# ResPublish

P. Baillehache

May 1, 2019

## Contents

<b>1</b>	<b>Definitions</b>	<b>2</b>
1.1	TextOMeter . . . . .	2
<b>2</b>	<b>Interface</b>	<b>3</b>
<b>3</b>	<b>Code</b>	<b>5</b>
3.1	respublish.c . . . . .	5
3.2	respublish-inline.c . . . . .	8
<b>4</b>	<b>Makefile</b>	<b>9</b>
<b>5</b>	<b>Unit tests</b>	<b>10</b>
<b>6</b>	<b>Unit tests output</b>	<b>11</b>

## Introduction

ResPublish is a C library providing structures and functions to log and/or display data in various way during execution of a process.

- TextOMeter: displays textual data in a dedicated Xterm from a running parent process.

It uses the `PBErr` library.

# 1 Definitions

## 1.1 TextOMeter

The solution implemented in ResPublish to display text in a dedicated Xterm (while the parent process is attached to another terminal) is resumed below. It is very probably not portable to systems other than the one it has been developed on.

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
// Main function
// Example code to display text in an independant Xterm from a running
// parent process.
// Compile with gcc -o main main.c
int main() {
    // Temporary file to transmit the tty and pid from the Xterm to the
    // parent process
    #define TMPFILENAME "./tmptty"
    // Ensure the temporary file doesn't exist
    remove(TMPFILENAME);
    // Fork between the parent process executing the main algorithm and
    // the child process running the Xterm
    pid_t pid = fork();
    // If we are in the child process
    if (pid == 0) {
        // Open a new Xterm, executing '(echo $$ && tty) > ./tmptty'
        // to save its pid and tty into the temporary file, followed
        // by a bash to avoid the window closing right after the command
        char cmd[] = "/usr/bin/xterm";
        char* argv[] = {
            "xterm",
            "-e",
            "(echo $$ && tty) > " TMPFILENAME " ; bash",
            NULL
        };
        execv(cmd, argv);
        // Never reach here as the child get replaced by the Xterm
    } else {
        // Else, we are in the parent process
        // Wait one second to give time to the Xterm to write its pid and
        // tty into the temporary file
        sleep(1);
        // Open the temporary file
        FILE* fp = fopen(TMPFILENAME, "r");
        // Variable to memorize the tty of the Xterm
        char tty[100] = {'\0'};
        // Read the pid and tty of the Xterm
        fscanf(fp, "%d %s\n", &pid, tty);
        // Close and remove the temporary file
        fclose(fp);
        remove(TMPFILENAME);
        // Display the information about the Xterm
        printf("Xterm attached to tty %s and has pid %d\n", tty, pid);
        // Open the tty
        FILE* fty = fopen(tty, "w");
        // Simulate a process sending info toward the Xterm and its own
        // console
    }
}
```

```

    for (int i = 0; i < 10; ++i) {
        fprintf(ftty, "Toward Xterm: %d\n", i);
        printf("Toward parent's terminal: %d\n", i);
        sleep(1);
    }
    // Close the tty
    fclose(ftty);
    // Kill the Xterm
    kill(pid, SIGKILL);
}
// Return success code
return 0;
}

```

## 2 Interface

```

// ===== RESPUBLISH.H =====

#ifndef RESPUBLISH_H
#define RESPUBLISH_H

// ===== Include =====

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#include "gset.h"
#include "pberr.h"

// ---- TextOMeter

// ===== Define =====

#define TEXTOMETER_TTY_FILENAME "/tmp/textometertty.tmp"
#define TEXTOMETER_TTY_MAXLENGTH 100
#define TEXTOMETER_XTERM_CMD "/usr/bin/xterm"

// ===== Data structure =====

typedef struct TextOMeter {
    // Title
    char* _title;
    // Width
    int _width;
    // Height
    int _height;
    // PID of the associated xterm
    pid_t _pid;
    // File pointer to the tty of the associated xterm
    char _tty[TEXTOMETER_TTY_MAXLENGTH];
    FILE* _fp;
} TextOMeter;

// ===== Functions declaration =====

```

```

// Create a new TextOMeter of 'width' columns and 'height' lines and
// 'title' as title of the attached xterm
// May return NULL if the creation of the Xterm failed
TextOMeter* TextOMeterCreate(char* const title,
    const int width, const int height);

// Free the memory used by the TextOMeter 'that'
void TextOMeterFree(TextOMeter** that);

// Clear the content of the TextOMeter 'that'
// Return true if the content could be cleared, false else
#ifdef BUILDMODE != 0
inline
#endif
bool TextOMeterClear(TextOMeter* const that);

// Print the string 'str' on the TextOMeter 'that'
// Return true if the content could be printed, false else
#ifdef BUILDMODE != 0
inline
#endif
bool TextOMeterPrint(TextOMeter* const that, const char* const str);

// Flush the stream of the TextOMeter 'that'
#ifdef BUILDMODE != 0
inline
#endif
void TextOMeterFlush(TextOMeter* const that);

// ---- EstimTimeToComp

// ===== Define =====

// ===== Data structure =====

typedef struct EstimTimeToComp {
    // Start time
    time_t _start;
    // ETC
    char _etc[100];
} EstimTimeToComp;

// ===== Functions declaration =====

// Create a new EstimTimeToComp
EstimTimeToComp EstimTimeToCompCreateStatic();

// Free the memory used by the EstimTimeToComp 'that'
void EstimTimeToCompFreeStatic(EstimTimeToComp* that);

// Reset the start time of the EstimTimeToComp 'that' to current time
void ETCReset(EstimTimeToComp* that);

// Estimate the ETC of the EstimTimeToComp 'that' given the percentage
// of completion 'comp'
// time(0) is expected to returned Thu Jan 1 00:00:00 1970
const char* ETCGet(EstimTimeToComp* const that, float comp);

// ---- PBMailer

// ===== Define =====

```

```

// ===== Data structure =====

typedef struct PBMailer {
    // Set of strings to send
    GSetStr _messages;
    // Target email address
    char* _to;
    // Minimum delay in seconds between two actual emails
    // Used to avoid flooding the target address
    time_t _delayBetweenEmails;
    // Time of last sent email
    time_t _lastEmailTime;
} PBMailer;

// ===== Functions declaration =====

// Create a new PBMailer toward the email adress 'to'
// _delayBetweenEmails is initialiwed to 60s
PBMailer PBMailerCreateStatic(const char* const to);

// Free the memory used by the PBMailer 'that'
// Flush the remaining strings if any
void PBMailerFreeStatic(PBMailer* that);

// Send the strings of the PBMailer 'that' if the last PBMailerSend
// call is at least _delayBetweenEmails seconds old, else do nothing
// with the subject 'subject'
// Uses the 'mail' command which is supposed to configure, up and
// running by the user
void PBMailerSend(PBMailer* const that, const char* const subject);

// Add a copy of the string 'str' to the PBMailer 'that' to be sent
// later with PBMailerSend()
void PBMailerAddStr(PBMailer* const that, const char* const str);

// Set the minimum delay between emails of the PBMailer 'that' to 'delay'
#if BUILDMODE != 0
inline
#endif
void PBMailerSetDelayBetweenEmails(PBMailer* const that, const time_t delay);

// Get the minimum delay between emails of the PBMailer 'that'
#if BUILDMODE != 0
inline
#endif
time_t PBMailerGetDelayBetweenEmails(PBMailer* const that);

// ===== Inliner =====

#if BUILDMODE != 0
#include "republish-inline.c"
#endif

#endif

```

## 3 Code

### 3.1 republish.c

```
// ===== RESPUBLISH.C =====

// ===== Include =====

#include "republish.h"
#if BUILDMODE == 0
#include "republish-inline.c"
#endif

// ===== Functions implementation =====

// Create a new TextOMeter of 'width' columns and 'height' lines and
// 'title' as title of the attached xterm
// May return NULL if the creation of the Xterm failed
TextOMeter* TextOMeterCreate(char* const title,
    const int width, const int height) {
    #if BUILDMODE == 0
        if (title == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'title' is null");
            PBErrCatch(ResPublishErr);
        }
        if (width <= 0) {
            ResPublishErr->_type = PBErrTypeInvalidArg;
            sprintf(ResPublishErr->_msg, "'width' is invalid (0<%d)", width);
            PBErrCatch(ResPublishErr);
        }
        if (height <= 0) {
            ResPublishErr->_type = PBErrTypeInvalidArg;
            sprintf(ResPublishErr->_msg, "'height' is invalid (0<%d)", height);
            PBErrCatch(ResPublishErr);
        }
    }
    #endif
    // Declare the new TextOMeter
    TextOMeter* that = NULL;
    // Ensure the temporary file to get the tty and pid doesn't exists
    remove(TEXTOMETER_TTY_FILENAME);
    // Fork to create the Xterm
    pid_t pid = 0;
    if ((pid = fork()) == 0) {
        // Create the Xterm
        char cmd[] = TEXTOMETER_XTERM_CMD;
        char geometry[100];
        sprintf(geometry, "%dx%d", width, height);
        char* argv[] = {
            "xterm",
            "-xrm",
            "'XTerm.vt100.allowTitleOps: false'",
            "-T",
            title,
            "-geometry",
            geometry,
            "-e",
            "(echo $$ && tty) > " TEXTOMETER_TTY_FILENAME " ; bash",
            NULL
        };
        if (execv(cmd, argv) == -1) {
```

```

        fprintf(stderr, "TextOMeter '%s' couldn't create the Xterm, "
            "execv failed (%d)\n", title, errno);
    }
} else {
    // Wait for the tty and pid from the Xterm
    FILE *fp = NULL;
    sleep(1);
    int wait = 10000;
    do {
        fp = fopen(TEXTOMETER_TTY_FILENAME, "r");
        wait--;
    } while (fp == NULL && wait > 0);
    if (fp == NULL) {
        fprintf(stderr,
            "TextOMeter '%s' couldn't read the tty and pid\n", title);
    } else {
        // Read the tty and pid from the Xterm
        char tty[100];
        if (fscanf(fp, "%d %s\n", &pid, tty) == EOF) {
            fprintf(stderr,
                "TextOMeter '%s' couldn't read the tty and pid\n", title);
        } else {
            // Allocate memory for the new TextOMeter
            that = PBErrMalloc(ResPublishErr, sizeof(TextOMeter));
            // Set properties
            that->_width = width;
            that->_height = height;
            that->_title = strdup(title);
            that->_pid = pid;
            strcpy(that->_tty, tty);
            // Open the tty to send message to the Xterm
            that->_fp = fopen(that->_tty, "w");
            // Clear the content of the TextOMeter
            TextOMeterClear(that);
            TextOMeterFlush(that);
        }
        // Close the temporary file and delete it
        fclose(fp);
        remove(TEXTOMETER_TTY_FILENAME);
    }
}
// Return the new TextOMeter
return that;
}

// Free the memory used by the TextOMeter 'that'
void TextOMeterFree(TextOMeter** that) {
    // Check argument
    if (that == NULL || *that == NULL)
        // Nothing to do
        return;
    // Close the file pointer to the tty
    if ((*that)->_tty != NULL)
        fclose((*that)->_fp);
    // Kill the terminal
    if (kill((*that)->_pid, SIGTERM) == -1) {
        fprintf(stderr,
            "Couldn't kill the TextOMeter '%s'\n", (*that)->_title);
    }
    // Free memory
    free((*that)->_title);
    free(*that);
}

```

```

    *that = NULL;
}

// ---- EstimTimeToComp

// ===== Functions implementation =====

// Create a new EstimTimeToComp
EstimTimeToComp EstimTimeToCompCreateStatic() {
    // Declare the new EstimTimeToComp
    EstimTimeToComp that;
    // Set properties
    ETCReset(&that);
    that._etc[0] = '\0';
    // Return the new EstimTimeToComp
    return that;
}

// Free the memory used by the EstimTimeToComp 'that'
void EstimTimeToCompFreeStatic(EstimTimeToComp* that) {
    // Nothing to do
    (void)that;
}

// Reset the start time of the EstimTimeToComp 'that' to current time
void ETCReset(EstimTimeToComp* that) {
    #if BUILDMODE == 0
        if (that == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'that' is null");
            PBErrCatch(ResPublishErr);
        }
    #endif
    // Reset the start time to the current time
    that->_start = time(NULL);
}

// Estimate the ETC of the EstimTimeToComp 'that' given the percentage
// of completion 'comp'
// time(0) is expected to returned Thu Jan  1 00:00:00 1970
const char* ETCGet(EstimTimeToComp* const that, float comp) {
    #if BUILDMODE == 0
        if (that == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'that' is null");
            PBErrCatch(ResPublishErr);
        }
    #endif
    // Get the current time
    time_t cur = time(NULL);
    // If the percentage of completion is valid
    if (comp > 0.0 && comp <= 1.0) {
        // Calculate the estimated time to completion and store the result
        // in a string format
        time_t elapsed = cur - that->_start;
        time_t remain = (time_t)((float)elapsed / comp) - elapsed;
        struct tm* rtm = gmtime(&remain);
        sprintf(that->_etc, "%03dd:%02dh:%02dm:%02ds",
            (rtm->tm_year - 70) * 365 + rtm->tm_mon * 30 + rtm->tm_mday - 1,
            rtm->tm_hour, rtm->tm_min, rtm->tm_sec);
    } else {
        sprintf(that->_etc, "???d:??h:??m:??s");
    }
}

```



```

    }
    // Return the etc
    return that->_etc;
}

// ---- PBMailer

// ===== Functions implementation =====

// Create a new PBMailer toward the email adress 'to'
// _delayBetweenEmails is initialiwed to 60s
PBMailer PBMailerCreateStatic(const char* const to) {
#if BUILDMODE == 0
    if (to == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'to' is null");
        PBErrCatch(ResPublishErr);
    }
#endif
    // Declare the new PBMailer
    PBMailer that;
    // Set properties
    that._to = strdup(to);
    that._messages = GSetStrCreateStatic();
    that._delayBetweenEmails = 60;
    that._lastEmailTime = 0;
    // Return the new PBMailer
    return that;
}

// Free the memory used by the PBMailer 'that'
// Flush the remaining strings if any
void PBMailerFreeStatic(PBMailer* that) {
#if BUILDMODE == 0
    if (that == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'that' is null");
        PBErrCatch(ResPublishErr);
    }
#endif
    // Flush the remaing messages
    that->_delayBetweenEmails = 0;
    PBMailerSend(that, "PBMailerFreeStatic flushing remaining messages");
    // Free memory
    free(that->_to);
}

// Send the strings of the PBMailer 'that' if the last PBMailerSend
// call is at least _delayBetweenEmails seconds old, else do nothing
// with the subject 'subject'
// Uses the 'mail' command which is supposed to configure, up and
// running by the user
void PBMailerSend(PBMailer* const that, const char* const subject) {
#if BUILDMODE == 0
    if (that == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'that' is null");
        PBErrCatch(ResPublishErr);
    }
    if (subject == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'subject' is null");
    }
#endif
}

```

```

        PBErrCatch(ResPublishErr);
    }
#endif
    // Get the current time
    time_t curTime = time(NULL);
    // If the delay since the last email is above the threshold and there
    // are messages
    if (curTime - that->_lastEmailTime > that->_delayBetweenEmails &&
        GSetNbElem(&(that->_messages)) > 0) {
        // Calculate the length of the body
        int bodyLength = 0;
        GSetIterForward iter =
            GSetIterForwardCreateStatic(&(that->_messages));
        do {
            char* str = GSetIterGet(&iter);
            bodyLength += strlen(str);
        } while (GSetIterStep(&iter));
        // Create the body of the email
        char* body = malloc(bodyLength + 1);
        int insertPos = 0;
        while (GSetNbElem(&(that->_messages)) > 0) {
            char* str = GSetPop(&(that->_messages));
            sprintf(body + insertPos, "%s", str);
            insertPos += strlen(str);
            free(str);
        }
        // Save the body to a temporary file
        FILE* fp = fopen("./pbmailer.temp", "w");
        if (fp != NULL) {
            fprintf(fp, "%s", body);
            fclose(fp);
            // Create the command to send the email
            char* cmd = malloc(strlen(that->_to) + strlen(subject) + 50);
            sprintf(cmd, "mail -s \"%s\" %s < ./pbmailer.temp", subject,
                that->_to);
            // Send the email
            int ret = system(cmd);
            // Erase the temporary file
            ret = system("rm ./pbmailer.temp");
            // TODO should process the returned value
            (void)ret;
            // Free memory
            free(cmd);
        }
        // Update the last email time
        that->_lastEmailTime = time(NULL);
        // Free memory
        free(body);
    }
}

// Add a copy of the string 'str' to the PBMailer 'that' to be sent
// later with PBMailerSend()
void PBMailerAddStr(PBMailer* const that, const char* const str) {
    #if BUILDMODE == 0
        if (that == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'that' is null");
            PBErrCatch(ResPublishErr);
        }
    #endif
    // If the string is null do nothing

```

```

    if (str == NULL || strlen(str) == 0)
        return;
    // Add a copy of the string at the tail of the set of string
    // to send
    GSetAppend(&(that->_messages), strdup(str));
}

```

## 3.2 republish-inline.c

```

// ===== REPUBLISH-INLINE.C =====

// ---- TextOMeter

// ===== Functions implementation =====

// Clear the content of the TextOMeter 'that'
// Return true if the content could be cleared, false else
#if BUILDMODE != 0
inline
#endif
bool TextOMeterClear(TextOMeter* const that) {
    if BUILDMODE == 0
        if (that == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'that' is null");
            PBErrCatch(ResPublishErr);
        }
    #endif
    // Print on the tty as many line as the height of the Xterm
    for (int i = that->_height; i--;) {
        if (fprintf(that->_fp, "\n") < 0)
            return false;
        fflush(that->_fp);
    }
    return true;
}

// Print the string 'str' on the TextOMeter 'that'
// Return true if the content could be printed, false else
#if BUILDMODE != 0
inline
#endif
bool TextOMeterPrint(TextOMeter* const that, const char* const str) {
    if BUILDMODE == 0
        if (that == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'that' is null");
            PBErrCatch(ResPublishErr);
        }
        if (str == NULL) {
            ResPublishErr->_type = PBErrTypeNullPointer;
            sprintf(ResPublishErr->_msg, "'str' is null");
            PBErrCatch(ResPublishErr);
        }
    #endif
    // Print the string on the tty
    if (fprintf(that->_fp, "%s", str) < 0)
        return false;
    return true;
}

```

```

}

// Flush the stream of the TextOMeter 'that'
#if BUILDMODE != 0
inline
#endif
void TextOMeterFlush(TextOMeter* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'that' is null");
        PBErrCatch(ResPublishErr);
    }
#endif
    fflush(that->_fp);
}

// ---- PBMailer

// Set the minimum delay between emails of the PBMailer 'that' to 'delay'
#if BUILDMODE != 0
inline
#endif
void PBMailerSetDelayBetweenEmails(PBMailer* const that,
    const time_t delay) {
#if BUILDMODE == 0
    if (that == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'that' is null");
        PBErrCatch(ResPublishErr);
    }
#endif
    that->_delayBetweenEmails = delay;
}

// Get the minimum delay between emails of the PBMailer 'that'
#if BUILDMODE != 0
inline
#endif
time_t PBMailerGetDelayBetweenEmails(PBMailer* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        ResPublishErr->_type = PBErrTypeNullPointer;
        sprintf(ResPublishErr->_msg, "'that' is null");
        PBErrCatch(ResPublishErr);
    }
#endif
    return that->_delayBetweenEmails;
}

```

## 4 Makefile

```

# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0

```

```

all: pbmake_wget main xterm

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=respublish
$(repo)_EXENAME: \
$(repo)_EXENAME.o \
$(repo)_EXE_DEP \
$(repo)_DEP
$(COMPILER) 'echo "$$(repo)_EXE_DEP $$(repo)_EXENAME.o" | tr ' ' '\n' | sort -u' $(LINK_ARG) $$(repo)_LINK_ARG

$(repo)_EXENAME.o: \
$(repo)_DIR/$$(repo)_EXENAME.c \
$(repo)_INC_H_EXE \
$(repo)_EXE_DEP
$(COMPILER) $(BUILD_ARG) $$(repo)_BUILD_ARG 'echo "$$(repo)_INC_DIR" | tr ' ' '\n' | sort -u' -c $$(repo)_DIR)/

xterm: xterm.c
gcc -o xterm xterm.c

```

## 5 Unit tests

```

#include <stdlib.h>
#include <stdio.h>
#include "respublish.h"

void UnitTestTextOMeter() {
    TextOMeter* meterA = TextOMeterCreate("UnitTestA", 40, 20);
    TextOMeter* meterB = TextOMeterCreate("UnitTestB", 40, 20);
    if (meterA == NULL || meterB == NULL) {
        ResPublishErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ResPublishErr->_msg, "TextOMeterCreate NOK");
        PBErrCatch(ResPublishErr);
    }
    if (!TextOMeterClear(meterA) || !TextOMeterClear(meterB)) {
        ResPublishErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ResPublishErr->_msg, "TextOMeterClear NOK");
        PBErrCatch(ResPublishErr);
    }
    sleep(2);
    if (!TextOMeterPrint(meterA, "Message from UnitTestTextOMeterA") ||
        !TextOMeterPrint(meterB, "Message from UnitTestTextOMeterB")) {
        ResPublishErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ResPublishErr->_msg, "TextOMeterClear NOK");
        PBErrCatch(ResPublishErr);
    }
    TextOMeterFlush(meterA);
    TextOMeterFlush(meterB);
    sleep(2);
    TextOMeterFree(&meterA);
    TextOMeterFree(&meterB);
    if (meterA != NULL || meterB != NULL) {

```

```

    ResPublishErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ResPublishErr->_msg, "TextOMeterFree NOK");
    PBErrCatch(ResPublishErr);
}
printf("UnitTestTextOMeter OK\n");
}

void UnitTestEstimTimeToComp() {
    EstimTimeToComp etc = EstimTimeToCompCreateStatic();
    char* checkA[5] = {
        "???d:??h:??m:??s",
        "000d:00h:00m:03s",
        "000d:00h:00m:02s",
        "000d:00h:00m:01s",
        "000d:00h:00m:00s"
    };
    for (int i = 0; i < 5; ++i) {
        printf("%s\n", ETCGet(&etc, (float)i / 5.0));
        sleep(1);
        if (strcmp(etc._etc, checkA[i]) != 0) {
            ResPublishErr->_type = PBErrTypeUnitTestFailed;
            sprintf(ResPublishErr->_msg, "ETCGet NOK");
            PBErrCatch(ResPublishErr);
        }
    }
    ETCReset(&etc);
    char* checkB[5] = {
        "???d:??h:??m:??s",
        "576d:16h:53m:19s",
        "286d:08h:26m:38s",
        "191d:21h:37m:43s",
        "144d:16h:13m:16s"
    };
    for (int i = 0; i < 5; ++i) {
        printf("%s\n", ETCGet(&etc, (float)(i * i) / 50000000.0));
        sleep(1);
        if (strcmp(etc._etc, checkB[i]) != 0) {
            ResPublishErr->_type = PBErrTypeUnitTestFailed;
            sprintf(ResPublishErr->_msg, "ETCGet NOK");
            PBErrCatch(ResPublishErr);
        }
    }
    printf("UnitTestEstimTimeToComp OK\n");
}

void UnitTestPBMailer() {
    char* email = "Your@Email.net";
    PBMailer mailer = PBMailerCreateStatic(email);
    char* lineA = "UnitTestPBMailer, line A";
    char* lineB = "UnitTestPBMailer, line B";
    char* lineC = "UnitTestPBMailer, line C";
    PBMailerSend(&mailer, "");
    PBMailerAddStr(&mailer, lineA);
    PBMailerAddStr(&mailer, lineB);
    PBMailerSend(&mailer, "UnitTestPBMailer, subject 1");
    PBMailerAddStr(&mailer, lineC);
    PBMailerSend(&mailer, "UnitTestPBMailer, subject 1");
    PBMailerSetDelayBetweenEmails(&mailer, 10);
    if (mailer._delayBetweenEmails != 10) {
        ResPublishErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ResPublishErr->_msg, "PBMailerSetDelayBetweenEmails NOK");
        PBErrCatch(ResPublishErr);
    }
}

```

```

    }
    if (PBMailerGetDelayBetweenEmails(&mailer) != 10) {
        ResPublishErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ResPublishErr->_msg, "PBMailerGetDelayBetweenEmails NOK");
        PBErrCatch(ResPublishErr);
    }
    sleep(11);
    PBMailerSend(&mailer, "UnitTestPBMailer, subject 2");
    PBMailerAddStr(&mailer, lineA);
    PBMailerFreeStatic(&mailer);

    // Emails reveived:
    /*
    UnitTestPBMailer, subject 1
    > UnitTestPBMailer, line A
    > UnitTestPBMailer, line B

    UnitTestPBMailer, subject 2
    > UnitTestPBMailer, line C

    PBMailerFreeStatic flushing remaining messages
    > UnitTestPBMailer, line A
    */

    printf("UnitTestPBMailer OK\n");
}

void UnitTestAll() {
    UnitTestTextOMeter();
    UnitTestEstimTimeToComp();
    UnitTestPBMailer();
    printf("UnitTestAll OK\n");
}

int main() {
    //UnitTestAll();
    UnitTestPBMailer();
    // Return success code
    return 0;
}

```

## 6 Unit tests output

```

UnitTestTextOMeter OK
???d:??h:??m:??s
000d:00h:00m:03s
000d:00h:00m:02s
000d:00h:00m:01s
000d:00h:00m:00s
???d:??h:??m:??s
576d:16h:53m:19s
286d:08h:26m:38s
191d:21h:37m:43s
144d:16h:13m:16s
UnitTestEstimTimeToComp OK
UnitTestAll OK

```