LAMPIRAN A

Listing* Program *Software Primary Controller

```
//---------------------------------------------------------------------------
#include <stdlib.h>
#include <time.h>
#include <fstream.h>
#include <vcl.h>
#pragma hdrstop

#include "code_frmMain.h"
#include "code_frmConfigure.h"
#include "code_frmProgress.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

/*
[DEBUGGING PURPOSE]
#define DATA_PORT 0x378
#define STATUS_PORT 0x379
#define CONTROL_PORT 0x37A
*/

TfrmMain *frmMain;

/*
  Prototype untuk fungsi yang ada di inpout32.dll
  ------------------------------------------------------------
*/
typedef short __stdcall (*InputDll) (short PortAddress);
typedef void __stdcall (*OutputDll) (short PortAddress, short data);
//---------------------------------------------------------------

// ### Variables ###
HINSTANCE dll;
short port;
OutputDll Out32;
InputDll Inp32;
bool port_open = false;
bool general_usage = false;
bool hardware_connected = false;

//System Handler
String command;
bool boot = true;
int log_scroll_y = 0;
int sys_errors;
String sender_number;
String sender_message;
int instruction_no = 0; // none
int command_level = 1; // 1 - instruction, 2 - parameter
bool confirmation = false; // variable for CommandHandler 2nd parameter
int option;  // variable for CommandHandler parameter value
bool listed = false; // 'List file' command execution result
int available_file_option_max;

//Download Subsystem
int download_mode = 0; // default, 0=no-mode-running 1=single-download-mode
2=list-download-mode
HANDLE wget_hwnd;
String downloaded_file;
String URL;
char ** download_file_list;
int download_list_count;
int downloading_index = -1; // default -1

//Email Subsystem
bool attach_file = true;
String filename_2attach;

//Shortcuts Subsystem
bool enumerated = false;
```

```
char ** shortcuts; // file path
char ** files; // file title / filename
int total_shortcuts;
String selected_shortcut;

//---------------------------------------------------------------------------
__fastcall TfrmMain::TfrmMain(TComponent* Owner)
   : TForm(Owner)
{
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
// ===========================================================================
//                             COMPUTER I/O PORT
// ===========================================================================

bool TfrmMain::OpenPort()
{
   //Open DB25 parallel I/O port connection
   bool result = true;

   dll = LoadLibrary("inpout32.dll"); // Win32 API, load external library / DLL
      if(dll == NULL) {
         MessageDlg( "inpout32.dll not found, error.",mtError,TMsgDlgButtons() <<
mbOK,0);
         result = false;
      }
   Inp32 = (InputDll) GetProcAddress(dll,"Inp32");
      if(Inp32 == NULL) {
         MessageDlg( "Function could'nt loaded!, ERROR.",mtError,TMsgDlgButtons()
<< mbOK,0);
         result = false;
      }
   Out32 = (OutputDll) GetProcAddress(dll,"Out32");
      if(Out32 == NULL) {
         MessageDlg( "Function could'nt loaded!, ERROR.",mtError,TMsgDlgButtons()
<< mbOK,0);
         result = false;
      }

   if(result == true) {
      port_open = true;
      SetPortAsOutput();
   } else {
      port_open = false;
   }

   return result;
}
//---------------------------------------------------------------------------

bool TfrmMain::ClosePort()
{
   //Close DB25 parallel I/O port connection
   bool result;
   result = FreeLibrary(dll);

   if(result == false) {
      MessageDlg( "Couldn't close port, ERROR.",mtError,TMsgDlgButtons() <<
mbOK,0);
   } else {

   }

   return result;
}

void TfrmMain::SetPortAsOutput()
{
   //Set DB25 Parallel data port to be output
```

```
   // clear bit C5 of control register

   // clear bit nC3 of control register:
   //     r:1 -> turn-off 74LS373(IN) [DIRECT]
   //     r:0 -> turn-on 74LS373(OUT) [through inverter]
   unsigned char temp;

   temp = Inp32(CONTROL_PORT);

   temp = temp & 0xD7 ; // 1101.0111

   Out32(CONTROL_PORT, temp);
}

void TfrmMain::SetPortAsInput()
{
   //Set DB25 Parallel data port to be input
   // set bit C5 of control register

   // set bit nC3 of control register:
   //     r:1 -> turn-off 74LS373(OUT) [through inverter]
   //     r:0 -> turn-on 74LS373(IN) [DIRECT]
   unsigned char temp;

   temp = Inp32(CONTROL_PORT);

   temp = temp | 0x28 ; // 0010.1000

   Out32(CONTROL_PORT, temp);
}

//---------------------------------------------------------------------------
// ===========================================================================
//                         LOW LEVEL HARDWARE FUNCTIONS
// ===========================================================================

/*
   Data port - Communication lines / Communication port

   Control bit 0 - Clock
   Control bit 1 - Exit-loop function (not used thought)

   Status bit 4 - Busy flag
*/

void TfrmMain::HWClock() {
   // Give hardware a transition from High->Low->High (Falling+Rising Edge)
   // CONTROL PORT, BIT 0

   unsigned char tmp;

   Sleep(20);

   // Clear LOW
   tmp = Inp32(CONTROL_PORT);
   tmp = tmp & (0xFE); // 0b1111.1110
   Out32(CONTROL_PORT, tmp);

   Sleep(20);

   // Set HIGH
   tmp = Inp32(CONTROL_PORT);
   tmp = tmp | 0x01; // 0b0000.0001
   Out32(CONTROL_PORT, tmp);

   Sleep(20);

   // Clear LOW
   tmp = Inp32(CONTROL_PORT);
   tmp = tmp & (0xFE); // 0b1111.1110
```

```
   Out32(CONTROL_PORT, tmp);

   Sleep(20);
}

void TfrmMain::HWWaitBusy()
{
   unsigned char tmp;
   bool busy = true;
   // Wait for hardware signal "busy" until it goes low
   while(busy == true) {
      tmp = Inp32(STATUS_PORT); // S4
      tmp = tmp & 0x10; // 0b0001.0000
      if(tmp != 0x10) {
         busy = false;
      }
      Sleep(1);
      frmMain->Update();
   }
}

void TfrmMain::ToggleHeartbeat()
{
   // Send out different signal
   // CONTROL PORT, C2

   unsigned int tmp;

   tmp = Inp32(CONTROL_PORT);
   tmp = tmp & 0x04; // 0b0000.0100
   if(tmp == 0x04) {
      // Clearing C2
      tmp = Inp32(CONTROL_PORT);
      tmp = tmp & (!0x04); // 0b1111.1011
      Out32(CONTROL_PORT, tmp);
   } else {
      // Setting C2
      tmp = Inp32(CONTROL_PORT);
      tmp = tmp | 0x04; // 0b0000.0100
      Out32(CONTROL_PORT, tmp);
   }

}

// ============================================================================
//                         HARDWARE "SOFTWARE" FUNCTIONS
// ============================================================================

void TfrmMain::HWClearStorages()
{
   //Hardware clear recepient number phone and message
   unsigned char inst_code = 0x01;

   // > set instruction code
   Out32(DATA_PORT,inst_code);

   // > give clock, to confirm instruction code
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}
```

```
void TfrmMain::HWInputAscii(char ascii)
{
   //Computer input a message, char by char to the hardware
   unsigned char inst_code = 0x02;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > set parameter data
   Out32(DATA_PORT,ascii);
   // > give clock, to confirm parameter data, ASCII
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}

void TfrmMain::HWInputNumber(char number)
{
   //Computer input recepient number, digit by digit to the hardware
   unsigned char inst_code = 0x03;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > set parameter data
   Out32(DATA_PORT,number);
   // > give clock, to confirm parameter data, Number ASCII
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}

void TfrmMain::HWEncodeMessage(unsigned char mode)
{
   //Tell hardware to encode the message.
   unsigned char inst_code = 0x04;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > set parameter data, PDU_Mode : 0x01 -- 7-bit | 0x02 -- 8-bit | 0x03 -- 16-
bit
   Out32(DATA_PORT, mode );
   // > give clock, to confirm parameter data
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
```

```
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}

void TfrmMain::HWDecodeMessage()
{
   //Tell hardware to fetch message from mobile phone, and decode it.
   unsigned char inst_code = 0x05;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();

}

void TfrmMain::HWSendMessage()
{
   //Send the encoded message to mobile phone
   unsigned char inst_code = 0x06;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}

void TfrmMain::HWReadMessage()
{
   //Read the decoded message from hardware

   char tmp;
   unsigned char inst_code = 0x07;

   String number;
   String message;

   unsigned char data;
   unsigned char num_length = 0;
   unsigned char t_num;
   unsigned char msg_length = 0;
   unsigned char t_msg;

   number = "";
   message = "";

   SBar->Position = 0;
   SBar->Max = 0;

   // > set instruction code
```

```
Out32(DATA_PORT,inst_code);
// > give clock, to confirm instruction code
HWClock();

SetPortAsInput();

// > give clock, to confirm execution
HWClock();

//============================== receving begin here ======

num_length = Inp32(DATA_PORT);
t_num = num_length;
SBar->Max = (t_num * 2);
Sleep(10);
// > give clock, confirm data, number_length
HWClock();

while(num_length > 0) {

   // > get data from port!
   data = Inp32(DATA_PORT);

   // accumulate to number variable
   number = number + String(char(data));

   HWClock();
   num_length--;

   frmMain->Update();

   SBar->Position = SBar->Position + 1;

}

msg_length = Inp32(DATA_PORT);
t_msg = msg_length;
SBar->Max = (t_msg * 2);
SBar->Position = (SBar->Max/2);
Sleep(10);
// > give clock, confirm data, message_length
HWClock();

while(msg_length > 0) {

   // > get data from port!
   data = Inp32(DATA_PORT);

   // accumulate to message variable
   message = message + String(char(data));

   HWClock();
   msg_length--;
   frmMain->Update();

   SBar->Position = SBar->Position + 1;
}

//============================== receiving end here =======

// > give clock, to exit
HWClock();

SetPortAsOutput();

SBar->Position = 0;
SBar->Max = 1;

// > set output to variables
```

```
   sender_number = number;
   sender_message = message;

}

void TfrmMain::HWDeleteLastMessage()
{
   //Tell hardware to delete last received message in mobile phone
   unsigned char inst_code = 0x08;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}

char TfrmMain::HWGetMessage()
{
   //Get message from hardware by this instruction

   char msg;
   unsigned char inst_code = 0x09;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
   HWClock();

   // > Set port to (input-mode)
   SetPortAsInput();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   msg = Inp32(DATA_PORT);

   // > give clock, to confirm data
   HWClock();

   // > give clock, to exit
   HWClock();

   // > set port to (output-mode)
   SetPortAsOutput();

   return msg;
}

void TfrmMain::HWKeepConnection()
{
   //Keep connection alive between M.P.D Translator and the mobile phone
   //This ensure new message indication are sent properly to the M.P.D Translator

   unsigned char inst_code = 0x0B;

   // > set instruction code
   Out32(DATA_PORT,inst_code);
   // > give clock, to confirm instruction code
```

```
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   HWWaitBusy();

   // > give clock, to exit
   HWClock();
}

//---------------------------------------------------------------------------
// ===========================================================================
//                            MAIN PROGRAM HANDLER
// ===========================================================================
//---------------------------------------------------------------------------

void TfrmMain::LoadConfig()
{
   // Load user configuration files

   String file_config = ExtractFilePath(Application->ExeName) + "\\" +
"config.ini";

   // Check for config file? exist or not?
   // If exist then load it, else leave configuration at default values then save.
   if(FileExist(file_config) == false) {
      // File Not exist
      MessageBox(Handle,"User configuration not found!\nUsing default
settings\n(Please configure it now!)","Warning",MB_OK|MB_ICONEXCLAMATION);
      DefaultConfig();
      SaveConfig(); // save it for next settings

   } else {
      // File exist, load it up..

      char * retStr;
      retStr = (char*) calloc( 201, sizeof(char));
      memset(retStr,0,201);

      // General
      GetPrivateProfileString(
"General","Verbose","0",retStr,2,file_config.c_str());
      frmConfigure->chkVerbose->Checked = StrToInt(String(retStr));
      GetPrivateProfileString(
"General","Autostart","0",retStr,2,file_config.c_str());
      frmConfigure->chkAutostart->Checked = StrToInt(String(retStr));

      // Log
      GetPrivateProfileString( "Log","Cleaner","",retStr,2,file_config.c_str());
      frmConfigure->chkClearLog->Checked = StrToInt(String(retStr));
      GetPrivateProfileString( "Log","Counter","",retStr,20,file_config.c_str());
      frmConfigure->txtClearCount->Text = String(retStr);
      GetPrivateProfileString( "Log","Save","",retStr,2,file_config.c_str());
      frmConfigure->chkSaveLog->Checked = StrToInt(String(retStr));
      GetPrivateProfileString( "Log","Path","",retStr,200,file_config.c_str());
      frmConfigure->txtProgramLog->Text = String(retStr);

      // Hardware
      GetPrivateProfileString( "Hardware","Mode","",retStr,2,file_config.c_str());
      frmConfigure->cmbPort->ItemIndex = StrToInt(String(retStr));
      GetPrivateProfileString(
"Hardware","CustomPort","",retStr,10,file_config.c_str());
      frmConfigure->txtCustomPort->Text = String(retStr);
      frmConfigure->cmbPortChange(frmMain);

      // Downloader
      GetPrivateProfileString(
"Downloader","ProgramPath","",retStr,200,file_config.c_str());
```

```
      frmConfigure->txtDownloaderProgramPath->Text = String(retStr);
      GetPrivateProfileString(
"Downloader","SavePath","",retStr,200,file_config.c_str());
      frmConfigure->txtDownloaderSavePath->Text = String(retStr);
      GetPrivateProfileString(
"Downloader","LogPath","",retStr,200,file_config.c_str());
      frmConfigure->txtDownloaderLogPath->Text = String(retStr);
      GetPrivateProfileString(
"Downloader","ListPath","",retStr,200,file_config.c_str());
      frmConfigure->txtDownloadListPath->Text = String(retStr);

      // Email
      GetPrivateProfileString(
"Email","ProgramPath","",retStr,200,file_config.c_str());
      frmConfigure->txtEmailProgramPath->Text = String(retStr);

      // We don't read any sensitive information, the one who responsible for this
kind of information is emailer program (blat)
      //GetPrivateProfileString(
"Email","Username","",retStr,200,file_config.c_str());
      //frmConfigure->txtUsername->Text = String(retStr);
      //GetPrivateProfileString(
"Email","Password","",retStr,200,file_config.c_str());
      //frmConfigure->txtPassword->Text = String(retStr);

      GetPrivateProfileString(
"Email","SMTPServer","",retStr,200,file_config.c_str());
      frmConfigure->txtSMTPServer->Text = String(retStr);
      GetPrivateProfileString(
"Email","SSLPort","",retStr,20,file_config.c_str());
      frmConfigure->txtSSLPort->Text = String(retStr);
      GetPrivateProfileString(
"Email","SenderAddress","",retStr,200,file_config.c_str());
      frmConfigure->txtSenderAddress->Text = String(retStr);
      GetPrivateProfileString(
"Email","RecepientAddress","",retStr,200,file_config.c_str());
      frmConfigure->txtRecepientAddress->Text = String(retStr);
      GetPrivateProfileString(
"Email","MailSubject","",retStr,200,file_config.c_str());
      frmConfigure->txtMailSubject->Text = String(retStr);

      // Misc
      GetPrivateProfileString(
"Misc","AuthorizedNumber","",retStr,14,file_config.c_str());
      frmConfigure->txtAuthorizedNumber->Text = String(retStr);
      GetPrivateProfileString(
"Misc","PeriodicReport","",retStr,20,file_config.c_str());
      frmConfigure->chkPeriodicReport->Checked = StrToInt(String(retStr));
      GetPrivateProfileString(
"Misc","ReportInterval","",retStr,20,file_config.c_str());
      frmConfigure->txtReportInterval->Text = String(retStr);
      GetPrivateProfileString(
"Misc","ShortcutsDirectory","",retStr,200,file_config.c_str());
      frmConfigure->txtShortcutsDirPath->Text = String(retStr);

   }
}

void TfrmMain::DefaultConfig()
{
   // Revert all settings back into default values.
      // General
      frmConfigure->chkVerbose->Checked = false;
      frmConfigure->chkAutostart->Checked = true;

      // Log
      frmConfigure->chkClearLog->Checked = true;
      frmConfigure->txtClearCount->Text = "50";
      frmConfigure->chkSaveLog->Checked = false;
```

```
      frmConfigure->txtProgramLog->Text = ExtractFilePath(Application->ExeName) +
"logs\\log_program.txt";

      // Hardware
      frmConfigure->cmbPort->ItemIndex = 0;
      frmConfigure->txtCustomPort->Text = "";
      frmConfigure->cmbPortChange(frmMain);

      // Downloader
      frmConfigure->txtDownloaderProgramPath->Text = "";
      frmConfigure->txtDownloaderSavePath->Text = ExtractFilePath(Application-
>ExeName);
      frmConfigure->txtDownloaderLogPath->Text = ExtractFilePath(Application-
>ExeName) + "logs\\log_download.txt";
      frmConfigure->txtDownloadListPath->Text = "";

      // Email
      frmConfigure->txtEmailProgramPath->Text = "";
      frmConfigure->txtUsername->Text = "";
      frmConfigure->txtPassword->Text = "";
      frmConfigure->txtSMTPServer->Text = "";
      frmConfigure->txtSSLPort->Text = "";
      frmConfigure->txtSenderAddress->Text = "";
      frmConfigure->txtRecepientAddress->Text = "";
      frmConfigure->txtMailSubject->Text = "";

      // Security
      frmConfigure->txtAuthorizedNumber->Text = "";
      frmConfigure->chkPeriodicReport->Checked = true;
      frmConfigure->txtReportInterval->Text = "15";
      frmConfigure->txtShortcutsDirPath->Text = "";
}

void TfrmMain::SaveConfig()
{
   // Save user configuration

   String file_config = ExtractFilePath(Application->ExeName) + "\\" +
"config.ini";

   // Delete last configuration file
   DeleteFile(file_config);

   // General
   WritePrivateProfileString( "General","Verbose",String((int)frmConfigure-
>chkVerbose->Checked).c_str(),file_config.c_str());
   WritePrivateProfileString( "General","Autostart",String((int)frmConfigure-
>chkAutostart->Checked).c_str(),file_config.c_str());

   // Log
   WritePrivateProfileString( "Log","Cleaner",String((int)frmConfigure-
>chkClearLog->Checked).c_str(),file_config.c_str());
   WritePrivateProfileString( "Log","Counter",String(frmConfigure->txtClearCount-
>Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Log","Save",String((int)frmConfigure->chkSaveLog-
>Checked).c_str(),file_config.c_str());
   WritePrivateProfileString( "Log","Path",String(frmConfigure->txtProgramLog-
>Text).c_str(),file_config.c_str());

   // Hardware
   WritePrivateProfileString( "Hardware","Mode",String(frmConfigure->cmbPort-
>ItemIndex).c_str(),file_config.c_str());
   WritePrivateProfileString( "Hardware","CustomPort",String(frmConfigure-
>txtCustomPort->Text).c_str(),file_config.c_str());

   // Downloader
   WritePrivateProfileString( "Downloader","ProgramPath",String(frmConfigure-
>txtDownloaderProgramPath->Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Downloader","SavePath",String(frmConfigure-
>txtDownloaderSavePath->Text).c_str(),file_config.c_str());
```

```
   WritePrivateProfileString( "Downloader","LogPath",String(frmConfigure-
>txtDownloaderLogPath->Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Downloader","ListPath",String(frmConfigure-
>txtDownloadListPath->Text).c_str(),file_config.c_str());

   // Email
   WritePrivateProfileString( "Email","ProgramPath",String(frmConfigure-
>txtEmailProgramPath->Text).c_str(),file_config.c_str());
   // We don't store any sensitive information, the one who responsible for this
kind of information is emailer program (blat)
   //WritePrivateProfileString( "Email","Username",String(frmConfigure-
>txtUsername->Text).c_str(),file_config.c_str());
   //WritePrivateProfileString( "Email","Password",String(frmConfigure-
>txtPassword->Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Email","SMTPServer",String(frmConfigure-
>txtSMTPServer->Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Email","SSLPort",String(frmConfigure->txtSSLPort-
>Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Email","SenderAddress",String(frmConfigure-
>txtSenderAddress->Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Email","RecepientAddress",String(frmConfigure-
>txtRecepientAddress->Text).c_str(),file_config.c_str());
   WritePrivateProfileString( "Email","MailSubject",String(frmConfigure-
>txtMailSubject->Text).c_str(),file_config.c_str());

   // Misc
   WritePrivateProfileString("Misc","AuthorizedNumber",String(frmConfigure-
>txtAuthorizedNumber->Text).c_str(),file_config.c_str());
   WritePrivateProfileString("Misc","PeriodicReport",String((int)frmConfigure-
>chkPeriodicReport->Checked).c_str(),file_config.c_str());
   WritePrivateProfileString("Misc","ReportInterval",String(frmConfigure-
>txtReportInterval->Text).c_str(),file_config.c_str());
   WritePrivateProfileString("Misc","ShortcutsDirectory",String(frmConfigure-
>txtShortcutsDirPath->Text).c_str(),file_config.c_str());

   // -- Outside data processes --

      // Auto start at Windows Logon.
      // save a key to registry HKCU\Software\Microsoft\Windows\CurrentVersion\Run
      if(frmConfigure->chkAutostart->Checked == true) {
         // Create registry entry
         void * hKey;
         String path = Application->ExeName;

if(RegOpenKeyEx(HKEY_CURRENT_USER,"Software\\Microsoft\\Windows\\CurrentVersion\\R
un",0,KEY_ALL_ACCESS,&hKey) == ERROR_SUCCESS) {
            if(RegSetValueEx( hKey, "CMS_SMS_SYS", 0, REG_SZ, path.c_str(),
path.Length()) != ERROR_SUCCESS) {
               // Add report
               mmLog->Lines->Add("(!) Couldn't open registry, Autostart
configuration feature error!");
               log_scroll_y++;
            }
            RegCloseKey(hKey);
         } else {
            // Add report
            mmLog->Lines->Add("(!) Couldn't open registry, Autostart configuration
feature error!");
            log_scroll_y++;
         }
      } else {
         // Delete registry entry
         void * hKey;
         String path = Application->ExeName;

if(RegOpenKeyEx(HKEY_CURRENT_USER,"Software\\Microsoft\\Windows\\CurrentVersion\\R
un",0,KEY_ALL_ACCESS,&hKey) == ERROR_SUCCESS) {
            RegDeleteValue(hKey,"CMS_SMS_SYS");
            RegCloseKey(hKey);
         } else {
```

```
            // Add report
            mmLog->Lines->Add("(!) Couldn't open registry, Autostart configuration
feature error!");
            log_scroll_y++;
         }
      }

}

void __fastcall TfrmMain::FormActivate(TObject *Sender)
{
   if(boot == true) {

      // Initialize variables and startup value for each controls
         mmLog->Clear(); // clear the log memobox

         lblDownloaderStatus->Caption = "Inactive";
         PBar->Position = 0; // zero
         lblPercent->Caption = "-";
         lblURL->Caption = "-";

         lblCurrentTask->Caption = "-";
         lblCompUptime->Caption = "--:--:--";
         lblSysErrors->Caption = "0x";
         btnAutoResponse->Caption = "";
         sys_errors = 0;

         SBar->Position = 0;

         shortcuts = (char**) calloc(3,sizeof(char*));
         files = (char **) calloc(3,sizeof(char*));
         for(int i = 0; i < 3 ; i++) {
            shortcuts[i] = (char *) calloc(260,sizeof(char));
            memset(shortcuts[i],0,260);
            files[i] = (char *) calloc(15,sizeof(char));
            memset(files[i],0,15); // 14 char, 1 for null
         }

      // Load user configurations
      LoadConfig();

      // Wait for timer.. to continue Startup execution
      //      we don't use a loop, because the DoEvents function doesn't process
WM_QUIT / Application->Terminate();

      tmrStartup->Interval = 10000; // 10 sec.
      tmrStartup->Enabled = true;
      mmLog->Lines->Add("(i) Click configure button to configure the software.");
      mmLog->Lines->Add("(i) Click close window button to use computer/'general-
usage'");
      mmLog->Lines->Add("");
      mmLog->Lines->Add("(i) " + IntToStr(tmrStartup->Interval / 1000) + " seconds
for user confirmation ...");

   }
}

void __fastcall TfrmMain::tmrStartupTimer(TObject *Sender)
{
   if(boot == true) {
      mmLog->Clear();
      tmrStartup->Enabled = false;
      Startup();
   }
}

void TfrmMain::Startup()
{
   // WARNING: Beginning code section of this function, moved to FormActivate,
   // this because considering the 'general-usage' plan usage.
```

```
   // Booting up complete so far, don't re-loop again.
      boot = false;

   // Set GUI (CloseButton enable, disable CloseWindowButton)
      TBorderIcons tempBI = frmMain->BorderIcons;
      tempBI >> biSystemMenu;
      frmMain->BorderIcons = tempBI;
      btnConfigure->Enabled = false; // User cannot configure, once the system is
activated!
      btnAutoResponse->Enabled = true;
      btnCloseSoftware->Enabled = true;

   // set this program Always On Top
      //mmLog->Lines->Add("DEBUG: Always On Top DISABLED");
      //log_scroll_y++;
      SetWindowPos(Application-
>Handle,HWND_TOPMOST,0,0,0,0,SWP_NOMOVE|SWP_NOSIZE);
      SetWindowPos(frmMain->Handle,HWND_TOPMOST,0,0,0,0,SWP_NOMOVE|SWP_NOSIZE);

   // Open computer I/O Port
   // get settings first before open any port

      switch(frmConfigure->cmbPort->ItemIndex) {
         case 0:
            DATA_PORT = 0x378;
         break;
         case 1:
            DATA_PORT = 0x278;
         break;
         case 2:
            //Custom Port
            DATA_PORT = StrToIntDef(frmConfigure->txtCustomPort->Text,0x378); //
wrong input, revert to 0x378
         break;
      }

      STATUS_PORT = DATA_PORT + 1;
      CONTROL_PORT = DATA_PORT + 2;
      // open now
      OpenPort();

   // Log entry
      String time_str = String(Time());
      String date_str = String(Date());
      mmLog->Lines->Add("(System Started at " + date_str + " - " + time_str +
")");
      mmLog->Lines->Add("");
      log_scroll_y++;
      log_scroll_y++;
      // Verbose: "[System Started]"
      if(frmConfigure->chkVerbose->Checked == true) {
         SWSendMessage("(System Started)","");
      }

   // Give the 1st heartbeat
      // DEBUG, SKIP
      //mmLog->Lines->Add("DEBUG: Startup heartbeat skipped");
      //log_scroll_y++;

      // Add report
      mmLog->Lines->Add("(>) Sending startup-heartbeat");
      log_scroll_y++;

      tmrHWHeartbeatTimer(frmMain); // 1x
      frmMain->Update();
      Sleep(1000);
      tmrHWHeartbeatTimer(frmMain); // ..^ (part of)
      frmMain->Update();
      Sleep(1000);
```

```
   tmrHWHeartbeatTimer(frmMain); // 2x
   frmMain->Update();
   Sleep(1000);
   tmrHWHeartbeatTimer(frmMain); // ..^ (part of)
   frmMain->Update();
   Sleep(1000);
   tmrHWHeartbeatTimer(frmMain); // 3x
   frmMain->Update();
   Sleep(1000);
   tmrHWHeartbeatTimer(frmMain); // ..^ (part of)
   frmMain->Update();
   Sleep(1000);
   tmrHWHeartbeatTimer(frmMain); // 4x
   frmMain->Update();
   Sleep(1000);
   tmrHWHeartbeatTimer(frmMain); // ..^ (part of)
   frmMain->Update();
   Sleep(1000);


// Check for hardware availabillity
   // DEBUG, SKIP
   //mmLog->Lines->Add("DEBUG: Hardware Check Skipped!");
   //log_scroll_y++;
   //hardware_connected = true;


   SWHWCheck();
   if(hardware_connected == false) {
      // Add report
      mmLog->Lines->Add("(!) Hardware isn't connected!");
      log_scroll_y++;
      CloseSoftware();
   }


// Only proceed, if hardware is connected
if(hardware_connected == true) {

// Enumerate shortcuts first!
   // DEBUG, SKIP
   //mmLog->Lines->Add("DEBUG: Enumeration Process Skipped!");
   //log_scroll_y++;
   EnumerateShortcuts();

// Turn on Self-Response-System
   tmrAutoResponse->Enabled = true;
   if(tmrAutoResponse->Enabled == true) {
      btnAutoResponse->Caption = "Sys Auto Response : [ON]";
      mmLog->Lines->Add("(>) Self Response System activated");
      log_scroll_y++;
      // Normal/Verbose: "(System Online)"
      SWSendMessage("(System Online)","");
   } else {
      btnAutoResponse->Caption = "Sys Auto Response : [OFF]";
      mmLog->Lines->Add("(!) WARNING: Self Response System inactive!");
      log_scroll_y++;
      // Verbose: "(!) System couldn't start, malfunction!"
      if(frmConfigure->chkVerbose->Checked == true) {
         SWSendMessage("(!) System couldn't start, malfunction!","");
      }
   }

// Turn on hardware Heartbeat generator
   tmrHWHeartbeat->Enabled = true;

// Check for periodic-report feature (ON/OFF)
   if(frmConfigure->chkPeriodicReport->Checked == true) {
      // Add report
      mmLog->Lines->Add("(>) Periodic report feature enabled");
```

```
         log_scroll_y++;
         tmrPeriodicReport->Interval = StrToIntDef(frmConfigure-
>txtReportInterval->Text,15) * 60 * 1000; // wrong input, revert to 15 min.
         tmrPeriodicReport->Enabled = true;
      } else {
         mmLog->Lines->Add("(>) Periodic report feature disabled!");
         log_scroll_y++;
         tmrPeriodicReport->Enabled = false;
      }

   }

}

void __fastcall TfrmMain::tmrAutoResponseTimer(TObject *Sender)
{
   // LED Indicator
   LED1->Color = clGreen;
   tmrLED1->Enabled = true;

   // Keep connection alive between M.P.D Translator with Mobile Phone
   HWKeepConnection();

   // Call Process
   AutoResponse();

}

void TfrmMain::AutoResponse()
{
   /*
      SYSTEM AUTO RESPONSE

   1. Checking for a new message by some time intervals
   2. If new message are available, then
      3. Decode it
      4. Read it
      5. Check for it's number first? authorized or not?
         6. If authorized, then
         7. Execute the message by system plans, call CommandHandler
         8. Log message command
      9. Delete that message

   */

   bool unauthorized = false;

   if(HWGetMessage() == 0x01) {
      HWClearStorages(); // keep our system clean.
      HWDecodeMessage();
      HWReadMessage();
      if(sender_number == frmConfigure->txtAuthorizedNumber->Text) {
         // Add a report
         // Verbose: "Message received"
         mmLog->Lines->Add("(i) New message received!");
         log_scroll_y++;

         // Authorized user
         // CommandHandler();
         // ^ this function placed below, because we need complete this read
process first!
         // if we placed it here, when CloseSoftware called, it's gonna be a crash

      } else {
         // UNAUTHORIZED USER!!
         // Add a report
         mmLog->Lines->Add("(!) WARNING: Unauthorized message received!");
         log_scroll_y++;
         unauthorized = true;
```

```
      }
      HWDeleteLastMessage();

      // Clear last data on RAM
      HWClearStorages();

      // Process the command . . .
      if(unauthorized == true) {
         // Normal/Verbose: "WARNING: UNAUTHORIZED NUMBER!"
         SWSendMessage("(!) UNAUTHORIZED NUMBER!!","");
      } else {
         // Authorized user
         CommandHandler();
      }

   }

}

void __fastcall TfrmMain::tmrPeriodicReportTimer(TObject *Sender)
{
   // Periodic report feature
   // check if system is currently sending message, then skip this one.
   mmLog->Lines->Add("(i) -- Periodic report --");
   log_scroll_y++;
   SWSendMessage("(i) Periodic Report","");
}

void TfrmMain::CommandHandler()
{
   // Switch command
   /* HOW TO USE :

      There are 2 level in this Command Handler mechanism.
      1st level is for instruction/command names
      2nd level is for parameters, owned by an instruction that been called on 1st
level
          n level (actually inside 2nd level) is for n parameters
      * Please place correctly, according to it's level used *

      if(sender_message == "") {

      }
   */

   if(command_level == 2) { // Parameter mode

      // Add report
      mmLog->Lines->Add("\t(i) Parameter(s) data confirmed");

      // Going back to instruction mode again. (by default)
      command_level = 1;

      switch(instruction_no) {
         case 3:
            // List file <index>
            listed = false;
            SliceShortcuts(StrToIntDef(sender_message,-1)); // included with reply
plans.
         break;

         case 4:
            // Get file <option>

            // Confirmation answer
            if(confirmation == true) {
               if(sender_message == "Y") {
                  // Set selected shortcut
                  selected_shortcut = String(shortcuts[option-1]);
                  // Add report
```

```
                  mmLog->Lines->Add("\t(i) Sending file: " + selected_shortcut);
// String(files[option-1]));
                  log_scroll_y++;

                  // Send Mail
                  SendMail();

                  confirmation = false;
               } else {
                  if(sender_message == "N") {
                     // Do nothing, don't send mail!
                     selected_shortcut = ""; // reset
                     SWSendMessage("(i) File selection cancelled","\t");
                     // Add report
                     mmLog->Lines->Add("\t(i) File selection cancelled by user");
                     log_scroll_y++;
                  } else {
                     // Unrecognize input, try again!
                     command_level = 2;
                     SWSendMessage("(?) Unknown confirmation, try again!","\t");
                  }
               }
            } else {

               // Get option value
               option = StrToIntDef(sender_message,-1);
               if((option > 0) && (option < available_file_option_max+1)) {

                  // Add report
                  mmLog->Lines->Add("\t(i) File selected: " + String(files[option-
1]));
                  log_scroll_y++;

                  String confirm = "File: " + String(files[option-1]) + "? (Y/N)";
                  SWSendMessage(confirm,"\t");

                  // Add report
                  mmLog->Lines->Add("\t(i) Waiting for user confirmation..");
                  log_scroll_y++;

                  confirmation = true;
                  command_level = 2; // get again the 2nd parameter, Y/N
               } else {
                  // Add report
                  mmLog->Lines->Add("\t(!) Error option, try again!");
                  log_scroll_y++;
                  SWSendMessage("(?) Error option, try again!","\t");
                  confirmation = false;
                  command_level = 2; // error, try again!
               }
            }
         break;

         case 5:
            // Download <url>
            /*
               Extend if the last char is '<'
               Stop extend if the last char is '>'
               Cancel download if the current parameter is "<cancel>"
            */

            if(sender_message != "<cancel>") {

               if(sender_message != "<done>") {
                  URL = URL + sender_message;
                  command_level = 2; // get other parts of the URL
               } else {
                  StartDownload();
               }
```

```
            } else {

               // Cancel download
               URL = ""; // reset

               // Add report
               mmLog->Lines->Add("(i) Download cancelled by user");
               log_scroll_y++;

               // Normal/Verbose:
               SWSendMessage("(i) Download cancelled","\t");

            }
         break;

      }

   }


   if(command_level == 1) { // Instruction mode

      confirmation = false; // default

      if(sender_message == "Attention") {
         // Get attention from software
         lblCurrentTask->Caption = "Get Attention!";
         // No parameter
         // Add report
         mmLog->Lines->Add("(i) User command: Get Attention");
         log_scroll_y++;
         instruction_no = 1;
         if(sys_errors > 0) {
            SWSendMessage("(!) Sys errors detected, total " +
IntToStr(sys_errors),"\t");
         } else {
            SWSendMessage("(i) System OK!","\t");
         }
      }

      if(sender_message == "Power off") {
         // No parameter
         lblCurrentTask->Caption = "Shutdown";
         // Add report
         mmLog->Lines->Add("(i) User command: Computer, Power off");
         log_scroll_y++;
         instruction_no = 2;
         CloseSoftware();
      }

      if(sender_message == "List file") {
         if(enumerated == true) {
            // Parameter: <index>
            lblCurrentTask->Caption = "Get file list";
            // Add report
            mmLog->Lines->Add("(i) User command: Get file list");
            log_scroll_y++;
            instruction_no = 3;
            command_level = 2;
            int total = (total_shortcuts / 3);
            if((total_shortcuts % 3) > 0) {
               total++;
            }
            SWSendMessage("(?) Select index: 1-" + IntToStr(total),"\t");
            // Add report
            mmLog->Lines->Add("\t(i) Waiting for parameter(s) data...");
            log_scroll_y++;
         } else {
            // Add report
            mmLog->Lines->Add("\t(!) Error, enumarator malfunction");
```

```
      log_scroll_y++;
      SWSendMessage("(!) Shortcuts enumerator malfunction!","\t");
      sys_errors++;
   }
}

if(sender_message == "Get file") {
   if(listed == true) {
      // Parameter: <option>
      lblCurrentTask->Caption = "Upload file";
      // Add report
      mmLog->Lines->Add("(i) User command: Get file/Upload file");
      log_scroll_y++;
      instruction_no = 4;
      command_level = 2;
      confirmation = false;
      SWSendMessage("(?) Select option: 1-3 ?","\t");
      // Add report
      mmLog->Lines->Add("\t(i) Waiting for parameter(s) data...");
   } else {
      // Add report
      mmLog->Lines->Add("\t(!) You haven't list the file!");
      log_scroll_y++;
      SWSendMessage("(!) You haven't list the file!","\t");
   }
}

if(sender_message == "Download") {
   // Parameter: <index>
   // Add report
   mmLog->Lines->Add("(i) User command: Download");
   log_scroll_y++;
   URL = "";
   instruction_no = 5;
   command_level = 2;
   SWSendMessage("(?) Input URL now, if done send '<done>'","\t");
   SWSendMessage("(?) Otherwise, '<cancel>'","\t");
   // Add report
   mmLog->Lines->Add("\t(i) Waiting for parameter(s) data...");
}

if(sender_message == "Download list") {
   // No parameter
   instruction_no = 6;
   // Add report
   mmLog->Lines->Add("(i) User command: Download list");
   log_scroll_y++;
   StartDownloadList();
}

if(sender_message == "Download progress") {
   // No parameter
   // Add report
   mmLog->Lines->Add("(i) User command: Get download progress");
   log_scroll_y++;
   instruction_no = 7;
   // Send message about download progress
   if(tmrDownload->Enabled == true) {
      lblCurrentTask->Caption = "Get progress";
      String info;
      info = "(i) Download progress " + lblPercent->Caption + " completed.";
      SWSendMessage(info,"\t");
   } else {
      mmLog->Lines->Add("\t(?) No download process running! - fail");
      log_scroll_y++;
      SWSendMessage("(?) No download process running! - fail","\t");
   }
}

if(sender_message == "Abort download") {
```

```
         // No perameter
         // Add report
         mmLog->Lines->Add("(i) User command: Abort download");
         log_scroll_y++;
         instruction_no = 8;
         StopDownload();
         URL = ""; // delete last URL
      }

      if(sender_message == "Pause download") {
         // No parameter
         // Add report
         mmLog->Lines->Add("(i) User command: Pause download");
         log_scroll_y++;
         instruction_no = 9;
         PauseDownload();
      }

      if(sender_message == "Resume download") {
         // No parameter
         // Add report
         mmLog->Lines->Add("(i) User command: Resume download");
         log_scroll_y++;
         instruction_no = 10;
         ResumeDownload();
      }

      if(sender_message == "Restart download") {
         // No parameter
         // Add report
         mmLog->Lines->Add("(i) User command: Restart download");
         log_scroll_y++;
         instruction_no = 11;
         RestartDownload();
      }

      if(sender_message == "Download list progress") {
         // No parameter
         instruction_no = 12;
         // Add report
         mmLog->Lines->Add("(i) User command: Download list progress");
         log_scroll_y++;
         if(tmrDownloadList->Enabled == true) {
            // Send progress
            String msg = "(i) Download list progress " +
IntToStr(downloading_index+1) + "/" + IntToStr(download_list_count);
            SWSendMessage(msg,"\t");
            // Add report
            mmLog->Lines->Add("\t"+msg);
            log_scroll_y++;
         } else {
            // Add report
            mmLog->Lines->Add("\t(!) 'Download list' process isn't running");
            log_scroll_y++;
            // Send warning
            SWSendMessage("(!) 'Download list' process isn't running","\t");
         }
      }

      if(sender_message == "Stop download list") {
         // No parameter
         instruction_no = 13;
         // Add report
         mmLog->Lines->Add("(i) User command: Stop download list");
         log_scroll_y++;
         StopDownloadList();
      }

      if(sender_message == "Restart download list") {
         // No parameter
```

```
         instruction_no = 14;
         // Add report
         mmLog->Lines->Add("(i) User command: Restart download list");
         log_scroll_y++;
         RestartDownloadList();
      }

      if(sender_message == "Pause download list") {
         // No parameter
         instruction_no = 15;
         // Add report
         mmLog->Lines->Add("(i) User command: Pause download list");
         log_scroll_y++;
         PauseDownloadList();
      }

      if(sender_message == "Resume download list") {
         // No parameter
         instruction_no = 16;
         // Add report
         mmLog->Lines->Add("(i) User command: Resume download list");
         log_scroll_y++;
         ResumeDownloadList();
      }

      sender_message = "";
      sender_number = "";

   }

}

void TfrmMain::CloseSoftware()
{
   if(general_usage == false) {

      // Verbose: "Closing software!"
      if(frmConfigure->chkVerbose->Checked == true) {
         SWSendMessage("(i) Closing software!","");
      }

      // Close port first
      ClosePort();

      // Shutdown computer (delayed)
      Shutdown();

      // Add current time into log
      String time_str = String(Time());
      String date_str = String(Date());
      mmLog->Lines->Add("(i) Software closed at " + date_str + " - " + time_str);

      // Add computer_up_time to log
      mmLog->Lines->Add("(i) Computer uptime: " + lblCompUptime->Caption);
      log_scroll_y++;

      // Add system errors counter to log
      mmLog->Lines->Add("(!) System errors count: " + IntToStr(sys_errors));
      log_scroll_y++;

      // Save mmLog data to a log file
      // accumulate current logs data into mmAllLog
      mmAllLog->Lines->Add(mmLog->Text);
      if(frmConfigure->chkSaveLog->Checked == true) {
         mmLog->Clear();
         // check for existing log files
         if(FileExist(frmConfigure->txtProgramLog->Text) == true) {
            mmLog->Lines->LoadFromFile(frmConfigure->txtProgramLog->Text);
         }
         mmAllLog->Lines->Add("");
```

```
         mmAllLog->Lines->Add("");
         mmAllLog->Lines->Add(mmLog->Text);
         mmAllLog->Lines->SaveToFile(frmConfigure->txtProgramLog->Text);
      }

   } else {
      // General computer usage.
      // This mean that the computer is going to be used by user.
      // so dont process: Shutdown SaveLog, etc

      MessageBox(Handle,"Computer general usage preferred. Closing
software..","Acknowledged",MB_OK|MB_ICONINFORMATION);

      if(port_open == true) { // port_open var actually useful for this kind of
purpose
         ClosePort();
      }

   }

   // Freeing any used RAM/Memory variables
   if(download_list_count > 0) {
      for(int i = 0 ; i < download_list_count ; i++) {
         free(download_file_list[i]);
      }
      free(download_file_list);
   }

   for(int i = 0; i < 3 ; i++) {
      free(shortcuts[i]);
      free(files[i]);
   }
   free(shortcuts);
   free(files);

   // Terminate application
   Application->Terminate();
}

// ===========================================================================
//                              DOWNLOAD SUBSYSTEM
// ===========================================================================

void __fastcall TfrmMain::tmrDownloadListTimer(TObject *Sender)
{
   download_mode = 2;

   // Set status as well
   lblDownloadJob->Caption = IntToStr(downloading_index+1) + " / " +
IntToStr(download_list_count);

   // Monitor tmrDownload enable/disable
   if(tmrDownload->Enabled == false) {
      // current downloading_file_index has been completed! / fail! / etc
      // Next file!
      downloading_index++;
      if(downloading_index < download_list_count) {
         URL = String(download_file_list[downloading_index]);
         DownloadListPerFile();
      } else {
         // Turn off this timer
         tmrDownload->Enabled = false;
         tmrDownloadList->Enabled = false;
         // Add report
         mmLog->Lines->Add("\t(i) Download list has been completed!");
         log_scroll_y++;
         // Send info
         SWSendMessage("(i) Download list completed!","\t");
         download_mode = 0;
      }
```

```
   }
}

void TfrmMain::StartDownloadList()
{
   if(tmrDownloadList->Enabled == false) { // make sure downloader currently NOT
downloading a job

      if(download_list_count > 0) {
         // Freeing last used memory
         for(int i = 0 ; i < download_list_count ; i++) {
            free(download_file_list[i]);
         }
         free(download_file_list);
      }

      if(download_mode == 0) {
         // Open list file
         String list_file = frmConfigure->txtDownloadListPath->Text;
         String temp;

         if( FileExist(list_file) == true ) {

            char * retStr;
            retStr = (char*) calloc( 3001, sizeof(char));
            memset(retStr,0,3001);

            // Load selected download list file
            // Get the count
            GetPrivateProfileString(
"Properties","Count","0",retStr,3000,list_file.c_str());
            download_list_count = StrToIntDef(String(retStr),0);

            // Allocate for variables
            download_file_list = (char **) calloc( download_list_count ,
sizeof(char*));
            for(int i = 0 ; i < download_list_count ; i++) {
               download_file_list[i] = (char*) calloc( 3001, sizeof(char));
               memset(download_file_list[i],0,3001);
            }

            // Get the list
            for(int i = 0 ; i < download_list_count ; i++) {
               memset(retStr,0,3000);
               temp = "Addr" + IntToStr(i);
               GetPrivateProfileString(
"List",temp.c_str(),"",retStr,3000,list_file.c_str());
               URL = String(retStr);
               strncpy(download_file_list[i],retStr,strlen(retStr));
            }

            free(retStr);

            if(download_list_count > 0) {

               // Send info
               SWSendMessage("(i) Downloading the list...","\t");
               // Add report
               mmLog->Lines->Add("\t(i) Downloading the list...");
               log_scroll_y++;

               lblCurrentTask->Caption = "Downloading the list";

               // Setup download-list variables
               downloading_index = 0;

               // Set the 1st file, and download it
               URL = String(download_file_list[0]);
               DownloadListPerFile();
```

```
               // Activate Download timer
               download_mode = 2;
               tmrDownloadList->Enabled = true;

            } else {
               // Add report
               mmLog->Lines->Add("\t(!) URLs not found! - fail");
               log_scroll_y++;
               // Send warning
               SWSendMessage("(!) URLs not found! - fail","\t");
               sys_errors++;
            }

         } else {

            // Add report
            mmLog->Lines->Add("\t(!) Download list not found!");
            log_scroll_y++;

            // Send warning
            SWSendMessage("(!) Download list not found!","\t");
         }
      } else {
         // Add report
         mmLog->Lines->Add("\t(!) Download is currently running! - fail");
         log_scroll_y++;

         // Send warning
         SWSendMessage("(!) Download is currently running! - fail","\t");
      }

   } else {
      // Add report
      mmLog->Lines->Add("\t(!) Currently downloading a list! - fail");
      log_scroll_y++;
      // Send warning
      SWSendMessage("(!) Currently downloading a list! - fail","\t");
   }

}

void TfrmMain::StopDownloadList()
{
   if((download_mode == 2) && (tmrDownloadList->Enabled == true)) {
      // Turn off download-log monitor
      tmrDownloadList->Enabled = false;
      tmrDownload->Enabled = false;

      // Turn off download LED
      imgLED->Picture = imgLEDoff->Picture;

      // Set the status
      lblDownloaderStatus->Caption = "Aborted";
      lblCurrentTask->Caption = "Stop download list";

      // Close the downloader process
      TerminateProcess(wget_hwnd,1);

      Sleep(1000); // 1 sec delay

      // Delete the log file!
      DeleteFile(frmConfigure->txtDownloaderLogPath->Text);

      // Add report
      mmLog->Lines->Add("\t(i) Download list stopped!");
      log_scroll_y++;
      // Send info
      SWSendMessage("(i) Download list stopped!","\t");

      download_mode = 0; // no mode is running
```

```
   } else {
      // Add report
      mmLog->Lines->Add("\t(!) 'Download list' isn't running!");
      log_scroll_y++;
      SWSendMessage("(!) 'Download list' isn't running!","\t");
   }
}

void TfrmMain::RestartDownloadList()
{
   if((download_mode == 2) && (tmrDownloadList->Enabled == true)) {
      StopDownloadList();
      StartDownloadList();
   } else {
      // Add report
      mmLog->Lines->Add("\t(!) 'Download list' isn't running!");
      log_scroll_y++;
      SWSendMessage("(!) 'Download list' isn't running!","\t");
   }
}

void TfrmMain::PauseDownloadList()
{
   if((download_mode == 2) && (tmrDownloadList->Enabled == true)) {

      tmrDownloadList->Enabled = false;

      //Turn off download-log monitor
      tmrDownload->Enabled = false;

      //Set status..
      lblDownloaderStatus->Caption = "Paused";
      lblCurrentTask->Caption = "Pause download list";

      //Turn Download LED off
      imgLED->Picture = imgLEDoff->Picture;

      // this actually kill the wget process only, and does nothing
      TerminateProcess(wget_hwnd,1);

      // Add a report
      // Normal/Verbose: "Download paused!"
      mmLog->Lines->Add("\t(i) Download list paused");
      log_scroll_y++;
      SWSendMessage("(i) Download list paused!","\t");
   } else {
      // Add report
      mmLog->Lines->Add("\t(!) 'Download list' isn't running!");
      log_scroll_y++;
      SWSendMessage("(!) 'Download list' isn't running!","\t");
   }
}

void TfrmMain::ResumeDownloadList()
{
   if((download_mode == 2) && (tmrDownloadList->Enabled == false)) {

      String program_path = frmConfigure->txtDownloaderProgramPath->Text;
      // Add '-c' to program parameter (this '-c' means to continue downloading
last downloaded file)
      // This is the main difference between Start and Resume download
      String program_cmdline = String(" -P " + String(char(34)) + frmConfigure-
>txtDownloaderSavePath->Text + String(char(34)) + " -c -o " + String(char(34)) +
frmConfigure->txtDownloaderLogPath->Text + String(char(34)) + " -T 30 -t 2 " +
URL).c_str();

      STARTUPINFO startupinfo;
      memset(&startupinfo, 0, sizeof(startupinfo));
      startupinfo.cb = sizeof(startupinfo);
```

```
      PROCESS_INFORMATION procinfo;
      memset(&procinfo, 0, sizeof(procinfo));
      CreateProcess(program_path.c_str(),program_cmdline.c_str(), 0, 0, false,
CREATE_DEFAULT_ERROR_MODE | NORMAL_PRIORITY_CLASS, NULL, NULL, &startupinfo,
&procinfo);

      //check its running or not?
      if(procinfo.hProcess != NULL) {

         wget_hwnd = procinfo.hProcess;
         lblDownloaderStatus->Caption = "Resuming";
         lblCurrentTask->Caption = "Resuming download list";

         imgLED->Picture = imgLEDon->Picture;

         //Running, next process?
         tmrDownload->Enabled = true;
         tmrDownloadList->Enabled = true;

      } else {
         lblDownloaderStatus->Caption = "Failed to run!";
         imgLED->Picture = imgLEDoff->Picture;
      }

      // Add a report
      // Normal/Verbose: "Download resumed!"
      mmLog->Lines->Add("\t(i) Download list resumed");
      log_scroll_y++;
      SWSendMessage("(i) Download list resumed!","\t");

   } else {
      // Add report
      mmLog->Lines->Add("\t(!) Wrong download mode / isn't paused!");
      log_scroll_y++;
      SWSendMessage("(!) Wrong download mode / isn't paused!","\t");
   }
}

void TfrmMain::DownloadListPerFile()
{
   String program_path = frmConfigure->txtDownloaderProgramPath->Text;
   String program_cmdline = String(" -P " + String(char(34)) + frmConfigure-
>txtDownloaderSavePath->Text + String(char(34)) + " -o " + String(char(34)) +
frmConfigure->txtDownloaderLogPath->Text + String(char(34)) + " -T 30 -t 2 " +
URL).c_str();

   STARTUPINFO startupinfo;
   memset(&startupinfo, 0, sizeof(startupinfo));
   startupinfo.cb = sizeof(startupinfo);
   PROCESS_INFORMATION procinfo;
   memset(&procinfo, 0, sizeof(procinfo));
   CreateProcess(program_path.c_str(),program_cmdline.c_str(), 0, 0, false,
CREATE_DEFAULT_ERROR_MODE | NORMAL_PRIORITY_CLASS, NULL, NULL, &startupinfo,
&procinfo);

   //check its running or not?
   if(procinfo.hProcess != NULL) {

      wget_hwnd = procinfo.hProcess;
      lblDownloaderStatus->Caption = "Downloading";
      lblCurrentTask->Caption = "Downloading list...";

      imgLED->Picture = imgLEDon->Picture;
      lblPercent->Caption = "0%";
      PBar->Position = 0;

      //Running, next process?
      tmrDownload->Enabled = true;

      Sleep(100); // wait 0.1 sec
```

```
      tmrDownloadTimer(frmMain); // call to get downloaded_file variable filled
with data.

      // Add a report
      mmLog->Lines->Add("\t(>) Downloading file (" + IntToStr(downloading_index+1)
+ "): " + downloaded_file);
      log_scroll_y++;

   } else {

      imgLED->Picture = imgLEDoff->Picture;
      tmrDownload->Enabled = false;
      lblDownloaderStatus->Caption = "Failed to run!";
      // Add a report
      mmLog->Lines->Add("\t(!) Failed to run downloader program... - aborted");
      sys_errors++;
      log_scroll_y++;

      // Normal/Verbose: "Failed to download"
      SWSendMessage("(!) Couldn't run downloader! - aborted","\t");

   }
}

// ======================[ SINGLE DOWNLOAD FUNCTION ]=========================

void __fastcall TfrmMain::tmrDownloadTimer(TObject *Sender)
{
   ParseDownloadLog();
}

void TfrmMain::StartDownload()
{
   if(tmrDownload->Enabled == false && download_mode == 0) { // check currently
downloading something or not?
      String program_path = frmConfigure->txtDownloaderProgramPath->Text;
      String program_cmdline = String(" -P " + String(char(34)) + frmConfigure-
>txtDownloaderSavePath->Text + String(char(34)) + " -o " + String(char(34)) +
frmConfigure->txtDownloaderLogPath->Text + String(char(34)) + " -T 30 -t 2 " +
URL).c_str();

      STARTUPINFO startupinfo;
      memset(&startupinfo, 0, sizeof(startupinfo));
      startupinfo.cb = sizeof(startupinfo);
      PROCESS_INFORMATION procinfo;
      memset(&procinfo, 0, sizeof(procinfo));
      CreateProcess(program_path.c_str(),program_cmdline.c_str(), 0, 0, false,
CREATE_DEFAULT_ERROR_MODE | NORMAL_PRIORITY_CLASS, NULL, NULL, &startupinfo,
&procinfo);

      //check its running or not?
      if(procinfo.hProcess != NULL) {

         wget_hwnd = procinfo.hProcess;
         lblDownloaderStatus->Caption = "Downloading";
         lblCurrentTask->Caption = "Downloading file";

         imgLED->Picture = imgLEDon->Picture;
         lblPercent->Caption = "0%";
         PBar->Position = 0;

         //Running, next process?
         tmrDownload->Enabled = true;

         Sleep(1000); // wait 1 sec
         tmrDownloadTimer(frmMain); // call to get downloaded_file variable filled
with data.

         // Add a report
         mmLog->Lines->Add("\t(>) Downloading file: " + downloaded_file);
```

```
         log_scroll_y++;
         // Normal/Verbose: "Downloading file..."
         SWSendMessage("(>) Downloading file...","\t");

         download_mode = 1; // set to single mode
         lblDownloadJob->Caption = "";

      } else {
         imgLED->Picture = imgLEDoff->Picture;
         tmrDownload->Enabled = false;
         lblDownloaderStatus->Caption = "Failed to run!";
         // Add a report
         mmLog->Lines->Add("\t(!) Failed to run downloader program... - aborted");
         sys_errors++;
         log_scroll_y++;

         // Normal/Verbose: "Failed to download"
         SWSendMessage("(!) Couldn't run downloader! - aborted","\t");

         sys_errors++;

      }
   } else {
      // Add report
      mmLog->Lines->Add("\t(!) List mode is running - fail");
      log_scroll_y++;
      SWSendMessage("(!) List mode is running - fail","\t");
   }

}

void TfrmMain::PauseDownload()
{
   if((URL != "") && (tmrDownload->Enabled == true) && (download_mode == 1)) {

      tmrDownload->Enabled = false;
      //Set status..
      lblDownloaderStatus->Caption = "Paused";
      lblCurrentTask->Caption = "Pause download";

      //Turn off download-log monitor
      tmrDownload->Enabled = false;

      //Turn Download LED off
      imgLED->Picture = imgLEDoff->Picture;

      // this actually kill the wget process only, and does nothing
      TerminateProcess(wget_hwnd,1);

      // Add a report
      // Normal/Verbose: "Download paused!"
      mmLog->Lines->Add("\t(i) Download paused");
      log_scroll_y++;
      SWSendMessage("(i) Download paused!","\t");

   } else {
      // Add report
      mmLog->Lines->Add("\t(?) Pause download, unknown! - fail");
      log_scroll_y++;
      SWSendMessage("(?) Pause download, unknown! - fail","\t");
   }
}

void TfrmMain::ResumeDownload()
{
   if((URL != "") && (tmrDownload->Enabled == false) && (download_mode == 1)) {

      String program_path = frmConfigure->txtDownloaderProgramPath->Text;
      // Add '-c' to program parameter (this '-c' means to continue downloading
last downloaded file)
```

```
      // This is the main difference between Start and Resume download
      String program_cmdline = String(" -P " + String(char(34)) + frmConfigure-
>txtDownloaderSavePath->Text + String(char(34)) + " -c -o " + String(char(34)) +
frmConfigure->txtDownloaderLogPath->Text + String(char(34)) + " -T 30 -t 2 " +
URL).c_str();

      STARTUPINFO startupinfo;
      memset(&startupinfo, 0, sizeof(startupinfo));
      startupinfo.cb = sizeof(startupinfo);
      PROCESS_INFORMATION procinfo;
      memset(&procinfo, 0, sizeof(procinfo));
      CreateProcess(program_path.c_str(),program_cmdline.c_str(), 0, 0, false,
CREATE_DEFAULT_ERROR_MODE | NORMAL_PRIORITY_CLASS, NULL, NULL, &startupinfo,
&procinfo);

      //check its running or not?
      if(procinfo.hProcess != NULL) {

         wget_hwnd = procinfo.hProcess;
         lblDownloaderStatus->Caption = "Resuming";
         lblCurrentTask->Caption = "Resuming download";

         imgLED->Picture = imgLEDon->Picture;

         //Running, next process?
         tmrDownload->Enabled = true;

      } else {
         lblDownloaderStatus->Caption = "Failed to run!";
         imgLED->Picture = imgLEDoff->Picture;
      }

      // Add a report
      // Normal/Verbose: "Download resumed!"
      mmLog->Lines->Add("\t(i) Download resumed");
      log_scroll_y++;
      SWSendMessage("(i) Download resumed!","\t");

   } else {
      // Add report
      mmLog->Lines->Add("\t(?) Resume download, unknown! - fail");
      log_scroll_y++;
      SWSendMessage("(?) Resume download, unknown! - fail","\t");
   }
}

void TfrmMain::StopDownload()
{
   if((tmrDownload->Enabled == true) && (download_mode == 1)) {
      // Turn off download-log monitor
      tmrDownload->Enabled = false;
      tmrDownloadList->Enabled = false;

      // Turn off download LED
      imgLED->Picture = imgLEDoff->Picture;

      // Set the status
      lblDownloaderStatus->Caption = "Aborted";
      lblCurrentTask->Caption = "Stop download";

      // Close the downloader process
      TerminateProcess(wget_hwnd,1);

      Sleep(1000); // 1 sec delay

      // Delete the log file!
      DeleteFile(frmConfigure->txtDownloaderLogPath->Text);

      //Delete the donwloaded file
      DeleteFile(downloaded_file);
```

```
      // Normal/Verbose: "Download aborted!"
      // Add a report
      mmLog->Lines->Add("\t(i) Download aborted");
      log_scroll_y++;
      SWSendMessage("(i) Download aborted!","\t");

      download_mode = 0; // no mode running

   } else {
      // Add report
      mmLog->Lines->Add("\t(?) Abort download, unknown! - fail");
      log_scroll_y++;
      SWSendMessage("(?) Abort download, unknown! - fail","\t");
   }

}

void TfrmMain::RestartDownload()
{
   if(URL != "" && download_mode == 1) {
      StopDownload();
      // Delay 5 seconds*
      for(int i = 0; i < 5001; i++) {
         Sleep(1);
      }
      StartDownload();
   } else {
      // Add report
      mmLog->Lines->Add("\t(?) Restart download, No URL / wrong mode - fail");
      log_scroll_y++;
      SWSendMessage("(?) Restart dwnld, URL?/wrong mode - fail","\t");
   }
}

// ======================[ MISC. DOWNLOAD FUNCTION ]=========================

void TfrmMain::ParseDownloadLog()
{
   ifstream log;
   const int TEMP_LENGTH = 50;
   char txtcmp[TEMP_LENGTH+1];
   char temp[TEMP_LENGTH+1];
   memset(temp,0,TEMP_LENGTH+1);
   char single_temp;
   char filename[256];
   char percent[4];
   int i_percent = -1;
   int i_temp;
   int i_for;
   String logPath = frmConfigure->txtDownloaderLogPath->Text;
   log.open(logPath.c_str());

   /* HOW THIS PARSER WORK

      Data:
         This is not a 100% safe things, so please beware of it.
      Comparison text:
         "This"
      How the comparing process works:

         _x__x__x_[T][h][i][s]  < Comparison box

         [ ][ ][ ][ ][ ][ ][T]  < Serial input a char 1 by 1, shift left
         [ ][ ][ ][ ][ ][T][h]
         [ ][ ][ ][ ][T][h][i]
         [ ][ ][ ][T][h][i][s]  > Return OK here!
         [ ][ ][T][h][i][s][ ]
         [ ][T][h][i][s][ ][i]
         [T][h][i][s][ ][i][s]
```

```
*/

while(log.eof() == false) {
   //FILO First In Last Out
   for(i_for = 0; i_for < TEMP_LENGTH; i_for++) {
      temp[i_for] = temp[i_for+1];
      /*
         temp[0] = temp[1];
         temp[1] = temp[2];
         temp[2] = temp[3];
         temp[3] = temp[4];
         ...
      */
   }
   log.get(temp[TEMP_LENGTH]);

   // monitor any text has pattern like this ".xxx%", x mean dont care
   if((temp[TEMP_LENGTH-4] == '.') && (temp[TEMP_LENGTH]=='%')) {
      memset(percent, 0, 4);
      percent[0] = temp[TEMP_LENGTH-3];
      percent[1] = temp[TEMP_LENGTH-2];
      percent[2] = temp[TEMP_LENGTH-1];
      i_temp = StrToInt(String(percent));
      if(i_temp > i_percent) {
         i_percent = i_temp;
         PBar->Position = i_percent;
         lblPercent->Caption = IntToStr(i_percent) + "%";
      }
   }

   // monitor any text has pattern like this "1xx%", x mean dont care
   if((temp[TEMP_LENGTH-3]=='1')&&(temp[TEMP_LENGTH]=='%')) {
      tmrDownload->Enabled = false;
      PBar->Position = 100;
      lblPercent->Caption = "100%";
      lblDownloaderStatus->Caption = "Finished!";
      imgLED->Picture = imgLEDoff->Picture;
      URL = ""; // reset
      // Add a report to log
      mmLog->Lines->Add("\t(i) Download has been finished");
      log_scroll_y++;;
      if(download_mode != 2) {
         // this apply to single-download only
         // Normal/Verbose: "Download Complete!"
         SWSendMessage("(i) Download has been finished","\t");
      }
      // Delete the log file!
      DeleteFile(frmConfigure->txtDownloaderLogPath->Text);
      download_mode = 0;
   }

   // monitor any text that has this pattern "failed: timed out."
   strnrcpy(txtcmp,temp,strlen("failed: timed out."),TEMP_LENGTH);
   if(strncmp(txtcmp,"failed: timed out.",strlen("failed: timed out.")) == 0) {
      tmrDownload->Enabled = false;
      PBar->Position = 0;
      lblDownloaderStatus->Caption = "Failed!";
      imgLED->Picture = imgLEDoff->Picture;
      // Add a report to log
      mmLog->Lines->Add("\t(!) Download stopped, downloader timed out!");
      log_scroll_y++;;
      //Delete the donwloaded file
      DeleteFile(downloaded_file);
      // Delete the log file!
      DeleteFile(frmConfigure->txtDownloaderLogPath->Text);
      // Normal/Verbose: "Download failed, timed out!"
      sys_errors++;
      SWSendMessage("(!) Download failed, timed out!","\t");
      download_mode = 0;
```

```
      }

      // monitor any text that has this pattern "Giving up."
      strnrcpy(txtcmp,temp,strlen("Giving up."),TEMP_LENGTH);
      if(strncmp(txtcmp,"Giving up.",strlen("Giving up.")) == 0) {
         tmrDownload->Enabled = false;
         PBar->Position = 0;
         lblDownloaderStatus->Caption = "Failed!";
         imgLED->Picture = imgLEDoff->Picture;
         // Add a report to log
         mmLog->Lines->Add("\t(!) Download stopped, downloader giving up!");
         log_scroll_y++;;
         //Delete the donwloaded file
         DeleteFile(downloaded_file);
         // Delete the log file!
         DeleteFile(frmConfigure->txtDownloaderLogPath->Text);
         // Normal/Verbose: "Download failed, give up!"
         sys_errors++;
         SWSendMessage("(!) Download failed, give up!","\t");
         download_mode = 0;
      }

      // monitor any text that has this pattern "... failed: Host not found."
      strnrcpy(txtcmp,temp,strlen("... failed: Host not found."),TEMP_LENGTH);
      if(strncmp(txtcmp,"... failed: Host not found.",strlen("... failed: Host not
found.")) == 0) {
         tmrDownload->Enabled = false;
         PBar->Position = 0;
         lblDownloaderStatus->Caption = "Failed!";
         imgLED->Picture = imgLEDoff->Picture;
         // Add a report to log
         mmLog->Lines->Add("\t(!) Download stopped, host not found!");
         log_scroll_y++;;
         //Delete the donwloaded file
         DeleteFile(downloaded_file);
         // Delete the log file!
         DeleteFile(frmConfigure->txtDownloaderLogPath->Text);
         // Normal/Verbose: "Download failed: 404"
         sys_errors++;
         SWSendMessage("(!) Download failed: 404","\t");
         download_mode = 0;
      }

      // monitor any text that has this pattern "ERROR 404"
      strnrcpy(txtcmp,temp,strlen("ERROR 404"),TEMP_LENGTH);
      if(strncmp(txtcmp,"ERROR 404",strlen("ERROR 404")) == 0) {
         tmrDownload->Enabled = false;
         PBar->Position = 0;
         lblDownloaderStatus->Caption = "Failed!";
         imgLED->Picture = imgLEDoff->Picture;
         // Add a report to log
         mmLog->Lines->Add("\t(!) Download stopped, 404 code!");
         log_scroll_y++;;
         //Delete the donwloaded file
         DeleteFile(downloaded_file);
         // Delete the log file!
         DeleteFile(frmConfigure->txtDownloaderLogPath->Text);
         // Normal/Verbose: "Download failed: 404"
         sys_errors++;
         SWSendMessage("(!) Download failed: 404","\t");
         download_mode = 0;
      }

      // get this pattern, and the resolve the filename "=> `filename'"
      if((temp[TEMP_LENGTH-3]=='=')&&(temp[TEMP_LENGTH]=='`')) {
         single_temp = '-';
         memset(filename,0,256);
         int filename_index = 0;
         while(single_temp != '\'') {
            log.get(single_temp);
```

```
            if(single_temp == '/') {
               // change the filesystem view
               // from linux type ( / ) to windows type ( \ )
               single_temp = '\\';
            }
            if(single_temp != '\'') {
               //this conditional remove the last "'" quote
               filename[filename_index] = single_temp;
               filename_index++;
            }
         }

      }

   }

   log.close();

}

// ============================================================================
//                            E-MAIL SUBSYSTEM
// ============================================================================

void TfrmMain::SettingMail()
{
   // save user sensitive configuration by calling `blat -installSMTP`

   /*

   <server addr>      *      The SMTP Server you want Blat to connect to for
sending eMail.
   <sender's email addr>      *      This is the from address of the message.
   <try n times>      **+    How many times you would like Blat to try to send to
the server before giving up (default=1)
   <port>      **+    The TCP/IP port the above SMTP Server is listening on
(defaults; SMTP=25, NNTP=119)
   <profile>   **+    The Profile name to associate with this set of parameters
stored in the Registry. This allows you to keep multiple configurations.
   <username>  **     If your server requires authentication, you can store the
username here.
   <password>  **     If your server requires authentication, you can store the
password here.

   */

   String config;
   config = frmConfigure->txtEmailProgramPath->Text + "\\blat.exe -installSMTP ";

   config += frmConfigure->txtSMTPServer->Text;
   config += " " + frmConfigure->txtSenderAddress->Text;
   config += " -";
   config += " " + frmConfigure->txtSSLPort->Text;
   config += " SMS_Sys";
   config += " " + frmConfigure->txtUsername->Text;
   config += " " + String(char(34)) + frmConfigure->txtPassword->Text +
String(char(34));

   system(config.c_str());

}

void TfrmMain::SendMail()
{
   if(selected_shortcut != "") {

      // call the main program `blat`
      String program = "blat.exe ";
      program = program + "-p SMS_Sys ";
      program = program + "-to " + frmConfigure->txtRecepientAddress->Text + " ";
```

```
      // Attachment file
      program = program + "-attach " + String(char(34)) + selected_shortcut +
String(char(34)) + " ";

      // Mail subject
      program = program + "-subject " + String(char(34)) + frmConfigure-
>txtMailSubject->Text + String(char(34)) + " ";

      // Mail body
      program = program + "-body " + String(char(34)) + "(This mail is from C.R.S-
SMS System)" +  String(char(34)) + " ";

      //add "cd /d <program_path>"
      String alamat = frmConfigure->txtEmailProgramPath->Text;
      program = "cd /d " + alamat + "\n" + program;

      // write pause (Debug-purpose)
      //program = program + "\npause";

      //Write the program output into file
      ofstream progOut;
      progOut.open(String(alamat + "\\program.bat").c_str());

      progOut.write(program.c_str(),strlen(program.c_str()));

      progOut.close();

      // Execute program
      system(String(alamat + "\\program.bat").c_str());

      //remove the program.bat
      DeleteFile(alamat + "program.bat");

      // Add to report
      mmLog->Lines->Add("\t(i) E-mail has been sent!");
      log_scroll_y++;

      // Normal/Verbose: "E-mail has been sent!"
      SWSendMessage("(i) E-mail has been sent!","\t");

   } else {
      //MessageBox(Handle,"Shortcut hasn't been choosen
yet!","Error",MB_OK|MB_ICONERROR);
      // Add to report
      mmLog->Lines->Add("\t(!) Shortcut hasn't been choosen yet! -- Send e-mail
aborted");
      log_scroll_y++;
      //Normal/Verbose: "Shortcuts hasn't been choosen yet!"
      SWSendMessage("(!) Shortcuts hasn't been choosen yet!","\t");
   }
}

// ============================================================================
//                       SHORTCUT ENUMERATION SUBSYSTEM
// ============================================================================

void TfrmMain::EnumerateShortcuts()
{
   // Enumerate files (shortcuts) in pointed directory

   mmLog->Text = mmLog->Text + ("(>) Enumerating recent shortcuts... ");
   log_scroll_y++;

   String dir = frmConfigure->txtShortcutsDirPath->Text + "\\*"; // wildcard char
for search any file
   int total_item = 0;
   int shortcuts_detected = 0;
   HANDLE ret;
   WIN32_FIND_DATA search_result;
```

```
   if(frmConfigure->txtShortcutsDirPath->Text != "") {

      // [first, get the file count in pointed directory]
      ret = FindFirstFile(dir.c_str(),&search_result);

      if(ret != INVALID_HANDLE_VALUE) {

         if((String(search_result.cFileName) != "..") &&
(String(search_result.cFileName) != ".")) {
            total_item++;
         }

         while(FindNextFile(ret, &search_result) != 0){
            if((String(search_result.cFileName) != "..") &&
(String(search_result.cFileName) != ".")) {
               total_item++;
            }
         }

         FindClose(ret);
      }

      // [prepare variables for filenames and do search for another files]
      int index_filename = 0;
      char ** filenames;
      filenames = (char**) calloc(total_item,sizeof(char *));
      for(int k = 0 ; k < total_item ; k++){
         filenames[k] = (char *) calloc(260,sizeof(char));
         memset(filenames[k],0,260);
      }

      ret = FindFirstFile(dir.c_str(),&search_result);

      if(ret != INVALID_HANDLE_VALUE) {

         // don't include ".." or "."
         if((String(search_result.cFileName) != "..") &&
(String(search_result.cFileName) != ".")) {

strncpy(filenames[index_filename],search_result.cFileName,strlen(search_result.cFi
leName));
            index_filename++;
         }

         while(FindNextFile(ret, &search_result) != 0){
            // don't include ".." or "."
            if((String(search_result.cFileName) != "..") &&
(String(search_result.cFileName) != ".")) {

strncpy(filenames[index_filename],search_result.cFileName,strlen(search_result.cFi
leName));
               index_filename++;
            }
         }

         FindClose(ret);
      }

      // [Create filetime & systemtime variables]
      SYSTEMTIME userview;
      FILETIME *create_time;
      create_time = (FILETIME *) calloc(total_item,sizeof(FILETIME));

      // [Sort files by Create time (Descending)]
      // Create variable
      char ** sorted_files;
      sorted_files = (char **) calloc(total_item, sizeof(char *));
      for(int k = 0 ; k < total_item ; k++) {
         sorted_files[k] = (char *) calloc(260,sizeof(char));
```

```
         memset(sorted_files[k],0,260);
      }
      bool * done;
      done = (bool *) calloc(total_item, sizeof(bool));
      memset(done, false, total_item);

      // Open files and get their filetime (create time) properties
      String current_file;
      HANDLE file_handle;
      for(int i = 0 ; i < total_item; i++) {
         // Open the file with GENERIC_READ
         current_file = frmConfigure->txtShortcutsDirPath->Text + "\\" +
String(filenames[i]);
         file_handle = CreateFile(current_file.c_str(), GENERIC_READ,
FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL);

         // Get the filetime
         GetFileTime( file_handle ,&create_time[i],NULL,NULL);

         // Close file handle
         CloseHandle(file_handle);
      }

      // Sort

         unsigned long latest_time = 0x00000000;
         unsigned long latest_date = 0x00000000;

         unsigned long now_time = 0;
         unsigned long now_date = 0;

         int indeks = 0;
         int save_index = 0;

         for(int j = 0; j < total_item ; j++) {
            // Selection order
            for(int i = 0; i < total_item ; i++) {

               // Get user view data, and convert them again to universal value
               FileTimeToSystemTime(&create_time[i],&userview);

               now_time =
calculateTime(userview.wHour,userview.wMinute,userview.wSecond);
               now_date =
calculateDate(userview.wDay,userview.wMonth,userview.wYear);

               if( (now_date > latest_date) || ((now_date ==
latest_date)&&(now_time >= latest_time)) ){ // hari terbaru[reset waktu ke 0].
atau (jika hari sama, maka waktu terdepan [reset keduanya ke nol])
                  if(done[i] == false) {
                     latest_time = now_time;
                     latest_date = now_date;

                     indeks = i;
                  }
               }
            }

            // after selection
            done[indeks] = true; // set this as done
            // add to final list
            strcpy(sorted_files[save_index],filenames[indeks]);
            save_index++;
            //lbFiles->Items->Add(String(tempList->Items->Strings[indeks]));

            // reset again the best record
            latest_time = 0;
            latest_date = 0;
            indeks = 0;
```

```
         }

      // [Save data in hidden UI]
      lbShortcutsFiles->Clear();
      // show all filenames into list box
      for(int k = 0 ; k < total_item ; k++) {
         lbShortcutsFiles->Items->Add(String(sorted_files[k]));
         String current_shortcut_file = frmConfigure->txtShortcutsDirPath->Text +
"\\" + String(sorted_files[k]);
         if(GetFileExtension(current_shortcut_file) == "lnk") {
            shortcuts_detected++;
         }
      }

      // [Remove invalid links]
      // read each shortcuts path and save them into variable
      // then remove any invalid path that indicated this is an invalid
shortcut/link
      int valid = 0;
      String current_shortcut_file;
      String path;
      frmMain->Enabled = false;
      frmProgress->Show();
      frmProgress->Bar->Position = 0;
      lbShortcutsPath->Clear();

      if(shortcuts_detected > 0) {

         // *** for faster performing execution  ***

         HRESULT hres;
         IShellLink *psl;
         IPersistFile *ppf;
         char * strPath;
         strPath = (char *) calloc(MAX_PATH+1, sizeof(char));
         WIN32_FIND_DATA wfd;
         CoInitialize(NULL);

         for(int c = 0 ; c < total_item ; c++) {

            current_shortcut_file = frmConfigure->txtShortcutsDirPath->Text + "\\"
+ String(sorted_files[c]);
            path = ""; // assume invalid link

            // Open only *.lnk files
            if(GetFileExtension(current_shortcut_file) == "lnk") {

               //PROCESS

               // [Get a pointer to the IShellLink interface.]
               hres = CoCreateInstance(
CLSID_ShellLink,NULL,CLSCTX_INPROC_SERVER,IID_IShellLink,(LPVOID*)&psl);
               if(hres != S_OK) {
                  //MessageBox(Handle,"Couldn't open a pointer to IShellLink
Interface!","Error",MB_OK|MB_ICONERROR);
               } else {

                  // [Get a pointer to the IPersistFile interface.]
                  hres = psl->QueryInterface(IID_IPersistFile,(void **) &ppf);
                  if(hres != S_OK) {
                     //MessageBox(Handle,"Couldn't open a pointer to IPersistFile
Interface!","Error",MB_OK|MB_ICONERROR);
                  } else {

                     // [Load the shortcut file]
                     WCHAR wsz[MAX_PATH];
                     MultiByteToWideChar(CP_ACP, 0, current_shortcut_file.c_str(),
-1, wsz, MAX_PATH);
                     hres = ppf->Load( wsz , STGM_READ);
```

```
                    if(hres != S_OK) {
                       //MessageBox(Handle,"Load shortcut
error","Error",MB_OK|MB_ICONEXCLAMATION);
                    } else {
                       // [Resolve the link.]
                       hres = psl->Resolve(Handle, SLR_NO_UI); // SLR_NO_UI =
don't warning the user with UI !
                       if(hres != S_OK) {
                          // WARNING: INVALID LINKS !!
                          //MessageBox(Handle,"Resolve the link
error","Error",MB_OK|MB_ICONEXCLAMATION);
                       } else {
                          // [Get the path to the link target.]
                          hres = psl->GetPath( strPath, MAX_PATH,
(WIN32_FIND_DATA*)&wfd, SLGP_UNCPRIORITY);
                          if(hres != S_OK) {
                             MessageBox(Handle,"Get path
error","Error",MB_OK|MB_ICONEXCLAMATION);
                          } else {
                             // [Copy output data to GUI]
                             path = String(strPath);
                             valid++;
                             }
                          }
                       }
                    }
                 }
              }

              //PROCESS END

           if(path == "") {
              // invalid path/link/shortcut file
              // delete file entry
              memset(sorted_files[c],0,260);
           }

           lbShortcutsPath->Items->Add(path);

           frmProgress->Bar->Position = (((c+1) * 100)/(total_item));

           frmMain->Update();
           frmProgress->Update();

        }

        CoUninitialize();  // < placed here, for faster performing execution
        ppf->Release();
        psl->Release();
        free(strPath);

     }

     frmProgress->Close();
     frmMain->Enabled = true;
     frmMain->Show();
     lblValidLinks->Caption = "Valid links total : " + String(valid);

     // [Remove any directory links from list]

     // copy/replace from sorted list into raw list
     lbShortcutsFiles->Items = lbShortcutsPath->Items;
     lbShortcutsPath->Clear();

     bool file_type = false;
     int file_count = 0;

     if(shortcuts_detected > 0) {
        for(int c = 0; c < total_item ; c++) {
```

```
            current_shortcut_file = lbShortcutsFiles->Items->Strings[c];

            if(current_shortcut_file != "") { // select only valid paths !

               file_type = false;

               // Check the path is available or not, still exist or not?

               HANDLE exist;
               exist =
CreateFile(current_shortcut_file.c_str(),0,FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN
_EXISTING,FILE_ATTRIBUTE_NORMAL,0);
               if(exist != INVALID_HANDLE_VALUE) {
                  file_type = true;
               }
               CloseHandle(exist);

               if(file_type == true) {
                  // if this selected path are file, not directory, then include
it into file count
                  lbShortcutsPath->Items->Add( lbShortcutsFiles->Items->Strings[c]
);
                  file_count++;
               } else {
                  // Directory type
                  // delete file entry
                  memset(sorted_files[c],0,260);
               }
            }

         }
      }

      lblFileCount->Caption = "File count : " + String(file_count);

      // [Show again the sorted file entry list]
      lbShortcutsFiles->Clear();
      for(int c = 0 ; c < total_item ; c++) {
         if(sorted_files[c][0] != 0){ // not null
            // add files entry into list
            lbShortcutsFiles->Items->Add(String(sorted_files[c]));
         }
      }

      // [Free memory used by variables]
      for(int k = 0 ; k < total_item ; k++) {
         free(sorted_files[k]);
      }
      free(sorted_files);
      free(create_time);
      for(int k = 0 ; k < total_item ; k++){
         free(filenames[k]);
      }
      free(filenames);

      mmLog->Text = mmLog->Text + "Done.";
      mmLog->Lines->Add("");
      enumerated = true;
      total_shortcuts = lbShortcutsFiles->Count;

      // My recent documents has been cleared, no shortcuts detected!
      if(shortcuts_detected == 0) {
         //MessageBox(Handle,"Recent shortcuts directory cleared! - no
records","Error",MB_ICONERROR);
         //Add to report
         mmLog->Lines->Add("(!) Recent shortcuts directory cleared! - no
records");
         log_scroll_y++;
         enumerated = false;
         total_shortcuts = 0;
```

```
         sys_errors++;
      } else {
         // Verbose: "Enumeration has been completed!"
         if(frmConfigure->chkVerbose->Checked == true) {
            SWSendMessage("(i) Enumeration has been completed!","");
         }
      }

   } else {
      //MessageBox(Handle,"Recent shortcuts directory has not been setup
yet!","Error",MB_ICONERROR);
      //Add to report
      mmLog->Lines->Add("(!) Recent shortcuts directory has not been setup yet!");
      log_scroll_y++;
      enumerated = false;
      total_shortcuts = 0;
      sys_errors++;
   }

}

void TfrmMain::SliceShortcuts(int index)
{
   // Index starts from 1.

   //After enumeration done, this operation can proceed
   if((enumerated == true) && (index != -1)) {

      // [Slice shortcut by 3 items per index]

      // refresh shortcuts , clear files array
      for(int i = 0 ; i < 3 ; i++) {
         memset(shortcuts[i],0,260);
         memset(files[i],0,15);
      }

      char * tmp;
      tmp = (char *) calloc(11,sizeof(char));
      memset(tmp, 0, 11);

      if( (index*3) <= total_shortcuts ) { // if selected index still below
total_shortcuts then it's OK
         for(int i = ((index-1)*3) ; i < (index*3) ; i++) {
            strcpy(shortcuts[(i-((index-1)*3))],lbShortcutsPath->Items-
>Strings[i].c_str());
            files[(i-((index-1)*3))][0] = ((i-((index-1)*3)) + 1) + 48; // convert
to ASCII
            files[(i-((index-1)*3))][1] = '.';
            files[(i-((index-1)*3))][2] = ' ';

            memset(tmp,0,11);
            strncpy(tmp, lbShortcutsFiles->Items->Strings[i].c_str(), 10);
            for(int h = 0; h < 10 ; h++) {
               files[(i-((index-1)*3))][h+3] = tmp[h];
            }

            files[(i-((index-1)*3))][13] = ' ';
         }
         available_file_option_max = 3;
      } else {

         int over;
         over = total_shortcuts % 3;
         if( (over < 3) && (over > 0) ) { // if selected index is select last few
shortcuts, then it's OK too
            for(int i = ((index-1)*3) ; i < (((index-1)*3)+over) ; i++) {
               strcpy(shortcuts[(i-((index-1)*3))],lbShortcutsPath->Items-
>Strings[i].c_str());
               files[(i-((index-1)*3))][0] = ((i-((index-1)*3)) + 1) + 48; //
convert to ASCII
```

```
               files[(i-((index-1)*3))][1] = '.';
               files[(i-((index-1)*3))][2] = ' ';

               memset(tmp,0,13);
               strncpy(tmp, lbShortcutsFiles->Items->Strings[i].c_str(), 10);
               for(int h = 0; h < 10 ; h++) {
                  files[(i-((index-1)*3))][h+3] = tmp[h];
               }

               files[(i-((index-1)*3))][13] = ' ';
            }
         }
         available_file_option_max = over;

      }

      // [SEND SMS to user]
      String allfiles = String(files[0]) + String(files[1]) + String(files[2]);
      SWSendMessage(allfiles,"\t");

      free(tmp);

      listed = true;

   } else {
      //MessageBox(Handle,"Enumeration process has not been
run!","Error",MB_OK|MB_ICONEXCLAMATION);
      mmLog->Lines->Add("(!) Enumeration process has not been run!");
      log_scroll_y++;
      // Normal/Verbose:
      SWSendMessage("(!) Enumeration process has not been run!","");
      sys_errors++;
   }
}

// ============================================================================
//                              Misc. Functions
// ============================================================================

void TfrmMain::DoEvents()
{
   //process Windows message that has been queued.
   MSG msg;

   while (PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE)) {
      if (GetMessage(&msg, NULL, 0, 0)) {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      } else {
         break;
      }
   }
}

void TfrmMain::Shutdown()
{
   // Add report
   mmLog->Lines->Add("(i) Executing shutdown program...");
   log_scroll_y++;

   // Shutdown computer by calling "shutdown.exe" Windows program.
   /* Set "shutdown" process parameters:
      Delay:  30 sec.
      Desc:   "C.R.S - SMS [Shutdown process]"
   */

   int delay = 30;
   String comment = "C.R.S - SMS [Shutdown process]";
   String windir = ""; // Default;
   String file = "shutdown.exe";
```

```
   String cmdline = " -s -t " + IntToStr(delay) + " -c " + String(char(34)) +
comment + String(char(34));

   // DEBUG
   //mmLog->Lines->Add("DEBUG: Shutdown process skipped!");
   //log_scroll_y++;
   ShellExecute(0, "open", file.c_str(), cmdline.c_str(), windir.c_str(),
SW_SHOW);

}

void TfrmMain::SWSendMessage(String message, String log_prefix)
{
   // Directly process send "message" to M.P.D Translator

   // DEBUG
   //mmLog->Lines->Add(log_prefix+"DEBUG: Send Message skipped");
   //log_scroll_y++;

   // Send message/SMS much simpler by calling this function
   if(message.Length() <= 42) {

      char * characters;
      characters = (char *) calloc(message.Length()+1,sizeof(char)); // +1 for
null
      memset(characters,0,message.Length()+1);
      strncpy(characters,message.c_str(),message.Length());

      // Add to report
      mmLog->Lines->Add(log_prefix+"(>) Sending SMS : \"" + message + "\"");
      log_scroll_y++;


      // Clear last data on RAM
      HWClearStorages();

      // Set recepient number
      SWSetNumber();

      // Input message char by char
      SBar->Max = message.Length() - 1;
      for(int i = 0 ; i < message.Length() ; i++) {
         HWInputAscii( characters[i] );
         SBar->Position = i;
         frmMain->Update();
      }

      HWEncodeMessage(0x01); // set default using 7-bit

      // Send message after encoding finished!
      HWSendMessage();

      // Clear last data on RAM
      HWClearStorages();


      free(characters);

      SBar->Max = 1;
      SBar->Position = 0;

   } else {
      // Add to report
      mmLog->Lines->Add("(!) Maximum SMS length reached! -- cancelling
operation!");
      log_scroll_y++;
      mmLog->Lines->Add("\t'"+message+"'");
      log_scroll_y++;
   }
```

```
}

void TfrmMain::SWSetNumber()
{
   //Save Authorized phone number into M.P.D Translator
   char * number;
   number = (char*) calloc(frmConfigure->txtAuthorizedNumber-
>Text.Length(),sizeof(char));
   strcpy(number,frmConfigure->txtAuthorizedNumber->Text.c_str());

   SBar->Max = frmConfigure->txtAuthorizedNumber->Text.Length() - 1;
   for(int i = 0 ; i < frmConfigure->txtAuthorizedNumber->Text.Length() ; i++) {
      HWInputNumber(number[i]);
      SBar->Position = i;
      frmMain->Update();
   }

   SBar->Max = 1;
   SBar->Position = 0;

   free(number);
}

void TfrmMain::SWHWCheck()
{
   // Check for hardware availability
   // using instruction no 0x01 -- Clear Storages

   //Hardware clear recepient number phone and message

   // Add report
   mmLog->Text = mmLog->Text + "(>) Checking for hardware... ";

   int timeout = 1000;
   int counter = 0;
   bool busy = true;
   char tmp;
   unsigned char inst_code = 0x01;

   // > set instruction code
   Out32(DATA_PORT,inst_code);

   // > give clock, to confirm instruction code
   HWClock();

   // > give clock, to confirm execution
   HWClock();

   // > wait for hardware, until busy pin goes LOW
   while(busy == true) {
      tmp = Inp32(STATUS_PORT); // S4
      tmp = tmp & 0x10; // 0b0001.0000
      if(tmp != 0x10) {
         busy = false;
         hardware_connected = true;
      }
      Sleep(1);
      counter++;
      frmMain->Update();
      if(counter > timeout) {
         break;

      }
   }

   // > give clock, to exit
   HWClock();

   mmLog->Text = mmLog->Text + "Done.";
   mmLog->Lines->Add("");
```

```
   log_scroll_y++;

   if(busy == true) {
      // Hardware not responding!
      // Assumed : There is no hardware connected!
      //MessageBox(Handle,"Hardware is not
connected!","Error",MB_OK|MB_ICONERROR);

      // Add report
      mmLog->Lines->Add("(!) HARDWARE NOT DETECTED, SHUTTING DOWN SYSTEM!");
      log_scroll_y++;

   }
}

int TfrmMain::strnrcpy(char * dest, char * src, unsigned int n, unsigned int
dest_size)
{
   //String copy that started from the right of the string!

   /*
      This explains how it works
       (index)  0  1  2  3  4  5  6  7  8
      A = [S][a][y][a][ ][B][u][d][i]
  (index)  0  1  2  3  4
      B = [ ][ ][ ][ ][ ]

      B = [B][u][d][i][ ] < this is our objective

       from above,
  we know that we can copy the source chars (A) started from index 5-8
       to destination chars (B) 0 - 3.
  this explains to us, so when we want to copy from right index, always get
       the source length first! and then get from where we want to copy (index=n)
   */

   //empty the destination array 1st as we want to place them later
   memset(dest,0,dest_size);

   unsigned int dest_len = dest_size;
   unsigned int src_len = strlen(src);
   unsigned int start_index = src_len - n;  // 5,6,7,8 in the example, return 5

   if((n <= dest_len)&&(n <= src_len)) {
      for(unsigned int i = 0; i <= n; i++) { // looping n-times.
         // 5 6 7 8 [start_index+i]
         // 0 1 2 3 [i]
         dest[i] = src[start_index+i];
      }
      return 0;
   } else {
      return 1; // error!
   }


}

String TfrmMain::GetFileExtension(String filename)
{
   // Extract file extension from a filename
   // if extension is not known, function returned "<ERROR>"

   String result = "";

   bool valid = false;
   int len;
   char * temp;
   temp = (char *) calloc(filename.Length(),sizeof(char));
   memset(temp, 0, filename.Length());
   strncpy(temp, filename.c_str(), filename.Length());
```

```
   len = filename.Length() - 1;

   for(int i = len; i >= 0; i--) {
      if(temp[i] == '.') {
         valid = true;
         i = -1;
      } else {
         result = String(temp[i]) + result;
      }
   }

   if(valid == false) {
      result = "<ERROR>";
   }

   free(temp);

   return result;
}

unsigned int TfrmMain::calculateTime(int hours, int minutes, int seconds)
{
   unsigned int result; // in seconds

   result = seconds;
   result += (minutes * 60);
   result += (hours * 3600);

   return result;
}

unsigned int TfrmMain::calculateDate(int days, int months, int years)
{
   unsigned int result; // in days

   result = days;
   result += (31) * months; // set 31, because it's the maximum month length
   result += (12 * 31) * years;

   return result;
}

bool TfrmMain::FileExist(String filename)
{
   // Check for existing file.
   // if not exist, then return false.
   // if exist, then return true.
   bool result = false;
   HANDLE exist;

   exist =
CreateFile(filename.c_str(),0,FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN_EXISTING,FIL
E_ATTRIBUTE_NORMAL,0);
   if(exist != INVALID_HANDLE_VALUE) {
      result = true;
   } else {
      result = false;
   }
   CloseHandle(exist);

   return result;
}

// ============================================================================
//                           Software GUI Controls
// ============================================================================
void __fastcall TfrmMain::btnConfigureClick(TObject *Sender)
{
   general_usage = true;
```

```
   tmrStartup->Enabled = false; // at startup, this timer was running, so disable
it, to prevent any misfunction

   mmLog->Clear();
   mmLog->Lines->Add("-- Software disabled --");
   mmLog->Lines->Add("# Configuring software settings #");
   mmLog->Lines->Add("Please restart software to apply the settings...");

   frmConfigure->ShowModal();
}

void __fastcall TfrmMain::tmrLED1Timer(TObject *Sender)
{
   LED1->Color = clBtnFace;
   tmrLED1->Enabled = false;
}

void __fastcall TfrmMain::tmrUpdateStatsTimer(TObject *Sender)
{
   // Display current download URL
      lblURL->Caption = URL;

   // Display counted errors
      lblSysErrors->Caption = IntToStr(sys_errors) + "x";

   // Calculate computer uptime and display it on GUI control.
      int uptime;
      String uptime_str;
      int hours;
      String hours_str;
      int minutes;
      String minutes_str;
      int seconds;
      String seconds_str;

      uptime = GetTickCount(); // scale is in milli-seconds
      uptime = uptime / 1000; // change scale to seconds;

      hours = (uptime/3600);
      minutes = (uptime / 60) - (hours * 60);
      seconds = uptime - (minutes * 60) - (hours * 3600);

      hours_str = IntToStr(hours);
      if(hours_str.Length() < 2) {
         hours_str = "0" + hours_str;
      }
      minutes_str = IntToStr(minutes);
      if(minutes_str.Length() < 2) {
         minutes_str = "0" + minutes_str;
      }
      seconds_str = IntToStr(seconds);
      if(seconds_str.Length() < 2) {
         seconds_str = "0" + seconds_str;
      }
      uptime_str = hours_str + ":" + minutes_str + ":" + seconds_str;

      lblCompUptime->Caption = uptime_str;
}

void __fastcall TfrmMain::btnCloseSoftwareClick(TObject *Sender)
{
   CloseSoftware();
}

void __fastcall TfrmMain::FormClose(TObject *Sender, TCloseAction &Action)
{
   // This event is for computer general usage, not used by remote system.
   if(boot == true) {
      general_usage = true;
```

```
      tmrStartup->Enabled = false; // at startup, this timer was running, so
disable it, to prevent any misfunction
      CloseSoftware();
   } else {
      // Disable close event!
      Action = caNone; // don't close
   }
}

void __fastcall TfrmMain::tmrLogTimer(TObject *Sender)
{
   // Auto-scroll for mmLog
   mmLog->Perform(EM_LINESCROLL,0,log_scroll_y);

   // Check for maximum lines length
   if(frmConfigure->chkClearLog->Checked == true) {
      if(log_scroll_y > StrToIntDef(frmConfigure->txtClearCount->Text,100)) { //
if wrong input, revert to 100
         mmAllLog->Lines->Add(mmLog->Text);
         mmLog->Clear();
         log_scroll_y = 0;
      }
   }

}

void __fastcall TfrmMain::btnAutoResponseClick(TObject *Sender)
{
   tmrAutoResponse->Enabled = ! tmrAutoResponse->Enabled;
   if(tmrAutoResponse->Enabled == true) {

      btnAutoResponse->Caption = "Sys Auto Response : [ON]";
      mmLog->Lines->Add("(>) Self Response System activated");
      log_scroll_y++;
   } else {
      btnAutoResponse->Caption = "Sys Auto Response : [OFF]";
      mmLog->Lines->Add("(!) WARNING: Self Response System inactive!");
      log_scroll_y++;
   }
}

void __fastcall TfrmMain::tmrHWHeartbeatTimer(TObject *Sender)
{
   // GUI LED
   if(LED2->Color == clRed) {
      LED2->Color = clBtnFace;
   } else {
      LED2->Color = clRed;
   }

   // Generate signal
   if(port_open == true) {
      ToggleHeartbeat();
   }
}


//===========================================================================
//                              DEBUGING CONTROLS
//===========================================================================
void __fastcall TfrmMain::btnDebugClick(TObject *Sender)
{
   general_usage = true;
   tmrStartup->Enabled = false; // at startup, this timer was running, so disable
it, to prevent any misfunction
   //CloseSoftware();

   // Open computer I/O Port
   // get settings first before open any port
```

```
      switch(frmConfigure->cmbPort->ItemIndex) {
         case 0:
            DATA_PORT = 0x378;
         break;
         case 1:
            DATA_PORT = 0x278;
         break;
         case 2:
            //Custom Port
            DATA_PORT = StrToIntDef(frmConfigure->txtCustomPort->Text,0x378); //
wrong input, revert to 0x378
         break;
      }

      STATUS_PORT = DATA_PORT + 1;
      CONTROL_PORT = DATA_PORT + 2;
      // open now
      OpenPort();

   mmLog->Clear();

   int choose;
   choose = MessageBox(Handle,"Set as Output?","PORT",MB_YESNO|MB_ICONQUESTION);
   if(choose == ID_YES) {
      SetPortAsOutput();
   } else {
      SetPortAsInput();
   }
}
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
#include <shlobj.h> // Open Directory Dialog library
#include <vcl.h>
#pragma hdrstop

#include "code_frmMain.h"
#include "code_frmConfigure.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmConfigure *frmConfigure;

bool mail_settings_change = false;

//---------------------------------------------------------------------------
__fastcall TfrmConfigure::TfrmConfigure(TComponent* Owner)
   : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TfrmConfigure::FormCreate(TObject *Sender)
{
   // Define items in cmbPort combobox
   cmbPort->Items->Add("LPT-1 : 0x378");
   cmbPort->Items->Add("LPT-2 : 0x278");
   cmbPort->Items->Add("Custom port, define your own");
   // Select an index number as default setting
   cmbPort->ItemIndex = 0;
}
//---------------------------------------------------------------------------
void __fastcall TfrmConfigure::cmbPortChange(TObject *Sender)
{
   if(cmbPort->ItemIndex == 2) {
      // "Custom port, define your own" selected...
      // Enable user set input to custom port textbox
      txtCustomPort->Enabled = true;
   } else {
      // Disable enter custom port textbox
      txtCustomPort->Enabled = false;
      txtCustomPort->Text = "";
   }
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::btnOKClick(TObject *Sender)
{
   ValidateDownloaderPaths();
   frmMain->SaveConfig();

   if(mail_settings_change == true) {
      // Install email settings on the software (blat)
      frmMain->SettingMail();

      // Clear the sensitive data from program
      frmConfigure->txtUsername->Clear();
      frmConfigure->txtPassword->Clear();
   }

   Close();
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::btnCancelClick(TObject *Sender)
{
   ValidateDownloaderPaths();
   Close();
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::btnDefaultsClick(TObject *Sender)
```

```
{
   int ask;
   ask = MessageBox(Handle,"Use default configuration?\nWARNING: All settings will
be LOST!\nProceed?","Default configuration",MB_YESNO|MB_ICONEXCLAMATION);
   if(ask == ID_YES) {
      frmMain->DefaultConfig();
   }
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::btnProgramLogSaveToClick(TObject *Sender)
{
   SaveDlg->Title = "Save program log file as ...";
   if(SaveDlg->Execute()) {
      txtProgramLog->Text = SaveDlg->FileName;
   }
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::btnDownloaderSaveLogClick(TObject *Sender)
{
   SaveDlg->Title = "Save downloader log file as ...";
   if(SaveDlg->Execute()) {
      txtDownloaderLogPath->Text = SaveDlg->FileName;
   }
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::btnDownloaderSavePathClick(TObject *Sender)
{
   String dir = OpenDirDlg();
   if(dir != "") {
      txtDownloaderSavePath->Text = dir;
   }
}
//---------------------------------------------------------------------------


void TfrmConfigure::ValidateDownloaderPaths()
{
   char * validate;
   validate = (char*) calloc( txtDownloaderSavePath-
>Text.Length()+1,sizeof(char));
   memset(validate,0,txtDownloaderSavePath->Text.Length()+1);

   strcpy(validate,txtDownloaderSavePath->Text.c_str());

   if(txtDownloaderSavePath->Text.Length() > 3) { // avoid X:\ path
      if(validate[txtDownloaderSavePath->Text.Length()-1] == '\\') {
         validate[txtDownloaderSavePath->Text.Length()-1] = 0;
         txtDownloaderSavePath->Text = String(validate);
      }
   }

   free(validate);
}
void __fastcall TfrmConfigure::FormShow(TObject *Sender)
{
   mail_settings_change = false;
   shpMod->Visible = false;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtEmailProgramPathChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TfrmConfigure::txtUsernameChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtPasswordChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtSMTPServerChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtSSLPortChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtSenderAddressChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtRecepientAddressChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmConfigure::txtMailSubjectChange(TObject *Sender)
{
   mail_settings_change = true;
   shpMod->Visible = true;
}
//---------------------------------------------------------------------------

String TfrmConfigure::OpenDirDlg()
{
   // Show open directory dialog

   String strAddress = "";
   char * address;
   address = (char*) calloc(MAX_PATH+1, sizeof(char));
   memset(address, 0, MAX_PATH+1);

   _browseinfoA bi;
   _ITEMIDLIST *retHndl;
   bool retFunc;

   bi.hwndOwner = Handle;
   bi.pidlRoot = 0;
   bi.pszDisplayName = address;
   bi.lParam = 0;
   bi.lpszTitle = "";
   bi.ulFlags = 0;
   bi.lpfn = 0;
   bi.iImage = 0;
```

```
   retHndl = SHBrowseForFolder(&bi);

   if(retHndl != NULL) {
      retFunc = SHGetPathFromIDList(retHndl, address);
      if(retFunc != false) {
         strAddress = String(address);
      } else {
         strAddress = "";
      }
   }

   free(address);

   return strAddress;

}
void __fastcall TfrmConfigure::btnOpenDwnldListClick(TObject *Sender)
{
   OpenDlg->Title = "Open download list file";
   if(OpenDlg->Execute()) {
      txtDownloadListPath->Text = OpenDlg->FileName;
   }
}
//---------------------------------------------------------------------------
```