**MINISTERUL EDUCATIEI REPUBLICII MOLDOVA UNIVERSITATEA TEHNICA A MOLDOVEI**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Inginerie Software si Automatica**

RAPORT

**Programarea în rețea**

Lucrare de laborator Nr. 4

Tema: Protocolul SMTP de transfer mesaje electronice. Comenzile protocolului SMTP. Protocolul POP3. Comenzile POP3 de autorizare, tranzacţie şi actualizare. Componente .Net/Java/MFC pentru elaborare client de poştă electronică

A elaborat: st. gr. TI-142 Chifa Vladislav

A verificat: lector asistent Ostapenco Stepan

Chișinău 2017

# Scopul lucrării

# Înțelegerea sistemului de poștă electronică și studiul acestuia în baza a două protocoale SMTP și POP3. Scopul costînd în crearea unei aplicații de poștă electronică pentru trimiterea de mesaje electonice și citirea acestor din cutia poștală a utilizatorului.

# <https://github.com/tarakan-x/PR_Lab>

# Protocolul SMTP

SMTP (Simple Mail Transfer Protocol) este un protocol simplu, folosit pentru transmiterea mesajelor în format electronic pe Internet.

Protocolul SMTP specifică modul în care mesajele de poştă electronică sunt transferate între procese SMTP aflate pe sisteme diferite. Procesul SMTP care  transmite un mesaj este numit client SMTP, iar procesul SMTP care primeşte mesajul este numit server SMTP.

Protocolul nu se referă la modul în care mesajul ce urmează a fi transmis este trecut de la utilizator către clientul SMTP, sau cum mesajul ce urmează a fi recepţionat de serverul SMTP este livrat destinatarului, nici la modul în care este memorat mesajul şi nici de câte ori clientul SMTP încearcă să transmită mesajul.

Obiectivul protocolului SMTP este de a trimite mail-uri într-un mod eficient. El este independent de sistemele care participă la comunicaţie, dacă se asigură un canal prin care datele să fie transmise într-un mod ordonat .

SMTP foloseşte următorul model de comunicaţie: transmiţătorul, ca urmare a unei cereri de transmisie a mail-ului, stabileşte o legătură bidirecţională cu receptorul, care poate fi destinatarul final al mail-ului sau doar un intermediar. De aceea  este necesar să se precizeze numele de *host al destin*aţiei finale precum şi utilizatorul căruia îi este destinat mesajul.

Un server SMTP trebuie să cunoască cel puţin următoarele comenzi :

* HELO - identificare computer expeditor;
* EHLO - identificare computer expeditor cu cerere de mod extins;
* MAIL FROM - specificare expeditorului;
* RCPT TO - specificarea destinatarului ;
* DATA - conţinutul mesajului;
* RSET – Reset;
* QUIT - termină sesiunea;
* HELP - ajutor pentru comenzi;
* VRFY – verificare o adresa;

**Protocolul POP3**

POP3-Post Office Protocol Version 3– protocol folosit pentru citirea mail-urilor. Permite utilizatorilor să se conecteze printr-o conexiune TCP pe portul 110 (default) la server și să descarce mesajele, după care să întrerupă conexiunea și să le citească off-line.

*Avantejele sistemului POP:*

Mesajele sunt afișate foarte repede după ce sunt descărcate de pe server. După ce sunt descărcate mesajele se gasesc pe calculatorul personal. Pe majoritatea serverelor spațiul este limitat, deoarece mesajele sunt descărcate pe calculator spațiul pentru mesaje este limitat doar de hard-disk-ul propriu, nu este limitat de spațiul pe care îl aveți pe server.

Toți clienții de postă electronică suportă acest protocol.

*Dezavantajele protocolului POP:*

Trebuie folosit un program pentru a descărca mesajele de pe server.

Mesajele sunt stocate pe calculatorul propriu și nu sunt accesibile de la alte calculatoare.

Mesajele trimise si mesajele în curs de scriere nu sunt accesibile de la alte calculatoare.

Mesajele sunt șterse de pe server și dacă aveți o problemă cu calcualtorul este posibil să pierdeți mesajele.

Mesajele sunt stocate în fișiere care nu sunt compatibile cu toți clienții de poștă electronică, dacă doriți să schimbați clientul, poate fi dificil să recuperați mesajele anterioare**.**

**Aplicatia**

În figura 1 este reprezentată fereastra aplicației pentru crearea unui mesaj , unde in prima secțiune setăm smtp server-ul (***smtp.gmail.com****)* ,port-ul lui ***587*** și facem conexiunea prin introducerea poștei noastre electronice si a parolei .Apoi in sectiunea a 2-a setem expeditorul și receptorul după care introducem subiectul mesajului nostru și însăși mesaju nostru .

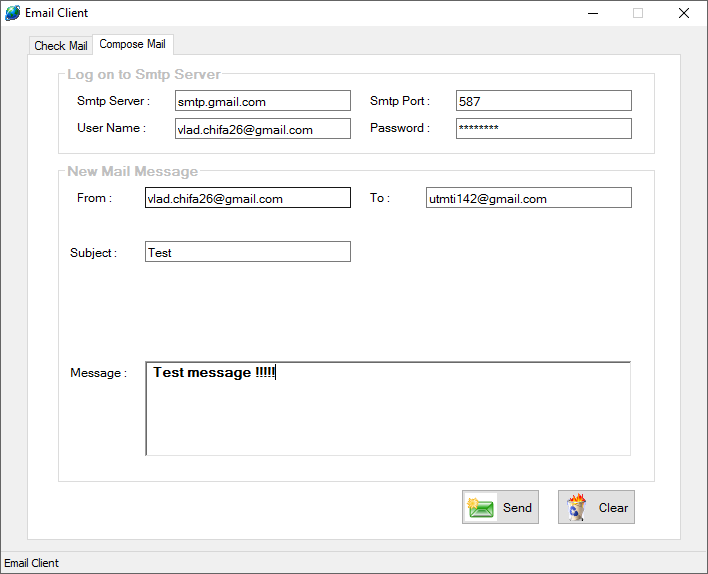


Figura 1-Setarea conexiunei cu serverul ***smtp*** și expedierea mesajului .

În figura 2 este prezentată interfața ne conectăm la serverul ***pop.gmail.com***  pentru a descărca mesajele primite .Dupăcum observăm din imagine , mesajul nostru a fost trimis cu succes și în secțiunea de jos putem vedea cine este expeditorul ,subiectul și conținutul mesajului primit.

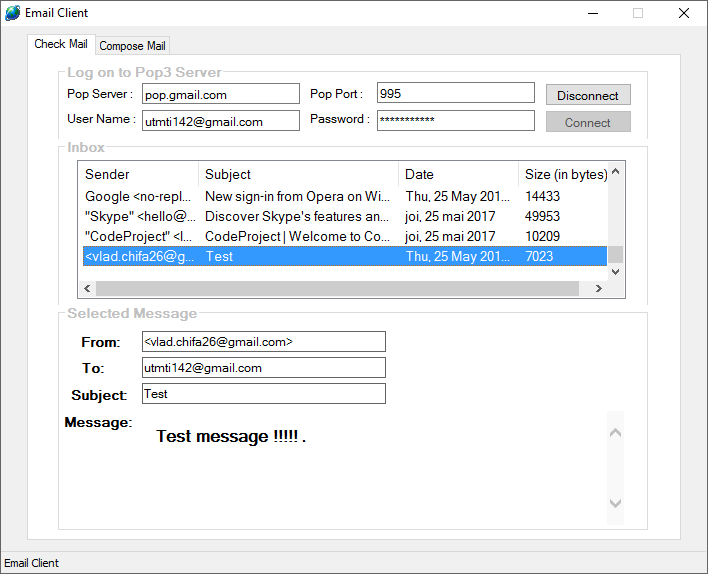


Figura 2 – Conectarea la POP server .

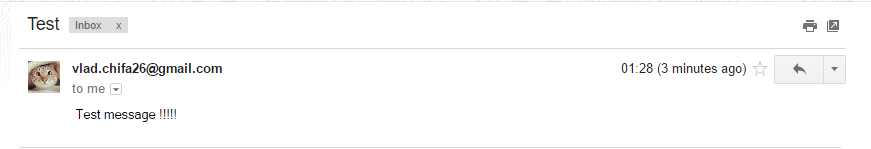


Figura 3 - Mesajul primit pe poștă.

**Concluzie:**

Efectuînd această lucrare de laborator am creat o aplicatie Email Client care foloseste protocolul SMTP pentru a efectua o expediere a unui mesaj și folosirea protocolului POP3 pentru descarcara si citirea mesajului trimis , aceste mesaje dupa descarcare se salveaza pe calculatorul propriu .

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Text.RegularExpressions;

using System.Collections;

using System.IO;

using System.Threading;

namespace Email\_Client

{

public partial class EmailClient : Form

{

bool \_lock = false;

Pop3Client pop = null;

string email = "";

int msg\_id = 0;

public EmailClient()

{

InitializeComponent();

}

private void EmailTimer\_Tick(object sender, EventArgs e)

{

if (SmtpServer.ContainsFocus)

this.SmtpServer.BackColor = Color.Ivory;

else

this.SmtpServer.BackColor = SystemColors.Window;

if (SmtpPort.ContainsFocus)

this.SmtpPort.BackColor = Color.Ivory;

else

this.SmtpPort.BackColor = SystemColors.Window;

if (UserName.ContainsFocus)

this.UserName.BackColor = Color.Ivory;

else

this.UserName.BackColor = SystemColors.Window;

if (Password.ContainsFocus)

this.Password.BackColor = Color.Ivory;

else

this.Password.BackColor = SystemColors.Window;

if (PopServer.ContainsFocus)

this.PopServer.BackColor = Color.Ivory;

else

this.PopServer.BackColor = SystemColors.Window;

if (PopPort.ContainsFocus)

this.PopPort.BackColor = Color.Ivory;

else

this.PopPort.BackColor = SystemColors.Window;

if (PopUserName.ContainsFocus)

this.PopUserName.BackColor = Color.Ivory;

else

this.PopUserName.BackColor = SystemColors.Window;

if (PopPassword.ContainsFocus)

this.PopPassword.BackColor = Color.Ivory;

else

this.PopPassword.BackColor = SystemColors.Window;

if (From.ContainsFocus)

this.From.BackColor = Color.Ivory;

else

this.From.BackColor = SystemColors.Window;

if (To.ContainsFocus)

this.To.BackColor = Color.Ivory;

else

this.To.BackColor = SystemColors.Window;

if (Subject.ContainsFocus)

this.Subject.BackColor = Color.Ivory;

else

this.Subject.BackColor = SystemColors.Window;

if (FromPopHeader.ContainsFocus)

this.FromPopHeader.BackColor = Color.Ivory;

else

this.FromPopHeader.BackColor = SystemColors.Window;

if (ToPopHeader.ContainsFocus)

this.ToPopHeader.BackColor = Color.Ivory;

else

this.ToPopHeader.BackColor = SystemColors.Window;

if (SubjectPopHeader.ContainsFocus)

this.SubjectPopHeader.BackColor = Color.Ivory;

else

this.SubjectPopHeader.BackColor = SystemColors.Window;

}

private void SmtpPort\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsDigit(e.KeyChar) && e.KeyChar != (char)8)

{

e.Handled = true;

}

}

private void Send\_Click(object sender, EventArgs e)

{

if (this.CheckInputValidation(SmtpServer.Text, SmtpPort.Text, UserName.Text, Password.Text, From.Text , To.Text,Text,Text))

{

if (this.EmailValidation(this.From.Text))

{

bool isRecipient = false;

if (this.To.Text.Length > 0)

{

if (this.RecipientsEmailValidation(this.To.Text))

{

isRecipient = true;

}

else

{

MessageBox.Show(this, "Recipients' email address is not in the correct format, in \"To: \" field.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

}

if (Internet.IsConnectedToInternet())

{

if (isRecipient == true)

{

Rtf2Html rtf = new Rtf2Html();

string Html = rtf.ConvertRtfToHtml(this.MailMessage);

MailMessage mail\_message = new MailMessage();

mail\_message.From = this.From.Text;

mail\_message.To = this.To.Text;

mail\_message.Subject = this.Subject.Text;

mail\_message.MailType = MailEncodingType.HTML;

mail\_message.MailPriority = MailSendPriority.NORMAL;

mail\_message.Message = Html;

Thread thread = new Thread(new ParameterizedThreadStart(this.SendEmail));

thread.Start(mail\_message);

}

else

{

return;

}

}

else

{

MessageBox.Show(this, "You must connect to the internet.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

else

{

MessageBox.Show(this, "Sender email address is not in the correct format.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void SmtpClear\_Click(object sender, EventArgs e)

{

this.SmtpServer.Text = "";

this.SmtpPort.Text = "";

this.UserName.Text = "";

this.Password.Text = "";

this.From.Text = "";

this.To.Text = "";

this.Subject.Text = "";

this.MailMessage.Text = "";

}

void smtp\_Disconnected(object sender, string Server)

{

if (this.statusStrip.InvokeRequired)

{

DisconnectEventHandler discon = new DisconnectEventHandler(this.smtp\_Disconnected);

this.Invoke(discon, new object[] { sender,Server });

}

else

{

this.ProgressLabel.Text = "Disconnected with the smtp server \"" + Server + "\"";

Thread.Sleep(500);

this.Send.Enabled = true;

this.ProgressLabel.Text = "Email Client";

}

}

void smtp\_EndedDataTransfer(object sender)

{

if (this.statusStrip.InvokeRequired)

{

DataTransferEventHandler data = new DataTransferEventHandler(this.smtp\_EndedDataTransfer);

this.Invoke(data, new object[] { sender });

}

else

{

this.ProgressLabel.Text = "Email message has sent";

}

}

void smtp\_StartedDataTransfer(object sender)

{

if (this.statusStrip.InvokeRequired)

{

DataTransferEventHandler data = new DataTransferEventHandler(this.smtp\_StartedDataTransfer);

this.Invoke(data, new object[] { sender});

}

else

{

this.ProgressLabel.Text = "Sending email message";

}

}

void smtp\_AuthenticationFinished(object sender, string userName)

{

if (this.statusStrip.InvokeRequired)

{

AuthenticateEventHandler auth = new AuthenticateEventHandler(this.smtp\_AuthenticationFinished);

this.Invoke(auth, new object[] { sender, userName });

}

else

{

this.ProgressLabel.Text = "Verification is completed";

}

}

void smtp\_AuthenticationBegan(object sender, string userName)

{

if (this.statusStrip.InvokeRequired)

{

AuthenticateEventHandler auth = new AuthenticateEventHandler(this.smtp\_AuthenticationBegan);

this.Invoke(auth, new object[] { sender, userName});

}

else

{

this.ProgressLabel.Text = "Verifying user name and password";

}

}

void smtp\_ConnectionEstablishing(object sender, string Server, int Port)

{

if (this.statusStrip.InvokeRequired)

{

ConnectEventHandler con = new ConnectEventHandler(this.smtp\_ConnectionEstablishing);

this.Invoke(con, new object[] { sender, Server, Port });

}

else

{

this.ProgressLabel.Text = "Connecting to smtp server \"" + Server + "\" on port " + Port;

}

}

void smtp\_ConnectionEstablished(object sender, string Server, int Port)

{

if (this.statusStrip.InvokeRequired)

{

ConnectEventHandler con = new ConnectEventHandler(this.smtp\_ConnectionEstablished);

this.Invoke(con, new object[] { sender, Server, Port });

}

else

{

this.ProgressLabel.Text = "Connection is established with the smtp server \"" + Server + "\"";

}

}

private void PopPort\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsDigit(e.KeyChar) && e.KeyChar != (char)8)

{

e.Handled = true;

}

}

private void Connect\_Click(object sender, EventArgs e)

{

if (this.CheckInputValidationForPop(this.PopServer.Text, this.PopPort.Text, this.PopUserName.Text, this.PopPassword.Text))

{

if (Internet.IsConnectedToInternet())

{

Thread th = new Thread(new ThreadStart(this.ReceiveEmails));

th.Start();

}

else

{

MessageBox.Show(this, "You must connect to the internet.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

private void pop\_Disconnected(object sender, string Server)

{

if (this.statusStrip.InvokeRequired)

{

DisconnectEventHandler discon = new DisconnectEventHandler(this.pop\_Disconnected);

this.Invoke(discon, new object[] { sender, Server });

}

else

{

this.ProgressLabel.Text = "Disconnected with the pop server \"" + Server + "\"";

Thread.Sleep(500);

this.Send.Enabled = true;

this.ProgressLabel.Text = "Email Client";

}

}

private void pop\_EndedDataReceiving(object sender)

{

if (this.statusStrip.InvokeRequired)

{

DataReceivingEventHandler data = new DataReceivingEventHandler(this.pop\_EndedDataReceiving);

this.Invoke(data, new object[] { sender });

}

else

{

this.ProgressLabel.Text = "Email message has received";

}

}

private void pop\_StartedDataReceiving(object sender)

{

if (this.statusStrip.InvokeRequired)

{

DataReceivingEventHandler data = new DataReceivingEventHandler(this.pop\_StartedDataReceiving);

this.Invoke(data, new object[] { sender });

}

else

{

this.ProgressLabel.Text = "Receiving email message";

}

}

private void pop\_AuthenticationFinished(object sender, string userName)

{

if (this.statusStrip.InvokeRequired)

{

AuthenticateEventHandler auth = new AuthenticateEventHandler(this.pop\_AuthenticationFinished);

this.Invoke(auth, new object[] { sender, userName });

}

else

{

this.ProgressLabel.Text = "Verification is completed";

}

}

private void pop\_AuthenticationBegan(object sender, string userName)

{

if (this.statusStrip.InvokeRequired)

{

AuthenticateEventHandler auth = new AuthenticateEventHandler(this.pop\_AuthenticationBegan);

this.Invoke(auth, new object[] { sender, userName });

}

else

{

this.ProgressLabel.Text = "Verifying user name and password";

}

}

private void pop\_ConnectionEstablished(object sender, string Server, int Port)

{

if (this.statusStrip.InvokeRequired)

{

ConnectEventHandler con = new ConnectEventHandler(this.pop\_ConnectionEstablished);

this.Invoke(con, new object[] { sender, Server, Port });

}

else

{

this.ProgressLabel.Text = "Connection is established with the pop server \"" + Server + "\"";

}

}

private void pop\_ConnectionEstablishing(object sender, string Server, int Port)

{

if (this.statusStrip.InvokeRequired)

{

ConnectEventHandler con = new ConnectEventHandler(this.pop\_ConnectionEstablishing);

this.Invoke(con, new object[] { sender, Server, Port });

}

else

{

this.ProgressLabel.Text = "Connecting to pop server \"" + Server + "\" on port " + Port;

}

}

private void Disconnect\_Click(object sender, EventArgs e)

{

if (this.pop != null)

{

DialogResult result = MessageBox.Show(this, "Do you want to disconnect with the pop server \"" + this.pop.Pop3Server + "\" ?", "Email Client", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.Yes)

{

try

{

this.pop.Disconnect();

this.MailMessages.Items.Clear();

this.FromPopHeader.Text = "";

this.ToPopHeader.Text = "";

this.SubjectPopHeader.Text = "";

this.PopMessage.DocumentText = "<html></html>";

MessageBox.Show(this, "You are disconnected with the pop server " + this.pop.Pop3Server + ".", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.EnableDisableConnectButton(true);

this.EnableDisableDisconnectButton(false);

}

catch (Pop3ClientException err)

{

MessageBox.Show(this, err.ErrorMessage, "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

private void MailMessages\_MouseClick(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right && this.MailMessages.SelectedItems.Count > 0)

{

ContextMenuStrip menu = new ContextMenuStrip();

Image image = (Image) Email\_Client.Properties.Resources.delete;

menu.Items.Add("Delete",image);

menu.ItemClicked += new ToolStripItemClickedEventHandler(menu\_ItemClicked);

menu.Show(Control.MousePosition);

}

else if (e.Button == MouseButtons.Left && this.MailMessages.SelectedItems.Count > 0)

{

int index = this.MailMessages.SelectedItems[0].Index;

index = index + 1;

this.email = "";

this.msg\_id = index;

this.FromPopHeader.Text = "";

this.ToPopHeader.Text = "";

this.SubjectPopHeader.Text = "";

this.PopMessage.DocumentText = "<html></html>";

Thread th = new Thread(new ThreadStart(this.FetchEmailCallBack));

th.Start();

}

}

private void menu\_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

{

DialogResult result = MessageBox.Show(this, "Do you want to delete the selected message?", "Email Client", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.Yes)

{

int index = this.MailMessages.SelectedItems[0].Index;

index = index + 1;

try

{

this.pop.DeleteEmail(index);

this.MailMessages.SelectedItems[0].Remove();

if (this.msg\_id == index)

{

this.FromPopHeader.Text = "";

this.ToPopHeader.Text = "";

this.SubjectPopHeader.Text = "";

this.PopMessage.DocumentText = "<html></html>";

}

MessageBox.Show(this, "Message having id " + index + " is deleted.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

catch (Pop3ClientException err)

{

MessageBox.Show(this, err.ErrorMessage, "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void MailMessages\_SelectedIndexChanged(object sender, EventArgs e)

{

if (!(this.MailMessages.SelectedItems.Count > 0))

{

this.FromPopHeader.Text = "";

this.ToPopHeader.Text = "";

this.SubjectPopHeader.Text = "";

this.PopMessage.DocumentText = "<html></html>";

}

}

// The delegate used to enable or disable a button control, in another thread

private delegate void EnableDisableEventHandler(bool enable);

private void EnableDisableSendButton(bool enable)

{

if (this.Connect.InvokeRequired)

{

EnableDisableEventHandler obj = new EnableDisableEventHandler(this.EnableDisableSendButton);

this.Invoke(obj, new object[] { enable });

}

else

{

this.Send.Enabled = enable;

}

}

private void EnableDisableConnectButton(bool enable)

{

if (this.Connect.InvokeRequired)

{

EnableDisableEventHandler obj = new EnableDisableEventHandler(this.EnableDisableConnectButton);

this.Invoke(obj, new object[] { enable });

}

else

{

this.Connect.Enabled = enable;

}

}

private void EnableDisableDisconnectButton(bool enable)

{

if (this.Connect.InvokeRequired)

{

EnableDisableEventHandler obj = new EnableDisableEventHandler(this.EnableDisableDisconnectButton);

this.Invoke(obj, new object[] { enable });

}

else

{

this.Disconnect.Enabled = enable;

}

}

// The delegate used to insert an item in the Inbox ListView

private delegate void InboxItemEventHandler(ListViewItem item);

private void InsertItem(ListViewItem item)

{

if (this.MailMessages.InvokeRequired)

{

InboxItemEventHandler obj = new InboxItemEventHandler(this.InsertItem);

this.Invoke(obj, new object[] { item });

}

else

{

this.MailMessages.Items.Add(item);

}

}

// The delegate used to write Pop Message in browser

private delegate void WritePopMessageEventHandler(string content);

private void WritePopMessage(string content)

{

if (this.PopMessage.InvokeRequired)

{

WritePopMessageEventHandler obj = new WritePopMessageEventHandler(this.WritePopMessage);

this.Invoke(obj, new object[] { content });

}

else

{

this.PopMessage.DocumentText = content.Trim();

}

}

// The delegate used to update Status Bar

private delegate void UpdateStatusBarEventHandler(string text);

private void UpdateStatusBar(string text)

{

if (this.statusStrip.InvokeRequired)

{

UpdateStatusBarEventHandler obj = new UpdateStatusBarEventHandler(this.UpdateStatusBar);

this.Invoke(obj, new object[] { text });

}

else

{

this.ProgressLabel.Text = text;

}

}

// The delegate used to update Pop message header

private delegate void UpdatePopMessageHeaderEventHandler(string from, string to, string subject);

private void UpdatePopMessageHeader(string from, string to, string subject)

{

if (this.FromPopHeader.InvokeRequired || this.ToPopHeader.InvokeRequired || this.SubjectPopHeader.InvokeRequired)

{

UpdatePopMessageHeaderEventHandler obj = new UpdatePopMessageHeaderEventHandler(this.UpdatePopMessageHeader);

this.Invoke(obj, new object[] {from,to,subject });

}

else

{

this.FromPopHeader.Text = from.Trim();

this.ToPopHeader.Text = to.Trim();

this.SubjectPopHeader.Text = subject.Trim();

}

}

// Helping methods for Pop3 Client user Interface

private void ReceiveEmails()

{

try

{

Pop3Client pop\_client = new Pop3Client();

pop\_client.Pop3Server = this.PopServer.Text;

pop\_client.Pop3Port = Convert.ToInt32(this.PopPort.Text);

pop\_client.UserName = this.PopUserName.Text;

pop\_client.Password = this.PopPassword.Text;

this.EnableDisableConnectButton(false);

pop\_client.ConnectionEstablishing += new ConnectEventHandler(this.pop\_ConnectionEstablishing);

pop\_client.ConnectionEstablished += new ConnectEventHandler(this.pop\_ConnectionEstablished);

pop\_client.AuthenticationBegan += new AuthenticateEventHandler(this.pop\_AuthenticationBegan);

pop\_client.AuthenticationFinished += new AuthenticateEventHandler(this.pop\_AuthenticationFinished);

pop\_client.StartedDataReceiving += new DataReceivingEventHandler(this.pop\_StartedDataReceiving);

pop\_client.EndedDataReceiving += new DataReceivingEventHandler(this.pop\_EndedDataReceiving);

pop\_client.Disconnected += new DisconnectEventHandler(this.pop\_Disconnected);

this.pop = pop\_client;

pop\_client.Connect();

this.EnableDisableDisconnectButton(true);

pop\_client.GetMailBoxDetails(); //it sets the TotalEmails and TotalEmailSize properties

if (pop\_client.TotalEmails >= 1)

{

this.UpdateStatusBar("Retrieving Emails");

this.FillInboxListView(pop\_client);

this.UpdateStatusBar("Email Client");

}

else

{

this.UpdateStatusBar("Email Client");

MessageBox.Show(this, "No email message exists in the inbox!.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (SmtpClientException err)

{

MessageBox.Show(this, err.ErrorMessage, "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

this.UpdateStatusBar("Email Client");

this.EnableDisableConnectButton(true);

this.EnableDisableDisconnectButton(false);

}

}

private void FetchEmailCallBack()

{

try

{

this.email = this.pop.FetchEmail(this.msg\_id);

this.UpdatePopMessageHeader(this.pop.From, this.pop.To, this.pop.Subject);

string content = "";

string content\_type = "";

string attached\_file\_name = "";

bool isHtmlIncluded = false;

int plain\_text\_message\_section = -1;

for (int i = 1; i <= this.pop.MailSections; i++)

{

this.pop.GetMailSection(i, ref content, ref content\_type, ref attached\_file\_name);

if (content\_type.ToLower().Equals("text/html"))

{

this.WritePopMessage(content);

isHtmlIncluded = true;

}

else if (content\_type.ToLower().Equals("base64"))

{

ListViewItem item = new ListViewItem(attached\_file\_name);

item.ImageIndex = 0;

}

else if (content\_type.ToLower().Equals("text/plain"))

{

plain\_text\_message\_section = i;

}

}

// this.InsertPopAttachedFileNames(attached\_file\_names);

if (isHtmlIncluded == false && plain\_text\_message\_section != -1)

{

this.pop.GetMailSection(plain\_text\_message\_section, ref content, ref content\_type, ref attached\_file\_name);

this.WritePopMessage(content);

}

this.UpdateStatusBar("Email Client");

}

catch (Pop3ClientException err)

{

MessageBox.Show(this, err.ErrorMessage, "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private bool CheckInputValidationForPop(string pop\_server, string pop\_port, string user\_name, string password)

{

if (pop\_server.Equals(""))

{

MessageBox.Show(this, "You must provide pop server address.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (pop\_port.Equals(""))

{

MessageBox.Show(this, "You must provide pop port number.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (user\_name.Equals(""))

{

MessageBox.Show(this, "You must provide username.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (password.Equals(""))

{

MessageBox.Show(this, "You must provide password.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

return true;

}

private void FillInboxListView(Pop3Client obj)

{

ArrayList sender = new ArrayList();

ArrayList subject = new ArrayList();

ArrayList date = new ArrayList();

ArrayList size = new ArrayList();

DateTime date\_time; string temp = "";

this.MailMessages.Items.Clear();

for (int i = 1; i <= obj.TotalEmails; i++)

{

string emailHeader = obj.FetchEmailTop(i, 0);

sender.Add(obj.From);

subject.Add(obj.Subject);

try

{

date\_time = DateTime.Parse(obj.Date);

temp = date\_time.ToString("D");

date.Add(temp);

}

catch(Exception)

{

date.Add(obj.Date);

}

size.Add(obj.GetMailSize(i));

}

for (int j = 0; j < sender.Count; j++)

{

ListViewItem item = new ListViewItem();

item.Text = sender[j].ToString();

item.SubItems.Add(subject[j].ToString());

item.SubItems.Add(date[j].ToString());

item.SubItems.Add(size[j].ToString());

this.InsertItem(item);

}

}

// Helping Methods for Smtp Client user interface

private void SendEmail(object mail\_msg)

{

try

{

MailMessage mail\_message = (MailMessage)mail\_msg;

SmtpClient smtp = new SmtpClient(this.SmtpServer.Text, Convert.ToInt32(this.SmtpPort.Text));

smtp.UserName = this.UserName.Text;

smtp.Password = this.Password.Text;

this.EnableDisableSendButton(false);

smtp.ConnectionEstablishing += new ConnectEventHandler(smtp\_ConnectionEstablishing);

smtp.ConnectionEstablished += new ConnectEventHandler(smtp\_ConnectionEstablished);

smtp.AuthenticationBegan += new AuthenticateEventHandler(smtp\_AuthenticationBegan);

smtp.AuthenticationFinished += new AuthenticateEventHandler(smtp\_AuthenticationFinished);

smtp.StartedDataTransfer += new DataTransferEventHandler(smtp\_StartedDataTransfer);

smtp.EndedDataTransfer += new DataTransferEventHandler(smtp\_EndedDataTransfer);

smtp.Disconnected += new DisconnectEventHandler(smtp\_Disconnected);

smtp.SendMail(mail\_message);

// MessageBox.Show(this,"Email message has sent.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (SmtpClientException obj)

{

// MessageBox.Show(this, obj.ErrorMessage, "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Error);

this.EnableDisableSendButton(true);

this.ProgressLabel.Text = "Email Client";

}

}

private bool CheckInputValidation(string smtp\_server, string smtp\_port, string user\_name, string password, string from, string to,string cc,string bcc)

{

if (smtp\_server.Equals(""))

{

MessageBox.Show(this, "You must provide smtp server address.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (smtp\_port.Equals(""))

{

MessageBox.Show(this, "You must provide smtp port number.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (user\_name.Equals(""))

{

MessageBox.Show(this, "You must provide username.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (password.Equals(""))

{

MessageBox.Show(this, "You must provide password.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

else if (from.Equals(""))

{

MessageBox.Show(this, "You must provide sender email address.", "Email Client", MessageBoxButtons.OK, MessageBoxIcon.Information);

return false;

}

return true;

}

private bool RecipientsEmailValidation(string recipient)

{

string[] splits = recipient.Split(new char[] { ',', ';' });

for (int i = 0; i < splits.Length; i++)

{

if (this.EmailValidation(splits[i]))

{

continue;

}

else

{

return false;

}

}

return true;

}

private bool EmailValidation(string email)

{

Regex regx = new Regex(@"([a-zA-Z\_0-9.-]+\@[a-zA-Z\_0-9.-]+\.\w+)", RegexOptions.IgnoreCase);

if (regx.IsMatch(email))

{

return true;

}

return false;

}

}

}