

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, tranzacție și actualizare. Componente .Net/Java/MFC pentru elaborare client de poștă 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 constând în crearea unei aplicații de poștă electronică pentru trimiterea de mesaje electronice 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 poștă electronică sunt transferate între **proces** SMTP aflate pe sisteme diferite. Procesul SMTP care transmite un mesaj este numit client SMTP, iar procesul SMTP care primește 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 recepționat de serverul SMTP este livrat destinatarului, nici la modul în care este memorat mesajul și 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 comunicație, dacă se asigură un canal prin care datele să fie transmise într-un mod ordonat .

SMTP folosește următorul model de comunicație: transmițătorul, ca urmare a unei cereri de transmisie a mail-ului, stabilește o legătură bidirecțională 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 destinației* finale precum și utilizatorul căruia îi este destinat mesajul.

Un server SMTP trebuie să cunoască cel puțin următoarele comenzi :

- HELO - identificare computer expeditor;
- EHLO - identificare computer expeditor cu cerere de mod extins;
- MAIL FROM - specificare expeditorului;
- RCPT TO - specificarea destinatarului ;
- DATA - conținutul 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 găsesc 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 poștă 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.

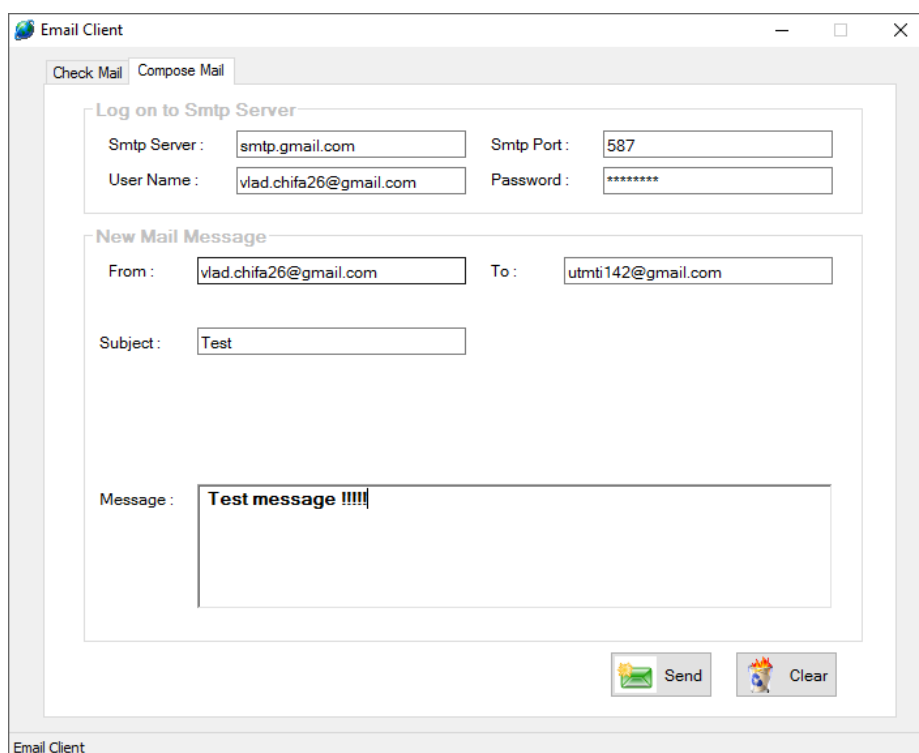
Mesajele trimise și mesajele în curs de scriere nu sunt accesibile de la alte calculatoare.

Mesajele sunt șterse de pe server și dacă aveți o problemă cu calculatorul 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 mesajul nostru .



The screenshot shows a window titled "Email Client" with two tabs: "Check Mail" and "Compose Mail". The "Compose Mail" tab is active. It contains two main sections:

- Log on to Smtip Server:** This section has four input fields:
 - Smtip Server: `smtp.gmail.com`
 - Smtip Port: `587`
 - User Name: `vlad.chifa26@gmail.com`
 - Password: `*****`
- New Mail Message:** This section has three input fields and a text area:
 - From: `vlad.chifa26@gmail.com`
 - To: `utmti142@gmail.com`
 - Subject: `Test`
 - Message: `Test message !!!!!`

At the bottom right of the form area, there are two buttons: "Send" (with a green envelope icon) and "Clear" (with a trash can icon). The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

Figura 1-Setarea conexiunii 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.

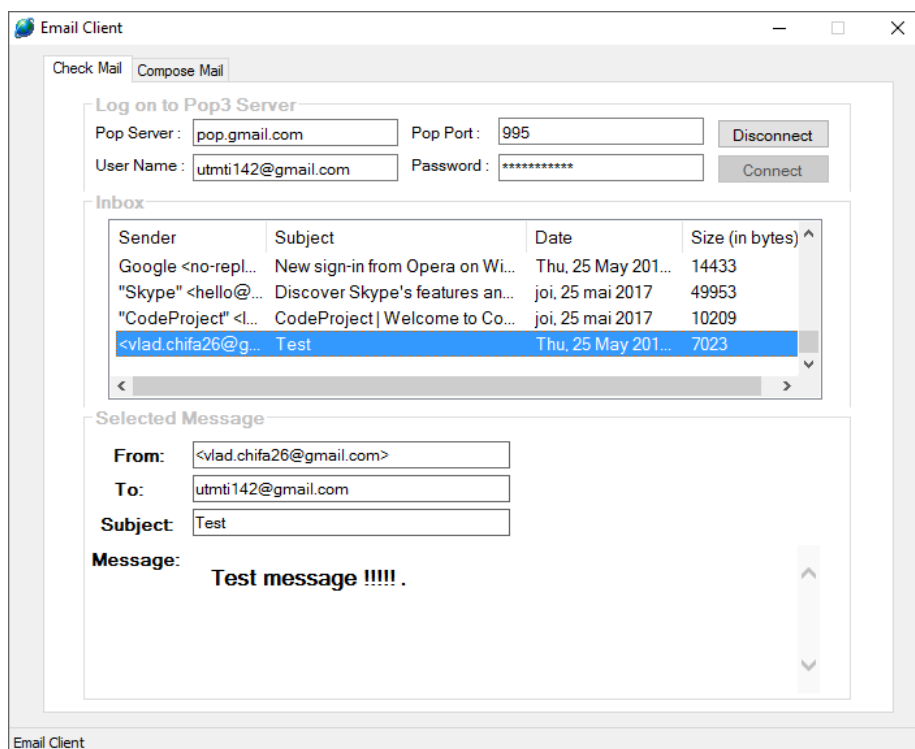


Figura 2 – Conectarea la POP server .

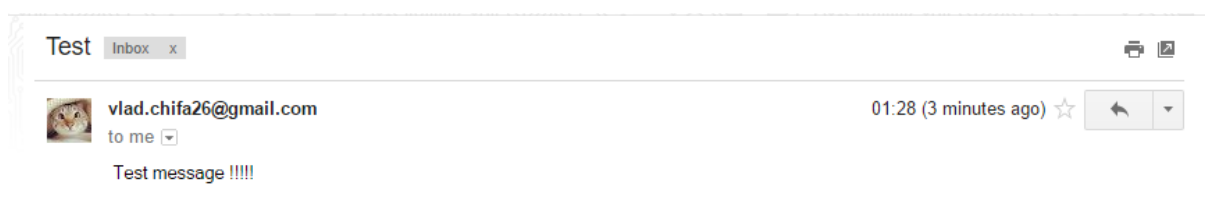


Figura 3 - Mesajul primit pe poștă.

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.ContainsFoc
us)

this.SubjectPopHeader.BackCol
or = Color.Ivory;
        else

this.SubjectPopHeader.BackCol
or = SystemColors.Window;
    }

```

```

        private void
SntpPort_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(Sm

```



```
tpServer.Text, SmtppPort.Text,
UserName.Text, Password.Text,
From.Text ,
To.Text,Text,Text))
```

```
{
    if
    (this.EmailValidation(this.Fr
om.Text))
```

```
{
    bool
isRecipient = false;
```

```
    if
    (this.To.Text.Length > 0)
    {
        if
        (this.RecipientsEmailValidati
on(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.IsConnectedToIntern
et())
```

```
{
    if
    (isRecipient == true)
```

```
Rtf2Html rtf = new
Rtf2Html();
```

```
string Html =
```

```
rtf.ConvertRtfToHtml(this.Mai
lMessage);
```

```
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.SmtppServer.Text = "";
        this.SmtppPort.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.InvokeRequi
        red)
        {

            DisconnectEventHandler discon
            = new
            DisconnectEventHandler(this.s
            mtp_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.InvokeRequi
    red)
    {
        DataTransferEventHandler data
        = new
        DataTransferEventHandler(this
        .smtp_EndedDataTransfer);

        this.Invoke(data, new
        object[] { sender });
    }
    else
    {
        this.ProgressLabel.Text =
        "Email message has sent";
    }
}

void
smtp_StartedDataTransfer(obje
ct sender)
{

```

```

        if
        (this.statusStrip.InvokeRequi
red)
        {

DataTransferEventHandler data
= new
DataTransferEventHandler(this
.smtp_StartedDataTransfer);

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

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

    void
smtp_AuthenticationFinished(o
bject sender, string
userName)
    {
        if
        (this.statusStrip.InvokeRequi
red)
        {

AuthenticateEventHandler auth
= new
AuthenticateEventHandler(this
.smtp_AuthenticationFinished)
;

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

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

```

```

    void
smtp_AuthenticationBegan(obje
ct sender, string userName)
    {
        if
        (this.statusStrip.InvokeRequi
red)
        {

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(o
bject sender, string Server,
int Port)
    {
        if
        (this.statusStrip.InvokeRequi
red)
        {

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.CheckInputValidationFor

```

```

Pop(this.PopServer.Text,
this.PopPort.Text,
this.PopUserName.Text,
this.PopPassword.Text))
        {
            if
(Internet.IsConnectedToIntern
et())
            {
                Thread th
= new Thread(new
ThreadStart(this.ReceiveEmail
s));

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.p
op_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.InvokeRequi
red)
    {

DataReceivingEventHandler
data = new
DataReceivingEventHandler(thi
s.pop_EndedDataReceiving);

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

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

private void
pop_StartedDataReceiving(objec
t sender)
{
    if
(this.statusStrip.InvokeRequi
red)
    {

DataReceivingEventHandler
data = new
DataReceivingEventHandler(thi
s.pop_StartedDataReceiving);

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

```

```

    }
    else
    {

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

private void
pop_AuthenticationFinished(ob
ject sender, string userName)
{
    if
(this.statusStrip.InvokeRequi
red)
    {

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(objec
t sender, string userName)
{
    if
(this.statusStrip.InvokeRequi
red)
    {

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(obj
ect sender, string Server,
int Port)
    {
        if
(this.statusStrip.InvokeRequi
red)
        {

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(ob
ject sender, string Server,
int Port)
    {
        if
(this.statusStrip.InvokeRequi
red)
        {

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 ==
MouseButton.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
== MouseButton.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_SelectedIndexCha
nged(object sender, EventArgs
e)
        {
            if
(! (this.MailMessages.Selected
Items.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(thi
s.EnableDisableSendButton);

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

```



```

        else
        {
this.Send.Enabled = enable;
        }
    }

    private void
EnableDisableConnectButton(bo
ol enable)
    {
        if
(this.Connect.InvokeRequired)
        {

EnableDisableEventHandler obj
= new
EnableDisableEventHandler(thi
s.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(thi
s.EnableDisableDisconnectButt
on);

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(ListVie
wItem item);

        private void
InsertItem(ListViewItem item)
        {
            if
(this.MailMessages.InvokeRequ
ired)
            {

InboxItemEventHandler obj =
new
InboxItemEventHandler(this.In
sertItem);

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

this.MailMessages.Items.Add(i
tem);
            }

        }

    // The delegate used
to write Pop Message in
browser
        private delegate void
WritePopMessageEventHandler(s
tring 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
UpdatePopMessageHeaderEventHa
ndler(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)
            {

UpdatePopMessageHeaderEventHa
ndler obj = new
UpdatePopMessageHeaderEventHa
ndler(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.EnableDisableConnectButt
on(false);

pop_client.ConnectionEstablis
hing += new
ConnectEventHandler(this.pop_
ConnectionEstablishing);

pop_client.ConnectionEstablis
hed += new
ConnectEventHandler(this.pop_
ConnectionEstablished);

pop_client.AuthenticationBega
n += new
AuthenticateEventHandler(this
.pop_AuthenticationBegan);

pop_client.AuthenticationFini
shed += new
AuthenticateEventHandler(this
.pop_AuthenticationFinished);

pop_client.StartedDataReceivi
ng += new
DataReceivingEventHandler(thi
s.pop_StartedDataReceiving);

pop_client.EndedDataReceiving
+= new
DataReceivingEventHandler(thi
s.pop_EndedDataReceiving);

pop_client.Disconnected +=
new
DisconnectEventHandler(this.p
op_Disconnected);

this.pop =
pop_client;

pop_client.Connect();

this.EnableDisableDisconnectB
utton(true);

pop_client.GetMailBoxDetails(
); //it sets the
TotalEmails and
TotalEmailSize properties

        if
(pop_client.TotalEmails >= 1)
        {
            this.UpdateStatusBar("Retrie
ving Emails");

            this.FillInboxListView(pop_cl
ient);

            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.EnableDisableConnectButt
on(true);

    this.EnableDisableDisconnectB
utton(false);
        }
    }

    private void
FetchEmailCallBack()
    {
        try
        {
            this.email =
this.pop.FetchEmail(this.msg_
id);

            this.UpdatePopMessageHeader(t
his.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().Equal
s("text/html"))
                {

                    this.WritePopMessage(content)
;

                    isHtmlIncluded = true;
                }
                else if
                (content_type.ToLower().Equal
s("base64"))
                {

                    ListViewItem item = new
ListViewItem(attached_file_na
me);

                    item.ImageIndex = 0;

                }
                else if
                (content_type.ToLower().Equal
s("text/plain"))
                {

                    plain_text_message_section =
i;

                }

                //

                this.InsertPopAttachedFileNam
es(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(st
ring 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].ToS
tring());

item.SubItems.Add(size[j].ToS
tring());

this.InsertItem(item);
    }
}

```

```

// Helping Methods
for Smtip Client user
interface

    private void
SendEmail(object mail_msg)
    {
        try
        {
            MailMessage
mail_message =
(MailMessage)mail_msg;

            SmtipClient
smtp = new
SmtipClient(this.SmtipServer.Te
xt,
Convert.ToInt32(this.SmtipPort
.Text));

            smtp.UserName
= this.UserName.Text;
            smtp.Password
= this.Password.Text;

this.EnableDisableSendButton(
false);

smtp.ConnectionEstablishing
+= new
ConnectEventHandler(smtp_Conn
ectionEstablishing);

smtp.ConnectionEstablished +=
new
ConnectEventHandler(smtp_Conn
ectionEstablished);

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_D
isconnected);

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;
}

```