Sikkim-Manipal University DE

Mail Server

Supports POP3 and SMTP Protocol

Arnav Mukhopadhyay

SYNOPSIS

RFC 1939 dictates the features and commands of Post Office Protocol Version 3 (POP3). Further, RFC 821, and updated version RFC 5321 [2008] dictates the features and commands of Simple Mail Transfer Protocol (SMTP). These two protocols form the basis of Mail Interchange System or Applications. The client side is implemented as software, such as Windows Live Mail, Microsoft Outlook, Microsoft Office Outlook, or any other mail software packages. This client software is either pre-installed with the operating system or can be downloaded for free or on some nominal charge. The Server version is implemented as a stand-alone or distributed application, that uses the Operating System based Network Socket API to let the client applications connect to them and interchange the required mailing information's as a byte stream. But these servers differs from any other servers on the Internet or Intranet, by the way they work or communicate with the client. The Mail Server must implement the POP3 and SMTP protocols at application layer, as guided by RFC 1939, RFC 821 or RFC 5321. These protocols are nothing but text commands that the client send to the server to perform the required work. The server in return, processes the client command sent the client back with either the query status, reply message or error report. The client receives this reply and proceeds as required or can say "QUIT" to disconnect from server.

Due to the complexity of the protocol's processing on server, Mail Servers are generally available as commercial packages. But, the limitations of these commercial packages lies with the volume of Mail processing did with them, and the higher cost of purchase or maintenance. These limitations make the commercial Mail Servers inefficient for implementation or testing on a small-office or a home based Intranet work systems.

Thus, we would design this Mail Server with the following objectives in mind:

i. Implement the minimal set of POP3 and SMTP command set as required by their respective RFCs.
ii. Minimize the cost of deployment and maintenance.

Amar Mukhopadhyay

iii. Can be deployed on any machine, with minimal hardware requirements.
iv. Interoperable over multiple client, operating systems and network environment, with ease of use.
v. Simplified UI and deployment requirement can even be used by novice users.

[illegible]

Manipal
INSPIRED BY LIFE

ACKNOWLEDGEMENT

I have been benefited from the advice and helped by many people, whom I would like to offer my thanks and without their continual help and support, I would not have completed the project or even started my work on it. The project idea was on my mind for a long time, but the force to implement it had been offered by our project guide and lecturer Sir Amitava Das and Srijit Sir, to whom I would like to, offer my whole hearted thanks. I wish to thank my parents for providing financial support and guidance throughout the course. And further extend it to our College principal and to the Sikkim-Manipal (DE) University to help us realize our dream. I would also like to thank other people who would remain nameless, for extending their help through the Internet Community (Code Project) to successfully complete my project.

[illegible]

PREFACE

Mail Systems had been prevailing for centuries. These systems had enabled communication between peoples across the World. With the introduction of modern technologies, this system had matured to a more sophisticated one. The Internet had brought a milestone to the snail mailing, providing speed and reduced the distances between people. Now the messages, which could have taken days to be relayed, take a few seconds to be conveyed. But as everything has a price to pay, the new technologies have gave birth to multi-national corporations that governs and controls this communication. With the advent of open-source software, these mailing technologies had gone mostly out of hand of these organizations, and again placed on to the hands of general people. Now standard (open-source) protocols have been defined to enable mailings and related communication systems. These protocols have been modified and updated for ages now, and have emerged into a new forefront for developers and users to use. But with lack of focus and increase in demand for professionalization, these protocols are been overestimated and developments in these sectors are losing in value, and again falling into the hands of the large organizations, which make available their technologies based on these standard protocols at high cost. But as I have understood, these protocols are mere command sets, which are clearly defined in RFCs that can be freely downloaded off the net, and can be easily implemented using any available platform, at almost no cost, depending on the deployment requirement. This is the main concern and development requirement as highlighted by this project.

[illegible]

INTRODUCTION

Network technologies had around been a long time, but the most of it had been a circumference around Internet. The benefit of this is the ability to connect people across the world through its amazing and innovative software products. One of this is the Mailing service. The Mailing service provides a way of connecting peoples and interchange messages and other documents. This service unlike other similar services had its analogies brought down from the existing Postal systems, which have existed for around centuries.

This product is targeted to establish the mailing system, more precisely stating the Server side of it. The Mail client is not established here, neither a preferred client software is named or specified, because any available commercial or non-commercial mail interchange software will just work fine.

The Mail Interchange between the server and the client over the network domain takes place with the help of a series of command and reply responses. The commands are passed from client to server, and in return, the corresponding response as reply to the command is passed from the server to the client. The mail message is also a possible response. The commands are in simple text format, which are grouped into a package forming rules or protocols. These protocols are defined in one of the Internet RFCs. Some examples of the Internet Mail Protocols are POP3, SMTP, IMAP, APOP, etc.

This project will be more concerned with the POP3 and SMTP. The POP3 and APOP shares a similar set of commands, but we will still consider the basics of POP3 with text authentication, without the need for TLS/SSL or (MD5 or RSA) Hash based authentication as laid down by the APOP services. Similarly ESMTP is an extension of SMTP protocol to achieve extended command set for mail interchange, but we will, again stick to the basic command sets of the SMTP protocol. The ESTP clients will be requested automatically by the SMTP server to fallback to SMTP protocol, otherwise they can disconnect. With all this facts introduced, we can head on to attain the objectives.

OBJECTIVE

This project is aimed at designing a Mail Server that establish and follows the POP3/SMTP protocol for mail interchange. Further, the application is aimed for small office or home based Intra-Network systems. The application interfaces or forms are designed such that, the server can be deployed and maintained or used by a novice user/administrator. It is also aimed at achieving low deployment and acquisition cost.

This server therefore, caters the following objectives:

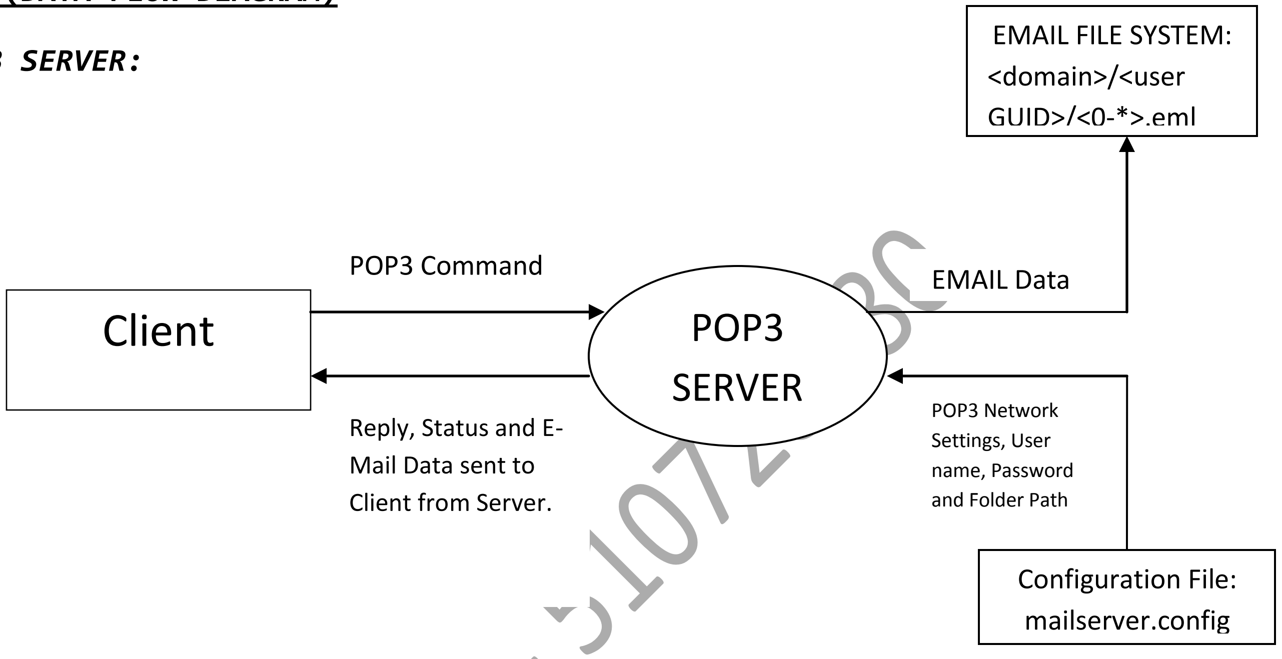
i. A POP3 and SMTP based mail server that is compatible with almost all available desktop based mail client.
ii. It can be deployed with minimal system requirements. This would minimize the system deployment cost.
iii. The User Interface is made as minimal as possible. These interfaces would be self-explainable and therefore, make it possible for novice users to use.
iv. The server uses most optimized code, and prevailing interfaces and components pre-tested by respective authorities. This would reduce the cost of testing and debugging, which would again reduce the cost of acquisition and in consequence, will reduce the overall cost of the product and increase the product lifetime.
v. The code will be deployed on a virtual machine platform provided by Microsoft .NET technologies, which have been extended to multiple platforms other than Microsoft Windows, to Linux, Solaris and also Mac OS. This reduces platform dependency, which would again transparency of product to multiple operating systems.
vi. The software could be deployed on Local Network, Intra-Network or Internet with no exceptions, but to reduce cost, some issues must be considered; which are discussed in the documentation as they arise.

Amar Mukhopadhyay

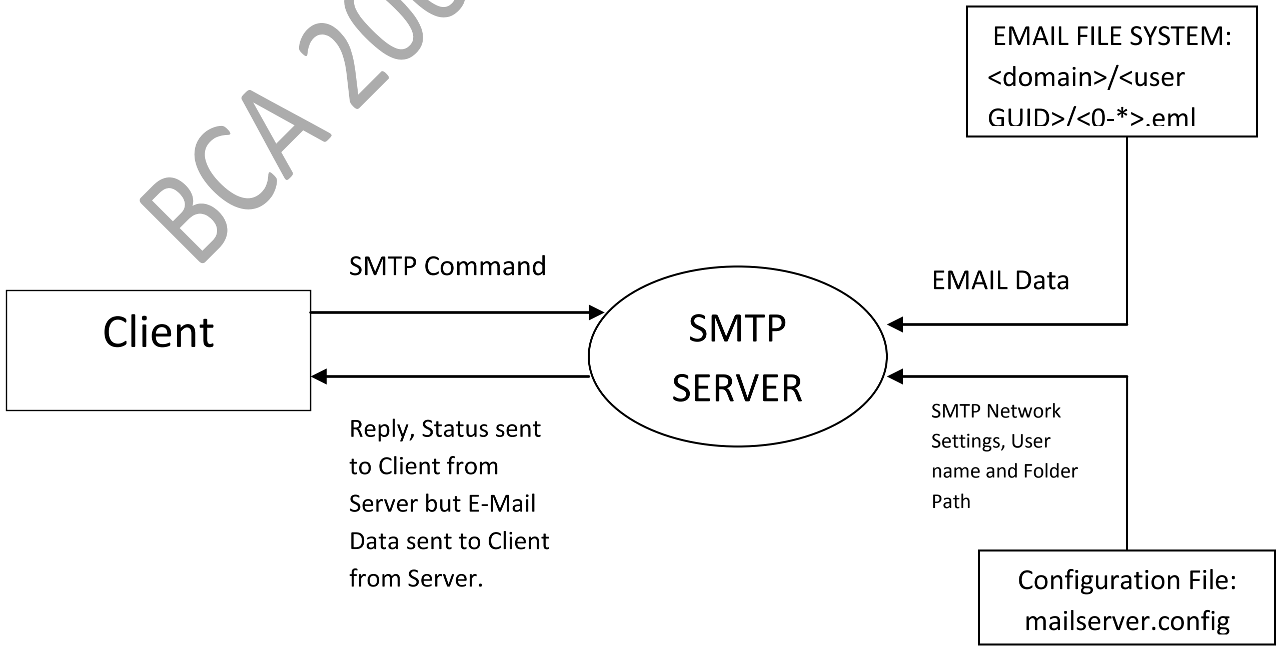
SYSTEM DESIGN

DFD (DATA FLOW DIAGRAM)

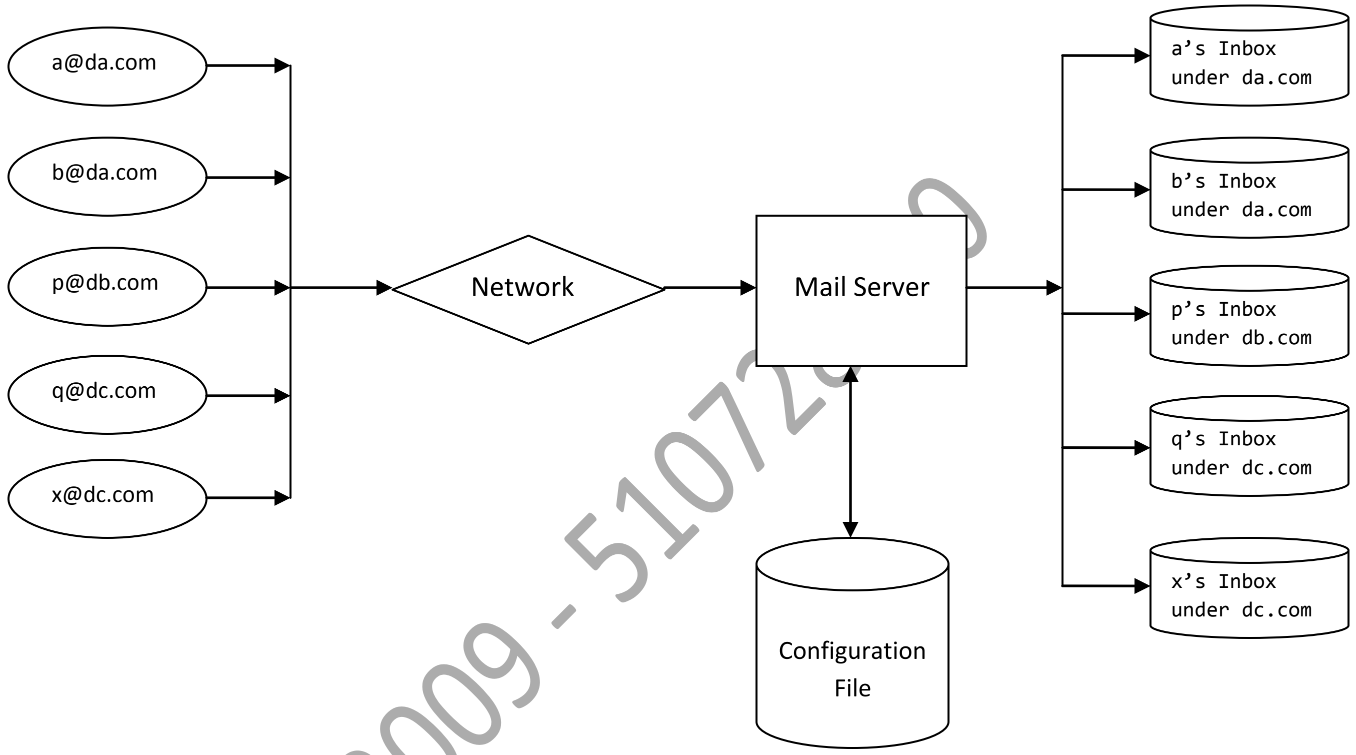
POP3 SERVER:


EMAIL FILE SYSTEM: <domain>/<user GUID>/<0-*>.eml
POP3 Command
Client
POP3 SERVER
EMAIL Data
Reply, Status and E-Mail Data sent to Client from Server.
POP3 Network Settings, User name, Password and Folder Path
Configuration File: mailserver.config

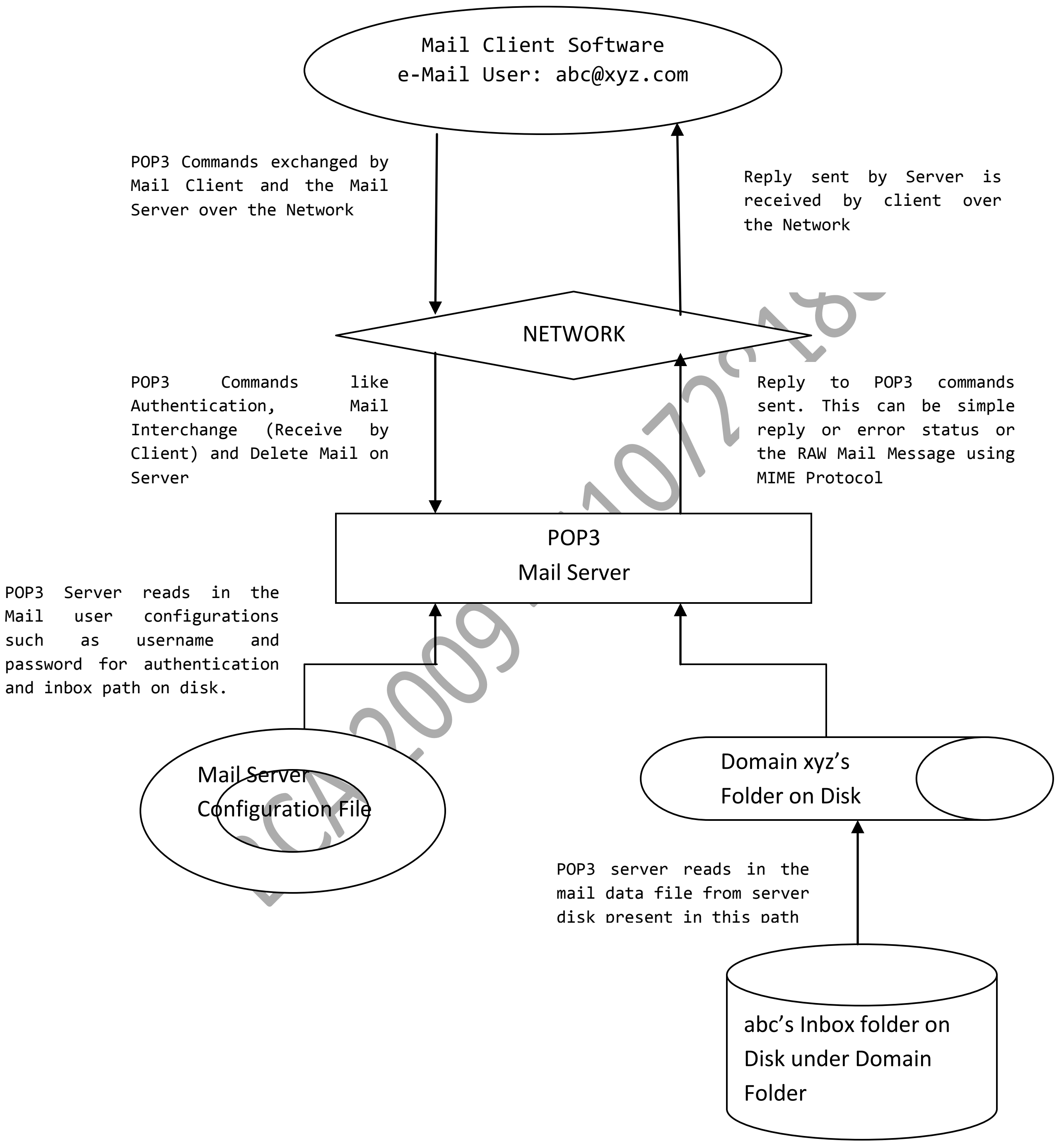
SMTP SERVER:


EMAIL FILE SYSTEM: <domain>/<user GUID>/<0-*>.eml
SMTP Command
Client
SMTP SERVER
EMAIL Data
Reply, Status sent to Client from Server but E-Mail Data sent to Client from Server.
SMTP Network Settings, User name and Folder Path
Configuration File: mailserver.config

Entity-Relationship Diagram (General)

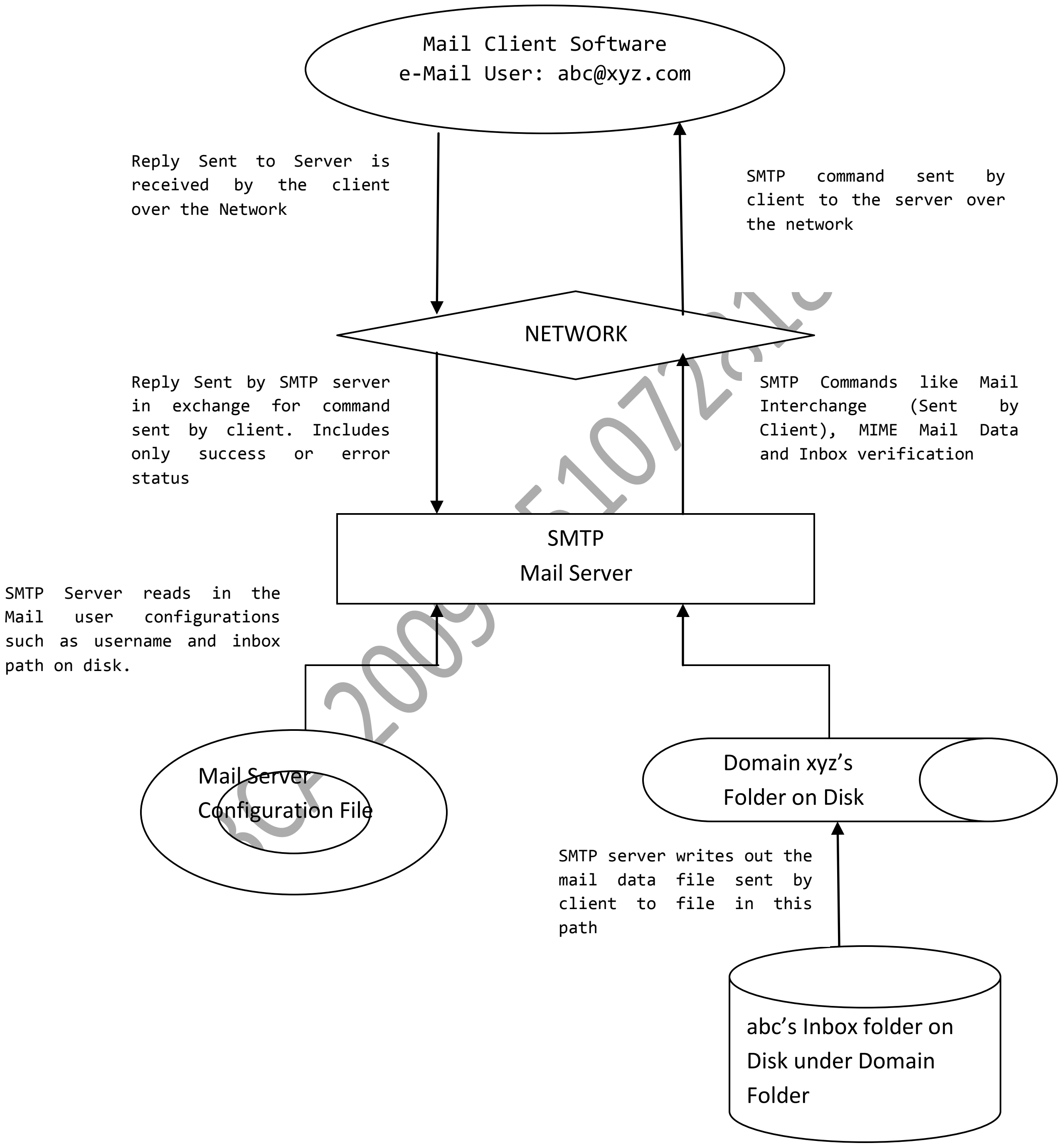


Entity-Relationship Diagram (POP3 Server)


Mail Client Software
e-Mail User: abc@xyz.com
POP3 Commands exchanged by Mail Client and the Mail Server over the Network
Reply sent by Server is received by client over the Network
NETWORK
POP3 Commands like Authentication, Mail Interchange (Receive by Client) and Delete Mail on Server
Reply to POP3 commands sent. This can be simple reply or error status or the RAW Mail Message using MIME Protocol
POP3
Mail Server
POP3 Server reads in the Mail user configurations such as username and password for authentication and inbox path on disk.
Mail Server Configuration File
Domain xyz's Folder on Disk
POP3 server reads in the mail data file from server disk present in this path
abc's Inbox folder on Disk under Domain Folder

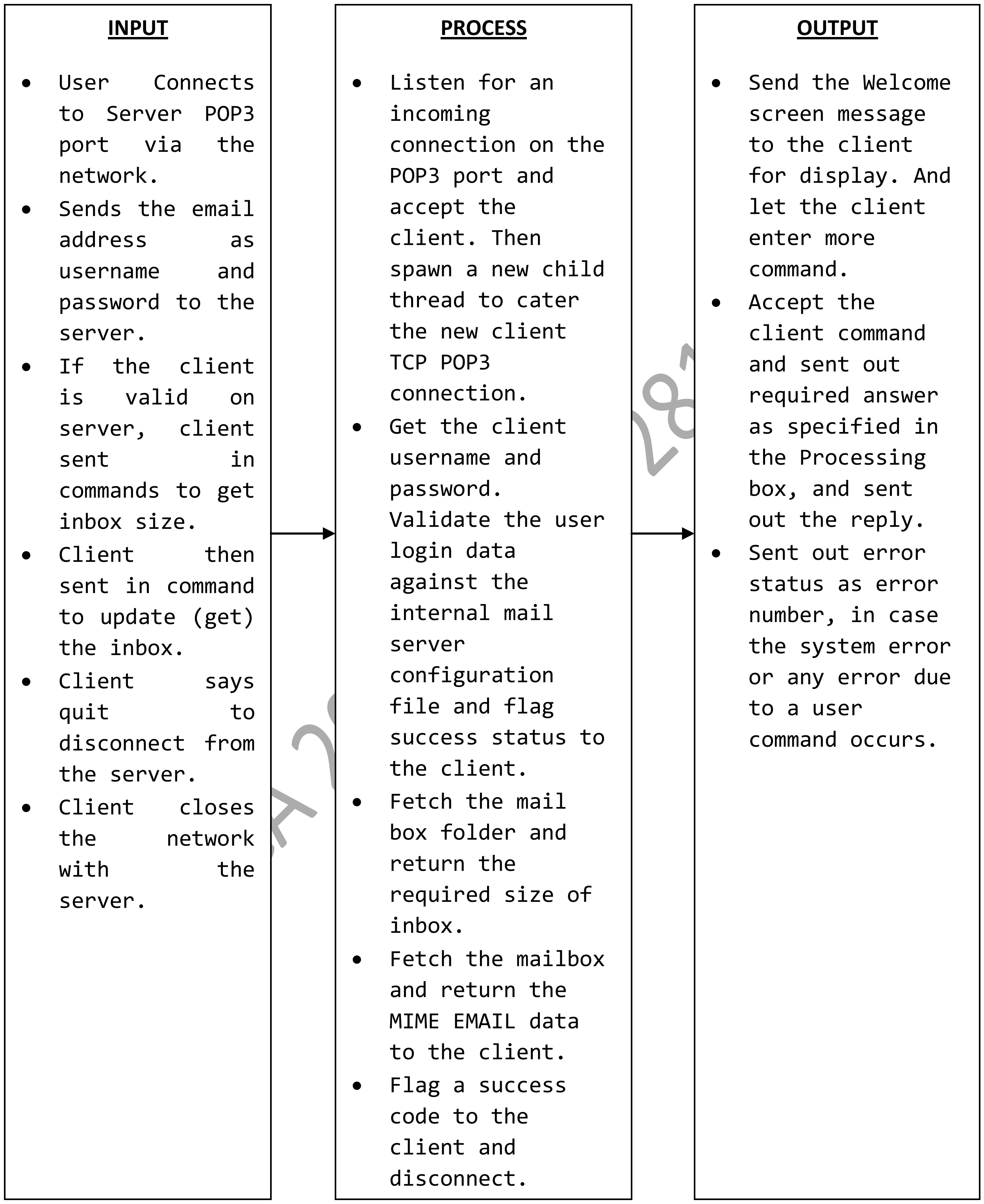
Amar Mukhopadhyay

Entity-Relationship Diagram (SMTP Server)


Mail Client Software
e-Mail User: abc@xyz.com
Reply Sent to Server is received by the client over the Network
SMTP command sent by client to the server over the network
NETWORK
Reply Sent by SMTP server in exchange for command sent by client. Includes only success or error status
SMTP Commands like Mail Interchange (Sent by Client), MIME Mail Data and Inbox verification
SMTP Mail Server
SMTP Server reads in the Mail user configurations such as username and inbox path on disk.
Mail Server Configuration File
Domain xyz's Folder on Disk
SMTP server writes out the mail data file sent by client to file in this path
abc's Inbox folder on Disk under Domain Folder

[illegible]

Hierarchy plus Input-Process-Output (HIPO) Chart for POP3 Server



INPUT

- User Connects to Server POP3 port via the network.
- Sends the email address as username and password to the server.
- If the client is valid on server, client sent in commands to get inbox size.
- Client then sent in command to update (get) the inbox.
- Client says quit to disconnect from the server.
- Client closes the network with the server.

PROCESS

- Listen for an incoming connection on the POP3 port and accept the client. Then spawn a new child thread to cater the new client TCP POP3 connection.
- Get the client username and password. Validate the user login data against the internal mail server configuration file and flag success status to the client.
- Fetch the mail box folder and return the required size of inbox.
- Fetch the mailbox and return the MIME EMAIL data to the client.
- Flag a success code to the client and disconnect.

OUTPUT

- Send the Welcome screen message to the client for display. And let the client enter more command.
- Accept the client command and sent out required answer as specified in the Processing box, and sent out the reply.
- Sent out error status as error number, in case the system error or any error due to a user command occurs.

Amar Mukhopadhyay

Hierarchy plus Input-Process-Output (HIPO) Chart for SMTP Server

INPUT	→	**PROCESS**	→	**OUTPUT**
• User Connects to Server SMTP port via the network. • User sent in commands to send mail to the SMTP server. It first sent in the command to identify the return path that is, user name of sender. And waits for success from server. • User sent in the target person mail id and waits for a success reply. • User sent in the MIME formatted EMAIL message and waits for a success message. • User repeats the above process to send more mails. • User is finally done, so it		• Listen for an incoming connection on the SMTP port and accept the client. Then spawn a new child thread to cater the new client TCP SMTP connection. • As the sender / client reply path / user name is sent, the server checks it against its database for validity against intruders. On success, it sent out a success flag or disconnects on error. • User verifies each sent in target mail ids against its database for their consecutive physical inbox path and redirects the mail data in MIME format to a file in the physical disk space. Sent out a error status if something goes wrong.		• Send the Welcome screen message to the client for display. And let the client enter more command. • Accept the client command and sent out required answer as specified in the Processing box, and sent out the reply. • Sent out error status and error message, in case the system error or any error due to a user command occurs and disconnect immediately.

Amar Mukhopadhyay

DATABASE

A mail exchanged by the client contains raw binary data that can contain symbols or indications, that may interfere with data format implementation of general databases. And also the relational database system introduces a high latency in the operation of the server. Thus, it had been decided to use the file system itself for the storage of the raw binary email data. In order to achieve this, and also to provide a degree of security along with mobilization of the database, and minimum possible IO latency we have the database designed as follows:

i. The domains and user name corresponding to the email addresses of the user are arranged as virtual folders and kept within the mail server configuration file named, mailserver.config. This results in centralized tracking of the mailing system and provides secure transaction.

ii. The username@domainname is stripped off into the domainname entry in the configuration file, where the username is placed as the sub-folder entry of the domainname entry. This folder has data entries like password, file system folder GUID string.

iii. The password is in plain text format, and is used during user account verification to initiate POP3 mail exchange.

iv. The GUID string entry is a randomly generated system GUID ID that represents the physical inbox path to the requested user mail account. This folder is a real folder that is created depending on the operating system or file system in use. But is generally created in the present working directory of the application itself.

v. The mail inbox physical folder contains the E-MAIL data files and has the extension name .eml. These

[illegible]

files are numbered from 0 onwards and are always unique. This naming scheme had been used to simplify the code and also quicken the mail interchange. Further, this reduce mail ID (sent by client) to mailing file (on server) transformation code, to further increase latency.

vi. The eml files contain the binary mail data as is sent by the client. This further increases compatibility as these files are compatible with Microsoft Outlook data file format and thus these can be opened on the server for debugging or testing purpose. This technique has a down side, as data is not encoded, anyone with access rights to these folder can open and read the data. Thus, this require adds an additional burden on the system administrator to carefully monitor and control access to this folder.

[illegible]

FEASIBILITY STUDY

The Project was designed to target a low end server platform deployment environment, since; it was specifically designed to meet the requirements of small-office or home. It is designed using Microsoft Visual C# .NET 2008, and makes extensive use of the 2008 exclusive features like LINQ, XML-LINQ and WCF. Thus, the minimum requirement is the Microsoft .NET Framework 3.5.

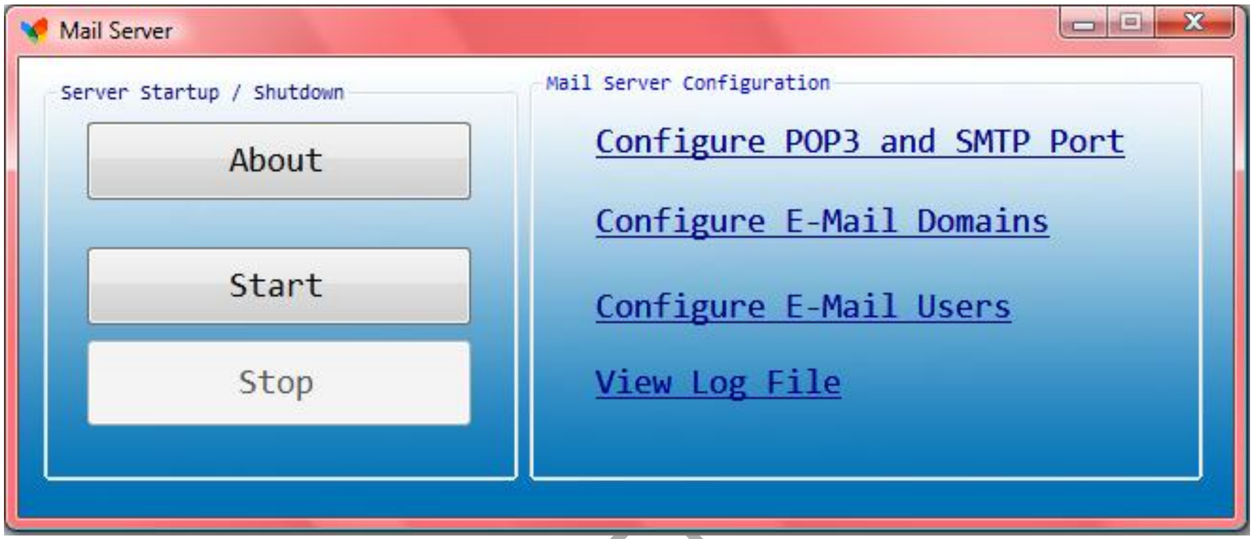
With reduced code and easy to use UI, the project can be deployed at minimal cost barrier.

To summarize the project requirement, we have:

- **SYSTEM REQUIRMENTS:-**
  - **Operating System:** Microsoft Windows XP (minimum), Linux, Solaris, etc.
  - **Platform:** Microsoft .NET 3.5 (Windows), MONO (other OS).
  - **Memory:** 64 MB (minimum)
  - **Network:** 10/100 MBPS LAN or higher data rate, as per number of client connections.
  - **Other requirements** must meet the deployment requirements of the Microsoft .NET 3.5 or MONO, whichever is used for executing the assembly.
- **Project Compilation requirements:-**
  - Programmed in Microsoft Visual C# 2008.
  - Requires XML, LINQ, WCF and Windows Form.
  - Requires Microsoft .NET Framework 3.5 during compilation.

[illegible]

SCREEN SHOTS & REPORTS



The above represents the main windows form that will be seen when the server application is opened. The "About" button will open the About dialog box, the Start button will start the server, Stop button will stop the server. The Start button must be clicked to start the server in listening mode, so that clients can exchange email messages. After the Stop button is clicked, the Server will stop listening and no more message interchange is possible after that.

The Configuration items are self-explanatory. The "Configure POP3 and SMTP Port" link is clicked to configure the server listening port and will be configurable using the "Configure POP3/SMTP Mail Server" dialog box. The Configure E-Mail Domains link will open up the "Configure the Domain" dialog, where you can add, remove the email domains. The "Configure E-Mail Users" link will open up the "Configure E-Mail User Accounts" dialog box, where you can add, remove or edit a user's inbox configuration and E-Mail ID.

When the mail server is opened, a notification icon is also displayed. Double clicking on the notification icon will show or hide the main windows form of the application.

Aman Mukhopadhyay

Form Code: Solution > MailServer.cs > MailServer.Designer.cs

```
namespace Mail_Server
{
    partial class MailServer
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be 
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new 
System.ComponentModel.ComponentResourceManager(typeof(MailServer));
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.lnkViewLog = new System.Windows.Forms.LinkLabel();
            this.lnkCfgUser = new System.Windows.Forms.LinkLabel();
            this.lnkCfgDomain = new System.Windows.Forms.LinkLabel();
            this.lnkCfgMail = new System.Windows.Forms.LinkLabel();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.btnAbout = new System.Windows.Forms.Button();
            this.btnStop = new System.Windows.Forms.Button();
            this.btnStart = new System.Windows.Forms.Button();
            this.notifyIcon1 = new 
System.Windows.Forms.NotifyIcon(this.components);
            this.groupBox1.SuspendLayout();
            this.groupBox3.SuspendLayout();
            this.SuspendLayout();
            // 
            // groupBox1
            // 
```

[illegible]

```
            this.groupBox1.BackColor = System.Drawing.Color.Transparent;
            this.groupBox1.Controls.Add(this.lnkViewLog);
            this.groupBox1.Controls.Add(this.lnkCfgUser);
            this.groupBox1.Controls.Add(this.lnkCfgDomain);
            this.groupBox1.Controls.Add(this.lnkCfgMail);
            this.groupBox1.ForeColor = System.Drawing.Color.MediumBlue;
            this.groupBox1.Location = new System.Drawing.Point(270, 7);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(356, 216);
            this.groupBox1.TabIndex = 0;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Mail Server Configuration";
            //
            // lnkViewLog
            //
            this.lnkViewLog.AutoSize = true;
            this.lnkViewLog.Font = new System.Drawing.Font("Consolas",
14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lnkViewLog.LinkColor = System.Drawing.Color.Navy;
            this.lnkViewLog.Location = new System.Drawing.Point(31, 153);
            this.lnkViewLog.Name = "lnkViewLog";
            this.lnkViewLog.Size = new System.Drawing.Size(140, 22);
            this.lnkViewLog.TabIndex = 3;
            this.lnkViewLog.TabStop = true;
            this.lnkViewLog.Text = "View Log File";
            this.lnkViewLog.LinkClicked += new
System.Windows.Forms.LinkLabelLinkClickedEventHandler(this.lnkViewLog_LinkCli
cked);
            //
            // lnkCfgUser
            //
            this.lnkCfgUser.AutoSize = true;
            this.lnkCfgUser.Font = new System.Drawing.Font("Consolas",
14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lnkCfgUser.LinkColor = System.Drawing.Color.Navy;
            this.lnkCfgUser.Location = new System.Drawing.Point(31, 113);
            this.lnkCfgUser.Name = "lnkCfgUser";
            this.lnkCfgUser.Size = new System.Drawing.Size(230, 22);
            this.lnkCfgUser.TabIndex = 2;
            this.lnkCfgUser.TabStop = true;
            this.lnkCfgUser.Text = "Configure E-Mail Users";
            this.lnkCfgUser.LinkClicked += new
System.Windows.Forms.LinkLabelLinkClickedEventHandler(this.lnkCfgUser_LinkCli
cked);
            //
            // lnkCfgDomain
            //
            this.lnkCfgDomain.AutoSize = true;
            this.lnkCfgDomain.Font = new System.Drawing.Font("Consolas",
14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lnkCfgDomain.LinkColor = System.Drawing.Color.Navy;
            this.lnkCfgDomain.Location = new System.Drawing.Point(31, 68);
            this.lnkCfgDomain.Name = "lnkCfgDomain";
            this.lnkCfgDomain.Size = new System.Drawing.Size(250, 22);
            this.lnkCfgDomain.TabIndex = 1;
            this.lnkCfgDomain.TabStop = true;
            this.lnkCfgDomain.Text = "Configure E-Mail Domains";
            this.lnkCfgDomain.LinkClicked += new
System.Windows.Forms.LinkLabelLinkClickedEventHandler(this.lnkCfgDomain_LinkC
licked);
            //
            // lnkCfgMail
            //
            this.lnkCfgMail.AutoSize = true;
            this.lnkCfgMail.Font = new System.Drawing.Font("Consolas",
14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lnkCfgMail.ForeColor = System.Drawing.Color.MediumBlue;
            this.lnkCfgMail.LinkColor = System.Drawing.Color.Navy;
            this.lnkCfgMail.Location = new System.Drawing.Point(31, 26);
            this.lnkCfgMail.Name = "lnkCfgMail";
            this.lnkCfgMail.Size = new System.Drawing.Size(290, 22);
            this.lnkCfgMail.TabIndex = 0;
            this.lnkCfgMail.TabStop = true;
            this.lnkCfgMail.Text = "Configure POP3 and SMTP Port";
            this.lnkCfgMail.LinkClicked += new
System.Windows.Forms.LinkLabelLinkClickedEventHandler(this.lnkCfgMail_LinkCli
cked);
            //
            // groupBox3
            //
            this.groupBox3.BackColor = System.Drawing.Color.Transparent;
            this.groupBox3.Controls.Add(this.btnAbout);
            this.groupBox3.Controls.Add(this.btnStop);
            this.groupBox3.Controls.Add(this.btnStart);
            this.groupBox3.ForeColor = System.Drawing.Color.DarkBlue;
            this.groupBox3.Location = new System.Drawing.Point(13, 12);
            this.groupBox3.Name = "groupBox3";
            this.groupBox3.Size = new System.Drawing.Size(251, 211);
            this.groupBox3.TabIndex = 2;
            this.groupBox3.TabStop = false;
            this.groupBox3.Text = "Server Startup / Shutdown";
            //
            // btnAbout
            //
            this.btnAbout.Font = new System.Drawing.Font("Consolas", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.btnAbout.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.btnAbout.Location = new System.Drawing.Point(22, 21);
            this.btnAbout.Name = "btnAbout";
            this.btnAbout.Size = new System.Drawing.Size(205, 43);
            this.btnAbout.TabIndex = 2;
            this.btnAbout.Text = "About";
```

[illegible]

```
            this.btnAbout.UseVisualStyleBackColor = true;
            this.btnAbout.Click += new
System.EventHandler(this.btnAbout_Click);
            //
            // btnStop
            //
            this.btnStop.Font = new System.Drawing.Font("Consolas", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.btnStop.ForeColor = System.Drawing.SystemColors.ControlText;
            this.btnStop.Location = new System.Drawing.Point(22, 136);
            this.btnStop.Name = "btnStop";
            this.btnStop.Size = new System.Drawing.Size(205, 47);
            this.btnStop.TabIndex = 1;
            this.btnStop.Text = "Stop";
            this.btnStop.UseVisualStyleBackColor = true;
            this.btnStop.Click += new
System.EventHandler(this.btnStop_Click);
            //
            // btnStart
            //
            this.btnStart.Font = new System.Drawing.Font("Consolas", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.btnStart.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.btnStart.Location = new System.Drawing.Point(22, 87);
            this.btnStart.Name = "btnStart";
            this.btnStart.Size = new System.Drawing.Size(205, 43);
            this.btnStart.TabIndex = 0;
            this.btnStart.Text = "Start";
            this.btnStart.UseVisualStyleBackColor = true;
            this.btnStart.Click += new
System.EventHandler(this.btnStart_Click);
            //
            // notifyIcon1
            //
            this.notifyIcon1.BalloonTipIcon =
System.Windows.Forms.ToolTipIcon.Info;
            this.notifyIcon1.BalloonTipText = "Mail Server";
            this.notifyIcon1.Icon =
((System.Drawing.Icon)(resources.GetObject("notifyIcon1.Icon")));
            this.notifyIcon1.Text = "Mail Server";
            this.notifyIcon1.Visible = true;
            this.notifyIcon1.MouseDoubleClick += new
System.Windows.Forms.MouseEventHandler(this.notifyIcon1_MouseDoubleClick);
            //
            // MailServer
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.SystemColors.WindowFrame;
            this.BackgroundImage =
global::Mail_Server.Properties.Resources.Untitled_2;
```

```
            this.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.ClientSize = new System.Drawing.Size(642, 242);
            this.Controls.Add(this.groupBox3);
            this.Controls.Add(this.groupBox1);
            this.DoubleBuffered = true;
            this.Font = new System.Drawing.Font("Consolas", 8.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MaximizeBox = false;
            this.Name = "MailServer";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Mail Server";
            this.Load += new System.EventHandler(this.MailServer_Load);
            this.FormClosed += new
System.Windows.Forms.FormClosedEventHandler(this.MailServer_FormClosed);
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.MailServer_FormClosing);
            this.groupBox1.ResumeLayout(false);
            this.groupBox1.PerformLayout();
            this.groupBox3.ResumeLayout(false);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.Button btnStop;
        private System.Windows.Forms.Button btnStart;
        private System.Windows.Forms.LinkLabel lnkCfgUser;
        private System.Windows.Forms.LinkLabel lnkCfgDomain;
        private System.Windows.Forms.LinkLabel lnkCfgMail;
        private System.Windows.Forms.LinkLabel lnkViewLog;
        private System.Windows.Forms.Button btnAbout;
        private System.Windows.Forms.NotifyIcon notifyIcon1;

    }
}
```

Form Code: Solution > MailServer.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Collections;

namespace Mail_Server
{
    /*
     * This is the main Windows Form class for the Mail Server Dialog.
     * This dialog is the entry dialog of the Mail Server
     */
    public partial class MailServer : Form
    {
        /*
         * The Default constructor for the MailServer class
         */
        public MailServer()
        {
            InitializeComponent();
        }

        /*
         * This function is invoked when the "Configure POP3/SMTP link is 
clicked.
         * This function is responsible for showing and updating the POP3 and 
SMTP
         * port configuration with the help of the configuration dialog.
         */
        private void lnkCfgMail_LinkClicked(object sender, 
LinkLabelLinkClickedEventArgs e)
        {
            try
            {
                CfgMailServer dlg = new CfgMailServer();
                dlg.ShowDialog(this);
            }
            catch
            {
            }
        }
```

[illegible]

```
        /*
         * This function is invoked when the "Configure E-Mail Domains" link
is clicked.
         * The function is responsible for showing the Mail Domains and is
also responsible
         * for adding and removing the Mail domains using the help of the
configuration dialog.
         */
        private void lnkCfgDomain_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            try
            {
                CfgDomain cfg = new CfgDomain();
                cfg.ShowDialog(this);
            }
            catch
            {
            }
        }

        /*
         * This function is invoked when the "Configure the E-Mail Users"
link is clicked.
         * The function is responsible for viewing, adding, updating or
removing an E-Mail
         * User and his/her account along with his Mail Inbox (if there is
any, in particular).
         */
        private void lnkCfgUser_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            try
            {
                CfgUser cfg = new CfgUser();
                cfg.ShowDialog(this);
            }
            catch
            {
            }
        }

        /*
         * This function is invoked when the form is loading.
         * It will just be used to execute any loadtime configuration.
         */
        private void MailServer_Load(object sender, EventArgs e)
        {
            Control.CheckForIllegalCrossThreadCalls = false;
            CServerLog.LogEntry("INFORMATION", "Mail Server application
started.");

            // The Server Listeners are not started. So why do we need the
Stop now? Just disable it.
```

```
            btnStop.Enabled = false;
        }

        /*
         * This function is invoked when the "View Log" link is clicked.
         * This function will open up a dialog box where the XML based Mail
Log file will be
         * displayed.
         */
        private void lnkViewLog_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            try
            {
                ServerLogViewer viewer = new ServerLogViewer();
                viewer.ShowDialog(this);
            }
            catch
            {
            }
        }

        /*
         * This function will be invoked when the MailServer dialog based
Form will be closed.
         * When this function is invoked, will indicate that this application
is ending.
         * So, just execute the task that we need to do, like saying Bye
Bye!!!
         */
        private void MailServer_FormClosed(object sender, FormClosedEventArgs
e)
        {
            CServerLog.LogEntry("INFORMATION", "Mail Server application
ended.");
        }

        /*
         * This function will be invoked when the button "About" is clicked.
         * This function will show the following informations in a
MessageBox.
         * The information displayed is generally about the project itself.
         */
        private void btnAbout_Click(object sender, EventArgs e)
        {
            StringBuilder sb = new StringBuilder();
            sb.Append("Mail Server:\n");
            sb.Append("     A small scale and simple mail server made for
home users and small office users.\n");
            sb.Append("\n");
            sb.Append("         Application written by Arnav
Mukhopadhyay.\n");
            sb.Append("         BCA Semester    -   6 (Project) [Session Jan
2010]\n");
```

[illegible]

```
            sb.Append("         Roll Number     -   510728180\n");
            sb.Append("         LC Code         -   02707\n");
            sb.Append("\n");
            sb.Append("     Requirements:\n");
            sb.Append("         Microsoft .NET Framework 3.5\n");
            sb.Append("         Microsoft Windows XP, Windows XP x64, Windows
Vista, Windows Vista x64, Windows 7 or above OS\n");
            sb.Append("\n");
            sb.Append("     Project Development Environment:\n");
            sb.Append("         Written using Visual C# and Microsoft .NET
Framework 3.5\n");
            sb.Append("         Compiled with Microsoft Visual Studio
2008\n");
            sb.Append("         Uses Thread, Sockets / WCF, LINQ, XML,
Windows Forms\n\n");

            MessageBox.Show(sb.ToString(),
                "About Mail Server",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }

        /*
         * This function is invoked when the button "Start" is clicked.
         * This will start the POP3 and SMTP server.
         */
        private void btnStart_Click(object sender, EventArgs e)
        {
            OnStartServer();
        }

        /*
         * This function is invoked when the button "Stop" is clicked.
         * This will stop the POP3 and SMTP server.
         */
        private void btnStop_Click(object sender, EventArgs e)
        {
            OnStopServer();
        }

        /*
         * This boolean variable will save the running state of the Server.
         * A "true" variable indicate that the server is running.
         */
        private bool bStarted = false;

        /*
         * This variable carries the reference to the Thread class that holds
the
         * POP3 server instance.
         */
        private Thread thPop3 = null;

        /*
```

[illegible]

```
          * This variable carries the reference to the Thread class that holds
the
          * SMTP server instance.
          */
         private Thread thSmtp = null;

         /*
          * This function is originally responsible for starting the server
instance.
          * This will initialize the Threads that actually starts the POP3 and
SMTP
          * listener and bind it to All available Network and wait for the
connections
          * that the client calls for on the Mail server.
          */
         private void OnStartServer()
         {
             if (bStarted == true)
             {
                 return;
             }

             try
             {
                 // Create a thread that will listen for client on POP3 port
                 thPop3 = new Thread(Pop3ThreadFunc);
                 thPop3.Start();

                 // Create a thread that will listen for client on Smtp port
                 thSmtp = new Thread(SmtpThreadFunc);
                 thSmtp.Start();
             }
             catch (Exception ex)
             {
                 CServerLog.LogEntry("EXCEPTION", ex.ToString());
                 MessageBox.Show("Exception occurred during starting of
Server. Please re-start the application.",
                     "Exception occurred",
                     MessageBoxButtons.OK,
                     MessageBoxIcon.Stop);
                 return;
             }

             btnStart.Enabled = false;
             btnStop.Enabled = true;
             lnkCfgMail.Enabled = false;
             bStarted = true;
             CServerLog.LogEntry("SUCCESS", "Mail Server Started
successfully.\nIt is now listening for incoming mail connections on POP3 and
SMTP port.");
         }

         /*
          * This function is originally responsible for Stopping the Threads
 responsible
          * for listening to POP3 and SMTP clients on the network.
          */
         private void OnStopServer()
         {
             if (bStarted == false)
             {
                 return;
             }

             if (thPop3 != null)
             {
                 if (thPop3.IsAlive == true)
                 {
                     thPop3.Abort();
                     thPop3.Join();
                 }
             }

             if (thSmtp != null)
             {
                 if (thSmtp.IsAlive == true)
                 {
                     thSmtp.Abort();
                     thSmtp.Join();
                 }
             }

             btnStart.Enabled = true;
             btnStop.Enabled = false;
             lnkCfgMail.Enabled = true;
             bStarted = false;
             CServerLog.LogEntry("SUCCESS", "Mail Server was Stopped
 successfully.\nIt will not interchange mails on any POP3 or SMTP ports.");
         }

         /*
          * This is the function invoked on an independent Thread.
          * This function initializes the POP3 listener TCP Socket and waits
 for the
          * incoming connection to be made by Mail clients.
          * On accepting a client, the server spawns child threads, each of
 which is responsible
          * for handling individual client's request.
          */
         private void Pop3ThreadFunc()
         {
             TcpListener lis = null;
             try
             {
                 CServerLog.LogEntry("INFORMATION", "Pop3 server
 starting...");
```

[illegible]

```
                try
                {
                    CServerCfg cfg = new CServerCfg();

                    // Read the POP3 listener port from the Mail Sever
configuration file.
                    int pop3Port =
int.Parse(cfg.Root.Element("Pop3Port").Value);

                    // Create a Tcp Listener to listen to the POP3 port on
all available
                    // Network interface.
                    lis = new TcpListener(IPAddress.Parse("0.0.0.0"),
pop3Port);

                    // Start the Listener Socket and bind it to the Network
interfaces.
                    lis.Start();
                    CServerLog.LogEntry("SUCCESS", "POP3 server started and
waiting for client to connect.");
                }
                catch (Exception ex)
                {
                    // Oops!!! I think something went wrong.... What to do
now????

                    // Show the Exception to user and Log it.
                    MessageBox.Show("Exception starting POP3 server: " +
ex.ToString(),
                        "POP3 Server cannot be started",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Stop);
                    CServerLog.LogEntry("FAILED", "Exception starting POP3
Server: " + ex.ToString());

                    // Rethrow it... So that we get out of this mess.
                    throw ex;
                }

                // Wait for listeners till eternity or until the Thread is
aborted.
                while (true)
                {
                    // Poll the POP3 TCP listener  for any client
                    // If we get true result, then we have a client waiting
to be accepted.
                    if (lis.Server.Poll(1000, SelectMode.SelectRead) == true)
                    {
                        // Log the client
                        CServerLog.LogEntry("INFORMATION", "A POP3 client is
waiting for connection.");

                        // get the client TCP Client socket interface and
spawn a new thread
```

```
                        // to service the POP3 client
                        TcpClient client = lis.AcceptTcpClient();
                        Thread th = new Thread(Pop3Client);
                        th.Start(client);
                    }
                    else
                    {
                        // Wait for 100ms before we re-poll
                        Thread.Sleep(100);
                    }
                }
            }
            catch
            {
                // Some exception occurred.... lets do nothing here
            }
            finally
            {
                // so now its done
                // let's stop the listener if there is one and that's it.
                if (lis != null)
                {
                    lis.Stop();
                }
            }
        }

        /*
         * This function is invoked by the POP3 listener, once for every
client request
         * made, and is on a new thread.
         * This enables to invoke multiple independent instance of this
function, thus
         * enabling the handling of multiple clients at once.
         */
        private void Pop3Client(object o)
        {
            TcpClient client = null;
            EndPoint remote = null;
            try
            {
                string userName = "";
                string password = "";
                string mailFolder = "";

                // Use the client as TCP Socket and gets it's value from the
object parameter
                client = o as TcpClient;

                // Retrieve the respective streams from the TCP client socket
to perform
                // buffered Network IO
                NetworkStream ns = client.GetStream();
                StreamReader sin = new StreamReader(ns,
System.Text.Encoding.ASCII, true);
                StreamWriter sout = new StreamWriter(ns,
System.Text.Encoding.ASCII);

                // Show the Welcome Message to POP3 client.
                sout.WriteLine("+OK POP3 ready at " + Environment.MachineName
+"\r");
                sout.Flush();

                // Get the Remote End Point IP
                remote = client.Client.RemoteEndPoint;
                CServerLog.LogEntry("POP3 CLIENT", "A POP3 client connected
from " + remote.ToString());

                string cmd;
                // Poll the POP3 client for commands and reply to them with
requisite request
                do
                {
                    // Poll this client socket to see if there exist any data
to be read
                    // If the return is true, the read the input command sent
by client
                    if (client.Client.Poll(1000, SelectMode.SelectRead) ==
true)
                    {
                        // Read a line of command from the Network input
stream
                        cmd = sin.ReadLine().Trim();

                        // Client wants to QUIT and so do Bye Bye to the
client.
                        if (cmd.ToUpper() == "QUIT")
                        {
                            sout.WriteLine("+OK Bye Bye!!!\r");
                            sout.Flush();
                            break;
                        }
                        // It is the "USER" command, so get the user name the
client want to
                        // authenticate for use
                        else if (cmd.Substring(0,Math.Min(4,
cmd.Length)).ToUpper() == "USER")
                        {
                            userName = cmd.Substring(5);
                            sout.WriteLine("+OK Please enter the
password.\r");
                            sout.Flush();
                            CServerLog.LogEntry("POP3 CLIENT AUTH USER", "The
client at " + remote.ToString() + " authorizing with user ID " + userName +
".");
                        }
                        // The client sent the "PASS" command, so get the
password and
                        // authenticate the account.
                        else if (cmd.Substring(0, Math.Min(4,
cmd.Length)).ToUpper() == "PASS")
                        {
                            password = cmd.Substring(5);
                            // Verify the User name and his password and in
turn also
                            // get his inbox location on the server.
                            // Everything is good, if this return true. false
value
                            // indicates a access violation or some kind of
error.
                            if (VerifyAccount(userName, password, ref
mailFolder) == true)
                            {
                                sout.WriteLine("+OK The Account has been
verified successfully.\r");
                                CServerLog.LogEntry("POP3 CLIENT AUTH", "The
client at " + remote.ToString() + " authorized with user ID " + userName + "
successfully.");
                            }
                            else
                            {
                                CServerLog.LogEntry("POP3 CLIENT AUTH
FAILED", "The client at " + remote.ToString() + " authorization with user ID
" + userName + " failed.");
                                sout.WriteLine("-ERR The User name or
Password is invalid. Please try again.\r");
                            }
                            sout.Flush();
                        }
                        // Client sent the "STAT" command, so send it the
mail statistics.
                        // The mail statistics is the number of mails on
server and the total
                        // size of the mails on server in octet.
                        // How to do it???
                        // List the files with extension .eml and get their
count and total size
                        // and send this values to the client.
                        else if (cmd.Trim('\r').ToUpper() == "STAT")
                        {
                            try
                            {
                                if (mailFolder == "")
                                {
                                    sout.WriteLine("-ERR Mailbox not found on
Server\r");
                                    sout.Flush();
                                    break;
                                }
                                DirectoryInfo dinfo = new
DirectoryInfo(Application.StartupPath + @"\" + mailFolder);
                                if (dinfo.Exists == false)
                                {
                                    sout.WriteLine("-ERR Server lost the
Mailbox\r");

                                    sout.Flush();
                                    break;
                                }

                                FileInfo[] finfoList =
dinfo.GetFiles("*.eml");

                                int count = 0;
                                long countSize = 0;
                                if (finfoList != null && finfoList.Length >
0)

                                {
                                    foreach (FileInfo finfo in finfoList)
                                    {
                                        if (finfo.Exists)
                                        {
                                            count++;
                                            countSize += finfo.Length;
                                        }
                                    }
                                }
                                sout.WriteLine("+OK {0} {1}\r", count,
countSize);

                                sout.Flush();
                            }
                            catch
                            {
                            }

                        }
                        // The client sent the "LIST" command, so send him
the statistics of

                        // the Mail on server and also the list of the mails
and their

                        // corresponding size to the client.
                        else if (cmd.Trim('\r').ToUpper() == "LIST")
                        {
                            try
                            {
                                if (mailFolder == "")
                                {
                                    sout.WriteLine("-ERR Mailbox not found on
Server\r");

                                    sout.Flush();
                                    break;
                                }
```

[illegible]

```
                                DirectoryInfo dinfo = new
DirectoryInfo(Application.StartupPath + @"\" + mailFolder);
                                if (dinfo.Exists == false)
                                {
                                    sout.WriteLine("-ERR Server lost the
Mailbox\r");

                                    sout.Flush();
                                    break;
                                }

                                FileInfo[] finfoList =
dinfo.GetFiles("*.eml");

                                int count = 0;
                                long countSize = 0;
                                if (finfoList != null && finfoList.Length >
0)

                                {
                                    foreach (FileInfo finfo in finfoList)
                                    {
                                        if (finfo.Exists)
                                        {
                                            count++;
                                            countSize += finfo.Length;
                                        }
                                    }
                                }
                                sout.WriteLine("+OK {0} messages  ({1}
octets)\r", count, countSize);
                                sout.Flush();

                                if (finfoList != null && finfoList.Length >
0)
                                {
                                    foreach (FileInfo finfo in finfoList)
                                    {
                                        if (finfo.Exists)
                                        {
                                            sout.WriteLine("{0} {1}\r",
finfo.Name.Substring(0, finfo.Name.IndexOf(".eml")).Trim(), finfo.Length);
                                            sout.Flush();
                                        }
                                    }
                                }

                                sout.WriteLine(".\r");
                                sout.Flush();
                            }
                            catch
                            {
                            }

                        }
                        // Client sent the "DELE" command, so lets delete the
EMAIL file number
                        // on the server
                        else if (cmd.Substring(0, Math.Min(4,
cmd.Length)).ToUpper() == "DELE")
                        {
                            try
                            {
                                string fname = cmd.Substring(5);
                                if (mailFolder == "")
                                {
                                    sout.WriteLine("-ERR Mailbox not found on
Server\r");

                                    sout.Flush();
                                    break;
                                }
                                FileInfo finfo = new
FileInfo(Application.StartupPath + @"\" + mailFolder + @"\" + fname +
".eml");
                                if (finfo.Exists)
                                {
                                    finfo.Delete();
                                }

                                sout.WriteLine("+OK Message {0} deleted\r",
fname);

                                sout.Flush();
                            }
                            catch
                            {
                            }

                        }
                        // Client sent the "RETR" command, so lets send it
the email file
                        // from the server as requested.
                        else if (cmd.Substring(0, Math.Min(4,
cmd.Length)).ToUpper() == "RETR")
                        {
                            try
                            {
                                string fname = cmd.Substring(5);
                                if (mailFolder == "")
                                {
                                    sout.WriteLine("-ERR Mailbox not found on
Server\r");

                                    sout.Flush();
                                    break;
                                }
                                int fileID = int.Parse(fname.Trim());
                                FileInfo finfo = new
FileInfo(Application.StartupPath + @"\" + mailFolder + @"\" + fileID +
".eml");
```

```
                                    if (finfo.Exists)
                                    {
                                        sout.WriteLine("+OK {0} octets\r", 
finfo.Length);
                                        sout.Flush();

                                        FileStream fs = finfo.Open(FileMode.Open, 
FileAccess.Read, FileShare.ReadWrite);
                                        int writeSize = (int) fs.Length;
                                        byte[] buffer = new byte[writeSize];
                                        writeSize = fs.Read(buffer, 0, 
writeSize);
                                        fs.Close();

                                        ns.Write(buffer, 0, writeSize);
                                        ns.Flush();
                                    }
                                    else
                                    {
                                        sout.WriteLine("-ERR E-Mail lost on 
server\r");
                                        sout.Flush();
                                        break;
                                    }

                                }
                                catch
                                {
                                }

                            }
                            // What is the client sending???? I don't know it.
                            // Send the client a ERROR message....
                            else
                            {
                                sout.WriteLine("-ERR The POP3 Server cannot 
understand the command.\r");
                                sout.Flush();
                            }
                        }
                        // So no command sent.... Wait for 100 ms and re-poll.
                        else
                        {
                            Thread.Sleep(100);
                            cmd = "";
                        }
                    }
                    while (client.Connected == true);
                }
                catch
                {
                    // Something wrong happened.... I don't want to know....
                }
                finally
```

```
            {
                // So we are done with Client anyway....
                // So close the client connection and log it, and say bye bye
to client.
                if (client != null)
                {
                    client.Close();
                    CServerLog.LogEntry("POP3 CLIENT CLOSED", "The Client at
" + remote.ToString() + " was closed.");
                }
            }
        }

        /*
         * This is the function solely reponsible for starting the SMTP
server.
         * This function creates the SMTP TCP listener and bind to all
available
         * network adapters and listens for SMTP clients to connect.
         * For eache client request it accepts, it spawns a new thread to
cater to
         * the service of the client.
         */
        private void SmtpThreadFunc()
        {
            TcpListener lis = null;
            try
            {
                CServerLog.LogEntry("INFORMATION", "Smtp server
starting...");
                try
                {
                    CServerCfg cfg = new CServerCfg();

                    // Read the SMTP port configuration from the XML based
Mail comfiguration file
                    int smtpPort =
int.Parse(cfg.Root.Element("SmtpPort").Value);

                    // Start the SMTP TCP listener interface and bind it to
                    // all network interfaces available on the assigned SMTP
port
                    lis = new TcpListener(IPAddress.Parse("0.0.0.0"),
smtpPort);

                    // Start the listening job
                    lis.Start();
                    CServerLog.LogEntry("SUCCESS", "SMTP server started and
waiting for client to connect.");
                }
                catch (Exception ex)
                {
                    // Something wrong just happened
                    // Display it to user and log it
                    MessageBox.Show("Exception starting SMTP server: " +
ex.ToString(),
                        "SMTP Server cannot be started",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Stop);
                    CServerLog.LogEntry("FAILED", "Exception starting SMTP
Server: " + ex.ToString());

                    // Rethrow the exception, so we can get out of this mess
                    throw ex;
                }

                // start to listen till eternity or till this thread is
aborted
                while (true)
                {
                    // Poll this socket for readable data which will return
true
                    // when there is client to be accepted.
                    if (lis.Server.Poll(1000, SelectMode.SelectRead) == true)
                    {
                        CServerLog.LogEntry("INFORMATION", "A SMTP client is
waiting for connection.");

                        // Accept the new SMTP client and spawn a new child
thread
                        // and let it cater the new client.
                        TcpClient client = lis.AcceptTcpClient();
                        Thread th = new Thread(SmtpClient);
                        th.Start(client);
                    }
                    else
                    {
                        // No client is available... So just wait for 100ms
and repoll again
                        Thread.Sleep(100);
                    }
                }
            }
            catch
            {
                // Something wrong happened... We don't want to know
            }
            finally
            {
                // So we are done with the listener....
                // Lets stop it and free the network adapters.
                if (lis != null)
                {
                    lis.Stop();
                }
            }

        }
        /*
         * This is the function that caters the SMTP client request.
         * This function is invoked as a result of the child thread created
         * for each SMTP client that connects.
         */
        private void SmtpClient(object o)
        {
            TcpClient client = null;
            EndPoint remote = null;
            try
            {
                // Create a place holder to hold the inbox folders, that 
corresponds
                // to the mail being send to by the client.
                ArrayList folderList = new ArrayList();

                // Get the TcpClient interface from the object
                client = o as TcpClient;

                // Retrieve the respective Network Stream from the Tcp Client 
socket
                NetworkStream ns = client.GetStream();
                StreamReader sin = new StreamReader(ns, 
System.Text.Encoding.ASCII, true);
                StreamWriter sout = new StreamWriter(ns, 
System.Text.Encoding.ASCII);

                // Show the Welcome message to client and flag it that the 
server is
                // ready to interchange messages.
                sout.WriteLine("220 " + Environment.MachineName + " SMTP 
ready.\r");
                sout.Flush();

                // Get the remote end IP address.
                remote = client.Client.RemoteEndPoint;
                CServerLog.LogEntry("SMTP CLIENT", "A SMTP client connected 
from " + remote.ToString());

                string cmd;

                // Poll the SMTP client for command and service the request.
                do
                {
                    // Poll this client socket for data and if the return 
result
                    // is a true value, then there is command waiting to be 
read.
                    if (client.Client.Poll(1000, SelectMode.SelectRead) == 
true)
                    {
                        // Read in the command sent by client.
                        cmd = sin.ReadLine().Trim();
```

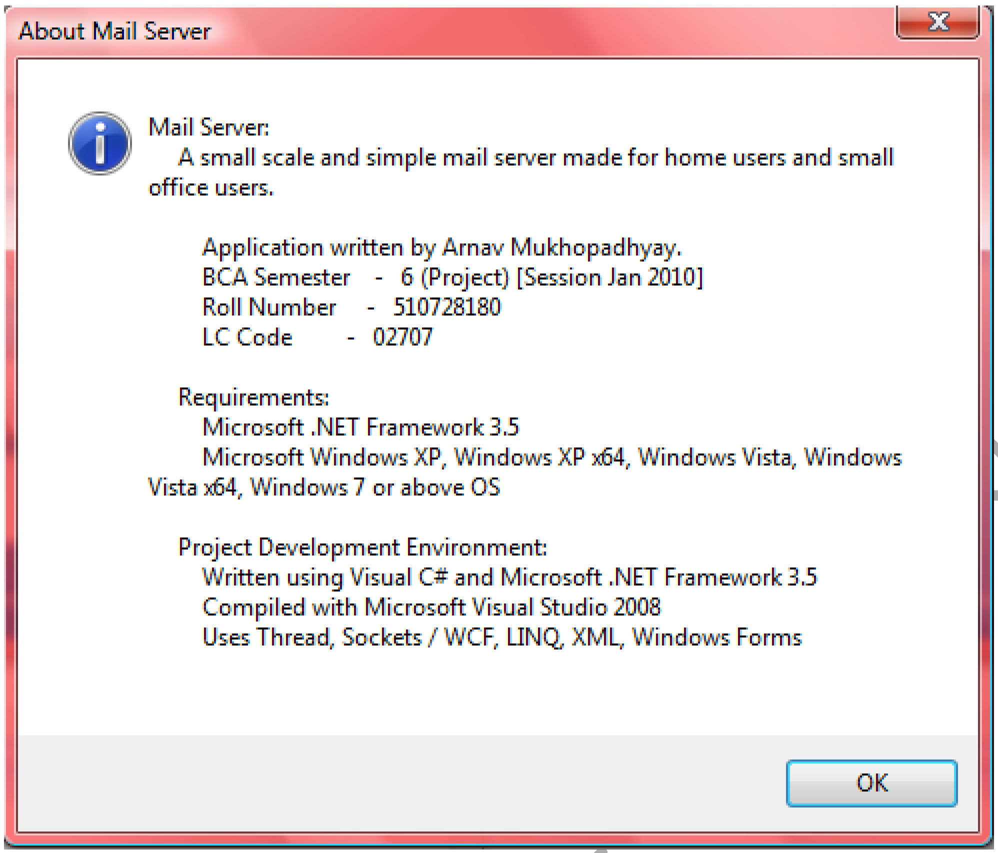
[illegible]

```
                        // Client wants to quit so Bye Bye it.
                        if (cmd.ToUpper() == "QUIT")
                        {
                            sout.WriteLine("221 Bye Bye!!!\r");
                            sout.Flush();
                            break;
                        }
                        // Client sent in "HELO" command, so log him at
server.
                        // This marks client as a valid SMTP client.
                        else if (cmd.Substring(0, Math.Min(4,
cmd.Length)).ToUpper() == "HELO")
                        {
                            string domainName =
cmd.Substring(5).Trim().Trim('\r');
                            sout.WriteLine("250 " + Environment.MachineName +
"\r");

                            sout.Flush();
                            CServerLog.LogEntry("SMTP CLIENT DOMAIN", "The
client at " + remote.ToString() + " is connected from " + domainName + ".");
                        }
                        // The client is an ESMTP client as it sent "EHLO". I
don't know
                        // ESMTP protocol... So please retry by sending in
"HELO"
                        else if (cmd.Substring(0, Math.Min(4,
cmd.Length)).ToUpper() == "EHLO")
                        {
                            string domainName =
cmd.Substring(5).Trim().Trim('\r');
                            sout.WriteLine("500 Not supported. Use HELO\r");
                            sout.Flush();
                            CServerLog.LogEntry("ESMTP CLIENT DOMAIN", "The
client at " + remote.ToString() + " is connecting from " + domainName + ".");
                        }
                        // Client sent "MAIL-FROM:" command, so validate the
client
                        // Check to see if the sender belongs to this server.
                        // I don't want spammer or relay services to send in
mail here.
                        // Also clear the receiver folder path place holder
array list as
                        // this marks the beginning of the new mail.
                        else if (cmd.Substring(0, Math.Min(10,
cmd.Length)).ToUpper() == "MAIL FROM:")
                        {
                            string senderEmail = cmd.Substring(11);
                            if (VerifyEMail(senderEmail) == true)
                            {
                                sout.WriteLine("250 OK\r");
                                folderList.Clear();
                            }
                            else
                            {
                                sout.WriteLine("500 ERR Email ID " +
senderEmail + " not found.\r");
                                sout.Flush();
                                break;
                            }
                            sout.Flush();
                        }
                        // Client sent "RCPT TO:", so check the attached
email addresses that
                        // they belong on this server, if not found, then
report the absence to
                        // the client and abort further mail transaction.
                        // If found, then get the inbox folder path on the
server and add to
                        // the folder place holder array list we created
before.
                        else if (cmd.Substring(0, Math.Min(8,
cmd.Length)).ToUpper() == "RCPT TO:")
                        {
                            string rcptEmail = cmd.Substring(9);
                            string inboxPath = "";
                            if (getInbox(rcptEmail, ref inboxPath) == true)
                            {
                                sout.WriteLine("250 OK\r");
                                folderList.Add(inboxPath);
                            }
                            else
                            {
                                sout.WriteLine("500 ERR Email ID " +
rcptEmail + " not found.\r");
                                sout.Flush();
                                break;
                            }
                            sout.Flush();
                        }
                        // Client sent in "DATA" command, which indicates
that the client
                        // wants to send in the email data. So hold it and
then copy to
                        // a new .eml file in the receiver's email folder
path.
                        else if (cmd.ToUpper() == "DATA")
                        {
                            // Flag the client with a message of how to send
in the mail

                            sout.WriteLine("354 Start mail input; end with
<CRLF>.<CRLF>\r");

                            sout.Flush();

                            // create a array list object to act as email
read-in buffer.

                            ArrayList dataList = new ArrayList();
                            byte ch;
                            // Read in the data and detect the end of mail
data. Otherwise,
                            // save it in the dataList buffer.
                            do
                            {
                                ch = (byte)ns.ReadByte();
                                dataList.Add(ch);

                                if (ch == '\n')
                                {
                                    ch = (byte)ns.ReadByte();
                                    dataList.Add(ch);

                                    if (ch == '.')
                                    {
                                        ch = (byte)ns.ReadByte();
                                        dataList.Add(ch);

                                        if (ch == '\r')
                                        {
                                            ch = (byte)ns.ReadByte();
                                            dataList.Add(ch);

                                            if (ch == '\n')
                                            {
                                                // End of Mail data
                                                break;
                                            }
                                        }
                                    }
                                }


                            }
                            while (true);

                            // now retrieve the Array of Client Inbox folder
names
                            // and save the email data in this folder as a
new .eml
                            // file.
                            Array dataArray = dataList.ToArray();
                            foreach (string folderName in folderList)
                            {
                                SaveData(folderName, dataArray);
                            }

                            // we are done saving so.... tell the client.
                            sout.WriteLine("250 OK\r");
                            sout.Flush();

                        }
                        // The user sent a blank command... We have nothing
to do
```

```
                        else if (cmd.Trim().Trim(new char[] { '\r', '\n' })
== "")
                        {

                        }
                        // Cannot understand what the user just sent in.
                        else
                        {
                            sout.WriteLine("500 The SMTP Server cannot
understand the command.\r");
                            sout.Flush();
                            break;
                        }
                    }
                    // There is no data to be read so... just wait for 100 ms
                    else
                    {
                        Thread.Sleep(100);
                        cmd = "";
                    }
                }
                while (client.Connected == true);
            }
            catch
            {
                // Some exception just occurred.... We don't want to know
            }
            finally
            {
                // We are done with the client. So close it now.
                if (client != null)
                {
                    client.Close();
                    CServerLog.LogEntry("SMTP CLIENT CLOSED", "The Client at
" + remote.ToString() + " was closed.");
                }
            }

        }

        /*
         * This function is called when the form is about to close.
         * So lets stop the server before this form closes.
         */
        private void MailServer_FormClosing(object sender,
FormClosingEventArgs e)
        {
            OnStopServer();
        }

        /*
         * Called to verify the email account against the internal user
database.
```

[illegible]

```
         * This requires the userName which is in format EMAIL@DOMAIN.XYZ
format,
         * along with the valid password for the acoount.
         * If the function returns true, then the mailFolder variable holds
the
         * EMAIL inbox path on the server, and also that the account is
authenticated
         * and valid for the transaction to proceed.
         * Otherwise, don't proceed with it... It is invalid.
         */
        private bool VerifyAccount(string userName, string password, ref
string mailFolder)
        {
            try
            {
                CServerCfg cfg = new CServerCfg();
                XElement elUsers = cfg.Root.Element("Users");
                foreach (XElement elDomain in elUsers.Nodes())
                {
                    foreach (XElement elUser in elDomain.Nodes())
                    {
                        string emailID = elUser.Name.LocalName + "@" +
elDomain.Name.LocalName;
                        string pass = elUser.Element("pass").Value;
                        string folder = elUser.Element("folder").Value;

                        if (emailID.ToLower() == userName.ToLower() &&
                           pass == password)
                        {
                            mailFolder = folder;
                            return true;
                        }
                    }
                }
            }
            catch
            {
            }
            return false;
        }

        /*
         * Called to verify the email account against the internal user
database.
         * This requires the userName which is in format EMAIL@DOMAIN.XYZ
format.
         * If the function returns true, the account is authenticated
         * and valid for the transaction to proceed.
         * Otherwise, don't proceed with it... It is invalid.
         */
        private bool VerifyEMail(string mailAddress)
        {
            try
            {
                mailAddress = mailAddress.ToLower().Trim();
                mailAddress = mailAddress.Trim(new char[] { '<', '>'});
                CServerCfg cfg = new CServerCfg();
                XElement elUsers = cfg.Root.Element("Users");
                foreach (XElement elDomain in elUsers.Nodes())
                {
                    foreach (XElement elUser in elDomain.Nodes())
                    {
                        string emailID = elUser.Name.LocalName + "@" +
elDomain.Name.LocalName;

                        if (emailID.ToLower() == mailAddress)
                        {
                               return true;
                        }
                    }
                }
            }
            catch
            {
            }
            return false;
        }

        /*
         * Called to retreive the email account inbox path on server from the
internal user database.
         * This requires the mailAddress in format EMAIL@DOMAIN.XYZ format.
         * If the function returns true, then the mailFolder variable holds
the
         * EMAIL inbox path on the server, and also that the account is
authenticated
         * and valid for the transaction to proceed.
         * Otherwise, don't proceed with it... It is invalid.
         */
        private bool getInbox(string mailAddress, ref string mailFolder)
        {
            try
            {
                mailAddress = mailAddress.ToLower().Trim();
                mailAddress = mailAddress.Trim(new char[] { '<', '>' });

                CServerCfg cfg = new CServerCfg();
                XElement elUsers = cfg.Root.Element("Users");
                foreach (XElement elDomain in elUsers.Nodes())
                {
                    foreach (XElement elUser in elDomain.Nodes())
                    {
                        string emailID = elUser.Name.LocalName + "@" +
elDomain.Name.LocalName;
                        string folder = elUser.Element("folder").Value;

                        if (emailID.ToLower() == mailAddress)
                        {
                            mailFolder = folder;
                            return true;
                        }
                    }
                }
            }
            catch
            {
            }
            return false;
        }

        /*
         * This function saves the char data in the Array reference in
mailData
         * to a new .eml file in inboxName path on the server.
         */
        private void SaveData(string inboxName, Array mailData)
        {
            // detect a filename to use
            string fileName;
            FileInfo finfo;

            for (int i = 1; ; i++)
            {
                fileName = Application.StartupPath + @"\" + inboxName + @"\"
+ i.ToString() + ".eml";
                finfo = new FileInfo(fileName);
                if (finfo.Exists == false)
                {
                    // got the required file
                    FileStream fs = finfo.Create();
                    foreach (byte ch in mailData)
                    {
                        fs.WriteByte(ch);
                    }
                    fs.Close();

                    // done
                    return;
                }
            }
        }

        /*
         * This function is called when the notification icon is called.
         * If the Mail Server dialog is visible, then it is hidden from view,
         * otherwise, it is hidden from the view.
         */
        private void notifyIcon1_MouseDoubleClick(object sender,
MouseEventArgs e)
        {
            if (this.Visible == true)
            {
                this.Hide();
            }
            else
            {
                this.Show();
            }
        }
    }
}
```


About Mail Server

Mail Server:
 A small scale and simple mail server made for home users and small office users.

 Application written by Arnav Mukhopadhyay.
 BCA Semester - 6 (Project) [Session Jan 2010]
 Roll Number - 510728180
 LC Code - 02707

 Requirements:
 Microsoft .NET Framework 3.5
 Microsoft Windows XP, Windows XP x64, Windows Vista, Windows Vista x64, Windows 7 or above OS

 Project Development Environment:
 Written using Visual C# and Microsoft .NET Framework 3.5
 Compiled with Microsoft Visual Studio 2008
 Uses Thread, Sockets / WCF, LINQ, XML, Windows Forms

This is the "About Mail Server" dialog box. This displays some general information about the Mail Server and related information.

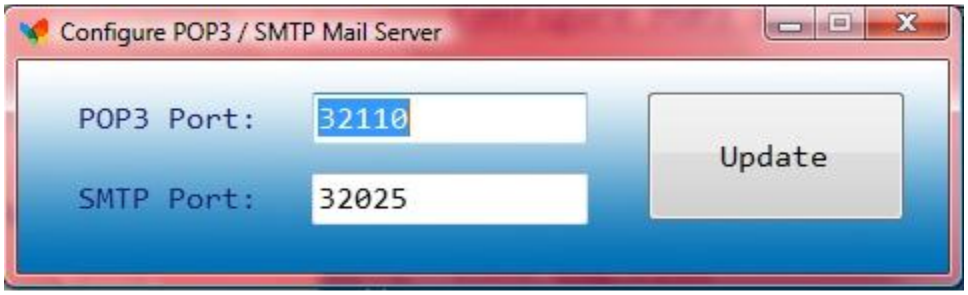
Form Code: Solution > MailServer.cs (Line 123 – 153)

```
        /*
         * This function will be invoked when the button "About" is clicked.
         * This function will show the following informations in a MessageBox.
         * The information displayed is generally about the project itself.
         */
        private void btnAbout_Click(object sender, EventArgs e)
        {
            StringBuilder sb = new StringBuilder();
            sb.Append("Mail Server:\n");
            sb.Append("     A small scale and simple mail server made for home users and small office users.\n");
            sb.Append("\n");
            sb.Append("         Application written by Arnav Mukhopadhyay.\n");
            sb.Append("         BCA Semester    -   6 (Project) [Session Jan 2010]\n");
```

[illegible]

```
            sb.Append("          Roll Number      -    510728180\n");
            sb.Append("          LC Code          -    02707\n");
            sb.Append("\n");
            sb.Append("      Requirements:\n");
            sb.Append("          Microsoft .NET Framework 3.5\n");
            sb.Append("          Microsoft Windows XP, Windows XP x64, Windows
Vista, Windows Vista x64, Windows 7 or above OS\n");
            sb.Append("\n");
            sb.Append("      Project Development Environment:\n");
            sb.Append("          Written using Visual C# and Microsoft .NET
Framework 3.5\n");
            sb.Append("          Compiled with Microsoft Visual Studio
2008\n");
            sb.Append("          Uses Thread, Sockets / WCF, LINQ, XML,
Windows Forms\n\n");

            MessageBox.Show(sb.ToString(),
                "About Mail Server",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }
```

[illegible]



This is the Configuration dialog box. The dialog box lets you enter the POP3 and SMTP port settings. On clicking the update button will update the server listening configuration. But if changes are made, the server must be stopped and restarted to use the new settings.

Form Code: Solution > CfgMailServer.cs > CfgMailServer.Designer.cs

```
namespace Mail_Server
{
    partial class CfgMailServer
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(CfgMailServer));
```

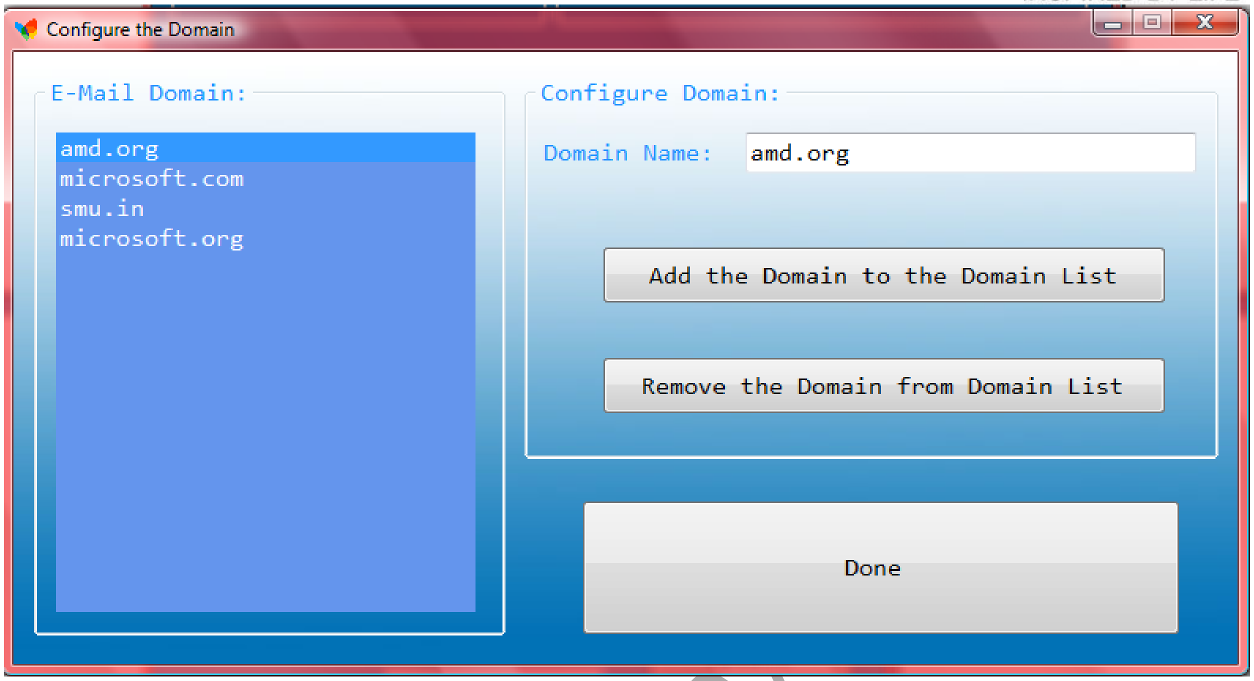
```
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.tbPop = new System.Windows.Forms.TextBox();
            this.tbSmtp = new System.Windows.Forms.TextBox();
            this.btnUpdate = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.BackColor = System.Drawing.Color.Transparent;
            this.label1.ForeColor = System.Drawing.Color.MidnightBlue;
            this.label1.Location = new System.Drawing.Point(26, 19);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(99, 19);
            this.label1.TabIndex = 0;
            this.label1.Text = "POP3 Port:";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.BackColor = System.Drawing.Color.Transparent;
            this.label2.ForeColor = System.Drawing.Color.MidnightBlue;
            this.label2.Location = new System.Drawing.Point(26, 59);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(99, 19);
            this.label2.TabIndex = 1;
            this.label2.Text = "SMTP Port:";
            //
            // tbPop
            //
            this.tbPop.Location = new System.Drawing.Point(147, 16);
            this.tbPop.MaxLength = 8;
            this.tbPop.Name = "tbPop";
            this.tbPop.Size = new System.Drawing.Size(138, 26);
            this.tbPop.TabIndex = 2;
            this.tbPop.TextChanged += new
System.EventHandler(this.tbPop_TextChanged);
            //
            // tbSmtp
            //
            this.tbSmtp.Location = new System.Drawing.Point(147, 56);
            this.tbSmtp.MaxLength = 8;
            this.tbSmtp.Name = "tbSmtp";
            this.tbSmtp.Size = new System.Drawing.Size(138, 26);
            this.tbSmtp.TabIndex = 3;
            this.tbSmtp.TextChanged += new
System.EventHandler(this.tbSmtp_TextChanged);
            //
            // btnUpdate
            //
            this.btnUpdate.Location = new System.Drawing.Point(314, 15);
            this.btnUpdate.Name = "btnUpdate";
            this.btnUpdate.Size = new System.Drawing.Size(130, 66);
            this.btnUpdate.TabIndex = 4;
            this.btnUpdate.Text = "Update";
            this.btnUpdate.UseVisualStyleBackColor = true;
            this.btnUpdate.Click += new
System.EventHandler(this.btnUpdate_Click);
            //
            // CfgMailServer
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 19F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackgroundImage =
global::Mail_Server.Properties.Resources.Untitled_2;
            this.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.ClientSize = new System.Drawing.Size(465, 106);
            this.Controls.Add(this.btnUpdate);
            this.Controls.Add(this.tbSmtp);
            this.Controls.Add(this.tbPop);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.DoubleBuffered = true;
            this.Font = new System.Drawing.Font("Consolas", 12F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.Margin = new System.Windows.Forms.Padding(4);
            this.MaximizeBox = false;
            this.Name = "CfgMailServer";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Configure POP3 / SMTP Mail Server";
            this.Load += new System.EventHandler(this.CfgMailServer_Load);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox tbPop;
        private System.Windows.Forms.TextBox tbSmtp;
        private System.Windows.Forms.Button btnUpdate;
    }
}
```

Form Code: Solution > CfgMailServer.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Linq;
using System.Xml.Linq;

namespace Mail_Server
{
    /*
     * This is the main class the Mail Server configuration dialog.
     */
    public partial class CfgMailServer : Form
    {
        /*
         * This variable holds the POP3 port number.
         */
        private int popPort = 110;

        /*
         * This variable holds the SMTP port number.
         */
        private int smtpPort = 25;

        /*
         * This is the default constructor for the CfgMailServer class.
         */
        public CfgMailServer()
        {
            InitializeComponent();
        }

        /*
         * This function is called when the form is loaded.
         * In this procedure, we would load the initial POP3 and SMTP
         * values from the Mail Configuration file, and display it in
         * their respective edit boxes.
         */
        private void CfgMailServer_Load(object sender, EventArgs e)
        {
            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();

                try
                {
                    popPort = int.Parse(cfg.Root.Element("Pop3Port").Value);
                }
                catch
                {
                    cfg.Root.Add(new XElement("Pop3Port",
popPort.ToString()));
                }

                try
                {
                    smtpPort = int.Parse(cfg.Root.Element("SmtpPort").Value);
                }
                catch
                {
                    cfg.Root.Add(new XElement("SmtpPort",
smtpPort.ToString()));
                }
            }
            catch
            {
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }

            tbPop.Text = popPort.ToString();
            tbSmtp.Text = smtpPort.ToString();
        }

        /*
         * This function is invoked when "Done" button is clicked.
         * Then save the new value of the POP3 and SMTP port numbers, if
modified.
         */
        private void btnUpdate_Click(object sender, EventArgs e)
        {
            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();
                cfg.Root.Element("Pop3Port").SetValue(popPort.ToString());
                cfg.Root.Element("SmtpPort").SetValue(smtpPort.ToString());

                MessageBox.Show("Mail Server Configuration updated and
saved.",
                    "Mail Server Configuration Updated",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
```

[illegible]

```
                MessageBox.Show("The Savings could not be updated.",
                    "Mail Server Update failed",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Warning);
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }

                this.Dispose();
            }
        }

        /*
         * This function is invoked when the user changes the text in SMTP
port edit box.
         * This function verifies if the text entered is valid as a SMTP port
number.
         */
        private void tbSmtp_TextChanged(object sender, EventArgs e)
        {
            try
            {
                smtpPort = int.Parse(tbSmtp.Text);
            }
            catch
            {
                tbSmtp.Text = smtpPort.ToString();
            }

        }

        /*
         * This function is invoked when the user changes the text in POP3
port edit box.
         * This function verifies if the text entered is valid as a POP3 port
number.
         */
        private void tbPop_TextChanged(object sender, EventArgs e)
        {
            try
            {
                popPort = int.Parse(tbPop.Text);
            }
            catch
            {
                tbPop.Text = popPort.ToString();
            }
        }
    }
}
```



This dialog is used to configure the E-Mail domains that will host the user email information. To add a domain to the domain list, just enter the domain name in "Domain Name:" dialog box and click on the "Add the Domain to the Domain List" button.

To remove an existing domain, just select the domain name from the "E-Mail Domain" list and then click on "Remove the Domain from Domain List". This will remove the domain and the user accounts hosted under it. It will further, remove and clear off any user email messages that are hosted under it.

Form Code: Solution > CfgDomain.cs > CfgDomain.Designer.cs

```
namespace Mail_Server
{
    partial class CfgDomain
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
```

```
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(CfgDomain));
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.lbDomains = new System.Windows.Forms.ListBox();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.btnRemove = new System.Windows.Forms.Button();
            this.btnAdd = new System.Windows.Forms.Button();
            this.tbDomain = new System.Windows.Forms.TextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.btnDone = new System.Windows.Forms.Button();
            this.groupBox1.SuspendLayout();
            this.groupBox2.SuspendLayout();
            this.SuspendLayout();
            //
            // groupBox1
            //
            this.groupBox1.BackColor = System.Drawing.Color.Transparent;
            this.groupBox1.Controls.Add(this.lbDomains);
            this.groupBox1.ForeColor = System.Drawing.Color.DodgerBlue;
            this.groupBox1.Location = new System.Drawing.Point(13, 16);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(299, 354);
            this.groupBox1.TabIndex = 0;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "E-Mail Domain:";
            //
            // lbDomains
            //
            this.lbDomains.BackColor = System.Drawing.Color.CornflowerBlue;
            this.lbDomains.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.lbDomains.ForeColor = System.Drawing.Color.White;
            this.lbDomains.FormattingEnabled = true;
            this.lbDomains.ItemHeight = 19;
            this.lbDomains.Location = new System.Drawing.Point(14, 35);
            this.lbDomains.Name = "lbDomains";
            this.lbDomains.Size = new System.Drawing.Size(266, 304);
            this.lbDomains.TabIndex = 0;
            this.lbDomains.SelectedIndexChanged += new
System.EventHandler(this.lbDomains_SelectedIndexChanged);
            //
            // groupBox2
            //
            this.groupBox2.BackColor = System.Drawing.Color.Transparent;
            this.groupBox2.Controls.Add(this.btnRemove);
            this.groupBox2.Controls.Add(this.btnAdd);
            this.groupBox2.Controls.Add(this.tbDomain);
            this.groupBox2.Controls.Add(this.label1);
            this.groupBox2.ForeColor = System.Drawing.Color.DodgerBlue;
            this.groupBox2.Location = new System.Drawing.Point(324, 16);
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(440, 242);
            this.groupBox2.TabIndex = 1;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "Configure Domain:";
            //
            // btnRemove
            //
            this.btnRemove.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.btnRemove.Location = new System.Drawing.Point(49, 177);
            this.btnRemove.Name = "btnRemove";
            this.btnRemove.Size = new System.Drawing.Size(358, 37);
            this.btnRemove.TabIndex = 4;
            this.btnRemove.Text = "Remove the Domain from Domain List";
            this.btnRemove.UseVisualStyleBackColor = true;
            this.btnRemove.Click += new
System.EventHandler(this.btnRemove_Click);
            //
            // btnAdd
            //
            this.btnAdd.ForeColor = System.Drawing.SystemColors.ControlText;
            this.btnAdd.Location = new System.Drawing.Point(49, 107);
            this.btnAdd.Name = "btnAdd";
            this.btnAdd.Size = new System.Drawing.Size(358, 37);
            this.btnAdd.TabIndex = 2;
            this.btnAdd.Text = "Add the Domain to the Domain List";
            this.btnAdd.UseVisualStyleBackColor = true;
            this.btnAdd.Click += new System.EventHandler(this.btnAdd_Click);
            //
            // tbDomain
            //
            this.tbDomain.Location = new System.Drawing.Point(140, 35);
            this.tbDomain.Name = "tbDomain";
            this.tbDomain.Size = new System.Drawing.Size(286, 26);
            this.tbDomain.TabIndex = 1;
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(8, 38);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(117, 19);
            this.label1.TabIndex = 0;
            this.label1.Text = "Domain Name:";
            //
            // btnDone
            //
            this.btnDone.Location = new System.Drawing.Point(360, 284);
            this.btnDone.Name = "btnDone";
            this.btnDone.Size = new System.Drawing.Size(380, 86);
            this.btnDone.TabIndex = 2;
            this.btnDone.Text = "Done ";
            this.btnDone.UseVisualStyleBackColor = true;
            this.btnDone.Click += new
System.EventHandler(this.btnDone_Click);
            //
            // CfgDomain
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 19F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackgroundImage =
global::Mail_Server.Properties.Resources.Untitled_2;
            this.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.ClientSize = new System.Drawing.Size(776, 388);
            this.Controls.Add(this.btnDone);
            this.Controls.Add(this.groupBox2);
            this.Controls.Add(this.groupBox1);
            this.DoubleBuffered = true;
            this.Font = new System.Drawing.Font("Consolas", 12F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.Margin = new System.Windows.Forms.Padding(4);
            this.MaximizeBox = false;
            this.Name = "CfgDomain";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Configure the Domain";
            this.Load += new System.EventHandler(this.CfgDomain_Load);
            this.groupBox1.ResumeLayout(false);
            this.groupBox2.ResumeLayout(false);
            this.groupBox2.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.ListBox lbDomains;
```

[illegible]

```
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button btnRemove;
        private System.Windows.Forms.Button btnAdd;
        private System.Windows.Forms.TextBox tbDomain;
        private System.Windows.Forms.Button btnDone;
    }
}
```

Form Code: Solution > CfgDomain.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Linq;
using System.IO;

namespace Mail_Server
{
    /*
     * This is the main form for Configuring E-Mail domain names.
     */
    public partial class CfgDomain : Form
    {
        /*
         * This is the default constructor for DfgDomain class.
         */
        public CfgDomain()
        {
            InitializeComponent();
        }

        /*
         * This function is called when the Domain configuration form is
loading.
         * Here we will load the XML configuration file, and enumerate
through
         * all the domain entries and display it in a list box on the form
for
         * the user to choose from.
         */
        private void CfgDomain_Load(object sender, EventArgs e)
        {
            CServerCfg cfg = null;
            try
```

[illegible]

```
    {
        cfg = new CServerCfg();
        try
        {
            XElement el = cfg.Root.Element("Domains");
            if (el == null)
                throw new Exception();

            foreach (XElement dmname in el.Nodes())
            {
                lbDomains.Items.Add(dmname.Name.LocalName);
            }

            if (lbDomains.Items.Count > 0)
            {
                lbDomains.SelectedIndex = 0;
            }
        }
        catch
        {
            cfg.Root.Add(new XElement("Domains"));
        }
    }
    catch
    {
    }
    finally
    {
        if (cfg != null)
        {
            cfg.Close();
        }
    }
}

/*
 * This function is invoked when the "Done" button is clicked.
 * So we are done, then destroy the form.
 */
private void btnDone_Click(object sender, EventArgs e)
{
    this.Dispose();
}

/*
 * Add the new domain to the list of list box on the form and also
 * to the mail configuration file.
 */
private void btnAdd_Click(object sender, EventArgs e)
{
    tbDomain.Text = tbDomain.Text.Trim();
    if (tbDomain.Text == "")
    {
        return;
```

```
            }

            if (lbDomains.Items.IndexOf(tbDomain.Text.ToLower()) >= 0)
            {
                return;
            }

            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();
                cfg.Root.Element("Domains").Add(new
XElement(tbDomain.Text.ToLower()));
                lbDomains.SelectedIndex =
lbDomains.Items.Add(tbDomain.Text.ToLower());
                MessageBox.Show("The Domain Name has been added to the Domain
List.",
                    "Domain Name Added",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
                MessageBox.Show("The Domain Name cannot be added to the
Domain List.",
                    "Domain Name Added Failed",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }

        }

        /*
         * This function will display the selected domain name on the list
box in tge
         * domain name text function, an dwill be invoked when the user
selects a
         * new domain name from the domain name list.
         */
        private void lbDomains_SelectedIndexChanged(object sender, EventArgs
e)
        {
            tbDomain.Text = lbDomains.Items[lbDomains.SelectedIndex] as
String;
        }

        /*
```

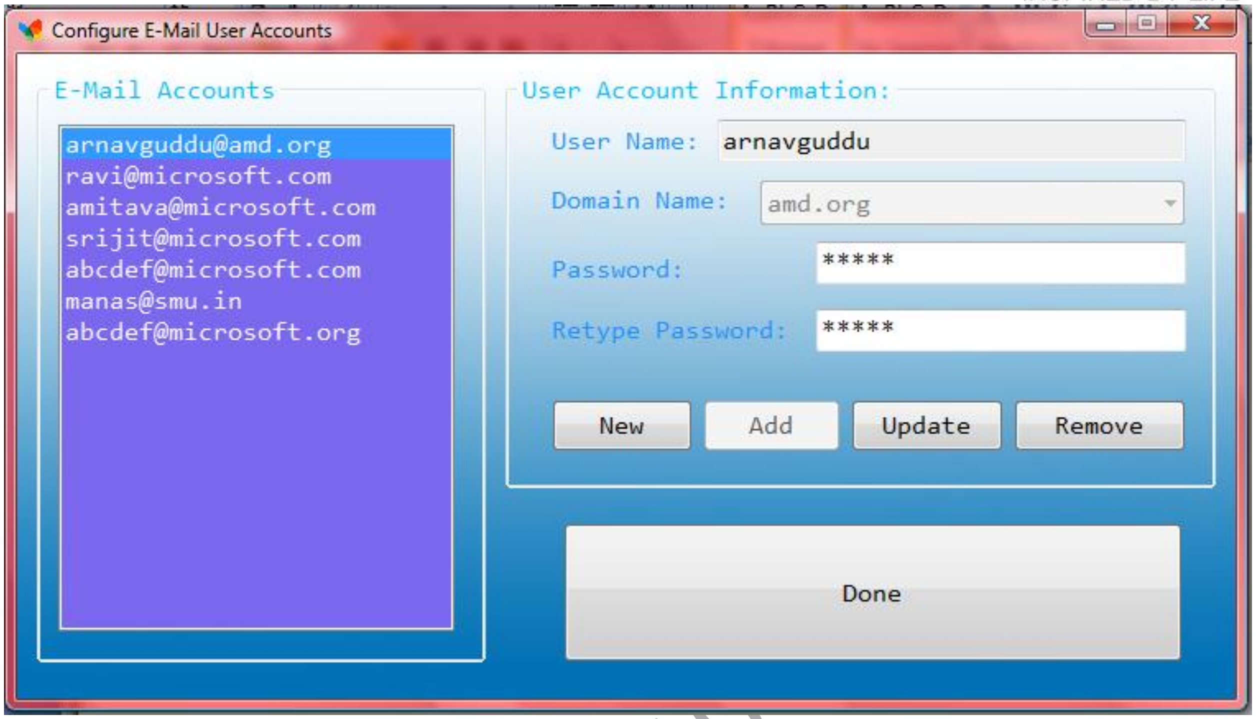
[illegible]

```
         * Remove the selected domain from the domain name list and
         * delete the domain name from Mail server configuration file. It
also
         * removes the users and inboxes configured under it.
         */
        private void btnRemove_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("You are about to delete the E-Mail Domain.
This will delete all the E-Mail Inbox configured with it and the dependent
user will loose all the mail content in their respective Inbox.\nAre you sure
with the delete option ?",
                                "Confirm Delete of Domain ?",
                                MessageBoxButtons.OKCancel,
                                MessageBoxIcon.Question) ==
DialogResult.Cancel)
                {
                        return;
                }

            CServerCfg cfg = null;
            try
            {

                cfg = new CServerCfg();

cfg.Root.Element("Domains").Element(lbDomains.Items[lbDomains.SelectedIndex].
ToString()).Remove();
                XElement udomain =
cfg.Root.Element("Users").Element(lbDomains.Items[lbDomains.SelectedIndex].To
String());
                if (udomain != null)
                {
                    foreach (XElement users in udomain.Nodes())
                    {
                        try
                        {
                            XElement ufolder = users.Element("folder");
                            DirectoryInfo dinfo = new
DirectoryInfo(Application.StartupPath + @"\" + ufolder.Value);
                            dinfo.Delete(true);
                        }
                        catch
                        {
                        }
                    }
                    udomain.Remove();
                }

                try
                {
                    lbDomains.Items.RemoveAt(lbDomains.SelectedIndex);
                }
                catch
                {
```

[illegible]

```
                }
                finally
                {
                    if (lbDomains.Items.Count > 0)
                    {
                        lbDomains.SelectedIndex = 0;
                    }
                }

                MessageBox.Show("The Domain name was removed successfully
from the Domain List.",
                    "Domain Name Removed",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
                MessageBox.Show("The Domain Name cannot be removed from the
Domain Name list.",
                    "Domain Name Removal Failed",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
            finally
            {
                cfg.Close();
            }
        }

    }
}
```

[illegible]



arnavguddu@amd.org
ravi@microsoft.com
amitava@microsoft.com
srijit@microsoft.com
abcdef@microsoft.com
manas@smu.in
abcdef@microsoft.org

User Name: arnavguddu
Domain Name: amd.org
Password: *****
Retype Password: *****

This dialog box is used to configure the Email user accounts. All the existing E-Mail accounts are listed in the "E-Mail Accounts" list.

To add a new user to the list, click on "New" button. Then enter the user name in "User Name" edit box, select a "Domain Name" from the list of Domain names. Then enter a password in "Password" edit box, and retype to verify in "Retype Password" dialog box. Then click on the "Add" button.

To edit an E-Mail account, select the E-Mail ID from the list and change the required values, then click on "Update" button to update the details.

To remove an user E-Mail ID, just click on the E-Mail ID from the list and click on the "Remove" button. This will remove the E-Mail account along with the messages. Be careful, once deleted the account information or its messages can never be recovered.

Once, everything is done, just click on the "Done" button and the form is closed.

Form Code: Solution > CfgUser.cs > CfgUser.Designer.cs

```
namespace Mail_Server
{
    partial class CfgUser
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(CfgUser));
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.lbUsers = new System.Windows.Forms.ListBox();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.btnRemove = new System.Windows.Forms.Button();
            this.btnUpdate = new System.Windows.Forms.Button();
            this.btnAdd = new System.Windows.Forms.Button();
            this.btnNew = new System.Windows.Forms.Button();
            this.cbDomains = new System.Windows.Forms.ComboBox();
            this.tbPassConfirm = new System.Windows.Forms.TextBox();
            this.tbPass = new System.Windows.Forms.TextBox();
            this.tbUser = new System.Windows.Forms.TextBox();
            this.label4 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label1 = new System.Windows.Forms.Label();
            this.btnDone = new System.Windows.Forms.Button();
            this.groupBox1.SuspendLayout();
            this.groupBox2.SuspendLayout();
```

[illegible]

```
            this.SuspendLayout();
            // 
            // groupBox1
            // 
            this.groupBox1.BackColor = System.Drawing.Color.Transparent;
            this.groupBox1.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.groupBox1.Controls.Add(this.lbUsers);
            this.groupBox1.ForeColor = System.Drawing.Color.DeepSkyBlue;
            this.groupBox1.Location = new System.Drawing.Point(12, 12);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(273, 358);
            this.groupBox1.TabIndex = 0;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "E-Mail Accounts";
            // 
            // lbUsers
            // 
            this.lbUsers.BackColor = System.Drawing.Color.MediumSlateBlue;
            this.lbUsers.ForeColor = System.Drawing.Color.White;
            this.lbUsers.FormattingEnabled = true;
            this.lbUsers.ItemHeight = 19;
            this.lbUsers.Location = new System.Drawing.Point(13, 31);
            this.lbUsers.Name = "lbUsers";
            this.lbUsers.Size = new System.Drawing.Size(241, 308);
            this.lbUsers.TabIndex = 0;
            this.lbUsers.SelectedIndexChanged += new
System.EventHandler(this.lbUsers_SelectedIndexChanged);
            // 
            // groupBox2
            // 
            this.groupBox2.BackColor = System.Drawing.Color.Transparent;
            this.groupBox2.Controls.Add(this.btnRemove);
            this.groupBox2.Controls.Add(this.btnUpdate);
            this.groupBox2.Controls.Add(this.btnAdd);
            this.groupBox2.Controls.Add(this.btnNew);
            this.groupBox2.Controls.Add(this.cbDomains);
            this.groupBox2.Controls.Add(this.tbPassConfirm);
            this.groupBox2.Controls.Add(this.tbPass);
            this.groupBox2.Controls.Add(this.tbUser);
            this.groupBox2.Controls.Add(this.label4);
            this.groupBox2.Controls.Add(this.label3);
            this.groupBox2.Controls.Add(this.label2);
            this.groupBox2.Controls.Add(this.label1);
            this.groupBox2.ForeColor = System.Drawing.Color.DeepSkyBlue;
            this.groupBox2.Location = new System.Drawing.Point(297, 12);
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(432, 252);
            this.groupBox2.TabIndex = 1;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "User Account Information:";
            // 
            // btnRemove
            // 
```

[illegible]

```
            this.btnRemove.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.btnRemove.Location = new System.Drawing.Point(309, 198);
            this.btnRemove.Name = "btnRemove";
            this.btnRemove.Size = new System.Drawing.Size(105, 32);
            this.btnRemove.TabIndex = 11;
            this.btnRemove.Text = "Remove";
            this.btnRemove.UseVisualStyleBackColor = true;
            this.btnRemove.Click += new
System.EventHandler(this.btnRemove_Click);
            //
            // btnUpdate
            //
            this.btnUpdate.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.btnUpdate.Location = new System.Drawing.Point(210, 198);
            this.btnUpdate.Name = "btnUpdate";
            this.btnUpdate.Size = new System.Drawing.Size(93, 32);
            this.btnUpdate.TabIndex = 10;
            this.btnUpdate.Text = "Update";
            this.btnUpdate.UseVisualStyleBackColor = true;
            this.btnUpdate.Click += new
System.EventHandler(this.btnUpdate_Click);
            //
            // btnAdd
            //
            this.btnAdd.ForeColor = System.Drawing.SystemColors.ControlText;
            this.btnAdd.Location = new System.Drawing.Point(120, 198);
            this.btnAdd.Name = "btnAdd";
            this.btnAdd.Size = new System.Drawing.Size(84, 32);
            this.btnAdd.TabIndex = 9;
            this.btnAdd.Text = "Add";
            this.btnAdd.UseVisualStyleBackColor = true;
            this.btnAdd.Click += new System.EventHandler(this.btnAdd_Click);
            //
            // btnNew
            //
            this.btnNew.ForeColor = System.Drawing.SystemColors.ControlText;
            this.btnNew.Location = new System.Drawing.Point(28, 198);
            this.btnNew.Name = "btnNew";
            this.btnNew.Size = new System.Drawing.Size(86, 32);
            this.btnNew.TabIndex = 8;
            this.btnNew.Text = "New";
            this.btnNew.UseVisualStyleBackColor = true;
            this.btnNew.Click += new System.EventHandler(this.btnNew_Click);
            //
            // cbDomains
            //
            this.cbDomains.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.cbDomains.FormattingEnabled = true;
            this.cbDomains.Location = new System.Drawing.Point(155, 65);
            this.cbDomains.Name = "cbDomains";
            this.cbDomains.Size = new System.Drawing.Size(258, 27);
            this.cbDomains.TabIndex = 7;
            //
            // tbPassConfirm
            //
            this.tbPassConfirm.Location = new System.Drawing.Point(189, 143);
            this.tbPassConfirm.MaxLength = 32;
            this.tbPassConfirm.Name = "tbPassConfirm";
            this.tbPassConfirm.PasswordChar = '*';
            this.tbPassConfirm.Size = new System.Drawing.Size(225, 26);
            this.tbPassConfirm.TabIndex = 6;
            //
            // tbPass
            //
            this.tbPass.Location = new System.Drawing.Point(189, 102);
            this.tbPass.MaxLength = 32;
            this.tbPass.Name = "tbPass";
            this.tbPass.PasswordChar = '*';
            this.tbPass.Size = new System.Drawing.Size(225, 26);
            this.tbPass.TabIndex = 5;
            //
            // tbUser
            //
            this.tbUser.Location = new System.Drawing.Point(129, 28);
            this.tbUser.MaxLength = 64;
            this.tbUser.Name = "tbUser";
            this.tbUser.Size = new System.Drawing.Size(285, 26);
            this.tbUser.TabIndex = 4;
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.ForeColor = System.Drawing.Color.DodgerBlue;
            this.label4.Location = new System.Drawing.Point(24, 146);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(153, 19);
            this.label4.TabIndex = 3;
            this.label4.Text = "Retype Password:";
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.ForeColor = System.Drawing.Color.DodgerBlue;
            this.label3.Location = new System.Drawing.Point(24, 109);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(90, 19);
            this.label3.TabIndex = 2;
            this.label3.Text = "Password:";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.ForeColor = System.Drawing.Color.DodgerBlue;
            this.label2.Location = new System.Drawing.Point(24, 67);
            this.label2.Name = "label2";
```

```
            this.label2.Size = new System.Drawing.Size(117, 19);
            this.label2.TabIndex = 1;
            this.label2.Text = "Domain Name:";
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.ForeColor = System.Drawing.Color.DodgerBlue;
            this.label1.Location = new System.Drawing.Point(24, 31);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(99, 19);
            this.label1.TabIndex = 0;
            this.label1.Text = "User Name:";
            //
            // btnDone
            //
            this.btnDone.Location = new System.Drawing.Point(332, 285);
            this.btnDone.Name = "btnDone";
            this.btnDone.Size = new System.Drawing.Size(377, 85);
            this.btnDone.TabIndex = 2;
            this.btnDone.Text = "Done";
            this.btnDone.UseVisualStyleBackColor = true;
            this.btnDone.Click += new
System.EventHandler(this.btnDone_Click);
            //
            // CfgUser
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 19F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackgroundImage =
global::Mail_Server.Properties.Resources.Untitled_2;
            this.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.ClientSize = new System.Drawing.Size(741, 392);
            this.Controls.Add(this.btnDone);
            this.Controls.Add(this.groupBox2);
            this.Controls.Add(this.groupBox1);
            this.DoubleBuffered = true;
            this.Font = new System.Drawing.Font("Consolas", 12F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.Margin = new System.Windows.Forms.Padding(4);
            this.MaximizeBox = false;
            this.Name = "CfgUser";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Configure E-Mail User Accounts";
            this.Load += new System.EventHandler(this.CfgUser_Load);
            this.groupBox1.ResumeLayout(false);
            this.groupBox2.ResumeLayout(false);
            this.groupBox2.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.ListBox lbUsers;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.Button btnRemove;
        private System.Windows.Forms.Button btnUpdate;
        private System.Windows.Forms.Button btnAdd;
        private System.Windows.Forms.Button btnNew;
        private System.Windows.Forms.ComboBox cbDomains;
        private System.Windows.Forms.TextBox tbPassConfirm;
        private System.Windows.Forms.TextBox tbPass;
        private System.Windows.Forms.TextBox tbUser;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button btnDone;
    }
}
```

Form Code: Solution > CfgUser.cs

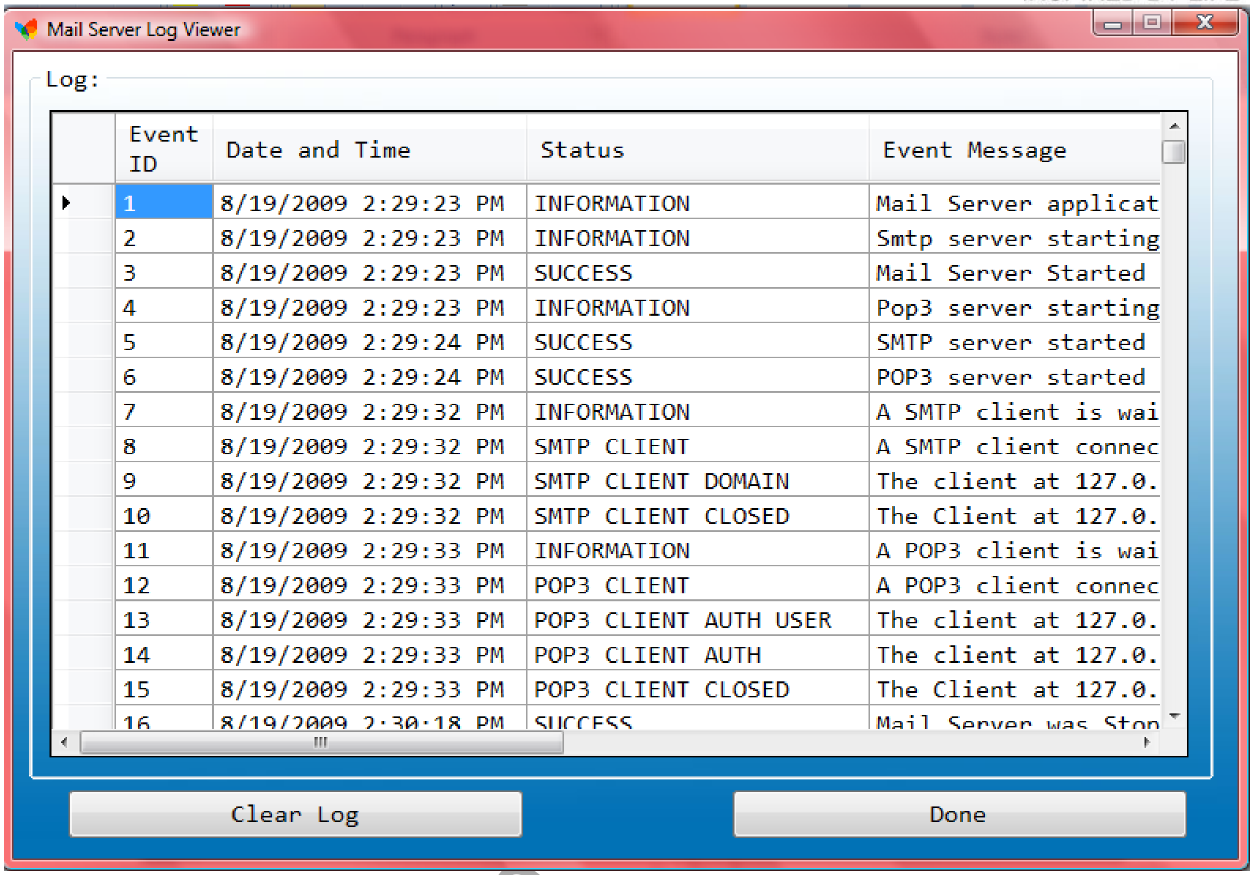
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Linq;
using System.IO;

namespace Mail_Server
{
    /*
     * This is the main class for User Account configuration dialog.
     */
    public partial class CfgUser : Form
    {
        /*
         * This is the default constructor for CfgUser class.
         */
        public CfgUser()
        {
            InitializeComponent();
        }

        /*
         * This function is invoked when the User Account Configuration
dialog is invoked.
         * Enumerate the Domain and its Users, and list the registered E-Mail
addresses in
         * the list box.
         */
        private void CfgUser_Load(object sender, EventArgs e)
        {
            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();
                XElement elDomain = cfg.Root.Element("Domains");
                if (elDomain != null)
                {
                    foreach (XElement el in elDomain.Nodes())
                    {
                        cbDomains.Items.Add(el.Name.LocalName);
                    }
                    if (cbDomains.Items.Count > 0)
                    {
                        cbDomains.SelectedIndex = 0;
                    }
                    else
                    {
                        // No E-Mail Domains specified
                        MessageBox.Show("The E-Mail server has no E-Mail
domains configured. Please configure domain names before adding an User
Account.",
                            "E-Mail Domain not Configured",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Stop);
                        this.Dispose();
                        return;
                    }
                }

                XElement elUsers = cfg.Root.Element("Users");
                if (elUsers != null)
                {
                    foreach (XElement elDomains in elUsers.Nodes())
                    {
                        foreach (XElement elUser in elDomains.Nodes())
                        {
                            lbUsers.Items.Add(elUser.Name.LocalName + "@" +
elDomains.Name.LocalName);
                        }
                    }
                    if (lbUsers.Items.Count > 0)
                    {
                        lbUsers.SelectedIndex = 0;
                        EnableReadOnly();
                    }
                    else
                    {
                        EnableWrite();
                    }
                }
                else
                {
                    // add the provision node for Users
                    cfg.Root.Add(new XElement("Users"));
                }
            }
            catch
            {
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }
        }

        /*
         * This function enables those dialog items, that can be enabled only
         * such that the email accounts can be updated and removed.
         */
        private void EnableReadOnly()
        {
            tbUser.ReadOnly = true;
            cbDomains.Enabled = false;
            btnAdd.Enabled = false;
            btnUpdate.Enabled = true;
            btnRemove.Enabled = true;
        }

        /*
         * This function enables those dialog items, that can be enabled only
         * when we need to add a new account or create a new account.
         */
        private void EnableWrite()
        {
            tbUser.ReadOnly = false;
            cbDomains.Enabled = true;
            btnAdd.Enabled = true;
            btnUpdate.Enabled = false;
            btnRemove.Enabled = false;
        }

        /*
```

```
         * This function is invoked when the "Done" button is clicked.
         * So we are done with account editing, and so we close this form.
         */
        private void btnDone_Click(object sender, EventArgs e)
        {
            this.Dispose();
        }

        /*
         * This function is invoked when we want to create a new E-Mail
account.
         */
        private void btnNew_Click(object sender, EventArgs e)
        {
            tbUser.Clear();
            if (cbDomains.Items.Count > 0)
            {
                cbDomains.SelectedIndex = 0;
            }
            tbPass.Clear();
            tbPassConfirm.Clear();
            EnableWrite();
        }

        /*
         * This function is invoked when we want to add the newly created E-
Mail
         * user account to the Mail Configuration file and also to the E-Mail
         * list box.
         */
        private void btnAdd_Click(object sender, EventArgs e)
        {
            tbUser.Text = tbUser.Text.Trim();
            tbPass.Text = tbPass.Text.Trim();

            if (tbUser.Text == "" || tbPass.Text == "")
            {
                MessageBox.Show("User Name or Password cannot be
blank.\nPlease enter a valid User Name and Password and try again.",
                    "Error in User Name or Password.",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
                return;
            }

            if(tbPass.Text != tbPassConfirm.Text)
            {
                MessageBox.Show("The entered passwords do not match. Please
enter the correct password and try again.",
                    "Wrong Password",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
                return;
            }
            /*
             * <Users>
             * ... ... ...
             *  <domain.xyz>
             *      <email>
             *          <pass>mypassword</pass>
             *          <folder>myemailfolder</folder>
             *      </email>
             *  <domain.xyz>
             * ... ... ...
             * </Users>
             */

            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();
                XElement el = cfg.Root.Element("Users");
                if (el == null)
                {
                    throw new Exception();
                }

                // Create the new folder Guid for email account
                string userdir = Guid.NewGuid().ToString().ToUpper();


                DirectoryInfo dirinfo = new
DirectoryInfo(Application.StartupPath);
                DirectoryInfo[] dirinfolist = dirinfo.GetDirectories();
                bool bFound;
                do
                {
                    bFound = true;
                    foreach (DirectoryInfo dinfo in dirinfolist)
                    {
                        if (dinfo.Name == userdir)
                        {
                            userdir = Guid.NewGuid().ToString().ToUpper();
                            bFound = false;
                            break;
                        }
                    }
                }
                while (bFound != true);

                // now create the unique user directory
                dirinfo = new DirectoryInfo(Application.StartupPath + @"\" +
userdir);
                dirinfo.Create();

                // now add this settings to configuration
                XElement elUser = new XElement(tbUser.Text);
                elUser.Add(new XElement("pass", tbPass.Text));
                elUser.Add(new XElement("folder", userdir));

                XElement elDomain = el.Element(cbDomains.Text);
                if (elDomain == null)
                {
                    elDomain = new XElement(cbDomains.Text);
                    elDomain.Add(elUser);
                    el.Add(elDomain);
                }
                else
                {
                    elDomain.Add(elUser);
                }

                lbUsers.SelectedIndex = lbUsers.Items.Add(tbUser.Text + "@" +
cbDomains.Text);

                MessageBox.Show("E-Mail Account was successfully created and
added to the list.",
                    "E-Mail Account created",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
                MessageBox.Show("E-Mail Account cannot be created at this
time. Please try again later.",
                    "E-Mail Account cannot be created",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }
        }

        /*
         * This function is invoked when the user selects a different user
account
         * to update or remove.
         */
        private void lbUsers_SelectedIndexChanged(object sender, EventArgs e)
        {
            EnableReadOnly();

            string usermail =
lbUsers.Items[lbUsers.SelectedIndex].ToString();
            string username = usermail.Substring(0, usermail.IndexOf("@"));
```

[illegible]

```
            string userdomain = usermail.Substring(usermail.IndexOf("@") +
1);

            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();
                XElement elDomain =
cfg.Root.Element("Users").Element(userdomain);
                if (elDomain == null)
                {
                    throw new Exception();
                }

                XElement elUser = elDomain.Element(username);
                if (elUser == null)
                {
                    throw new Exception();
                }

                tbUser.Text = elUser.Name.LocalName;
                cbDomains.SelectedIndex =
cbDomains.Items.IndexOf(elDomain.Name.LocalName);
                tbPass.Text = elUser.Element("pass").Value;
                tbPassConfirm.Text = tbPass.Text;
            }
            catch
            {
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }
        }

        /*
         * This function is invoked when the user clicks the "Update"
         * button and thereby, updates the E-Mail account settings in the
Mail
         * Configuration file.
         */
        private void btnUpdate_Click(object sender, EventArgs e)
        {
            // Update only updates the password
            tbPass.Text = tbPass.Text.Trim();
            if (tbPass.Text == "")
            {
                MessageBox.Show("Empty password string is not allowed. Please
enter a non-empty valid password and retry.",
                    "Invalid password",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
                return;
            }
            if (tbPass.Text != tbPassConfirm.Text)
            {
                MessageBox.Show("The entered password does not match. Please
enter the correct password before continuing.",
                    "Wrong password",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
                return;
            }

            CServerCfg cfg = null;
            try
            {
                cfg = new CServerCfg();
                XElement elDomain =
cfg.Root.Element("Users").Element(cbDomains.Items[cbDomains.SelectedIndex].To
String());
                XElement elUser = elDomain.Element(tbUser.Text);
                elUser.Element("pass").SetValue(tbPass.Text);

                MessageBox.Show("The E-Mail account was successfully
updated.",
                    "E-Mail Account update success",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
                MessageBox.Show("The E-Mail account could not be updated at
this time. Please try again later.",
                    "E-Mail Account Update Failed",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }

        }

        /*
         * This function is called when the user clicks the "Remove" button
         * which will remove the E-Mail account settings from the Mail
configuration
         * file and also removes the user's mail inbox and all the mail
content in it.
         */
        private void btnRemove_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("You are about to delete the E-Mail account.
This will delete the E-Mail Inbox and the user will loose all the mail
content in it.\nAre you sure with the delete option ?",
                                "Confirm Delete ?",
                                MessageBoxButtons.OKCancel,
                                MessageBoxIcon.Question) ==
DialogResult.Cancel)
                {
                    return;
                }

            CServerCfg cfg = null;
            try
            {

                cfg = new CServerCfg();

                XElement elDomain =
cfg.Root.Element("Users").Element(cbDomains.Items[cbDomains.SelectedIndex].To
String());
                XElement elUser = elDomain.Element(tbUser.Text);
                DirectoryInfo dinfo = new
DirectoryInfo(Application.StartupPath + @"\" +
elUser.Element("folder").Value);
                if (dinfo.Exists)
                {
                    dinfo.Delete(true);
                }

                elUser.Remove();
                cbDomains.Items.RemoveAt(cbDomains.SelectedIndex);
                if (cbDomains.Items.Count > 0)
                {
                    cbDomains.SelectedIndex = 0;
                }

                MessageBox.Show("The E-Mail account was successfully
removed.",
                    "E-Mail Account deleted",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
                MessageBox.Show("The E-Mail account cannot be removed at this
time. Please try again later.",
                    "E-Mail cannot be deleted",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);
            }
            finally
            {
                if (cfg != null)
                {
                    cfg.Close();
                }
            }
        }
    }
}
```

[illegible]



	Event ID	Date and Time	Status	Event Message
▸	1	8/19/2009 2:29:23 PM	INFORMATION	Mail Server applicat
	2	8/19/2009 2:29:23 PM	INFORMATION	Smtp server starting
	3	8/19/2009 2:29:23 PM	SUCCESS	Mail Server Started
	4	8/19/2009 2:29:23 PM	INFORMATION	Pop3 server starting
	5	8/19/2009 2:29:24 PM	SUCCESS	SMTP server started
	6	8/19/2009 2:29:24 PM	SUCCESS	POP3 server started
	7	8/19/2009 2:29:32 PM	INFORMATION	A SMTP client is wai
	8	8/19/2009 2:29:32 PM	SMTP CLIENT	A SMTP client connec
	9	8/19/2009 2:29:32 PM	SMTP CLIENT DOMAIN	The client at 127.0.
	10	8/19/2009 2:29:32 PM	SMTP CLIENT CLOSED	The Client at 127.0.
	11	8/19/2009 2:29:33 PM	INFORMATION	A POP3 client is wai
	12	8/19/2009 2:29:33 PM	POP3 CLIENT	A POP3 client connec
	13	8/19/2009 2:29:33 PM	POP3 CLIENT AUTH USER	The client at 127.0.
	14	8/19/2009 2:29:33 PM	POP3 CLIENT AUTH	The client at 127.0.
	15	8/19/2009 2:29:33 PM	POP3 CLIENT CLOSED	The Client at 127.0.
	16	8/19/2009 2:30:18 PM	SUCCESS	Mail Server was Stop

This is the Mail Server Log file. The file is a XML based log file that contains some DEBUG information that can be used to trace back some errors.

The “Clear Log” button will clear the Log file completely by removing all the entries.

The “Done” button will close the Log Viewer.

Form Code: Solution > ServerLogViewer.cs > ServerLogViewer.Designer.cs

```
namespace Mail_Server
{
    partial class ServerLogViewer
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(ServerLogViewer));
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.dgLog = new System.Windows.Forms.DataGridView();
            this.EventID = new
System.Windows.Forms.DataGridViewTextBoxColumn();
            this.DateAndTime = new
System.Windows.Forms.DataGridViewTextBoxColumn();
            this.Status = new
System.Windows.Forms.DataGridViewTextBoxColumn();
            this.EventMessage = new
System.Windows.Forms.DataGridViewTextBoxColumn();
            this.btnClear = new System.Windows.Forms.Button();
            this.btnDone = new System.Windows.Forms.Button();
            this.groupBox1.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.dgLog)).BeginInit();
            this.SuspendLayout();
            //
            // groupBox1
            //
            this.groupBox1.BackColor = System.Drawing.Color.Transparent;
            this.groupBox1.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.groupBox1.Controls.Add(this.dgLog);
            this.groupBox1.Location = new System.Drawing.Point(10, 7);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(751, 454);
            this.groupBox1.TabIndex = 0;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Log:";
            //
            // dgLog
            //
            this.dgLog.AllowUserToAddRows = false;
            this.dgLog.AllowUserToDeleteRows = false;
            this.dgLog.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
            this.dgLog.Columns.AddRange(new
System.Windows.Forms.DataGridViewColumn[] {
            this.EventID,
            this.DateAndTime,
            this.Status,
            this.EventMessage});
            this.dgLog.Location = new System.Drawing.Point(13, 30);
            this.dgLog.Name = "dgLog";
            this.dgLog.ReadOnly = true;
            this.dgLog.RowHeadersWidthSizeMode =
System.Windows.Forms.DataGridViewRowHeadersWidthSizeMode.DisableResizing;
            this.dgLog.ShowEditingIcon = false;
            this.dgLog.Size = new System.Drawing.Size(722, 410);
            this.dgLog.TabIndex = 0;
            //
            // EventID
            //
            this.EventID.AutoSizeMode =
System.Windows.Forms.DataGridViewAutoSizeColumnMode.AllCells;
            this.EventID.HeaderText = "Event ID";
            this.EventID.Name = "EventID";
            this.EventID.ReadOnly = true;
            this.EventID.SortMode =
System.Windows.Forms.DataGridViewColumnSortMode.NotSortable;
            this.EventID.Width = 87;
            //
            // DateAndTime
            //
            this.DateAndTime.AutoSizeMode =
System.Windows.Forms.DataGridViewAutoSizeColumnMode.AllCells;
            this.DateAndTime.HeaderText = "Date and Time";
            this.DateAndTime.Name = "DateAndTime";
            this.DateAndTime.ReadOnly = true;
            this.DateAndTime.SortMode =
System.Windows.Forms.DataGridViewColumnSortMode.NotSortable;
            this.DateAndTime.Width = 87;
            //
            // Status
            //
            this.Status.AutoSizeMode =
System.Windows.Forms.DataGridViewAutoSizeColumnMode.AllCells;
            this.Status.HeaderText = "Status";
            this.Status.Name = "Status";
            this.Status.ReadOnly = true;
            this.Status.SortMode =
System.Windows.Forms.DataGridViewColumnSortMode.NotSortable;
            this.Status.Width = 69;
            //
            // EventMessage
            //
            this.EventMessage.AutoSizeMode =
System.Windows.Forms.DataGridViewAutoSizeColumnMode.AllCells;
            this.EventMessage.HeaderText = "Event Message";
            this.EventMessage.Name = "EventMessage";
            this.EventMessage.ReadOnly = true;
            this.EventMessage.SortMode =
System.Windows.Forms.DataGridViewColumnSortMode.NotSortable;
            this.EventMessage.Width = 119;
            //
            // btnClear
            //
            this.btnClear.Location = new System.Drawing.Point(34, 467);
            this.btnClear.Name = "btnClear";
            this.btnClear.Size = new System.Drawing.Size(290, 32);
            this.btnClear.TabIndex = 1;
            this.btnClear.Text = "Clear Log";
            this.btnClear.UseVisualStyleBackColor = true;
            this.btnClear.Click += new
System.EventHandler(this.btnClear_Click);
            //
            // btnDone
            //
            this.btnDone.Location = new System.Drawing.Point(455, 467);
            this.btnDone.Name = "btnDone";
            this.btnDone.Size = new System.Drawing.Size(290, 32);
            this.btnDone.TabIndex = 2;
            this.btnDone.Text = "Done";
            this.btnDone.UseVisualStyleBackColor = true;
            this.btnDone.Click += new
System.EventHandler(this.btnDone_Click);
            //
            // ServerLogViewer
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 19F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackgroundImage =
global::Mail_Server.Properties.Resources.Untitled_2;
            this.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
            this.ClientSize = new System.Drawing.Size(776, 511);
            this.Controls.Add(this.btnDone);
            this.Controls.Add(this.btnClear);
            this.Controls.Add(this.groupBox1);
            this.DoubleBuffered = true;
            this.Font = new System.Drawing.Font("Consolas", 12F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
```

```
            this.Margin = new System.Windows.Forms.Padding(4, 4, 4, 4);
            this.MaximizeBox = false;
            this.Name = "ServerLogViewer";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Mail Server Log Viewer";
            this.Load += new System.EventHandler(this.ServerLogViewer_Load);
            this.groupBox1.ResumeLayout(false);

((System.ComponentModel.ISupportInitialize)(this.dgLog)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.DataGridView dgLog;
        private System.Windows.Forms.DataGridViewTextBoxColumn EventID;
        private System.Windows.Forms.DataGridViewTextBoxColumn DateAndTime;
        private System.Windows.Forms.DataGridViewTextBoxColumn Status;
        private System.Windows.Forms.DataGridViewTextBoxColumn EventMessage;
        private System.Windows.Forms.Button btnClear;
        private System.Windows.Forms.Button btnDone;
    }
}
```

Form Code: Solution > ServerLogViewer.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Mail_Server
{
    /*
     * This is the main class for Server Log Viewer dialog.
     */
    public partial class ServerLogViewer : Form
    {
        /*
         * Default constructor for the Server Log Viewer class
         */
        public ServerLogViewer()
        {
            InitializeComponent();
        }

        /*
         * This function is invoked when the form loads.
         * We would, in here, just refresh the log XML and reload it
         * and show it to user in a Data grid view.
         */
        private void ServerLogViewer_Load(object sender, EventArgs e)
        {
            ShowLog();
        }

        /*
         * This function will be invoked when the user clicks on the "Done"
button.
         * Here we will just close this dialog as we are done with it.
         */
        private void btnDone_Click(object sender, EventArgs e)
        {
            this.Dispose();
        }

        /*
         * This function will be called when the "Clear" button is clicked.
         * This will cause the XML log to clear and delete of all its
entries.
         * Then reload and refresh the data grid view.
         */
        private void btnClear_Click(object sender, EventArgs e)
        {
            try
            {
                CServerLog log = new CServerLog();
                log.ClearLog();
                MessageBox.Show("The Mail Server Log was successfully
cleared.",
                    "Mail Server Log Cleared",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
            catch
            {
                MessageBox.Show("The Mail Server log cannot be cleared now.
Please try again later.",
                    "Mail Server Log unable to Clear",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
            finally
            {
                ShowLog();
            }
        }
        /*
         * This function basically, loads the XML log file and
         * displays all the loag entry in a data grid view.
         */
        private void ShowLog()
        {
            try
            {
                // clear the data grid view
                dgLog.Rows.Clear();

                // load the server log
                CServerLog log = new CServerLog();
                int id = 0;

                // Enumerate all the log entries and display it in the data
grid view.
                foreach (ServerLogEntry se in log)
                {
                    id ++;
                    string[] strValues = new string[] {
                                                                           id.ToString(),

se.eventTime.ToString(),

se.eventStatus.ToString(),

se.eventMessage.ToString()
                                                                      };
                    dgLog.Rows.Add(strValues);
                }
            }
            catch
            {
                // Some error may happen... but we don't care
            }
        }
    }
}
```

Code: Solution > CServerCfg.cs

This file contains a helper utility API to edit, retrieve or use the Server configuration file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Linq;
using System.IO;

namespace Mail_Server
{
    /*
     * This is a utility API for loading, saving or updating the Mail
configuration.
     */
    class CServerCfg
    {
        /*
         * This private variable stores the path to the mail configuration
file.
         */
        private static string cfgFileName = Application.StartupPath +
@"\mailserver.config";

        /*
         * This static private variable will store the reference to the XML
file
         * that refers to the mail configuration file.
         * To optimize XML performance the XML file is loaded only once.
         */
        private static XDocument xdoc = null;

        /*
         * This is the default constructor for the CServerCfg class.
         * Here, load the XML configuration file, if it is loaded then do
nothing.
         * If the file is not loaded then, create the file and parse it.
         */
        public CServerCfg()
        {
            if (xdoc == null)
```

[illegible]

```
            {
                try
                {
                    LoadDoc();
                }
                catch
                {
                    try
                    {
                        CreateNewDoc();
                    }
                    catch
                    {
                    }
                }
            }
        }

        /*
         * Load the XML file from the disk.
         * But the file must exist.
         */
        private void LoadDoc()
        {
            xdoc = XDocument.Load(cfgFileName);
        }

        /*
         * Parse the XML file if the Load of Document failed.
         */
        private void ParseDoc()
        {
            xdoc = XDocument.Parse(cfgFileName);
        }

        /*
         * Create the new file and load it.
         */
        private void CreateNewDoc()
        {
            FileStream fs = File.Create(cfgFileName);

            byte[] bufData = System.Text.Encoding.ASCII.GetBytes("<?xml
version=\"1.0\" ?>\n");
            fs.Write(bufData, 0, bufData.Length);

            bufData = System.Text.Encoding.ASCII.GetBytes(@"<MailConfig>
</MailConfig>");
            fs.Write(bufData, 0, bufData.Length);

            fs.Close();

            try
            {
```

[illegible]

```
                LoadDoc();
            }
            catch
            {
                ParseDoc();
            }
        }

        /*
         * When Close (if called), the XML data is saved to file on disk and
updated.
         */
        public void Close()
        {
            try
            {
                xdoc.Save(cfgFileName);
            }
            catch
            {
            }
        }

        /*
         * A public read-only property that returns the reference to the XML
document
         * interface.
         */
        public XDocument Document
        {
            get
            {
                return xdoc;
            }
        }

        /*
         * A public read-only property that returns the reference to the
XElement
         * interface of the root element in the XML file.
         */
        public XElement Root
        {
            get
            {
                return xdoc.Element("MailConfig");
            }
        }
    }
}
```

Code: Solution > CServerLog.cs

This file contains a helper utility API to add or read the Server Log file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.Linq;
using System.IO;
using System.Windows.Forms;
using System.Collections;

namespace Mail_Server
{
    /*
     * This is a structure, that contains the representation for each event
in the XML
     * based Log file.
     */
    struct ServerLogEntry
    {
        public DateTime eventTime;
        public string eventStatus;
        public string eventMessage;
    }

    /*
     * The CServerLog class is the abstract representation for the Mail
Server log file.
     */
    class CServerLog : IDisposable, IEnumerable
    {
        /*
         * private static variable that holds the reference to the XML
document of
         * the server log.
         */
        private static XDocument doc = null;

        /*
         * private static variable that represents the path to the Mail
Server log file.
         */
```

[illegible]

```
        private static string logPath = Application.StartupPath +
@"\mailserver.log";

        /*
         * The default constructor of CServerLog file, which is responsible
to
         * open / create the server log file once, for reading (as
optimization).
         */
        public CServerLog()
        {
            if (doc == null)
            {
                OpenDoc();
            }
        }

        /*
         * This private function creates / opens the server log file.
         */
        private void OpenDoc()
        {
            try
            {
                FileInfo finfo = new FileInfo(logPath);
                if (finfo.Exists)
                {
                    OpenXml();
                }
                else
                {
                    FileStream fs = finfo.Create();
                    StreamWriter sout = new StreamWriter(fs);
                    sout.WriteLine("<?xml version=\"1.0\" encoding=\"utf-8\"
?>");

                    sout.WriteLine("<log>");
                    sout.WriteLine("</log>");
                    sout.Flush();
                    fs.Close();

                    OpenXml();
                }
            }
            catch
            {
            }
        }

        /*
         * This private function loads or parse the XML document.
         */
        private void OpenXml()
        {
            try
```

[illegible]

```
                {
                    doc = XDocument.Load(logPath);
                    if (doc == null)
                        throw new Exception();
                }
                catch
                {
                    try
                    {
                        doc = XDocument.Parse(logPath);
                    }
                    catch
                    {
                    }
                }
            }

            /*
             * This is the Dispose() method, which is called by the garbage
collector when
             * there is no reference to the object.
             */
            void IDisposable.Dispose()
            {
                Close();
            }

            /*
             * This function if called, whence the XML file is updated with new
Log
             * data.
             */
            public void Close()
            {
                if (doc == null)
                {
                    return;
                }

                try
                {
                    doc.Save(logPath);
                }
                catch
                {
                }
            }

            /*
             * This function is called to add a Log entry to the XML Mail server
log file.
             */
            public void WriteLog(string severityString,
                                 string eventString)
```

```
        {
            if (doc == null)
            {
                return;
            }

            try
            {
                XElement el = new XElement("message");
                el.Add(new XElement("date", DateTime.Now.ToString()));
                el.Add(new XElement("status", severityString));
                el.Add(new XElement("event", eventString));
                doc.Element("log").Add(el);
            }
            catch
            {
            }
        }

        /*
         * This function will be used by foreach to enumerate the log
entries.
         */
        IEnumerator IEnumerable.GetEnumerator()
        {
            if (doc == null)
            {
                yield break;
            }

            XElement elRoot = doc.Element("log");
            foreach (XElement el in elRoot.Nodes())
             {
                  ServerLogEntry se;
                  se.eventMessage = el.Element("event").Value;
                  se.eventStatus = el.Element("status").Value;
                  se.eventTime = DateTime.Parse(el.Element("date").Value);
                  yield return se;
             }
        }

        /*
         * This function is called to clear all the log entries in the XML
log
         * file.
         */
        public void ClearLog()
        {
            if (doc == null)
            {
                return;
            }

            try
            {
                Close();

                XElement elRoot = doc.Element("log");
                elRoot.RemoveAll();
            }
            catch
            {
            }
            finally
            {
                Close();
            }
        }

        /*
         * This static function is called to add a log entry without creating
a instance
         * of this class.
         * The function takes in the log entry and automatically saves it to
the
         * Log XML and updates it.
         */
        public static void LogEntry(string severityString,
                                    string eventString)
        {
            CServerLog log = null;
            try
            {
                log = new CServerLog();
                log.WriteLog(severityString, eventString);
            }
            catch
            {
            }
            finally
            {
                if (log != null)
                {
                    log.Close();
                }
            }
        }
    }
}
```

Code: Solution > Program.cs

This is the application startup file.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Mail_Server
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MailServer());
        }
    }
}
```

Code: Solution > Properties > AssemblyInfo.cs

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Mail Server")]
[assembly: AssemblyDescription("Mail Server for Small Office and Home based
POP3/SMTP client.")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Arnav Mukhopadhyay (Roll Number - 510728180) BCA
SMU")]
[assembly: AssemblyProduct("Mail Server")]
[assembly: AssemblyCopyright("Copyright © Arnav Mukhopadhyay 2009")]
[assembly: AssemblyTrademark("Arnav Mukhopadhyay (Roll Number - 510728180)
BCA SMU")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]
```

```
// The following GUID is for the ID of the typelib if this project is exposed
to COM
[assembly: Guid("5d04295e-19ea-499b-be88-97858564dc5f")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Code: Solution > Properties > Settings.settings > Settings.Designer.cs

```
//-----------------------------------------------------------------------------
---
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1434
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//-----------------------------------------------------------------------------
---

namespace Mail_Server.Properties
{


    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStud
io.Editors.SettingsDesigner.SettingsSingleFileGenerator", "9.0.0.0")]
    internal sealed partial class Settings :
global::System.Configuration.ApplicationSettingsBase
    {

        private static Settings defaultInstance =
((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized
(new Settings())));

        public static Settings Default
        {
            get
            {
```

[illegible]

```
                return defaultInstance;
            }
        }
    }
}
```

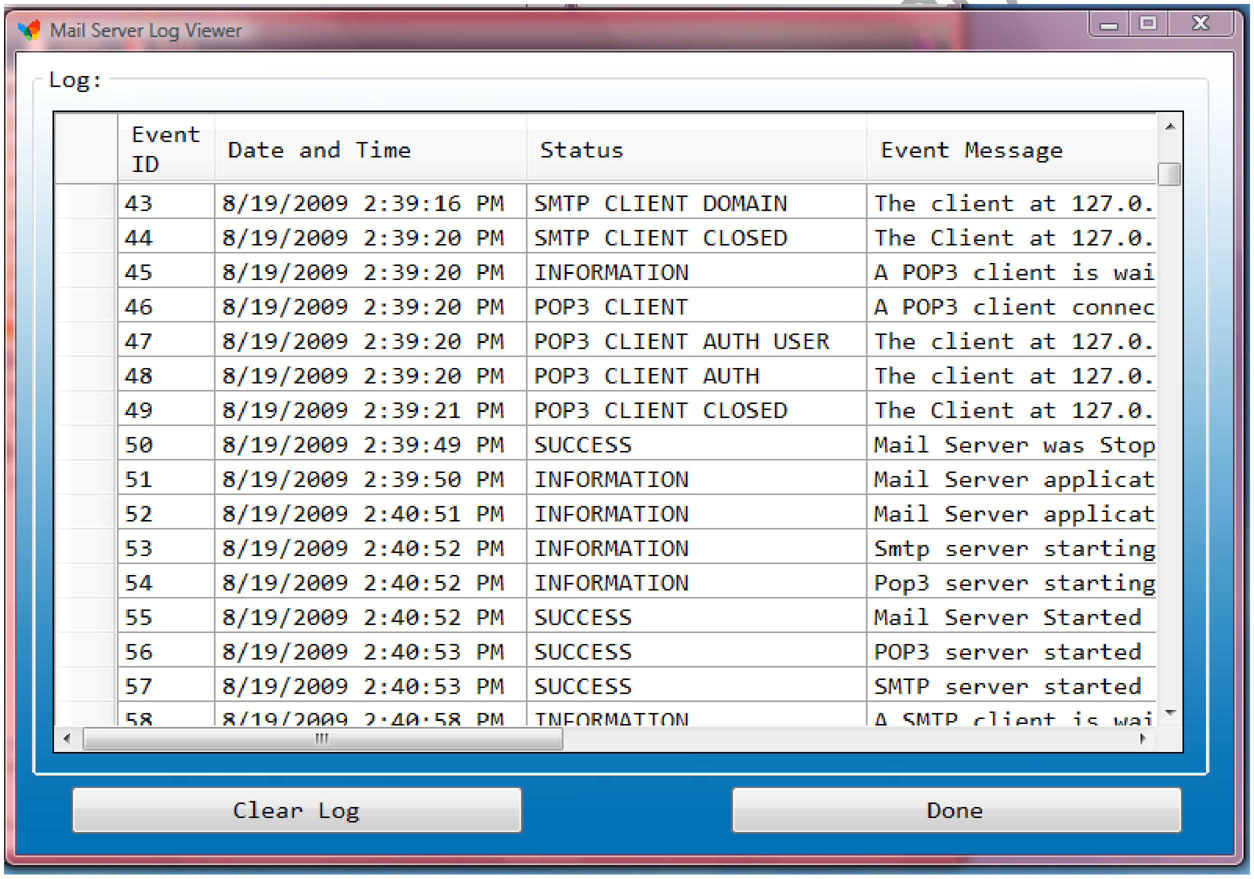
Code: Solution > Properties > Resources.resx > Resources.Designer.cs

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1434
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Mail_Server.Properties {
    using System;


    /// <summary>
    ///   A strongly-typed resource class, for looking up localized strings, etc.
    /// </summary>
    // This class was auto-generated by the StronglyTypedResourceBuilder
    // class via a tool like ResGen or Visual Studio.
    // To add or remove a member, edit your .ResX file then rerun ResGen
    // with the /str option, or rebuild your VS project.
    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedResourceBuilder", "2.0.0.0")]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal class Resources {

        private static global::System.Resources.ResourceManager resourceMan;

        private static global::System.Globalization.CultureInfo resourceCulture;


        [global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1811:AvoidUncalledPrivateCode")]
        internal Resources() {
        }

        /// <summary>
        ///   Returns the cached ResourceManager instance used by this class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.Compon
entModel.EditorBrowsableState.Advanced)]
        internal static global::System.Resources.ResourceManager
ResourceManager {
            get {
                if (object.ReferenceEquals(resourceMan, null)) {
                    global::System.Resources.ResourceManager temp = new
global::System.Resources.ResourceManager("Mail_Server.Properties.Resources",
typeof(Resources).Assembly);
                    resourceMan = temp;
                }
                return resourceMan;
            }
        }

        /// <summary>
        ///   Overrides the current thread's CurrentUICulture property for
all
        ///   resource lookups using this strongly typed resource class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.Compon
entModel.EditorBrowsableState.Advanced)]
        internal static global::System.Globalization.CultureInfo Culture {
            get {
                return resourceCulture;
            }
            set {
                resourceCulture = value;
            }
        }

        internal static System.Drawing.Bitmap Untitled_2 {
            get {
                object obj = ResourceManager.GetObject("Untitled-2",
resourceCulture);
                return ((System.Drawing.Bitmap)(obj));
            }
        }
    }
}
```

[illegible]

REPORTS

The software demonstrated here is a server application and thus, has no printable output or hard copy of reports. The only available information that can be considered as the report is that available online using the Log Viewer. A sample screenshot is shown below:



Event ID	Date and Time	Status	Event Message
43	8/19/2009 2:39:16 PM	SMTP CLIENT DOMAIN	The client at 127.0.
44	8/19/2009 2:39:20 PM	SMTP CLIENT CLOSED	The Client at 127.0.
45	8/19/2009 2:39:20 PM	INFORMATION	A POP3 client is wai
46	8/19/2009 2:39:20 PM	POP3 CLIENT	A POP3 client connec
47	8/19/2009 2:39:20 PM	POP3 CLIENT AUTH USER	The client at 127.0.
48	8/19/2009 2:39:20 PM	POP3 CLIENT AUTH	The client at 127.0.
49	8/19/2009 2:39:21 PM	POP3 CLIENT CLOSED	The Client at 127.0.
50	8/19/2009 2:39:49 PM	SUCCESS	Mail Server was Stop
51	8/19/2009 2:39:50 PM	INFORMATION	Mail Server applicat
52	8/19/2009 2:40:51 PM	INFORMATION	Mail Server applicat
53	8/19/2009 2:40:52 PM	INFORMATION	Smtp server starting
54	8/19/2009 2:40:52 PM	INFORMATION	Pop3 server starting
55	8/19/2009 2:40:52 PM	SUCCESS	Mail Server Started
56	8/19/2009 2:40:53 PM	SUCCESS	POP3 server started
57	8/19/2009 2:40:53 PM	SUCCESS	SMTP server started
58	8/19/2009 2:40:58 PM	INFORMATION	A SMTP client is wai

It shows the Event Identification number, Date and Time of Event logging, Status information, and the Event Message details. The Log file is in itself a XML file, used for simplifying the coding mechanism and providing use of ease and also the LINQ on XML provided by the new .NET framework version 3.5.

[illegible]

TESTING PLAN

This document describes the user acceptance plan for the Mail Server application. The overall test strategy is composed of the following tests in a sequence:

1. **Component testing:** This will test each components of the Mail Server.
2. **Integration testing:** This will ensure that the dependent components can communicate and integrate with each other, so that the whole system works correctly and reliably.
3. **Validation testing:** This will ensure that the system will work correctly in a pseudo-live environment.
4. **User Acceptance Testing:** This will ensure that the functionalities provided are acceptable to the user. This will be the final step performed in testing before the application is officially published for public use.

Test Scope

The scope of the user acceptance testing covers:

- Version 1 of the Mail Server application.
- User-interfacing functionalities defined by set of use cases.
- Server accessibility and compatibilities defined by the project scope.

The aim of this test is to determine how closely the application fits the user specification and client requirements, and to identify and resolve the issues when and where they arise. Also, the testing serves to compile a data set and corresponding results that can be used during subsequent test cycle, to test for non-regression of the software in later releases or after the application is in maintenance. The working practices are user dependable and are therefore, considered outside the scope of the testing.

Amar Mukhopadhyay

The sequence of acceptance testing was completed successfully, so that the application and its required infrastructure are considered stable and reliable. Acceptance testing take for granted the user’s perspective that is, how the application is used and whether it meets the quality criteria.

The change requests will be sent to the development team as actionable documentation. Change criteria will be determined by the test team and development team prior to the beginning of the testing. For instance, criteria may include *impact* to *desired functionality, amount of code impacted by proposed change, and design required by proposed change*. The tester will evaluate the criteria. The test lead will determine the requirements for the changes. Once a bug is detected as Change required, the bug report will be translated into a Change request and passed on to the requirement.

Preconditions

The following items are required before testing can take place:

- A complete and coherent functional specification of the Mail Server application expressed as use cases and usage scenario.
- A complete and validation-tested release of the Mail Server application, delivered according to the delivery plan.
- An agreed-upon procedure for dealing with any anomalies that are discovered during the testing process.
- A set of test specifications describing how each functional area is to be acceptance tested.
- An implemented test environment for the testing.
- Sufficient, suitable resources to carry out testing.
- Available standards for the acceptance testing.

Test Priorities

During testing, the following quantities will be tested in the order of priority:

- Functionality – whether the required functions are available and working as expected.

- Usability – how user-friendly, compatible and intuitive the system is.
- Security – how well-protected and guaranteed application and user data is.
- Performance – whether response times are within acceptable limits.
- Customization – how straight-forward it is to use the application in new and unpredicted way.

Test Techniques

The following tests are performed:

- Scripted tests – these tests are performed based on usage, compatibilities, based on predicted data set to obtain predicted outputs. Any deviation will be recorded and marked at tolerance or failure.
- Un-scripted test – Based on scripted tests, the scenario being modified to what-if possibilities.
- Compatibility test – This will modify the scenario based on scripted tests, but will only test the server application for compatibility and issues regarding different client applications.
- Usability test – This will test the application based on the complexity of interaction.
- Performance statistics – generation of performance information to check against desired performance criteria.

Test Organization

Roles and Responsibilities

The following roles are defined:

- Tester – carries out test according to test plans, and reports the result.
- Product Manager – ensures that the tests are carried out successfully from a user perspective.

- Test support – provides technical assistance, such as test environment configuration, and non-technical assistance, such as methodological support.

Team meetings are organized quickly, which involves test managers, testers, and product managers. At these meetings, the progress of the test process will be reported, any issues will be discussed, and actions will be agreed upon.

[illegible]

Manipal
INSPIRED BY LIFE

COST ESTIMATION

The software is designed using Microsoft Visual Studio 2008, requiring Microsoft Visual C# 2008, thereby using Microsoft .NET platform 2008. To minimize further cost, the server uses text mode authentication, flat file and XML database.

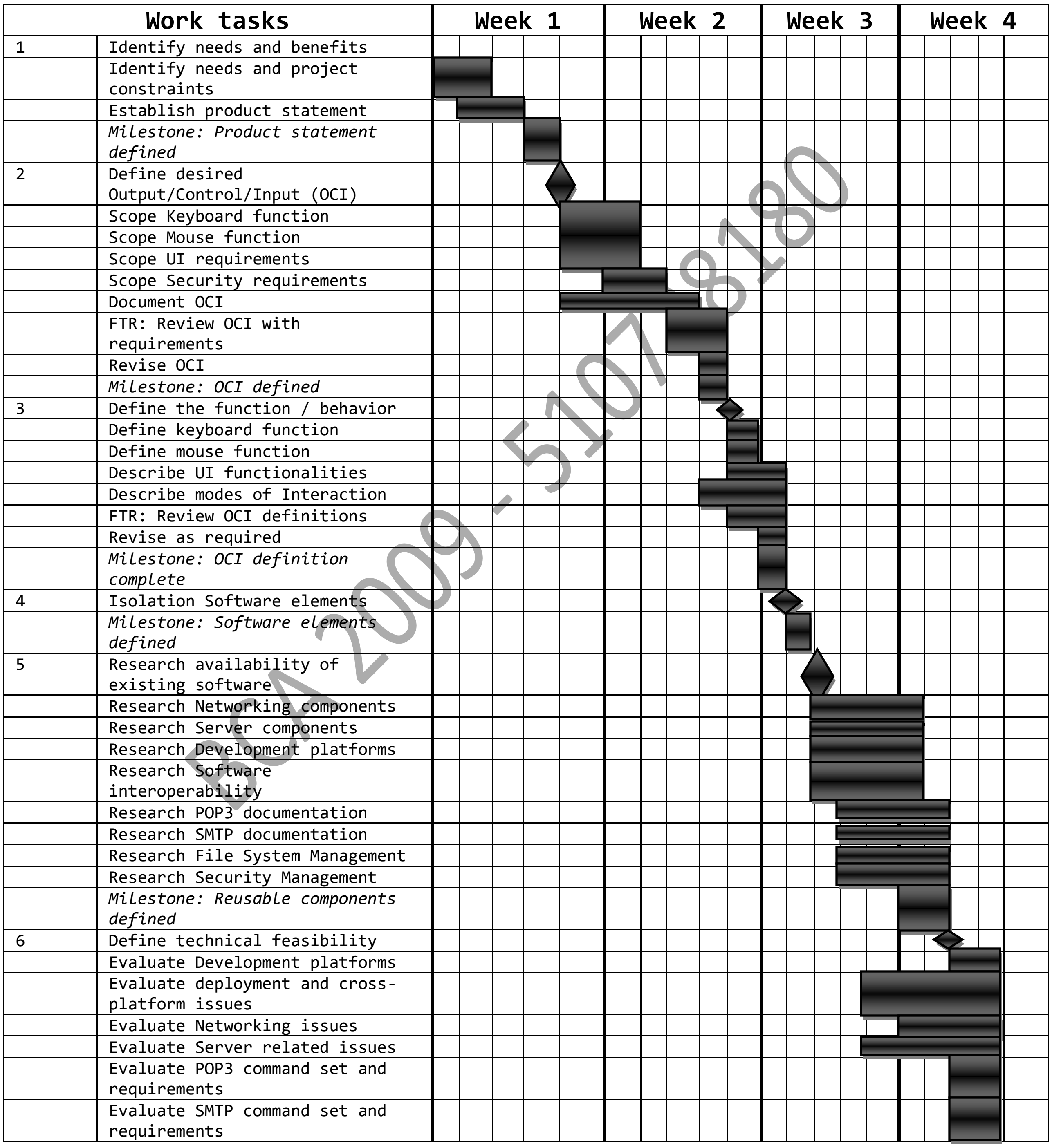
Cost of Deployment:

1.	**HARDWARE COST:**	**Rs. 15,000 /-**
2.	**SOFTWARE COST:***18hrs@Rs.10/-*	**Rs. 180 /-**
		Rs. 15,180 /-

[illegible]

TIME CHART

GANTT CHART (Timeline Charts)



Work tasks	
1	Identify needs and benefits
	Identify needs and project constraints
	Establish product statement
	Milestone: Product statement defined
2	Define desired Output/Control/Input (OCI)
	Scope Keyboard function
	Scope Mouse function
	Scope UI requirements
	Scope Security requirements
	Document OCI
	FTR: Review OCI with requirements
	Revise OCI
	Milestone: OCI defined
3	Define the function / behavior
	Define keyboard function
	Define mouse function
	Describe UI functionalities
	Describe modes of Interaction
	FTR: Review OCI definitions
	Revise as required
	Milestone: OCI definition complete
4	Isolation Software elements
	Milestone: Software elements defined
5	Research availability of existing software
	Research Networking components
	Research Server components
	Research Development platforms
	Research Software interoperability
	Research POP3 documentation
	Research SMTP documentation
	Research File System Management
	Research Security Management
	Milestone: Reusable components defined
6	Define technical feasibility
	Evaluate Development platforms
	Evaluate deployment and cross-platform issues
	Evaluate Networking issues
	Evaluate Server related issues
	Evaluate POP3 command set and requirements
	Evaluate SMTP command set and requirements

[illegible]

	Milestone: Technical feasibility assessed																		■	■	
7	Make quick estimate of size																			■	
8	Create a Scope definition																			■	■
	Review scope documents with Project requirements																			■	■
	Revise documents as required																				■
	Milestone: Scope document complete																				■

FUTURE SCOPE

This project is aimed at creating at creating a mail server at minimum cost of purchase and deployment, and is specifically designed for small office or home. But it is extendible to include more authenticated methods and server oriented applications. It can further be extended to include secure socket layer over TCP for POP and SMTP data interchange that would secure the communication. This would however require a certificate with a private and public key for both client to encode and server to decode the ciphered message.

Further, a new authentication protocol for POP3 introduced is APOP (or authenticated POP), can be included in the coding. This authentication is based on challenge string generated using MD5 hashing algorithm. This further secures the mail interchange.

SMTP again can be extended further to include ESMTP (Extended Simple Mail Transfer Protocol). This would enable mailing software, in both Server and Client mode, to provide more special commands. Some of these commands are UIDL, VRFY, HELP, EHLO, etc.

But these extensions require more aggressive programming and more advanced terminologies, which can, at this moment, be considered beyond the scope of this documentation but is available in some commercial software.

CONCLUSION

Thus, we conclude this documentation with the following concise points that can be considered as the positive impact of the project:

1. It is a low cost product.
2. It is capable of handling multiple clients at one time. The number of clients serviced at once depends on the maximum number of threads per process, available memory size, network speed and TCP connection slot.
3. It is mainly targeted for small-office and home based servers.
4. The server (term) can be any computer system, ranging from laptop to desktop, from low end architecture to high end mainframe depending on the client requirement.
5. The limitation for deployment of the server is the minimum requirement of the Microsoft .NET 3.5 platform or alternative platforms, if used.
6. The software implements the minimum and mandatory command sets of POP3 and SMTP protocol, with security; thus, providing a professional yet simple to use mail server.
7. It is developed on Microsoft .NET 3.5 platform and thus, provides a cross-platform support.
8. It supports almost any Mail interchange applications for mail interchange.
9. But due to security limitations, this application do not provide for mail relay services to interchange message with other mail servers.

[illegible]

BIBLOGRAPHY

1. RFC Documentation: **RFC 1734 (POP3 Authentication)**
    a. http://james.apache.org/server/rfclist/smtp/rfc0821.txt
2. RFC Documentation: **RFC 1939 (POP3 Protocol)**
    a. http://www.faqs.org/rfcs/rfc1939.html
3. RFC Documentation: **RFC 0821 (SMTP Protocol)**
    a. http://james.apache.org/server/rfclist/smtp/rfc0821.txt
4. **Professional C# 2008** – *Wiley-India Edition*
5. **C#** - *SMU Study Material*
6. Programmer's Community: www.codeproject.com
7. **Microsoft Software Developer Network** (*Webpage and Offline Help*)

[illegible]