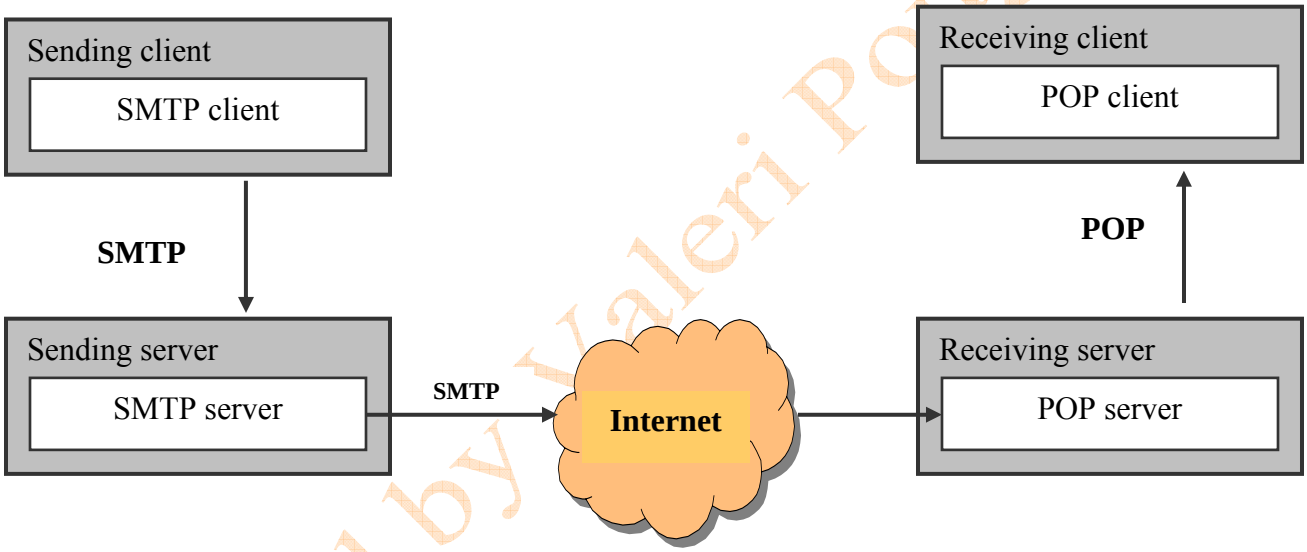
Lecture 5

E-mail in ASP.NET

How to send e-mail

When you create a web application, you often need to send e-mail messages from the application. For instance, when a user makes a purchase from an e-commerce site, a web application usually sends an e-mail to the customer that confirms the order. Or, if a serious errors, the web application often sends an e-mail message to the support staff that documents the error.

To send e-mail you, usually, use ***mail client*** software such as Microsoft Outlook or Outlook Express that allows you to send and retrieve e-mail messages. This type software communicates with a ***mail server*** that actually sends and retrieves your e-mail messages. Most likely, your mail server software is provided by your Internet Service Provider (ISP) or through your company. The diagram below shows how this works.



The two protocols that are commonly used to send and retrieve e-mail messages are ***SMTP*** and ***POP***. The third protocol you should know about is ***MIME***, which stands for ***Multi-purpose Internet Message Extension***. Unlike *SMTP* or *POP, MIME* isn't used to transfer e-mail messages. Instead, it defines how the content of an e-mail message and its attachments are formatted.

Three e-mail protocols

Protocol	Description
SMTP	***Simple Mail Transfer Protocol*** is used to send a message from one server to another.
POP	***Post Office Protocol*** is used by mail clients to retrieve messages from mail servers. Currently, POP is in version 3 and is known as POP3.
MIME	The ***Multi-purpose Internet Message Extension*** specifies the type of content that can be sent as a message or attachment.

Three common reasons for sending e-mail from an ASP.Net application

- **To confirm receipt of an order.** When the user completes an order, the application can e-mail a confirmation of the order to the user.
- **To remind a registered user of a forgotten password.** If the user forgets his/her password, the application can email the password to the e-mail address that's on file for the user.
- **To notify support personnel of a problem.** If a problem like an unhandled exception occurs, the application can email a message that summarized the problem to the appropriate person.

Configuring the SMTP service in IIS

Before you can test an ASP.Net application that sends e-mail messages, you must enable the e-mail server that's built into IIS and configure it to send mail from your application.

Install and Configure SMTP Virtual Servers in IIS

In order to send e-mail from an ASP.NET Web application, you must have the Simple Mail Transfer Protocol (SMTP) service of Internet Information Services (IIS) installed and configured on your server. The IIS SMTP service is a simple component for forwarding e-mail messages to an SMTP server for delivery.

Installing the SMTP Service

The SMTP service is not installed by default with IIS. You must install the SMTP service using **Control Panel**. Installing the SMTP service creates a default SMTP configuration that you can then customize by using IIS Manager.

To install the SMTP service on IIS 6.0

1. On the **Start** menu, click **Control Panel**, double-click **Add or Remove Programs**, and then click **Add/Remove Windows Components**.
2. In the **Components** list, click **Application Server**, and then click **Details**.
3. In the **Subcomponents of Application Server** list, click **Internet Information Services (IIS)**, and then click **Details**.
4. In the **Subcomponents of Internet Information Services (IIS)** list, select the **SMTP Service** check box, and then click **OK**.
5. Click **Next**. If you are prompted for the Windows CD or the network install path, follow the instructions.
6. Click **Finish**.

When you install the SMTP service, a default SMTP server configuration is created with a message store in *C:\Inetpub\Mailroot*.

When you are setting up the SMTP service, you can configure global settings for the SMTP service, as well as settings for individual components of the virtual server. The IIS SMTP service is a relay agent only. E-mail messages are forwarded to an SMTP server for delivery.

Configuring the SMTP Virtual Server

Installing the SMTP service will create a new node in IIS Manager. In order to configure the SMTP virtual server, you must start IIS Manager.

Important

You must be logged on as a member of the Administrators group on the local computer to perform the following procedure (or procedures), or you must have been delegated the appropriate authority.

To start IIS Manager from the Run dialog box

1. On the **Start** menu, click **Run**.
2. In the **Open** box, type ***inetmgr*** and then click **OK**.

To start IIS Manager from the Administrative Services console

1. On the **Start** menu, click **Run**. In the **Run** text box, type **control panel**, and then click **OK**.
2. In the **Control Panel** window, click **Administrative Tools**.
3. In the **Administrative Tools** window, click **Internet Information Services**.

Default Settings

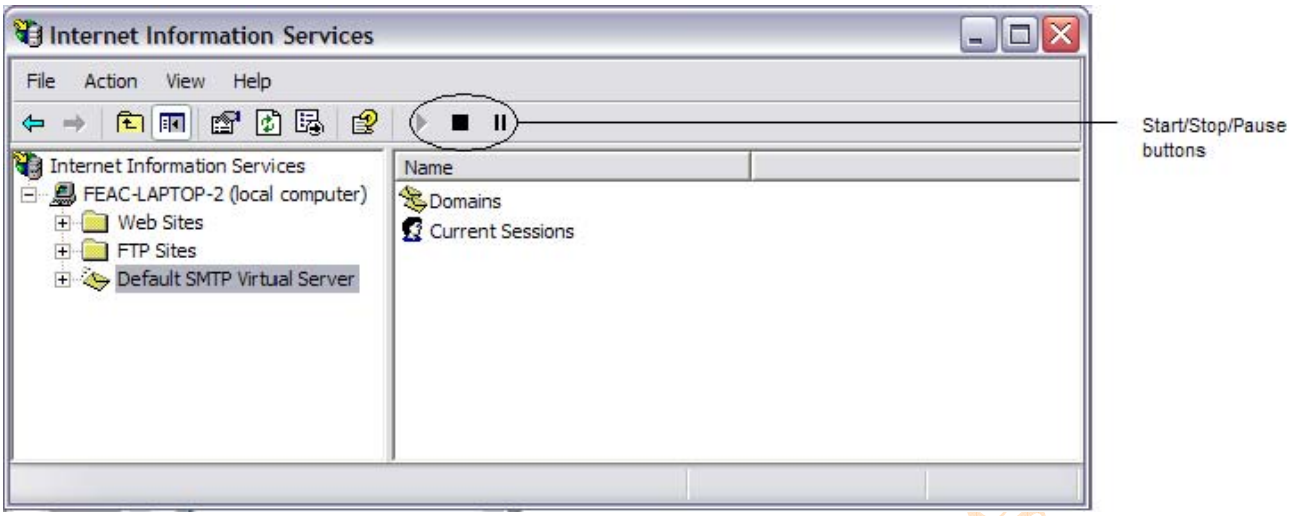
The default SMTP virtual server has the following default settings. If you create a new virtual server, you can configure default settings using the New Virtual Server Wizard.

- **Name**: The name of the virtual server that appears in IIS Manager. You can change the name of the virtual server in IIS Manager. Simply right-click the virtual server and then click **Rename**.
- **IP address/TCP port**: All unassigned/25.You can use the **General** tab in the **SMTP virtual server properties** dialog box to change this setting. If you change this setting, you must specify an IP address and TCP port combination that is not being used by another SMTP virtual server. TCP port 25 is both the default TCP port and the recommended TCP port. More than one virtual server can use the same TCP port, but they must be configured with different IP addresses. If you do not set a unique IP address and TCP port combination, the SMTP virtual server will not start.

Default domain: The domain name that is listed on the **Computer Name** tab in **System Properties**. The default domain is used to stamp messages from addresses that do not have a domain. An SMTP virtual server can have only one default domain, and it cannot be deleted. To change the name of the default domain in IIS Manager, double-click the virtual server, and then double-click **Domains**. Right-click the local (or default) domain, and then click **Rename**.

- **Home directory**: *C:\Inetpub\Mailroot*.The home directory is the root of your SMTP content directories, and it must be local to the computer on which the SMTP service runs.

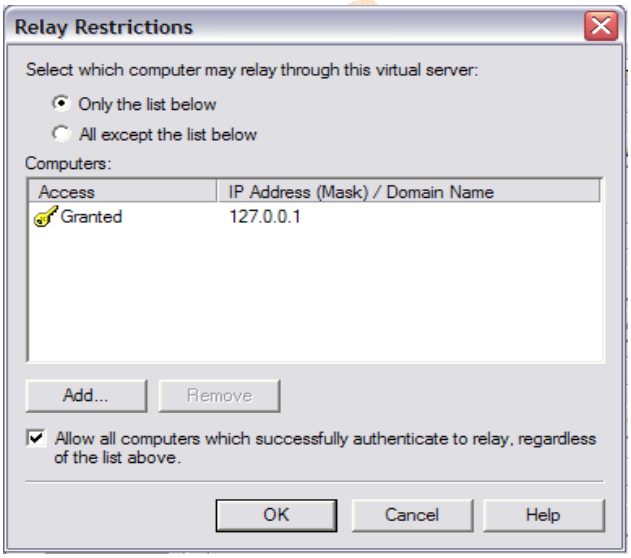
How to make sure *SMTP* is running on your local server

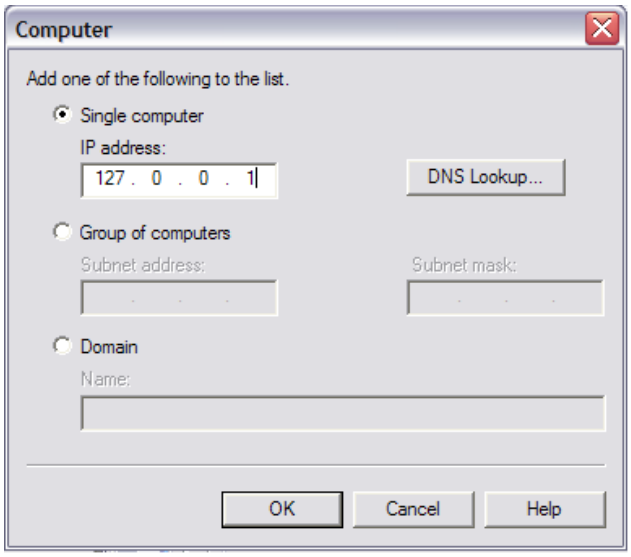


1. Open the Control Panel, double-click Administrative Tools, then double-click icon Internet Information Services (Windows XP) to open the IIS management console shown above. (In Windows 2000, this icon named Internet Information Manager)
2. Expand the tree to display your local computer's IIS server and the services running under it.
3. Select the *Default SMTP Virtual Server* service.
4. If the *SMTP* service is not running, the Stop button (■) will be disabled. In that case, click the Start button (►) to start the SMTP service.

Once you've started the SMTP service, you must configure it so that it allows your ASP.Net applications to send mail. To do that, you enable relaying for the local computer. This allows the mail server to accept mail from a specified computer and deliver it to the intended recipient. To configure the *SMTP* service to rely mail from the local computer, open the *Relay Restrictions* dialog box as described below.

How to allow relaying for the local host





1. In IIS management console, right-click the Default *SMTP* Virtual Server and choose the Properties command.
2. Click the *Access* tab, then click the *Relay* button. This displays the *Relay Restrictions* dialog box shown above.
3. Click the *Add* button to display the *Computer* dialog box shown above. Then select *Single Computer*, type, *127.0.0.1* into IP address field, click *OK* to close the *Computer* dialog box, and click *OK* again to close the *Relay Restrictions* dialog box.

How to create an e-mail message

Constructors and properties of the *MailMessage* class

Constructor	**Description**
MailMessage()	Creates an empty mail message
MailMessage(from, to)	Creates a mail message with the ***to*** and ***from*** addresses specified as strings or *MailAddress* objects.
MailMessage(from, to, subject, body)	Creates a mail message with the to address, from address, subject, and body specified as string
Property	**Description**
From	A *MailAddress* object for the message sender..
To	A collection of *MailAddress* objects for the message recipients.
Cc	A collection of *MailAddress* objects for the copy message recipients.
Bcc	A collection of *MailAddress* objects for the blind copy recipients.
Subject	The subject line for the message.
Body	The body of the message.
IsBodyHtml	A Boolean value that indicates if the body of the message contains HTML.
Attachments	A collection of *MailAttachment* objects.

Constructors of the *MailAddress* class

Constructor	Description
MailAddress(address)	Creates an email address with the specified address string
MailAddress(address, displayname)	Creates an email address with the specified address and display string.

Code that creates an e-mail message with a carbon copy

```
private void SendTextMessageCC(string fromAddress, string fromName, string toAddress,
        string subject, string body, string ccAddress)
{
        MailAddress fromAdd = new MailAddress(fromAddress, fromName);
        MailAddress toAdd = new MailAddress(toAddress);
        MailAddress ccAdd = new MailAddress(ccAddress);
        msg.Subject = subject;
        msg.Body = body;
        msg.CC.Add(ccAdd);
}
```

Another way to create a message

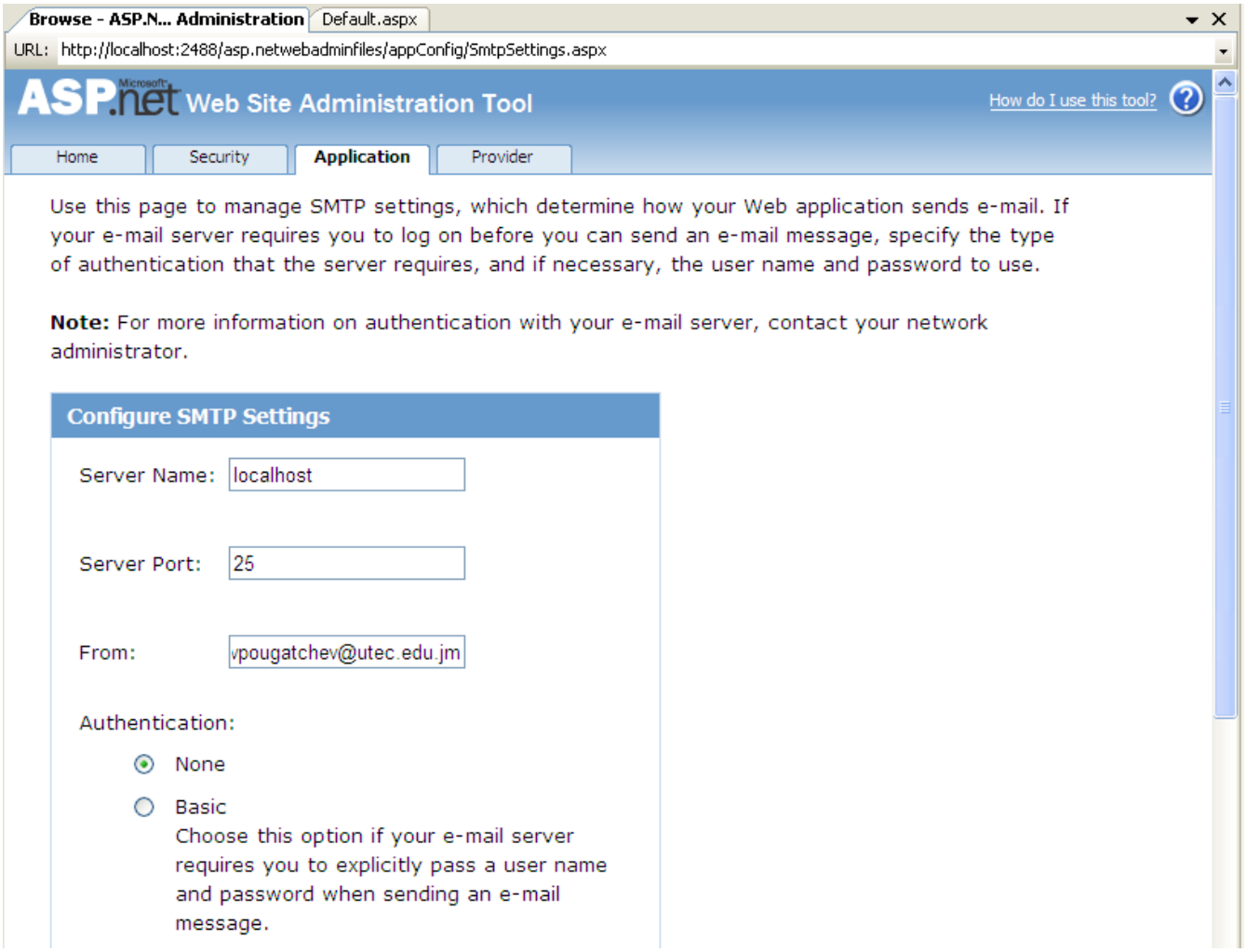
```
MailMessage msg = new MailMessage(fromAddress, toAddress, subject, body)

msg.CC.Add(new MailAddress(CCAddress));
```

You can send a message to any number of recipients. To do that, you need to use the first constructor for the *MailMessage* class shown below, to create an empty mail message. Then, you can create a *MailAddress* object for the sender and assign it to the *Form* property of the message. And you can create a *MailAddress* object for each recipient and add it to the collection of *MailAddress* object returned by the *To* property of the mail message. For example, to create a message that will be sent to two people, your code will look something like this:

```
MailMassage msg = new MailMessage();

msg.From = new MailAddress(vpougatchev@utech.edu.jm);
msg.To.Add(new MailAddress(tsmith@utech.edu.jm));
msg.To.Add(new MailAddress(akelly@utech.edu.jm));
```

How to send an e-mail message

After you create an e-mail message, you use the *SMTPClient* class shown below to send the message. The technique you use to do that depends on the message you're sending and on whether you have set the *SMTP* configuration settings for the application. To set the *SMTP* configuration settings, you use the Web Site Administration Tool:



Then, these settings are saved in the *web.config* file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.net>
    <mailSettings>
      <smtp from="vpougatchev@utec.edu.jm">
        <network host="localhost" password="" userName="" />
      </smtp>
    </mailSettings>
  </system.net>
</configuration>
```

Constructors and methods of the *SmtpClient* class

Constructor	Description
SmtpClient()	Creates a client using the settings specified in the *web.config* file
SmtpClient (name)	Creates a client that can send e-mail to the specified *SMTP* server.
SmtpClient(name, port)	Creates a client that can send e-mail to the specified *SMTP* server and port.
Property	**Description**
Send(message)	Sends the specified *MailMessage* object
Send(from, to, subject, body)	Creates and sends an email message using the specified from, to, subject, and body strings.

Code that sends a message using settings in the *web.config* file

```
SmtpClient client = new SmtpClient();
client.Send(msg);
```

If you haven't set the SMTP configuration options, or if you want to override these options, you can specify the domain name of the server when you create the *SmtpClient* object. This is illustrated in code below.

Code that creates and sends a message to a named server

```
SmtpClient client = new SmtpClient("localhost");
client.Send(fromAddress, toAddress, subject, body);
```

Here you do not need to specify a port number. That's because the default port number is 25, which is also the default port for the IIS server. If you use a server at different port, you'll have to specifiy the port number.

The code-behind of the *Default.aspx* of the *CH21Email* application, which sends e-mail

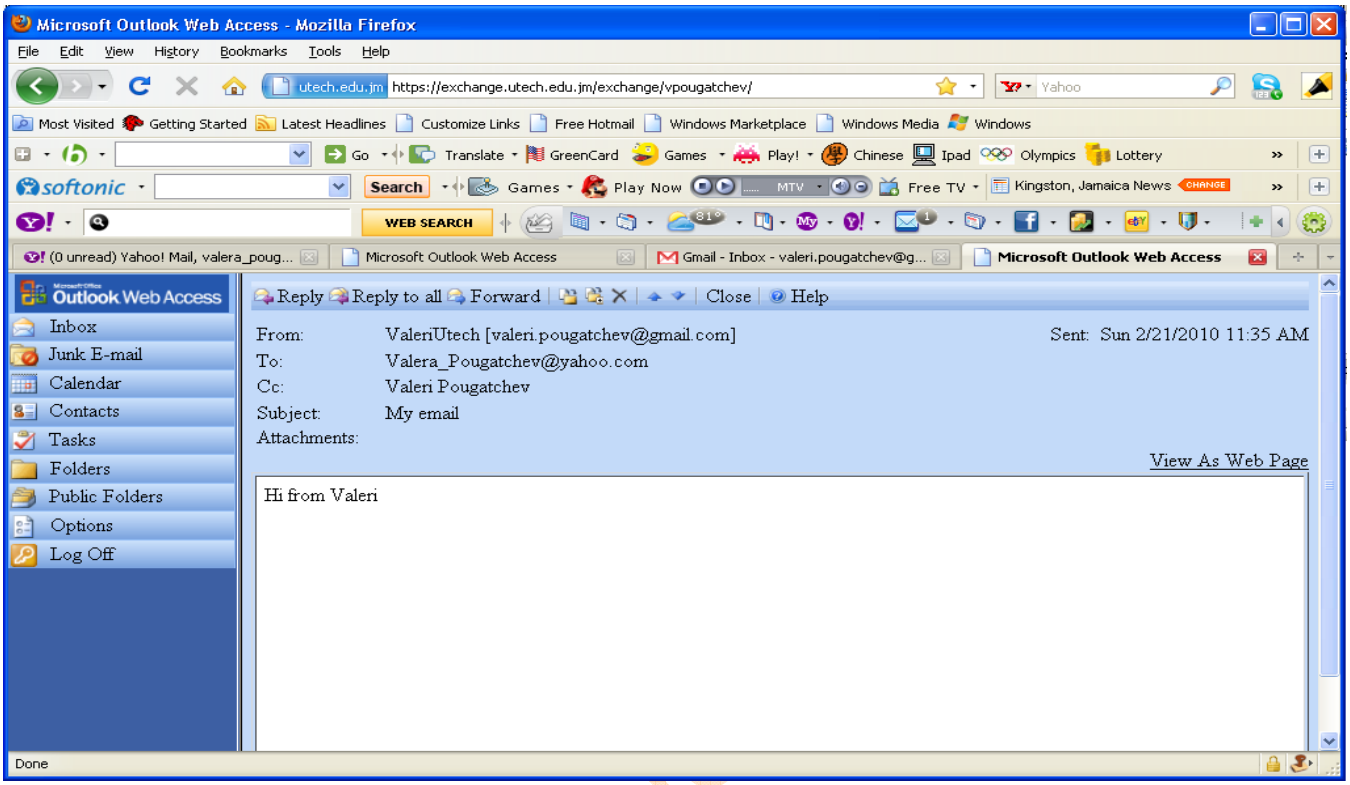
```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        MailAddress fromAdd = new MailAddress("vpougatchev@utech.edu.jm", "ValeriUtech");
        MailAddress toAdd = new MailAddress("Valera_Pougatchev@yahoo.com");
        MailAddress ccAdd = new MailAddress("vpougatchev@utech.edu.jm");

        MailMessage msg = new MailMessage(fromAdd, toAdd);

        msg.Subject = "My email";
```

```
        msg.Body = "Hi from Valeri";
        msg.CC.Add(ccAdd);

        SmtpClient client = new SmtpClient();

        client.EnableSsl = true;

        client.Send(msg);
    }
}
```

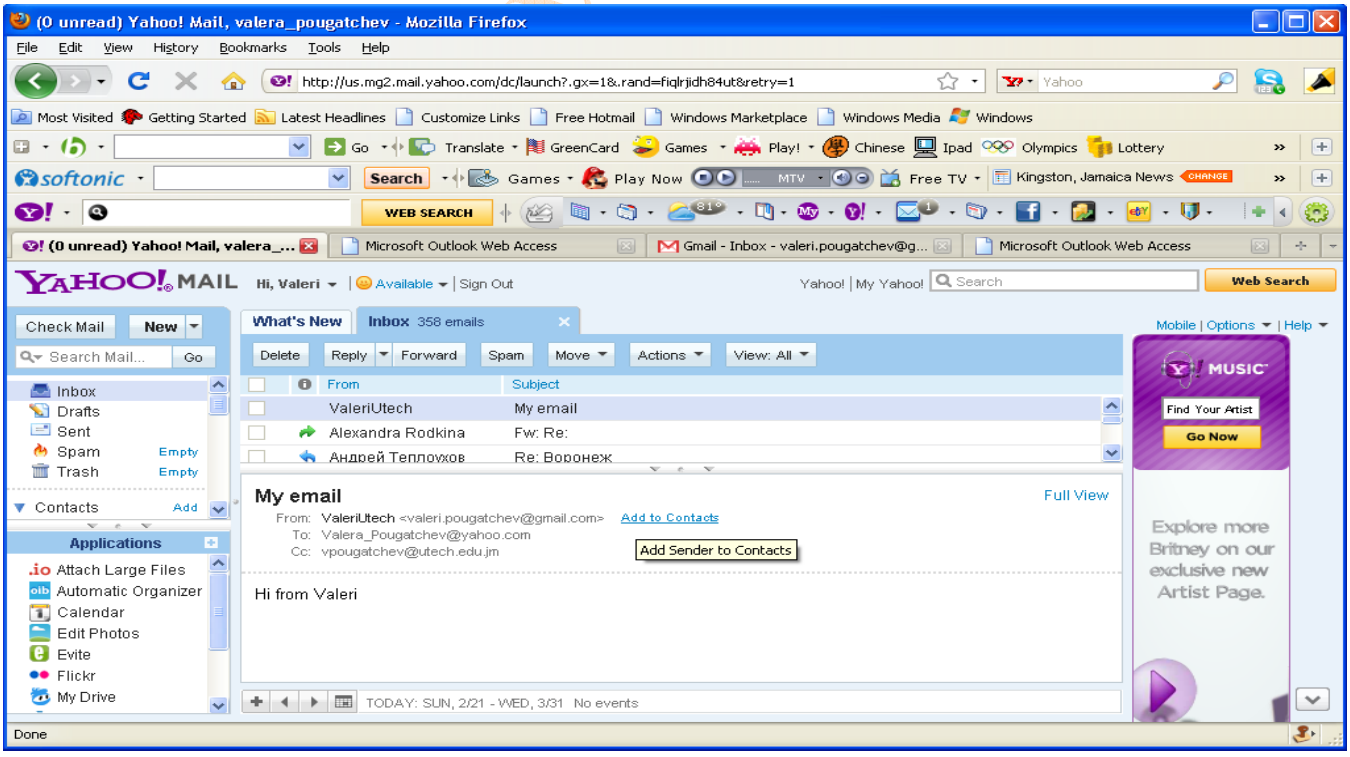
The *web.config* file is follow:

```
<?xml version="1.0"?>
<configuration>
        <system.net>
                <mailSettings>
                        <smtp deliveryMethod="Network" from="valeri.pougatchev@gmail.com">
                                <network
                                        port="587"
                                        host="smtp.gmail.com"
                                        password="<your password>"
                                        userName="valeri.pougatchev@gmail.com"
                                />
                        </smtp>
                </mailSettings>
        </system.net>
        <system.web>
                <compilation debug="true"/>
        </system.web>
</configuration>
```

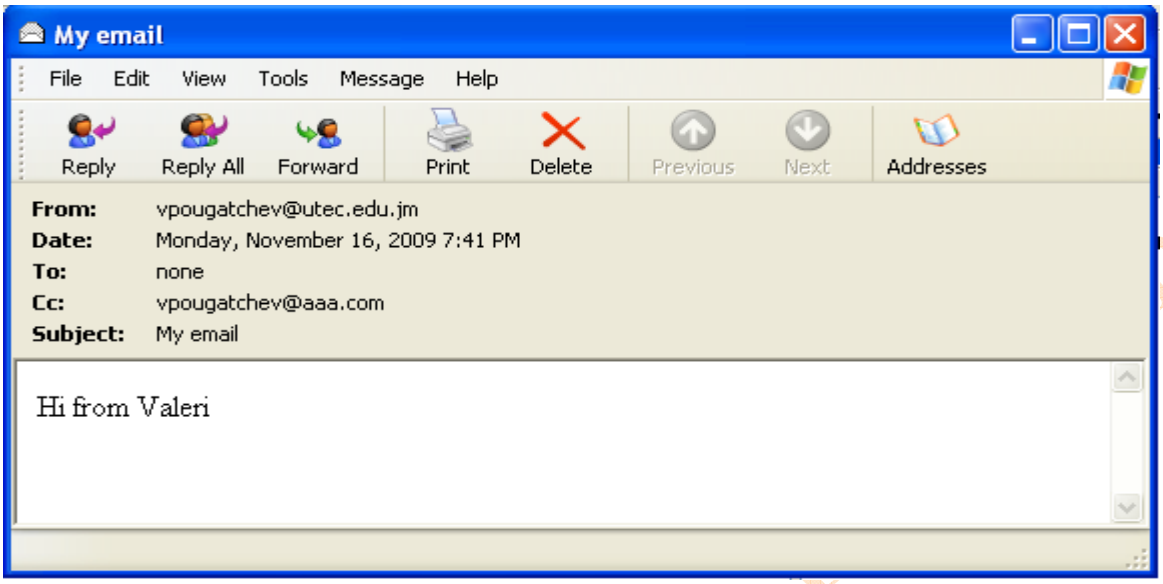
The user will receive the message in *vpougatchev@utech.edu.jm* sent by that application from *valeri.pougatchev@gmail.com* e-mail address:



The user will receive the message in *valera_pougatchev@yahoo.com* sent by that application from *valeri.pougatchev@gmail.com* e-mail address:



If process of sending is failed for some reason of receiver, the message will be send to the *C:\Inetpub\mailroot\Queue* address on localhost machine and named something like *NTFS_be094d6501cab31200000011*. It can be viewed by Outlook Express like:



How to add an attachment to an e-mail message

An ***attachment*** is a file that's sent along with an e-mail message.

The syntax for creating an attachment

attachement = new Attachment(filename);

One way to create a new attachment and add it to a message

string filename = "C:\\HalloweenStore\\Attachments\\ReturnPolicy.doc";

Attachment attach = new Attachment(fileName);

MailMessage msg = new MailMessage(fromAddress, toAddress, subject, body);

msg.Attachments.Add(attach);

Another way to create a new attachment and add it to a message

string filename = "C:\\HalloweenStore\\Attachments\\ReturnPolicy.doc";

MailMessage msg = new MailMessage(fromAddress, toAddress, subject, body);

msg.Attachments.Add(new Attachment(fileName));

Since the *SMTP* protocol is designed to send text messages, not binary files, any e-mail attachement for a binary file must vbe converted to text format before it can be sent. Then, the text attachement must be converted back to a binary file when it's received. The most common format for converting attached binary files to text and back to binary is called ***UUEncode***, and it's used by default. The other available format for converting binary files is called *Base64*.

How to create an HTML message

By default, e-mail message consist of the plain text with no formatting. However, you can create a formatted message by using *HTML* as the *MIME* type, described in example bolowe:

```
private void SendConfirmation()
{
        MailMessage msg = new MailMessage(halloween@utech.edu.jm, email);

        msg.Subject = "Order Confirmation";
        msg.Body = this.GetConfirmationMessage();
        msg.IsBodyHtml = true;

        string file = "C\\ASP.NET 2.0 WebSites\\EmailHalloweenStore\\Images\\banner.jpg";

        msg.Attachments.Add(new SmtpClient("localhost"));
        client.Send(msg);
}

Private string GetConfirmationMessage()
{
        string message = "<HTML><head><title>Order confirmation</tutle></head>"
                        + "<body><img src='banner.jpg' alt='Halloween Store' />"
                        + "<br /><br /><h3>Thanks for your order, "
                        + "dvCustomer[0]["FirstName"].ToString() + "
                        + "dvCustomer[0]["LastName"].ToString() + "!</h3>"
                        + "<p>Your order total is " + total.ToString("c")
                        + ".</p></body></HTML>";
}
```

Using a custom error handling

When an error occurs in an ASP.NET application, an exception is thrown. Then, if the exception isn't handled by the application, an ASP.NET Server Error is displayed. This page includes an error message, aportion of the source code that threw the unhandled exception, and other debugging information. Since this type of error page usually isn't appropriate for the users of an application, you typically replace the generic error pages with your own custom error pages after you're done testing the web site but before you go live with it.

An introduction to custom error handling

Four ways to display a custom error page when an exception occures

- Use *try-catch* statement to catch exceptions as they occure, then redirect or transfer to a custom error page.
- Use the *Page-error* event handler in a code-behind file to catch unhandled exceptions at the page level, then redirect or transfer to a custom error page.
- Use the *Application_error* event handler in the *global.asax* file to catch unhandled exceptions at the application level, then redirect or transfer to a custom error page.
- Use the *customError* element of the *web.config* file to specify custom error pages that are displayed for specific types of HTTP errors. This technique is used to display custom error pages when common HTTP errors such as a 404 – Not Found error occur.

Getting anf using the *Exception* object for an error

To get the properties and methods of the *Exception* and *HttpServerUtility* classes you can use the *Exception* object for an error.

Common properties of the *Exception* class

Property	Description
Message	A message that describes the error
Source	The name of the application or object that caused the error
InnerException	The *Exception* object that caused the exception at the application level

Methods of the *HttpServerUtility* class for working with exceptions

Method	Description
GetLastError()	Gets the most recent exception
ClearError()	Clears the most recent exception

Code that gets the *Exception* object at the method level

```
try
{
        Statements that could throw an exception
}
catch (Exception ex)
{
        Statement that use the Exception object named ex
}
```

The example below shows how to get the *Exception* object within the *Page_Error* event handler. This event handler is executed automatically if an exception isn't handled by the other methods of a code-behind file:

```
Exception ex = Server.GetLastError;
```

The next example shows how to get the *Exception* object within the *Application_Error* event handler of the *global.asax* file. This event handler is executed automatically if an exception isn't handled by of the methods in the code-behind file including the *Page-Error* event handler. In the *Application_Error* event handler, however, the *GetLastError* method doesn't return the correct *Exception* object. That's because a second exception called *HttpUnhandledException* is thrown if an exception occurs and exception isn't handled by a try-catch statement or a *Page_Error* event handler. As a result, the *GetLastError* method returns the *HttpUnhandledException* exception, you must use the *InnerException* property of the *Exception* object that's returned by the *GetLastError* method.

```
Exception ex = Server.GetLastError().InnerException;
```

Althogh you might think that you could use the *ClearError* method to clear the *HttpUnhandledException* exception and then use *GetLastError* to get the original exception, that won't work. That's because you can only use *GetLastError* to get the *Exception* object for the last exception that occures. In this case, that's the *HttpUnhandledException* exception.

The last example shows how you migth use the properties of an *Exception* object as you test an application.

```
Server.ClearError();

Response.Write(ex.Message + "<br />" + ex.Source);
```

Here, the *Write* method of the *HttpResponse* object is used to display the *Message* and *Source* propertes on the page. Notice that before this method is executed, the *ClearError* method of the *Server* object is used to clear the error. That way, the ASP.NET Server Error page won't be displayed.

Methods that redirect to a custom error page

There are three ways to redirect to a custom error page when an exception occures in an ASP.NET application.

A *try-catch* statement that redirects to a custom error page if an exception occures during a database operation

```
try
{
        SqlDataSource1.Insert();
}
catch (Exception ex)
{
        Session["Exception"] = ex;
        Response.Redirect("ErrorPage.aspx")
}
```

A *Page_Error* event handler that redirects to a custom error page

```
private void Page_Error(object sender, EventArgs e)
{
        Exception ex = Server.GetLastError();
        Session["Exception"] = ex;
        Response.Redirect("ErrorPage.aspx");
}
```

An *Application_Error* event handler in the *global.asax* file that redirects to a cusot error page

```
 void Application_Error(object sender, EventArgs e)
 {
        Exception ex = Server.GetLastError().InnerException;
        Session["Exception"] = ex;
        Response.Redirect("ErrorPage.aspx");
 }
```

The code fo a custom error page

The *aspx* code for the body of the *Error* page

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Image ID="Image1" runat="server"
                ImageUrl="~/Images/banner.jpg" /><br />
            <h2>Oh no!</h2>
            Something unexpected happened, which caused us to drop what we were doing
            and come here. We're sorry for the inconvenience.<br /><br />
            <asp:Button ID="btnReturn" runat="server"
                Text="Return to Order Page" PostBackUrl="~/Order.aspx"
        </div>
    </form>
</body>
```

The *C#* code for the *Error* page

```
using System.Net.Mail;

public partial class ErrorPage : System.Web.UI.Page
{
    Protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            Exception ex = (Exception) Session["Exception"];

            this.SendEmail(ex);

            Session.Remove("Exception");
        }
    }

    private void SendEmail(Exception ex)
    {
        string body = "An exception occured at "
                + DateTime.Now.ToLongTimeString()
                + " on " + DateTime.Now.ToLongDateString()
                + "<br />" + ex.Message;

        MailMessage msg = new MailMessage("halloween@utech.edu.jm"),
            support@utech.edu.jm);

        msg.Subject = "Exception in Halloween application";
```

```
            msg.Body = body;
            msg.IsBodyHtml = true;

            SmtpClient client = new SmtpClient("localhost");

            client.Send(msg);
        }
}
```

Handling the Http errors with *web.config* file

Not all unrecoverable errors cause ASP.NET to throw an exception. Below are some error conditions result in *HTTP* errors that are handled by the web server itself. For these errors, you can use the customErrors element in the web.config file to specify custom error pages.

A *customErrors* element in the *web.config* file that designed custom error page

```
<customErrors mode="On" defaultRedirect="DefaultError.aspx"></>
        <error status="404" redirect="E404.aspx"
        <error status="500" redirect="E500.aspx"
</customError>
```

Common *Http* error codes

Code	Description
401	Unauthorized request. The client must be authorized to access the resources.
403	Forbidden request. The client is not allowed to access the resource.
404	File Not Found. The recourse could not be located.
500	Internal Server Error. This is usually the result of an unhandled exception.

You can add a *customError* element to the *web.config* file using the web Site Administration Tool. To do that, select the *Website* → *ASP.NET Configuration* command, display the *Administration* tab, click the *Define Defaul Error Page* link, and then select the appropriate settings.