**SENDING EMAIL IN ASP**

Suppose for a moment that you’re the webmaster for an online store, and you want to send an email confirmation to each customer who places an order. Rather than manually typing an email to every customer about every order, you could automate the process using ASP.NET.

The namespace that groups the .NET mail-related classes is System.Net.Mail. The most useful classes in this namespace are:

**SmtpClient**

contains functionality for sending email

**MailMessage**

represents an email message

**Attachment**

represents an email attachment

**AttachmentCollection**

represents a collection of Attachment objects

**MailAddress**

represents an email address

**MailAddressCollection**

represents a collection of email addresses

A core set of features is common to most email programs. For instance, they all enable you to send an email to someone by typing the recipient’s email address in a To field. You are also able to specify who the email is from, the subject of the message, and the body content of the email. All these properties—and more—are available through the MailMessage class. Here’s a partial list of the properties that MailMessage supports:

**From**

specifies the address from which the email message is to be sent

**To**

specifies the address to which the email message is to be sent

**CC**

specifies the carbon copy field of the email message

**Bcc**

specifies the blind carbon copy field of the email message

**Attachments**

a collection of items or files attached to the email message

**Subject**

specifies the subject of the email message

**Body**

defines the body of the email message

**IsBodyHtml**

True if the message is in HTML format; False otherwise (defaults to False) Other properties of MailMessage that you may need to use include AlternateViews, BodyEncoding, DeliveryNotificationOptions, Headers, Priority, ReplyTo, Sender, and SubjectEncoding. The From field has the MailAddress type which represents an email address. The To, CC, and Bcc properties are of the MailAddressCollection type, and represent a collection of MailAddress objects. As you can see, there are lots of classes and properties that let you define email messages. However, to be able to send these messages, you need access to a SMTP server. The standard email protocol of the Internet is **Simple Mail Transfer Protocol** (SMTP). When you use ASP.NET to send an email, the message is relayed through one or more SMTP servers on the way to its final destination. Most ISPs provide an SMTP. server for customers’ use; alternatively, if you’re using IIS, you can make use of Windows’ built-in SMTP Server.

**PROGRAM CODE:**

Imports System.Net.Mail

Public Class Form1

Private Sub Label2\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label2.Click

End Sub

Private Sub Form1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

End Sub

Private Sub Button1\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

If TextBox2.Text = "" Then

ErrorProvider1.SetError(TextBox2, "Please Enter Password")

End If

If TextBox1.Text = "" Then

ErrorProvider1.SetError(TextBox1, "Please Enter Username")

End If

If TextBox3.Text = "" Then

ErrorProvider1.SetError(TextBox3, "Please Enter Receivers mail ID")

End If

If TextBox4.Text = "" Then

ErrorProvider1.SetError(TextBox4, "Please Enter Subject")

End If

If TextBox5.Text = "" Then

ErrorProvider1.SetError(TextBox5, "Please Enter Message Body")

End If

Try

Dim smtpserver As New SmtpClient

Dim mail As New MailMessage

smtpserver.Credentials = New Net.NetworkCredential(TextBox1.Text, TextBox2.Text)

smtpserver.Port = 587

smtpserver.Host = "smtp.gmail.com"

smtpserver.EnableSsl = True

mail = New MailMessage

mail.From = New MailAddress(TextBox1.Text)

mail.Subject = TextBox4.Text

mail.Body = TextBox5.Text

smtpserver.Send(mail)

MsgBox("Message sent successfully")

Catch ex As Exception

MsgBox("There is problem with connection")

End Try

End Sub

End Class

Sending e-mail with CDOSYS

CDO (Collaboration Data Objects) is a Microsoft technology that is designed to simplify the creation of messaging applications.

CDOSYS is a built-in component in ASP. We will show you how to use this component to send e-mail with ASP.

How about CDONTs?

Microsoft has discontinued the use of CDONTs on Windows 2000, Windows XP and Windows 2003. If you have used CDONTs in your ASP applications, you should update the code and use the new CDO technology.

Examples using CDOSYS

Sending a text e-mail:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.TextBody="This is a message."  
myMail.Send  
set myMail=nothing  
%>

Sending a text e-mail with Bcc and CC fields:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.Bcc="someoneelse@somedomain.com"  
myMail.Cc="someoneelse2@somedomain.com"  
myMail.TextBody="This is a message."  
myMail.Send  
set myMail=nothing  
%>

Sending an HTML e-mail:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.HTMLBody = "<h1>This is a message.</h1>"  
myMail.Send  
set myMail=nothing  
%>

Sending an HTML e-mail that sends a webpage from a website:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.CreateMHTMLBody "http://www.w3schools.com/asp/"  
myMail.Send  
set myMail=nothing  
%>

Sending an HTML e-mail that sends a webpage from a file on your computer:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.CreateMHTMLBody "file://c:/mydocuments/test.htm"  
myMail.Send  
set myMail=nothing  
%>

Sending a text e-mail with an Attachment:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.TextBody="This is a message."  
myMail.AddAttachment "c:\mydocuments\test.txt"  
myMail.Send  
set myMail=nothing  
%>

Sending a text e-mail using a remote server:

< %  
Set myMail=CreateObject("CDO.Message")  
myMail.Subject="Sending email with CDO"  
myMail.From="mymail@mydomain.com"  
myMail.To="someone@somedomain.com"  
myMail.TextBody="This is a message."  
myMail.Configuration.Fields.Item \_  
("http://schemas.microsoft.com/cdo/configuration/sendusing")=2  
'Name or IP of remote SMTP server  
myMail.Configuration.Fields.Item \_  
("http://schemas.microsoft.com/cdo/configuration/smtpserver")="smtp.server.com"  
'Server port  
myMail.Configuration.Fields.Item \_  
("http://schemas.microsoft.com/cdo/configuration/smtpserverport")=25   
myMail.Configuration.Fields.Update  
myMail.Send  
set myMail=nothing

%>

**WORKING WITH FILES**

The tasks of writing to and reading from text files with the aid of the classes that work with files and streams. These classes include:

**File**

contains methods for working with files

**FileStream**

represents a stream for reading and writing to files

**StreamReader**

reads characters from a text file

**StreamWriter**

writes characters to a text file

**Path**

contains methods for manipulating a file or directory

For the most part, we read from and write to text files by using the File class to return a stream. If we want to write to a text file, we use the StreamWriter class; conversely, we use the StreamReader class to read from a text file. Before our ASP.NET page can read and write files to your hard disk, the ASP.NET page must have permissions to access the file we’re trying to read or write. The task of setting the permissions depends on our context. create a new web form called **WriteFile.aspx**. Make sure you *aren’t* using a code-behind file or a master page. Next, enter the code shown here in bold:

<%@ Page Language="VB" %>

**<%@ Import Namespace="System.IO" %>**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>**Writing to Text Files**</title>

</head>

<body>

<form id="form1" runat="server">

<div>

**<p>Write the following text within a text file:</p>**

**<asp:TextBox ID="myText" runat="server" />**

**<asp:Button ID="writeButton" Text="Write" runat="server"**

**OnClick="WriteText" />**

</div>

</form>

</body>

</html>

As you can see, we import the System.IO namespace—the namespace that contains the classes for working with text files—first. Next, we add a TextBox control to handle collection of the user-entered text, and a Button control to send the information to the server for processing. Next, in the <head> tag, we’ll create the WriteText method mentioned in the OnClick attribute of the Button. Using statement is used to reduce the use of complete namespace while using the classes. For example if someone has to use the FileInfo class which is in System.IO namespace instead of writing System.IO.FileInfo one can write Using System.IO; in the declaration section and then use the fileinfo class directly. This method will write the contents of the TextBox to the text file:

FOR C#

<script runat="server">

**void WriteText(Object s, EventArgs e)**

**{**

**using (StreamWriter streamWriter = File.CreateText(**

**@"C:\myText.txt"))**

**{**

**streamWriter.WriteLine(myText.Text);**

**}**

**}**

</script>

FOR VB

<script runat="server">

**Sub WriteText(ByVal s As Object, ByVal e As EventArgs)**

**Using streamWriter As StreamWriter = File.CreateText( \_**

**"C:\myText.txt")**

**streamWriter.WriteLine(myText.Text)**

**End Using**

**End Sub**

</script>

**Reading Content from a Text File**

Just as you used the CreateText and AppendText methods of the File class to return a new StreamWriter object, you can use the OpenText method of the File class to return a new StreamReader. Once the StreamReader has been established, you can loop through the text file using a While loop in conjunction with the object’s ReadLine method to examine the contents of the text file. To experiment with the process of reading from text files, create a new web form named ReadFile.aspx in the same way that you created WriteFile.aspx, and add this code to it:

<%@ Page Language="VB" %>

**<%@ Import Namespace="System.IO" %>**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>**Reading from Text Files**</title>

</head>

<body>

<form id="form1" runat="server">

<div>

**<asp:Button ID="readButton" Text="Read" runat="server"**

**OnClick="ReadText" />**

**<br />**

**<asp:Label ID="resultLabel" runat="server" />**

</div>

</form>

</body>

</html>

As you can see, we’ve simply added a Button and Label to the page. When the user clicks the button, the Click event will be raised and the ReadText method will be called. Let’s add this method next. It will read the text from the text file and write it out to the Label control:

FOR VB

<script runat="server">

Sub ReadText(ByVal s As Object, ByVal e As EventArgs)

Dim inputString As String

resultLabel.Text = ""

Using streamReader As StreamReader = \_

File.OpenText(MapPath("myText.txt"))

inputString = streamReader.ReadLine()

While (inputString <> Nothing)

resultLabel.Text &= inputString & "<br />"

inputString = streamReader.ReadLine()

End While

End Using

End Sub

</script>

FOR C#

<script runat="server">

void ReadText(Object s, EventArgs e)

{

string inputString;

resultLabel.Text = "";

using (StreamReader streamReader =

File.OpenText(MapPath("myText.txt")))

{

inputString = streamReader.ReadLine();

while (inputString != null)

{

resultLabel.Text += inputString + "<br />";

inputString = streamReader.ReadLine();

}

}

}

</script>

We declare a new string variable named inputString to hold the text we’ll read from the text file. Next, we set the text value of the Label control to an empty string. We do this in case the user presses the Read button when the Label already contains text from a previous click.

The next thing our method has to do is call the OpenText method of the File class to return a new StreamReader, again passing in the full path to the text file. And, once again, we’re using the Using construct to ensure the stream object is disposed of after we finish working with it.