

Sending a HTML and Plain Text E-newsletter with ASP.NET, Part 1

When developing a website, especially for advertising, marketing, or recruiting agency, the ability to keep in touch with your audience through an attractive e-newsletter that provides information about current events and news surrounding your company is essential to keep your visitors in touch, in tuned and wanting more. Most of the time, developers in charge of the website purchase mailing list programs that can be expensive, difficult to customize, both in appearance and in the script, and hard to setup on the remove server. The reason for this is simple; they don't have the knowledge or experience to develop one from scratch.

However, if you're familiar with HTML and CSS, using ASP.NET, you can develop an e-newsletter easily as long as you keep the expectations of what can be achieved visually through email within reason. In this article, we'll take a completed web page and learn how to integrate the HTML markup into an ASP.NET web form that will send the email to a recipient of our choice. Throughout the article, we'll identify the following: best methods used for the layout, formatting the visual appearance of the newsletter with CSS, recommended practices for using images, and last, how to write a script that will send both an html and plain text version of the newsletter.

If you would like to learn how to create an e-newsletter using the tools you already have in a practical context please do follow along below.

See Finished Web Page

<http://midwestwebdesign.net/tutorials/aspnetwebmail/email.html>

Examining the finished web page: Layout

Since this article's purpose is about integrating and sending an e-newsletter through ASP.NET, we'll focus on moving the finished newsletter into an ASP.NET page. As you can see from looking at the newsletter in a browser, there's nothing extraordinary about the design or the code behind it. Continuing, if you look at the source of the web page, you'll notice a fairly archaic markup that is reminiscent of the early 90's when web developers thought that using inline styles for CSS and table layouts were here to stay. Obviously that has changed, and today developers are pushing the limits on pure CSS layouts. However, we didn't do that in the newsletter. Our reason is simple; support for advanced CSS such as positioning in email clients is marginal at best. As a result, it's recommended to keep the layout in tables and use CSS, in our case, inline styles to format other characteristics of the page, such as headings, paragraphs, etc.

Examining the finished web page: DOCTYPES

Even though support for CSS in email is limited, the pros of using a DOCTYPE outweigh the cons of not using one.

Examining the finished web page: Inline styles

You'll notice there are no linked style sheets or embedded styles, that's because some email clients strip those out, so we'll stick with inline styles.

Examining the web page: Images

You'll notice we limited our use of images. There are several reasons for this. For starters, images take time to download and display in an email client. If you send a large size image through email to a visitor with a slower Internet connection, the image will take several seconds if not minutes to download and display, which could result in an unpleasant viewing experience. Furthermore, and arguably more important, images that are large in file size take time to be processed by the server that is sending the email, and by the server that is receiving the email. As a result, the desired number of images used in e-newsletters need to be limited. Also, it should be mentioned we're using an absolute path to retrieve our images. The reason(s) for this will be explained when moving our finished web page into the code behind file of our ASP.NET page.

Requirements for ASP.NET

Before creating our project and corresponding ASP.NET page, make sure you have .NET version 3.5 installed on your development machine. The reason for this will be explained in more detail later in the article, but it deals with needing to use Ajax server side controls.

Creating the project & ASP.NET page

In order to create the ASP.NET page that will have access to the Ajax server side controls, we'll need to create a new web project through either Microsoft Visual Web Developer Express or Microsoft Visual Studio. For the purposes of this article, we'll use the former, since it's a free download. From the program, follow these steps to create the new web project:

- From the main menu, select *File:New Project*
- For project type, select *Visual C#*, and then *Web*
- For templates, select *ASP.NET web application*

- In the name field, type *SendEmail*
- Choose an appropriate directory for your solution
- Left click *OK*

Your new web project will be created with a web form (*default.aspx*), and its corresponding code behind file, as well as a configuration file, *web.config*. Double click *default.aspx*; the default markup will be the following:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="default.aspx.cs"
Inherits="SendEmail._default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

    </div>
  </form>
</body>

</html>
```

We'll modify the markup shown above with ours. Add the following inside *default.aspx*:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="default.aspx.cs"
Inherits="SendEmail._default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
</head>
<body>
  <h3>Send E-Newsletter</h3>
  <form id="form1" runat="server">
    <asp:TextBox ID="txtTo" runat="server"/>
    <asp:Button ID="sendEmailBtn" runat="server" Text="Send Email"
OnClick="sendEmailBtn_Click"/><asp:Label ID="lblMessage" runat="server"/>
  </form>
</body>
```

</html>

As you can see from the code above, we added the following:

- <h3>
 - Gives our page a title of *Send E-Newsletter*
- <form runat="server">
 - Every ASP.NET web form must have a form tag with an attribute of runat equal to server
- <asp:TextBox
 - A server side text box control with the following parts:
 - ID="txtTo"
 - Gives our text box control a unique name so we can reference it in our code behind file
 - Runat="server"
 - Allows our button control to be run from the server with the resulting output being read by the browser
- <asp:Button
 - A server side button control with the following parts:
 - ID="sendEmailBtn"
 - Gives our button control a unique name so we can reference it in our code behind file
 - Runat="server"
 - Allows our button control to be run from the server with the resulting output being read by the browser
 - Text="Send Email"
 - Gives our button a visual text to indicate to our visitors what the button can do
 - Onclick="sendEmailBtn_Click"/>
 - Gives our button control a server-side event handler that will be used to send our newsletter
- <asp:Label
 - A server side label control with the following parts:
 - ID="lblMessage"
 - Gives our label control a unique name so we can reference it in our code behind file
 - Runat="server"
 - Allows our button control to be run from the server with the resulting output being read by the browser

The label control is used to indicate whether our email was sent successfully. Its text property will be set to a literal text string from our code behind file after the email has been sent. Save your changes and keep the file open.

Adding the send email event handler

When moving the HTML markup to our code behind file (done later in the article), we need to place it within our send mail event handler. In order to do this, let's first add an event handler to our code behind

file. To do this, double click the send email button from *default.aspx*. After double clicking the button, the code behind file should open automatically and inserted the following code:

```
protected void sendEmailBtn_Click(object sender, EventArgs e)
{
}
```

As you can see from the code above, by double clicking the button, the program automatically added the correct event handler, `sendEmailBtn_Click`, which is the same name as our `onClick` attribute of our button control, with the appropriate parameters.

Add the mail name space

In order to eventually send the e-newsletter in an email, we need to add the name space that will allow us to create a mail message object. In our code behind file, add the following below the existing name spaces:

```
using System.Web.UI.WebControls;
using System.Net.Mail;
```

As you can see from the code above, we added *System.Net.Mail* name space in order to expose the mail message class. There's a reason why didn't use *System.Web.Mail*, which is explained later in the article

Create variable for html email and mail message object

Continuing, let's create the variable that will hold our HTML markup from our finished page, as well as create a mail message object that will be used to specify who the email is from and who the email is going to. In our code behind file, in the send email event handler, add the following code:

```
protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    MailMessage mailMsg = new MailMessage("webmaster@domain.net", txtTo.Text);
    mailMsg.Subject = "MWD E-Newsletter";
    mailMsg.IsBodyHtml = true;}
}
```

As you can see from the code above, we added the following: a string variable named *bodyHTML* and set its value to empty. This makes the contents of our variable read-only. Continuing, we create an instance of a new object, *mailMsg* from the mail message class and pass in two parameters. The mail message class constructor accepts two parameters at a minimum, which can include any of the following: the sender of the email, the recipient of the email, the subject, or the body. In our case, for the sender, we used an email address. It's important to note here that hosts will not let you send an email from an account that isn't active on your domain. As a result, create a generic name, such as webmaster@domain.net to serve as the sender. For the recipient, we used our text box control, which allows us to send our e-newsletter to any email address we choose. Lastly, we use the *subject* property of the mail message object and set it to "MWD E-Newsletter", and set the *IsBodyHtml* property to true, since this is an HTML email. Before moving on, now would be a good time to save your changes.

Adding the HTML markup to our bodyHTML variable

In order to put the HTML markup of our finished page into our variable, we set our variable to accept a literal (verbatim) string as shown below:

```
protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    string bodyPlain = string.Empty;
    MailMessage mailMsg = new MailMessage("webmaster@domain.net", txtTo.Text);
    mailMsg.Subject = "MWD E-Newsletter";
    mailMsg.IsBodyHtml = true;
    bodyHTML = @"";
}
```

As you can see from the code above, we set our variable to accept a literal (verbatim) string of text by using the at (@) sign, following by two double quotes. In other words, using the at (@) symbol allows C# to read our html markup exactly as we have it.

Next, to get our finished web page into our variable, follow these steps:

- Open the finished web page in a browser
- Right click and select *View Source*
- Press Cntrl + A on your keyboard to select all the markup
- With the markup highlighted, right click and select *Copy*

In our code behind file, place your mouse cursor in the double quotes of our variable, right click and select *Paste*. You'll notice the code behind file doesn't like the html markup. The reason for this is simple, we haven't escaped our double and single quotes correctly. To resolve the conflicts, use *default_htmlversion.aspx* as a reference. As you can see by looking at the example file, our finished web page is simply copied inside the double quotes of our variable, thus we replaced the double quotes from our original html markup with single quotes. Now would be a good time to save your changes.

Working with images in emails

You may have noticed in the reference file that we linked directly to our images by using an absolute path. As mentioned earlier in the article, when sending newsletters in html format, you generally want to use images sparingly. For the contents of the email, you don't want to force the recipients to download multiple images, because they might have a slower Internet connection, or more importantly, most email clients, including Outlook by Microsoft, block images by default because they can contain malicious code. In this scenerio, once an image is downloaded that contains malicious code, the code can then be used to take over the computer and use it as a zombie to attack other computers, or do a variety of other tasks that you don't know about. Unfortunately, there's nothing from a code point of view to force recipients to unblock images, you're simply at the mercy of the person viewing the email.

From a code point of view, you have another option in ASP.NET other than the absolute method to send images in email. Instead of using an absolute path, you could embed the image in the email by creating a linked resource variable. In this case, the image becomes part of the actual email stream in memory. This may seem like a better alternative because you're instructing ASP.NET that the email will contain an image as part of the email, rather than relying on a hard coded path. However, this can become

problematic when sending the email. In this case, since the image becomes part of the email, it will increase the overall size of the email being sent. While this may not appear to be a big issue, it can become one in certain situations.

For example, if you decide to send this email to 50 plus people, each time the mail server creates the email message, it not only creates enough memory to send the email, but it also includes enough memory to embed the image. In other words, an email message that may be 11 kilobytes (in our case) using the direct linking option, may end up being double that because you're embedding the image in the same memory stream as the email, which can cause time out errors. And to complicate matters, some hosts don't allow embedded images in code because of security limitations in a shared hosting environment. To complicate the issue further, .NET version 1.1 uses the *System.Web.Mail* name space, which doesn't include an option for embedding an image. This is one of many reasons you want to use the new mail space instead. As a result, this is why it's generally a better option to link directly to an image instead of embedding.

Create the alternative view

At this point, we have created the variable that holds the contents of our e- newsletter, as well as creating the mail message object that allows us to send the newsletter to a recipient. However, we still need to add a HTML view so that our mail message object knows how to handle the email when sending it to an email client. In other words, just because we have our finished web page in a variable, we haven't told ASP.NET how to handle the variable when sending it to the mail message object. We do this by adding the following right below our existing code as shown below:

```
protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    string bodyPlain = string.Empty;
    MailMessage mailMsg = new MailMessage("webmaster@domain.net", txtTo.Text);
    mailMsg.Subject = "MWD E-Newsletter";
    mailMsg.IsBodyHtml = true;
    bodyHTML = @"";

    AlternateView htmlView =
    AlternateView.CreateAlternateViewFromString(bodyHTML, null, "text/html");
}
```

As you can see from the code above, we create an *htmlView* variable from the *AlternateView* class, and set it to the *CreateAlternateViewFromString* method that accepts three parameters: (1) the variable that you're creating the view on, (2) the encoding type, which can be null, and (3) the MIME type, which in our case is HTML. Next, we need to add this view to our mail message object as shown below:

```
protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    string bodyPlain = string.Empty;
    MailMessage mailMsg = new MailMessage("webmaster@domain.net", txtTo.Text);
    mailMsg.Subject = "MWD E-Newsletter";
    mailMsg.IsBodyHtml = true;
    bodyHTML = @"";
```

```

AlternateView htmlView =
AlternateView.CreateAlternateViewFromString(bodyHTML, null, "text/html");
mailMsg.AlternateViews.Add(htmlView);
}

```

As you can see from the code above, we use our mail message object, set its property, *AlternateViews*, and then call its *add* method, and pass in our *htmlView* variable.

Creating the SMTP object, setting the port, host and sending the mail

At this point, we have everything we need in preparation of actually sending the email. To send the email, we need to create an smtp object that will let us authenticate to our smtp server, set our port, host, and send our email to our recipients. We do this by adding the following code:

```

protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    string bodyPlain = string.Empty;
    MailMessage mailMsg = new MailMessage("webmaster@domain.net", txtTo.Text);
    mailMsg.Subject = "MWD E-Newsletter";
    mailMsg.IsBodyHtml = true;
    bodyHTML = @"";

AlternateView htmlView =
AlternateView.CreateAlternateViewFromString(bodyHTML, null, "text/html");
mailMsg.AlternateViews.Add(htmlView);

SmtpClient smtp = new SmtpClient();
smtp.Port = 25;
smtp.Host = "mail.domain.net";
smtp.Send(mailMsg);
}

```

As you can see from the code above, we instantiated a new object of the smtp client class by using its constructor method. Next, we set the port to the standard 25 by using the port property of our smtp object. Continuing, we set the host to our mail server by using the host property of our object. Finally, we send our mail message by using the send method and passing in our mail message object as a parameter. It should be noted at this point to check with your host for specifics on how to send emails from a script, specifically if they require additional steps to authenticate to their SMTP server, since the above is a generic way of looking at it.

Before saving the file and closing it, we need to set our label control to a literal text string of “success” and clear out our text field after the email is sent by adding the following code:

```

protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    string bodyPlain = string.Empty;
    MailMessage mailMsg = new MailMessage("webmaster@domain.net", txtTo.Text);
    mailMsg.Subject = "MWD E-Newsletter";
    mailMsg.IsBodyHtml = true;
    bodyHTML = @"";
}

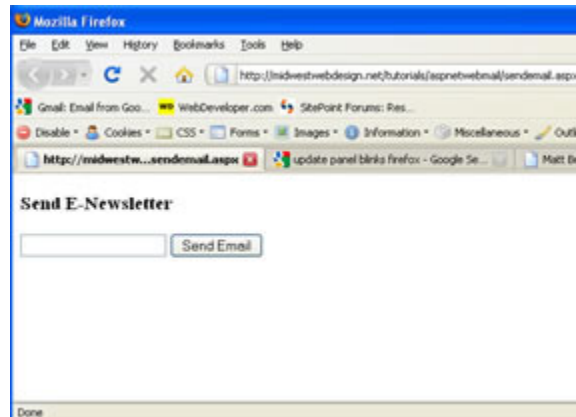
```

```
AlternateView htmlView =  
AlternateView.CreateAlternateViewFromString(bodyHTML, null, "text/html");  
mailMsg.AlternateViews.Add(htmlView);  
  
SmtpClient smtp = new SmtpClient();  
smtp.Port = 25;  
smtp.Host = "mail.domain.net";  
smtp.Send(mailMsg);  
lblMessage.Text = "success";  
txtTo.Text = "";  
}
```

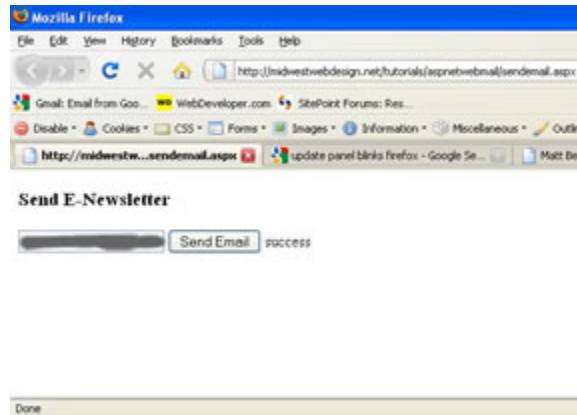
Setting the text property of our label control serves as a visual indicator that our email was sent successfully. Setting the text property of our recipient text box control (our text field) to an empty value, gives a better visual experience to our user allowing them to send the newsletter to another email address.

Sending the email

For the purposes of this article, it's assumed you're sending this email from a remote host. With that, all we need to do is upload the finished file to your remote host, type the full address to request the page, and you should see the following display:



Type the desired recipient in the text field, and after clicking the “Send Email” button, you should see the success message appear after a few seconds:



If everything went right, you should receive an email in a few minutes. If you don't, and you received no errors when executing the script, follow a few simple steps before contacting your host:

- Verify the recipient email address is correct
- Set a break point inside the send mail event handler and run the page from Visual Web Developer Express as shown below

```

17
18 protected void sendEmailBtn_Click(object sender, EventArgs e)
At Default.aspx.cs, line 18 ('SendEmail._default', line 8)
20     string bodyHTML = string.Empty;
21     string bodyPlain = string.Empty;
22     MailMessage mailMsg = new MailMessage("rt

```

The red dot shown in the image above indicates a break point. When clicking the “Send Email” button, if an error occurs in code, control of the page will revert from the browser back to Visual Web Developer Express showing you the error. As a last resort:

- Make sure you're authenticating properly, which may involve asking your host for further details

The results

After sending the email to Outlook, Gmail, and Hotmail, the results were as follows:

- Outlook: looked fine except the header and footer background color does not fill the entire area. In other words, there's some white showing, however, after debugging for awhile, it was decided it was good enough.
- Gmail: looked fine
- Hotmail: looked fine

One thing to note about Gmail and Hotmail is that due to their advertisements, the overall width of the layout may look reduced. There's nothing we can do to solve that issue. Also worth noting, as mentioned before, the images will be blocked initially in all three, unless you change your settings appropriately.

Houston, we have a problem...

At this point, if you refresh your browser, either by clicking the refresh button within the browser or by pressing F5 on your keyboard it would send another email to the recipient. The reason for this is a bit technical but understandable if we take it in steps. First, when we initially clicked the send email button, that caused a post back because the onclick attribute is associated to a server side event, which subsequently sends the email. However, since you're still in the "state" of a post back, refreshing your browser or pressing F5 on your keyboard causes the send mail event to execute again because refreshing the page is considered a "full" post back.

At this point, the only way to prevent the mail from being sent again is to use your mouse and left click in the address bar and press enter on your keyboard to return to an "original state". Obviously, this is an issue. If the person using the script kept hitting the refresh button thinking they would go back to the "original state", they would in fact send the email again to the same person. As a result, we need a way to refresh the page without sending the email again. We have two solutions that work: 1) create a database table with a date time column associated to the recipients email address. Then using Structured Query Language (SQL) we update the date time column with the current date from our C# script, or 2) use Ajax server side controls to control what part of the page we want to refresh without causing a full page post back. Since the first solution will be part two of the series, we'll go with solution two for this article.

Adding the Ajax server side controls

Double click *default.aspx* to open our web form and add the following code to the existing markup:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="default_ajax.aspx.cs" Inherits="SendEmail.default_ajax" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"/>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
        <asp:TextBox ID="txtTo" runat="server"/>
        <asp:Button ID="sendEmailBtn" runat="server" Text="Send Email"
OnClick="sendEmailBtn_Click"/>
        <asp:Label ID="lblMessage" runat="server"/>
      </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```

As you can see from the code above, we added two Ajax server side controls. Briefly, Ajax stands for Asynchronous JavaScript and XML and is a way for developers to instruct a web page when to initiate

between a client or server side request. This becomes very efficient when working with complex web applications and limiting when the application needs to make a server side request.

Continuing, from the above code, we can see that we first added a script manager control. In order to use any Ajax server side control in ASP.NET, you must have one script manager tag. This tag acts as a gate keeper for our page and manages all client side scripts (JavaScript) needed to handle asynchronous calls from a server. Next, we added an update panel control. This control contains two children tags, content template and triggers. In our case, we added a content template tag, which essentially makes any content inside, such as our existing web form controls Ajax enabled. Save your changes, and we'll explain how this solves our problem.

So, here's how the Ajax server side control solves our problem: think of our entire page as a box, then think of our update panel as another smaller box within our larger box. When a visitor clicks the send email button, it causes the update panel to execute the click event associated to the button, which sends the email. After this, when we refresh our page, the only part of our page which refreshes is our update panel, known as a partial page post back. As a result, there's no full page post back, thus, no duplicate email is sent.

It should be noted, this is why we needed to create our web project with .NET version 3.5. By default, when the project is created our web.config file is created with the necessary name spaces and assemblies needed to use the Ajax controls. You can make Ajax controls work with .NET version 2.0, but would've made the article much more difficult to follow at this specific point. Save your file and use *default_ajax.aspx* as a reference if needed.

Providing a web version of the email

One additional detail you can do for users who have difficulty reading the html version in their email client, is to provide a link at the top of the newsletter that will point readers to a web version of the newsletter for easier viewing. If you want to implement this, view *default_webversion.aspx*.

Creating a plain text version of the email

You may be asking the question: what if my users will only accept plain text in emails? Fortunately, there's a way to accommodate this in ASP.NET. Let's create a plain text version in steps as follows using *default_plaintext.aspx* as a reference:

- First, right below our variable that holds our html content, create another variable to hold our plain text content as shown below:

```
protected void sendEmailBtn_Click(object sender, EventArgs e)
{
    string bodyHTML = string.Empty;
    string bodyPlain = string.Empty;
```

As you can from the code above, we create another variable, *bodyPlain*, and set its value to an empty string, which makes the variable read only.

- Second, right after ending our html content, we add our content to our plain text variable as shown below:

```
</body>  
</html>'";
```

```
bodyPlain = @"...plain text version here";
```

As you can see from the code above, we applied the same at (@) sign to our string as we did for our html content. Recall from earlier the at (@) sign is treated as a literal (verbatim) string, allowing C# to read our plain text string exactly as we have it. Continuing, we needed a way to indicate navigation. We simply used hypens for this. Additionally, we need a way to separate our headings logically. The easiest way to do this is to use asterisks before and after each heading to denote a change in the flow of content. You can of course, use any logic or flow process you want. Finally, to account for the images, we provide the absolute link, that way, users can easily click the link if they choose to.

- Third, we need to create the plain text view for our email, which is shown below:

```
AlternateView plainView =  
AlternateView.CreateAlternateViewFromString(bodyPlain, null, "text/plain");
```

As you can see from the code above, we create an *plainView* variable from the *AlternateView* class, and set it to the *CreateAlternateViewFromString* method that accepts three parameters: (1) the variable that you're creating the view on, (2) the encoding type, which can be null, and (3) the MIME type, which in our case is plain text.

- Fourth and finally, we need to add this view to our mail message object as shown below:

```
mailMsg.AlternateViews.Add(plainView);
```

As you can see from the code above, we use our mail message object, set its property, *AlternateViews*, and then call it's *add* method, and pass in our *plainView* variable.

Testing the plain text version

If you want to see the plain text version of your email, comment out the line that adds the html view to the mail message object as shown below:

```
//mailMsg.AlternateViews.Add(htmlView);
```

Save your file and send the mail. You should receive the plain text version only.

Spam

Before finishing the article, it should be mentioned, in most cases, our email shouldn't be treated as spam or be blocked. However, there's no guarantee that wouldn't happen. Depending on the rules set on the firewall or router of the recipients account, our e-newsletter could be blocked or treated as spam. This could be especially true if the receiving email servers only accept emails from trusted senders. If this is the case, the only option you have is to talk with your system administrator to see what options you may have.

Summary

In this article you learned how to take a finished web page and integrate it into an ASP.NET page that's capable of sending an html and plain text newsletter to any specified recipient. Furthermore, you learned the following:

- How to structure the html version for maximized compatibility of email clients
- How to create a send mail event and add the following code inside:
 - Variables to hold html and plain text versions of the email
 - Mail message object used to store who's sending the mail, the recipient, subject and indicating the mail message contains html
 - How to include images in our email without creating additional file size
- How to create an SMTP object and set corresponding values
- How to work with Ajax server side controls to prevent sending multiple emails to the same person

In part two of this series, we'll look at how you can adjust the existing ASP.NET page to send the same email to multiple recipients using a database table, which will involve modifying our existing ASP.NET page and using structured query language (SQL). For now, take the knowledge gained in this article and use it to create any email newsletter of your dreams!

If you have questions, please contact me at the address below:

<http://midwestwebdesign.net/tutorials/contact.aspx>