Sending and Receiving Mail in C#

Sending Mails[1]

This lesson focuses on how to send mail messages in .NET Framework via a SMTP server. It firstly discusses the techniques which .NET Framework provides you to send mail messages. After that, it discusses types available for you when working with SMTP servers. Next, it discusses how to implement these techniques and to send mails from a .NET client.

At the end of this lesson, there is a sample application, Geming Mail+, which is used to send mails from a various SMTP servers. This application is open-source, so you can download its code freely.

Introduction

Simple Mail Transport Protocol or simply SMTP provides a way for applications to connect to the mail server and send mail messages via server's exposed SMTP service.

Before .NET 2.0, you were to access SMTP via classes in System.Web.Mail namespace which resides in System.Web.dll library. With the release of .NET 2.0, System.Web.Mail classes became deprecated and replaced with classes in System.Net.Mail namespace which exists in System.dll library. That means that you still can use classes of System.Web.Mail, however, you will receive warnings indicate that those classes are deprecated and you should use classes from System.Net.Mail namespace.

Type Overview

System.Net.Mail Types

System.Net.Mail namespace includes many types each of which provides a special feature. In fact, the most time you will not need to use all of them or to know them at all. However, being aware of what .NET provides to you for accessing SMTP is better to help you evolving your SMTP client application in many sides and in various degrees. Here are the most common classes of System.Net.Mail:

- **SmtpClient**: One of the essential classes provides you with means of connecting to the SMTP server and sending mail messages. Before starting using this class and send a message, you must initialize server properties like Host, Port, and EnableSsl to allow communicating with the SMTP server. SmtpClient also provides you with some methods like the Send method that sends a specific message synchronously, and SendAsync to send it asynchronously.
- **MailMessage**:The message to be sent using the SmtpClient class. This class exposes many properties specific to the message like To, CC, Bcc, Subject, and Body properties that corresponds to the message fields.
- **MailAddress**: Encapsulates a mail address. Provides the DisplayName and Address properties.
- **MailAddressCollection**: A collection of MailAddress objects. This collection is used inside the MailMessage object in the properties To, CC, and Bcc.
- **Attachment**: Encapsulates an attached file.
- **AttachmentCollection**: A collection of Attachment objects. Used in the MailMessage class in its

1 http://www.codeproject.com/Articles/66257/Sending-Mails-in-NET-Framework.aspx

Attachments property.

System.Web.Mail Types

Besides types in System.Net.Mail, for whose interested in .NET 1.0 and descendent before .NET 2.0, we will cover types of System.Web.Mail briefly. In fact, they are very few types, actually, they are only three classes and three enumerations, and they serve the same as types in System.Net.Mail.

Classes in System.Web.Mail:

- **SmtpMail**: Serves the same as System.Net.Mail.SmtpClient. However, it exposes only a single property SmtpServer. Plus, it exposes methods for sending mail messages.
- **MailMessage**: Encapsulates message related information and data like To, CC, BCC, Subject, and Body fields.
- **MailAttachment**: Encapsulates an attachment. MailMessage exposes a list of MailAttachment objects via its Attachments property.

SMTP Servers

In order to connect to a SMTP server you need to be aware of four things:

- **Server address**: Like smtp.example.com.
- **Port number**: Usually 25, and sometimes 465. Depends on server’s configuration.
- **SSL**: You need to know if the server requires a SSL (Secure Socket Layer) connection or not. To be honest, most servers require SSL connections.
- **Credentials**: You need to know if the server accepts default credentials of the user or requires specific credentials. Credentials are simply the username and password of the user. All e-mail service providers require specific credentials. For example, to connect to your Gmail’s account and send mails via Gmail’s SMTP server, you will need to provide your mail address and password.

The following is a list of some of the major e-mail service providers who provide SMTP services for their clients:

Name	**Server Address**	**Port**	**SSL Required?**
Live	smtp.live.com	25	Yes
Gmail	smtp.gmail.com	465, 25, or 587	Yes
Yahoo!	plus.smtp.mail.yahoo.com	465, 25, or 587	Yes

Implementation

The following is a simple code segment uses classes from System.Net.Mail namespace to send mail messages via GMail's SMTP server.

Do not forget to add a using statement for the **System.Net.Mail** namespace.

```
     // C# Code
    MailMessage msg = new MailMessage();

    // Your mail address and display name.
    // This what will appear on the From field.
    // If you used another credentials to access
    // the SMTP server, the mail message would be
    // sent from the mail specified in the From
    // field on behalf of the real sender.
    msg.From =
        new MailAddress("example@gmail.com", "Example");

    // To addresses
    msg.To.Add("friend_a@example.com");
    msg.To.Add(new MailAddress("friend_b@example.com", "Friend B"));

    // You can specify CC and BCC addresses also

    // Set to high priority
    msg.Priority = MailPriority.High;

    msg.Subject = "Hey, a fabulous site!";

    // You can specify a plain text or HTML contents
    msg.Body =
        "Hello everybody,<br /><br />" +
        "I found an interesting site called <a
       href=\"http://JustLikeAMagic.WordPress.com\">" +
        "Just Like a Magic</a>";

    msg.IsBodyHtml = true;

    // Attaching some data
    msg.Attachments.Add(new Attachment("D:\\Site.lnk")) ;

    // Connecting to the server and configuring it
    SmtpClient client = new SmtpClient();

    client.Host = "smtp.gmail.com";
    client.Port = 578;
    client.EnableSsl = true;
    // The server requires user's credentials
    // not the default credentials

    client.UseDefaultCredentials = false;
    // Provide your credentials
    client.Credentials = new System.Net.NetworkCredential("example@gmail.com",
"buzzwrd");
    client.DeliveryMethod = SmtpDeliveryMethod.Network;


    // Use SendAsync to send the message asynchronously
    client.Send(msg);
```

Receiving Mails[2]

I have just worked on a project where I needed to download e-mails from a POP3 server. I research some of the libraries out there and settled on *OpenPop.NET*. OpenPop.NET has a great set of examples of connecting to a POP3 server, but what I want to show you below are some examples that OpenPop.NET doesn't go over

2 http://www.formatyourbrain.com/openpop-net-helpful-tips/

or elaborate on.

E-mail Addresses

There are four potential e-mail addresses that you can grab.

1. To – who the message is to (multiple entries)
2. From – who sent the message
3. CC – who was carbon copied on the message (multiple entries)
4. BCC – who was blind carbon copied on the message (multiple entries)

The ‘To’, ‘CC’, ‘BCC’ are similar in retrieving the data. The From is separate since it provides more information that the other fields and also that there can’t be more than one sender on the message.

From E-mail Address

First we need to gain access to the header section of the e-mail we are importing.

To gain access to each of these e-mails you have to access them via the RfcMailAddress type. This is easily done as follows.

```
Pop3Client connection = new Pop3Client();
connection.Connect(serverURL, port, ssl);
connection.Authenticate(userName, password);
Message emailMessage = connect.GetMessage(i); // number of the message you are getting
MessageHeader emailHeader = emailMessage.Headers; // grab the headers
```

After you have the connection and the headers stored, you need to gain access to the From object. It’s a part of the RfcMailAddress type.

```
RfcMailAddress emailFrom = emailHeader.From;
```

The *From e-mail address* is simple since we know the e-mail can only be sent from one individual e-mail address. Each of the members of the object are straight forward.

```
string fromAddress = emailFrom.Address;
string fromName = emailFrom.DisplayName;
string fromComplete = emailHeader.From.ToString();
```

You will notice the last line of code. This is an extremely quick way to get both the name and the address in one string variable. If you use that method you will need to split up the data yourself. I prefer the other two calls, that way I can access the data without having to parse any of it.

To, CC, BCC E-mail Addresses

The To, CC, BCC e-mail address are all similar, you won’t be able to use the “From” example when retrieving the data. What works for one of these types will work for the other. For this instance we will say the e-mail that you are retrieving has multiple “To” recipients. This will requiring looping through the data to get each of the e-mail addresses. Before we even loop we need to store the e-mails into a List. I have put all the code together.

```
 List<RfcMailAddress> emailTo = emailHeader.To;

// Loop through the list and concatenate the e-mail addresses
string strAllEmails = "";

foreach(RfcMailAddress tempTo in emailTo)
{
     if(tempTo.HasValidMailAddress)
     {
         strAllEmails += tempTo.MailAddress + ",";
     }
}
```

To accomplish this for the CC and the BCC e-mail addresses you just need to change the variable you are grabbing to be the .Cc or .Bcc.

E-mail Body

Grabbing the body of the e-mail message is a little bit trying. You need to actually grab the MessagePart (in the e-mail address we used the MessageHeader). You will make that call as follows:

```
MessagePart emailMessage = connection.GetMessage(intMessageNumber).MessagePart;
```

Once you have the MessagePart you can then access the following methods .FindFirstPlainTextVersion() and .FindFirstHtmlVersion(), plain text and HTML respectively. But once you have these set up you need to be able to actually see the Body as text (whether it is HTML text or plain text). Below is the final call on how to get the desired body text from the e-mail message.

```
Message emailMessage = connection.GetMessage(intMessageNumber); // get the full message

// Then get the type of body text you desire.
string plainBody = emailMessage.FindFirstPlainTextVersion().GetBodyAsText();
string htmlBody = emailMessage.FindFirstHtmlVersion().GetBodyAsText();
```

Even though we call another method .GetBodyAsText() it still will return the needed value even if the body is HTML (tags and all).

Attachments

We can’t forget about the attachments. What’s a good e-mail without the ability to attach a file?
First you will need to test whether the e-mail has an attachment. If it does we need to set up a situation where we can loop through all the attachments, in case there is more than one. Below is a bit of code that will save the attachments to a folder on the local drive.

```
foreach (MessagePart emailAttachment in emailMessageMain.FindAllAttachments())
{
       //-- Set variables
       string OriginalFileName = emailAttachment.FileName;
       string ContentID = emailAttachment.ContentId; // If this is set then the
       attachment is inline.
string ContentType = emailAttachment.ContentType.MediaType; // type of attachment
pdf, jpg, png, etc.

//-- write the attachment to the disk
File.WriteAllBytes(path + OriginalFileName, emailAttachment.Body); //overwrites
MessagePart.Body with attachment
}
```

The documentation on the OpenPop.NET site is pretty straight forward and can help with anything I didn't cover here.

Warnings

- GMail can be accessed by OpenPOP.NET if you set the right settings in your account setup. The problem is Gmail does things a little different that a normal POP3 server. I would suggest that instead of trying to access gmail as a POP3 server that you access it as an IMAP server (which is has the ability to do). This will save you a lot of grief down the road.
- OpenPop.NET does not handle IMAP servers.