Chapter 15

Mail

E-mail is an extremely popular communication mechanism. Unlike the telephone and most other ways of communicating, e-mail is easy for applications to use. For a developer, e-mail is an effective way to allow an application to send files or reports to users and to notify users of problems or events.

The .NET Framework version 2.0 adds the *System.Net.Mail* namespace, which provides classes that enable you to easily create and transmit e-mail messages. Messages can include plain text, HTML, and file attachments. At a high level, sending e-mail has two steps: creating the mail message, and then sending the message to an SMTP (Simple Mail Transfer Protocol) server. Lesson 1 covers how to create a message, and Lesson 2 covers how to send the message.

NOTE .NET 2.0

The *System.Net.Mail* namespace is new in .NET Framework version 2.0.

Exam objectives in this chapter:

- Send electronic mail to a Simple Mail Transfer Protocol (SMTP) server for delivery from a .NET Framework application. (Refer *System.Net.Mail* namespace)
  - *MailMessage* class
  - *MailAddress* class and *MailAddressCollection* class
  - *SmtpClient* class, *SmtpPermission* class, and *SmtpPermissionAttribute* class
  - *Attachment class*, *AttachmentBase* class, and *AttachmentCollection* class
  - *SmtpException* class, *SmtpFailedRecipientException* class, and *SmtpFailedRecipientsException* class
  - *SendCompletedEventHandler* delegate
  - *LinkedResource* class and *LinkedResourceCollection* class
  - *AlternateView* class and *AlternateViewCollection* class

Lessons in this chapter:

Before You Begin

To complete the lessons in this chapter, you should be familiar with Microsoft Visual Basic or C# and be comfortable with the following tasks:

- Create a console application in Microsoft Visual Studio using Visual Basic or C#.
- Add references to system class libraries to a project.

Lesson 1: Creating a Mail Message

Creating an e-mail message can be simple or complex. At its simplest, an e-mail message has a sender, recipient, subject, and body. These simple messages can be created with a single line of code using the .NET Framework. At their most complex, e-mail messages can have custom encoding types, multiple views for plain text and HTML, attachments, and images embedded within HTML.

After this lesson, you will be able to:

- Describe the process of creating and sending an e-mail.
- Create a *MailMessage* object.
- Attach one or more files to an e-mail message.
- Create HTML e-mails with or without pictures.
- Catch and respond to different exceptions that might be thrown while creating a message.

Estimated lesson time: 30 minutes

The Process of Creating and Sending an E-mail Message

To create and send an e-mail message, follow these steps:

1. Create a *MailMessage* object. *MailMessage* and other mail-related classes are in the *System.Net.Mail* namespace.
2. If you did not specify the recipients in the *MailMessage* constructor, add them to the *MailMessage* object.
3. If you need to provide multiple views (such as plain text and HTML), create *AlternateView* objects and add them to the *MailMessage* object.
4. If necessary, create one or more *Attachment* objects and add them to the *MailMessage* object.
5. Create an *SmtpClient* object, and specify the SMTP server.
6. If the SMTP server requires clients to authenticate, add credentials to the *SmtpClient* object.
7. Pass your *MailMessage* object to the *SmtpClient.Send* method. Alternatively, you can use *SmtpClient.SendAsync* to send the message asynchronously.

Steps 5 through 7 are described in detail in Lesson 2.

How to Create a *MailMessage* Object

The *MailMessage* object has four different constructors that allow you to create a blank *MailMessage*; specify both the sender and recipient; or specify the sender, recipient, subject, and message body. If you are creating a simple message with a single recipient, you can do the bulk of the work in the *MailMessage* constructor:

```
' VB
Dim m As MailMessage = New MailMessage _
    ("jane@contoso.com", _
     "ben@contoso.com", _
     "Quarterly data report.", _
     "See the attached spreadsheet.")

// C#
MailMessage m = new MailMessage
   ("jane@contoso.com",
    "ben@contoso.com",
    "Quarterly data report.",
    "See the attached spreadsheet.");
```

NOTE Sending quick messages

You can also use an overload of the *SmtpClient.Send* method to send an e-mail without creating a *MailMessage* object. *SmtpClient* is described in Lesson 2.

You can specify the sender and the recipient as either a string or *MailAddress* object. The *MailAddress* object allows you to specify an e-mail address, a display name, and an encoding type, as the following code sample demonstrates:

```
' VB
Dim m As MailMessage = New MailMessage _
    (New MailAddress("lance@contoso.com", "Lance Tucker"), _
     New MailAddress("ben@contoso.com", "Ben Miller"))

// C#
MailMessage m = new MailMessage
    (new MailAddress("lance@contoso.com", "Lance Tucker"),
     new MailAddress("ben@contoso.com", "Ben Miller"));
```

NOTE Encoding types

Specifying the encoding type for e-mail addresses is rarely necessary.

If you need to specify multiple recipients, use the blank *MailMessage* constructor. Then add *MailAddress* objects to the *MailMessage.To* property (which is of the

MailAddressCollection type), and specify *MailMessage.From*, *MailMessage.Subject*, and *MailMessage.Body*:

```
' VB
Dim m As MailMessage = New MailMessage()
m.From = New MailAddress("lance@contoso.com", "Lance Tucker")
m.To.Add(New MailAddress("james@contoso.com", "James van Eaton"))
m.To.Add(New MailAddress("ben@contoso.com", "Ben Miller"))
m.To.Add(New MailAddress("burke@contoso.com", "Burke Fewel"))
m.Subject = "Quarterly data report."
m.Body = "See the attached spreadsheet."

// C#
MailMessage m = new MailMessage();
m.From = new MailAddress("lance@contoso.com", "Lance Tucker");
m.To.Add(new MailAddress("james@contoso.com", "James van Eaton"));
m.To.Add(new MailAddress("ben@contoso.com", "Ben Miller"));
m.To.Add(new MailAddress("burke@contoso.com", "Burke Fewel"));
m.Subject = "Quarterly data report.";
m.Body = "See the attached spreadsheet.";
```

Additionally, you can add recipients to the *MailMessage.Cc* and *MailMessage.Bcc* properties in exactly the same way as you would add recipients to *MailMessage.From*. Recipients specified with *MailMessage.Cc* will receive the message, and their names will show up on the CC line of the e-mail, which is visible to all recipients. Recipients specified with *MailMessage.Bcc* will receive the message, but their names will not be visible to other recipients. BCC stands for "blind carbon copy," a term that originated when people made duplicates of typed paper memos using carbon paper.

NOTE The risk of BCC

Instead of using BCC, you should send a separate copy of your message to each recipient that you want to receive a blind copy. The problem with BCC is that spam filters frequently block messages that do not have the recipient's e-mail address in the From header. Therefore, if you use BCC, the message will very likely be filtered.

MailMessage has the following less frequently used properties:

- ***DeliveryNotificationOptions*** Instructs the SMTP server to send a message to the address specified in *MailMessage.From* if a message is delayed, fails, or is successfully delivered or relayed to another server. The enumeration is of type *DeliveryNotificationOptions*, and the values are *OnSuccess*, *OnFailure*, *Delay*, *None*, and *Never*.
- ***ReplyTo*** The e-mail address that replies will be sent to. Because the .NET Framework does not act as an e-mail client, and therefore your application will not

typically be receiving e-mail, in most cases you should simply set *MailMessage.From* to the address that should receive replies instead of using *ReplyTo*.

- ***Priority*** The priority of the message. This does not in any way affect how the .NET Framework or the mail server handles the message. However, the priority might be visible in the recipient's e-mail client. The priority is also useful for filtering automatically generated e-mail based on the priority of the event that initiated the e-mail. This enumeration is of type *MailPriority* and can have values of *Normal*, *High*, and *Low*.

How to Attach Files

To attach a file, add it to the *MailMessage.Attachments AttachmentCollection* by calling the *MailMessage.Attachments.Add* method. The simplest way to add a file is to specify the file name:

```
' VB
Dim m As MailMessage = New MailMessage()
m.Attachments.Add(New Attachment("C:\boot.ini"))

// C#
MailMessage m = new MailMessage();
m.Attachments.Add(new Attachment(@"C:\boot.ini"));
```

You can also specify a MIME (Multipurpose Internet Mail Extensions) content type using the *System.Net.Mime.MediaTypeNames* enumeration. There are special MIME types for text and images, but you will typically specify *MediaTypeNames.Application.Octet*. The following code sample (which requires *System.IO* and *System.Net.Mime* in addition to *System.Net.Mail*) demonstrates how to use a *Stream* as a file attachment and how to specify the MIME type:

```
' VB
Dim m As MailMessage = New MailMessage()
Dim sr As Stream = New FileStream("C:\Boot.ini", FileMode.Open, FileAccess.Read)
m.Attachments.Add(New Attachment(sr, "myfile.txt", MediaTypeNames.Application.Octet))

// C#
MailMessage m = new MailMessage();
Stream sr = new FileStream(@"C:\Boot.ini", FileMode.Open, FileAccess.Read);
m.Attachments.Add(new Attachment(sr, "myfile.txt", MediaTypeNames.Application.Octet));
```

As the previous example demonstrates, you should specify a filename when creating an attachment from a *Stream* object. Otherwise, the attachment will be labeled with a generic name such as "application_octect-stream.dat". Because the file extension would be incorrect, users would not be able to easily open the attachment in the correct application.

How to Create HTML E-mails

To create an HTML e-mail message, supply HTML-tagged content for *MailMessage.Body* and set the *MailMessage.IsBodyHtml* attribute to True, as the following code sample demonstrates:

```
' VB
Dim m As MailMessage = New MailMessage
m.From = New MailAddress("lance@contoso.com", "Lance Tucker")
m.To.Add(New MailAddress("burke@contoso.com", "Burke Fewel"))
m.Subject = "Testing HTML"

' Specify an HTML message body
m.Body = "<html><body><h1>My Message</h1><br>This is an HTML message.</body></html>"
m.IsBodyHtml = True

' Send the message
Dim client As SmtpClient = New SmtpClient("smtp.contoso.com")
client.Send(m)

// C#
MailMessage m = new MailMessage();
m.From = new MailAddress("lance@contoso.com", "Lance Tucker");
m.To.Add(new MailAddress("burke@contoso.com", "Burke Fewel"));
m.Subject = "Testing HTML";

// Specify an HTML message body
m.Body = "<html><body><h1>My Message</h1><br>This is an HTML message.</body></html>";
m.IsBodyHtml = true;

// Send the message
SmtpClient client = new SmtpClient("smtp.contoso.com");
client.Send(m);
```

MailMessage.Subject is always plain text. You can define *MailMessage.Body* just like any HTML Web page. However, most e-mail clients will ignore the <head> section, will ignore any client-side scripts, and will not automatically download images from Web sites.

To embed images into an HTML message so that they appear when the user clicks the message (without requiring the user to explicitly choose to download images), use the *AlternateView* and *LinkedResource* classes. First, create an HTML message using *AlternateView*, and then add images using *LinkedResource*, as the following sample code demonstrates:

```
' VB
' Create the HTML message body
' Reference embedded images using the content ID
Dim htmlBody As String = "<html><body><h1>Picture</h1><br>" + _
    "<img src=""cid:Pic1""></body></html>"
```

```
Dim avHtml As AlternateView = AlternateView.CreateAlternateViewFromString _
    (htmlBody, Nothing, MediaTypeNames.Text.Html)

' Create a LinkedResource object for each embedded image
Dim pic1 As LinkedResource = New LinkedResource("pic.jpg", MediaTypeNames.Image.Jpeg)
pic1.ContentId = "Pic1"
avHtml.LinkedResources.Add(pic1)

' Create an alternate view for unsupported clients
Dim textBody As String = "You must use an e-mail client that supports HTML messages"
Dim avText As AlternateView = AlternateView.CreateAlternateViewFromString _
    (textBody, Nothing, MediaTypeNames.Text.Plain)

' Add the alternate views instead of using MailMessage.Body
Dim m As MailMessage = New MailMessage
m.AlternateViews.Add(avHtml)
m.AlternateViews.Add(avText)

' Address and send the message
m.From = New MailAddress("lance@contoso.com", "Lance Tucker")
m.To.Add(New MailAddress("james@contoso.com", "James van Eaton"))
m.Subject = "A picture using alternate views"
Dim client As SmtpClient = New SmtpClient("smtp.contoso.com")
client.Send(m)

// C#
// Create the HTML message body
// Reference embedded images using the content ID
string htmlBody = "<html><body><h1>Picture</h1><br><img src=\"cid:Pic1\"></body></html>";
AlternateView avHtml = AlternateView.CreateAlternateViewFromString
    (htmlBody, null, MediaTypeNames.Text.Html);

// Create a LinkedResource object for each embedded image
LinkedResource pic1 = new LinkedResource("pic.jpg", MediaTypeNames.Image.Jpeg);
pic1.ContentId = "Pic1";
avHtml.LinkedResources.Add(pic1);

// Create an alternate view for unsupported clients
string textBody = "You must use an e-mail client that supports HTML messages";
AlternateView avText = AlternateView.CreateAlternateViewFromString
    (textBody, null, MediaTypeNames.Text.Plain);

// Add the alternate views instead of using MailMessage.Body
MailMessage m = new MailMessage();
m.AlternateViews.Add(avHtml);
m.AlternateViews.Add(avText);

// Address and send the message
m.From = new MailAddress("lance@contoso.com", "Lance Tucker");
m.To.Add(new MailAddress("james@contoso.com", "James van Eaton"));
m.Subject = "A picture using alternate views";
SmtpClient client = new SmtpClient("smtp.contoso.com");
client.Send(m);
```

This code produces the HTML message shown in Figure 15-1 (assuming pic.jpg was a picture of a cute puppy stored in the same folder as the assembly).

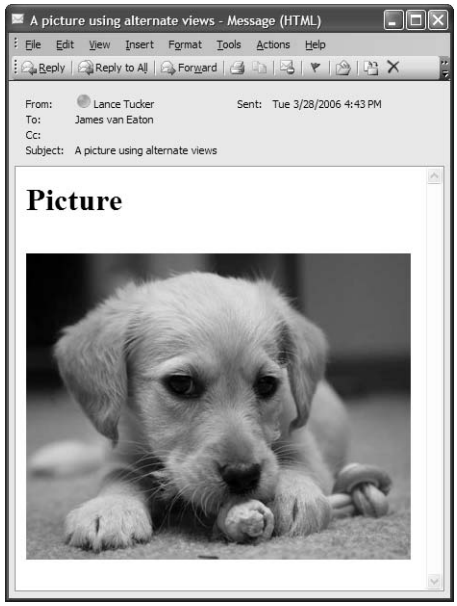

Figure 15-1 Using *AlternateView* and *LinkedResource* to embed images in emails

To reference images attached as linked resources from your HTML message body, use "cid:*contentID*" in the <img> tag. Then specify the same name for the *LinkedResource.ContentID* property. In the previous example, the body of the HTML message contained the tag <img src=\"cid:Pic1\">, and *LinkedResource.ContentID* was set to "Pic1". Each *LinkedResource* should have a unique *ContentID*.

Lab: Create a *MailMessage* Object

In this lab, you create a *MailMessage* object based on user input into a Microsoft Windows Forms application that is provided for you. If you encounter a problem completing an exercise, the completed projects are available on the companion CD in the Code folder.

1. Use Windows Explorer to copy either the C# or Visual Basic version of the Chapter15\Lesson1-Exercise1 folder from the companion CD to your My Documents\Visual Studio Projects\ folder. Then open the solution.
2. Add the *System.Net.Mail* namespace to your code.

3. The runtime calls the *sendButton_Click* method when you click the Send button in the form. You now need to add logic to create a *MailMessage* object using user input, as the following code demonstrates:

```
' VB
' Create a MailMessage object
Dim mm As MailMessage = New MailMessage

// C#
// Create a MailMessage object
MailMessage mm = new MailMessage();
```

4. Now, specify the sender and recipient. Note that it's okay if the user hasn't typed display names for the To and From addresses, because the *MailAddress* constructor can accept a Null value for the second string. This code demonstrates how to do this:

```
' VB
' Define the sender and recipient
mm.From = New MailAddress(fromEmailAddress.Text, fromDisplayName.Text)
mm.To.Add(New MailAddress(toEmailAddress.Text, toDisplayName.Text))

// C#
// Define the sender and recipient
mm.From = new MailAddress(fromEmailAddress.Text, fromDisplayName.Text);
mm.To.Add(new MailAddress(toEmailAddress.Text, toDisplayName.Text));
```

5. Next, define the subject and body, as the following code demonstrates:

```
' VB
' Define the subject and body
mm.Subject = subjectTextBox.Text
mm.Body = bodyTextBox.Text
mm.IsBodyHtml = htmlRadioButton.Checked

// C#
// Define the subject and body
mm.Subject = subjectTextBox.Text;
mm.Body = bodyTextBox.Text;
mm.IsBodyHtml = htmlRadioButton.Checked;
```

6. Build and run your project in Debug mode. Without typing anything, click the Send button. Notice that the runtime throws an *ArgumentException*. Create a Try/Catch block around your code to catch an *ArgumentException* and display a message to the user, as the following code demonstrates:

```
' VB
Try
    ' Method logic omitted
Catch ex As ArgumentException
```

```
    MessageBox.Show("You must type from and to e-mail addresses", _
        "Invalid input", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

// C#
try
{
    // Method logic omitted
}
catch (ArgumentException)
{
    MessageBox.Show("You must type from and to e-mail addresses",
        "Invalid input", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

NOTE Why not manually validate input?

In Chapter 3, "Searching, Modifying, and Encoding Text," you learned how to use regular expressions to validate user input. There are even specific examples in Chapter 3 for validating e-mail addresses. Although you could use regular expressions to validate input in this example, it would be redundant. The *MailMessage* object includes logic for validating input, as the next step demonstrates. In some circumstances, security might be important enough to justify having two levels of validation. In this example, the additional complexity and increased chance of introducing a bug are not justified.

7. Build and run your project in Debug mode again. This time, type "hello" into the To and From e-mail address boxes, and then click the Send button. Notice that the runtime throws a *FormatException* if the user provides an e-mail address in an invalid format. Add a *Catch* block to catch a *FormatException* and display a message to the user, as the following code demonstrates:

```
' VB
Catch ex As FormatException
    MessageBox.Show("You must provide valid from and to e-mail addresses", _
        "Invalid input", MessageBoxButtons.OK, MessageBoxIcon.Error)

// C#
catch (FormatException)
{
    MessageBox.Show("You must provide valid from and to e-mail addresses",
        "Invalid input", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

8. Build and run your project. Populate the form's text boxes with valid data, and click Send. Nothing will happen yet because you haven't written code to send the message. You will complete this application in Lesson 2.

Lesson Summary

- To send an e-mail message, create a *MailMessage* object, specify the sender, subject, and body; and add recipients, alternate views, and file attachments. Then create an instance of the *SmtpClient* class, and call *SmtpClient.Send* or *SmtpClient.SendAsync* (covered in Lesson 2).
- The *MailMessage* object includes constructors that allow you to create simple messages with a single line of code. More complex messages and messages with multiple recipients will require adding more code.
- To attach a file, create an instance of the *Attachment* class and add it to the *MailMessage.Attachments* collection.
- Creating an HTML message without images is as simple as specifying HTML in the body and setting *MailMessage.IsBodyHtml* to True. If you need to include images in the message, create an *AlternateView* object and a *LinkedResource* object for each image.

Lesson Review

You can use the following questions to test your knowledge of the information in Lesson 1, "Creating a Mail Message." The questions are also available on the companion CD if you prefer to review them in electronic form.

NOTE Answers

Answers to these questions and explanations of why each answer choice is right or wrong are located in the "Answers" section at the end of the book.

---

1. From which of the following sources can you attach a file to an e-mail message? (Choose all that apply.)
   - **A.** Local file system
   - **B.** *Stream*
   - **C.** Web site
   - **D.** An incoming e-mail message
2. Which of the following are required to send an HTML e-mail message? (Choose all that apply.)
   - **A.** Set *MailMessage.Body* to an HTML message.
   - **B.** Set *MailMessage.Head* to an HTML header.

C. Set *MailMessage.IsBodyHtml* to True.

D. Set *MailMessage.Subject* to an HTML message.

3. Which of the following HTML tags correctly references the following linked resource as an image?

```
' VB
' Create a LinkedResource object for each embedded image
Dim lr As LinkedResource = New LinkedResource("myPic.jpg", MediaTypeNames.Image.Jpeg)
lr.ContentId = "myPic"

// C#
// Create a LinkedResource object for each embedded image
LinkedResource lr = new LinkedResource("myPic.jpg", MediaTypeNames.Image.Jpeg);
lr.ContentId = "myPic";
```

A. `<img src="cid:myPic">`

B. `<img src="cid:myPic.jpg">`

C. `<img src="myPic">`

D. `<img src="myPic.jpg">`

4. You want to send an HTML message that is also viewable in clients that do not support HTML. Which class should you use?

A. *LinkedResource*

B. *Attachment*

C. *AlternateView*

D. *SmtpClient*

Lesson 2: Sending Mail

Often, sending an e-mail message is simple and requires only two lines of code. However, as with any network communication, it can become much more complex. If the server is unresponsive, you need to allow users to decide whether to wait for or cancel the message transfer. Some servers require valid user credentials, so you might need to provide a username and password. When possible, you should enable the Secure Sockets Layer (SSL) to encrypt the message in transit and reduce your security risks.

After this lesson, you will be able to:

- Configure an SMTP server, and send an e-mail message.
- Provide a username and password when required to authenticate to an SMTP server.
- Enable SSL to encrypt SMTP communications.
- Send a message asynchronously to allow your application to respond to the user or perform other tasks while an e-mail is being sent.
- Catch and respond to different exceptions that might be thrown while sending a message.

Estimated lesson time: 30 minutes

How to Send a Message

Once you create a message, you need to send it through an SMTP (Simple Message Transfer Protocol) server, which in turn will forward it to the recipient. In the .NET Framework, the *SmtpClient* class represents the SMTP server. Most of the time, sending a message is as simple as this sample code (where "smtp.contoso.com" is the name of the local SMTP server):

```
' VB
Dim m As MailMessage = New MailMessage _
    ("jane@contoso.com", _
     "ben@contoso.com", _
     "Quarterly data report.", _
     "See the attached spreadsheet.")
Dim client As SmtpClient = New SmtpClient("smtp.contoso.com")
client.Send(m)

// C#
MailMessage m = new MailMessage
   ("jane@northrup.org",
    "ben@northrup.org",
    "Quarterly data report.",
```

```
    "See the attached spreadsheet.");
SmtpClient client = new SmtpClient("smtp.contoso.com");
client.Send(m);
```

To send a message, call *SmtpClient.Send*.

NOTE The *SmtpClient.PickupDirectoryLocation* property

The *SmtpClient.PickupDirectoryLocation* property is intended for Web applications. To send messages through an IIS-based SMTP server, specify the host name of the IIS server just like you would any other SMTP server. To specify the local computer, use "localhost" or "127.0.0.1".

Real World

Tony Northrup

Years ago, I used to recommend that everyone install IIS, enable the SMTP service on the computer that would be running their application, and send messages through the local SMTP server. This process is much more efficient than using a remote SMTP server because the local IIS SMTP server will directly contact each recipient's SMTP servers to send the message. Additionally, you don't have to deal with network connections.

However, this approach became unreliable as organizations battled against spam. Organizations now block incoming messages from large blocks of IP addresses. Even if you are not a spammer, messages sent from a local SMTP server might be rejected if your IP address appears on one of these lists. To make it more complicated, you might be able to send messages to most domains but be rejected by others.

Ultimately, the job of managing SMTP servers is now too complex to have every user running their own mail server. As a result, I now recommend people use their ISP's or their organization's mail server for outgoing messages.

How to Handle Mail Exceptions

Many things can go wrong when sending e-mail. For example, the mail server might not be available, the server might reject your user credentials, or the server might determine that one of your recipients is invalid. In each of these circumstances, the runtime will throw an exception, and your application must be prepared to catch the exception.

When you send a message, you should always be prepared to catch an *SmtpException*. Messages are frequently rejected because the SMTP server cannot be found, the server identified the message as spam, or the server requires a username and password.

You should also catch *SmtpFailedRecipientException*, which the runtime will throw if the SMTP server rejects a recipient e-mail address. SMTP servers reject only local recipients. In other words, if you are sending a message to tony@contoso.com and are using Contoso's SMTP server, the server is likely to reject the message if tony@contoso.com is not a valid address (causing the runtime to throw a *SmtpFailedRecipientException*). However, if you are sending a message to the same invalid address using Fabrikam's SMTP server, the SMTP server will not be able to identify that the e-mail address is invalid. Therefore, the runtime will not throw an exception.

Table 15-1 summarizes these and other exceptions that the runtime might throw.

Table 15-1 Exceptions that Can Occur when Sending Mail

Situation	Exception	Synchronous or Asynchronous
You did not define the server hostname.	*InvalidOperationException*	Both
The server hostname could not be found.	*SmtpException* with an inner *WebException*	Both
You are sending a message to a recipient at your local mail server, but the recipient does not have a mailbox.	*SmtpFailedRecipient-Exception*	Synchronous
You are not a valid user, or other message transmission problems.	*SmtpException*	Both

When using *SmtpClient.SendAsync*, invalid recipients and several other events do not result in an exception. Instead, the runtime will call the *SmtpClient.SendCompleted* event. If *AsyncCompletedEventArgs.Error* is not Null, an error occurred.

How to Configure Credentials

To reduce spam, all SMTP servers should reject messages from unauthorized users when the message recipients are not hosted by the SMTP server. SMTP servers

provided by ISPs typically determine whether a user is authorized based on the user's IP address; if you are part of the ISP's network, you are allowed to use the SMTP server.

Other SMTP servers (and some ISP SMTP servers) require users to provide a valid username and password. To use the default network credentials, set *SmtpClient.UseDefaultCredentials* to True. Alternatively, you can set *SmtpClient.Credentials* to *CredentialCache.DefaultNetworkCredentials* (in the *System.Net* namespace), as the following code demonstrates:

```
' VB
Dim client As SmtpClient = New SmtpClient("smtp.contoso.com")
client.Credentials = CredentialCache.DefaultNetworkCredentials

// C#
SmtpClient client = new SmtpClient("smtp.contoso.com");
client.Credentials = CredentialCache.DefaultNetworkCredentials;
```

To specify the username and password, create an instance of the *System.Net.NetworkCredential* class and use it to define *SmtpClient.Credentials*. The following example shows hard-coded credentials; however, you should always prompt the user for credentials:

```
' VB
Dim client As SmtpClient = New SmtpClient("smtp.contoso.com")
client.Credentials = New NetworkCredential("user", "password")

// C#
SmtpClient client = new SmtpClient("smtp.contoso.com");
client.Credentials = new NetworkCredential("user", "password");
```

How to Configure SSL

Another important security-related property is *SmtpClient.EnableSsl*. When you set this value to True, the runtime will encrypt the SMTP communications using SSL. Not all SMTP servers support SSL, but you should always enable this property if support is available.

NOTE .NET 2.0

SmtpClient.EnableSsl is new in .NET 2.0.

How to Send a Message Asynchronously

Sending an e-mail message often takes less than a second. Other times, however, the SMTP server might be slow or completely unresponsive, causing your application to wait for the value specified by *SmtpClient.Timeout*. While your application waits for

the SMTP server (up to 100 seconds by default), your application will be unresponsive and the cursor will change to an hourglass. Users don't have much patience for unresponsive applications, and there is a good chance they will terminate your application.

Fortunately, you can send e-mails asynchronously to enable your application to respond to the user while you wait for the message to be sent. You can even give the user the opportunity to cancel the e-mail transmission. To send a message asynchronously, perform these tasks:

1. Create a method to respond to the *SmtpClient.SendCompleted* event. This method needs to determine whether the transmission was successful, unsuccessful, or cancelled.
2. Add your event handler to *SmtpClient.SendCompleted*.
3. Call *SmtpClient.SendAsync*.
4. Optionally, provide the user the opportunity to cancel the e-mail by calling the *SmtpClient.SendAsyncCancel* method.

For example, the following method responds to an *SmtpClient.SendCompleted* event (it requires the *System.ComponentModel* namespace):

```
' VB
Sub sc_SendCompleted(ByVal sender As Object, ByVal e As AsyncCompletedEventArgs)
    If e.Cancelled Then
        Console.WriteLine("Message cancelled")
    Else
        If Not (e.Error Is Nothing) Then
            Console.WriteLine("Error: " + e.Error.ToString)
        Else
            Console.WriteLine("Message sent")
        End If
    End If
End Sub

// C#
void sc_SendCompleted(object sender, AsyncCompletedEventArgs e)
{
    if (e.Cancelled)
        Console.WriteLine("Message cancelled");
    else if (e.Error != null)
        Console.WriteLine("Error: " + e.Error.ToString());
    else
        Console.WriteLine("Message sent");
}
```

And the following code creates the *SmtpClient* object, adds the event handler, calls the asynchronous send, and then immediately cancels the send. Naturally, in real code, you would wait for the user to initiate a cancellation. This code assumes a *MailMessage* object named *mm* already exists.

```
' VB
SmtpClient sc = new SmtpClient("server_name");

' Add the event handler
AddHandler sc.SendCompleted, AddressOf sc_SendCompleted

' Send the message asynchronously
sc.SendAsync(mm, Nothing)

' Cancel the send
sc.SendAsyncCancel()


// C#
sc = new SmtpClient("server_name");

// Add the event handler
sc.SendCompleted += new SendCompletedEventHandler(sc_SendCompleted);

// Send the message asynchronously
sc.SendAsync(mm, null);

// Cancel the send
sc.SendAsyncCancel();
```

SmtpClient.SendAsync accepts two parameters: the *MailMessage* object to be sent and a generic *Object*. You can specify Null or any other object for the second parameter; it is strictly for your own use. The .NET Framework simply passes it to the event handler. If you were sending multiple messages asynchronously, you could use the second parameter to keep track of which message generated the event.

Lab: Send an E-mail Message

In this lab, you complete the application you created in Lesson 1 by sending the e-mail message. If you encounter a problem completing an exercise, the completed projects are available on the companion CD in the Code folder.

▶ Exercise 1: Create an *SmtpClient* Object to Send a *MailMessage* Object

In this exercise, you extend an existing application to create an *SmtpClient* object and send an e-mail message.

1. Use Windows Explorer to copy either the C# or Visual Basic version of the Chapter 15\Lesson2-Exercise1 folder from the companion CD to your My Documents\ Visual Studio Projects\ folder. Then open the solution. Alternatively, you can continue working with the application you created in Lesson 1, Exercise 1, adding new code right after the mail message definitions.
2. Add the *System.Net* namespace to your code. (You will need the *System.Net.NetworkCredential* class.)
3. Write code to create an instance of *SmtpClient*, enable SSL if required, and configure credentials if required. The following code demonstrates this:

```
' VB
' Configure the mail server
Dim sc As SmtpClient = New SmtpClient(serverTextBox.Text)
sc.EnableSsl = sslCheckBox.Checked
If Not String.IsNullOrEmpty(usernameTextBox.Text) Then
    sc.Credentials = New NetworkCredential(usernameTextBox.Text, passwordTextBox.Text)
End If

// C#
// Configure the mail server
SmtpClient sc = new SmtpClient(serverTextBox.Text);
sc.EnableSsl = sslCheckBox.Checked;
if (!String.IsNullOrEmpty(usernameTextBox.Text))
    sc.Credentials = new NetworkCredential(usernameTextBox.Text, passwordTextBox.Text);
```

4. Write code to send the e-mail message, and notify the user that the message was sent successfully, as the following code demonstrates:

```
' VB
' Send the message and notify the user of success
sc.Send(mm)
MessageBox.Show("Message sent successfully. ", _
    "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

// C#
// Send the message and notify the user of success
sc.Send(mm);
MessageBox.Show("Message sent successfully.",
    "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

5. Build and run your project in Debug mode. Type e-mail addresses in the From and To boxes, and then click the Send button without typing the server name. Notice that the runtime throws an *InvalidOperationException*. Create a *Catch* block to catch an *InvalidOperationException*, and display a message to the user, as the following code demonstrates:

```
' VB
Catch ex As InvalidOperationException
```

```
    MessageBox.Show("Please provide a server name.", "No SMTP server provided", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)

// C#
catch (InvalidOperationException)
{
    MessageBox.Show("Please provide a server name.",
        "No SMTP server provided", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

6. Re-run your project. This time, provide an invalid server name, and then click Send. Notice that the runtime throws an *SmtpException*. Using the debugger, examine the exception. The *SmtpException* object doesn't provide a useful description of the problem. However, the inner exception object is a *WebException* instance, which describes the problem accurately as an invalid DNS name. Write code to catch *SmtpException* and display the message from the base exception type, as the following sample demonstrates:

```
' VB
Catch ex As SmtpException
    ' Invalid hostnames result in a WebException InnerException that provides
    ' a more descriptive error, so get the base exception
    Dim inner As Exception = ex.GetBaseException
    MessageBox.Show("Could not send message: " + inner.Message, _
        "Problem sending message", MessageBoxButtons.OK, MessageBoxIcon.Error)

// C#
catch (SmtpException ex)
{
    // Invalid hostnames result in a WebException InnerException that provides
    // a more descriptive error, so get the base exception
    Exception inner = ex.GetBaseException();
    MessageBox.Show("Could not send message: " + inner.Message,
        "Problem sending message", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

7. Re-run your project. This time, type your SMTP server's hostname. In the To box, provide an invalid e-mail address in your local domain. Then click Send. The runtime should throw an *SmtpFailedRecipientException* (though SMTP server behavior can vary). Write code to catch *SmtpFailedRecipientException* and display a message to the user:

```
' VB
Catch ex As SmtpFailedRecipientException
    MessageBox.Show("The mail server says that there is no mailbox for " + _
        toEmailAddress.Text + ".", "Invalid recipient", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
```

```
// C#
catch (SmtpFailedRecipientException)
{
    MessageBox.Show("The mail server says that there is no mailbox for " +
        toEmailAddress.Text + ".", "Invalid recipient",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

8. Finally, re-run your project and send yourself an e-mail message to verify that your application works properly.

▶ Exercise 2: Send an E-mail Asynchronously

In this exercise, you modify the application you created earlier to allow the user to cancel the message before the transaction is completed. To do this, you change the *SmtpClient.Send* method to *SmtpClient.SendAsync*, change the Send button to Cancel while a message is being sent, and respond to a user clicking the Cancel button.

1. Use Windows Explorer to copy either the C# or Visual Basic version of the Chapter 15\ Lesson2-Exercise 2 folder from the companion CD to your My Documents\Visual Studio Projects\ folder. Then open the solution. Alternatively, you can continue working with the application you created in the previous exercise.
2. First, comment out the existing *SmtpClient.Send* line.
3. You need to respond after the message is sent, so add an event handler to the *SmtpClient.SendCompleted* event. Then call *SmtpClient.SendAsync*, and pass the *MailMessage* object. Remove the code that displays a message box indicating that the message was transmitted successfully, because the runtime will immediately continue processing and will not wait for the message to be successfully transmitted. The following code demonstrates this:

```
' VB
' Send the message and notify the user of success
' sc.Send(mm)
AddHandler sc.SendCompleted, AddressOf sc_SendCompleted
sc.SendAsync(mm, Nothing)

// C#
// Send the message and notify the user of success
//               sc.Send(mm);
sc.SendCompleted += new SendCompletedEventHandler(sc_SendCompleted);
sc.SendAsync(mm, null);
```

4. To allow access to the *SmtpClient* variable from more than one method, move the variable declaration to the class level. You still need to define the variable by setting the value equal to *serverTextBox.Text* in the *sendButton_Click* method, however.

5. After you start sending the message, you need to give the user an opportunity to cancel the transmission. You could add a second button labeled Cancel, or you could change the Send button to Cancel. Either way, if the user clicks Cancel, you need to call *SmtpClient.SendAsyncCancel*. The following code demonstrates how to do this by adding an *If* statement to the *sendButton_Click* method that determines whether the user has clicked the button while in a Send or Cancel state:

```
' VB
If sendButton.Text = "Send" Then
    ' ... code omitted for simplicity ...

    ' Send the message and notify the user of success
    ' sc.Send(mm)
    AddHandler sc.SendCompleted, AddressOf sc_SendCompleted
    sc.SendAsync(mm, Nothing)
    sendButton.Text = "Cancel"
Else
    sc.SendAsyncCancel()
End If

// C#
if (sendButton.Text == "Send")
{
    // ... code omitted for simplicity ...

    // Send the message and notify the user of success
    //                sc.Send(mm);
    sc.SendCompleted += new SendCompletedEventHandler(sc_SendCompleted);
    sc.SendAsync(mm, null);
    sendButton.Text = "Cancel";
}
else
{
    sc.SendAsyncCancel();
}
```

6. Next, write a method to respond to the *SmtpClient.SendCompleted* event. Based on the variable name *sc*, Visual Studio generated a method named *sc_SendCompleted*. Within the method, you need to perform the following tasks:

   - If the *SendAsync* was cancelled, display a message confirming the cancellation.
   - If there was an error, display the error message to the user.
   - If the transmission was successful, inform the user.

The following code demonstrates this. If you are using Visual Basic, you will need to add the *System.ComponentModel* namespace:

```
' VB
Sub sc_SendCompleted(ByVal sender As Object, ByVal e As AsyncCompletedEventArgs)
    If e.Cancelled Then
        MessageBox.Show("Message cancelled.", "Cancelled", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)
    Else
        If Not (e.Error Is Nothing) Then
            MessageBox.Show("Error: " + e.Error.ToString, "Error", _
                MessageBoxButtons.OK, MessageBoxIcon.Error)
        Else
            MessageBox.Show("Message sent successfully.", "Success", _
                MessageBoxButtons.OK, MessageBoxIcon.Information)
        End If
    End If
    sendButton.Text = "Send"
End Sub

// C#
void sc_SendCompleted(object sender, AsyncCompletedEventArgs e)
{
    if (e.Cancelled)
        MessageBox.Show("Message cancelled.",
            "Cancelled", MessageBoxButtons.OK, MessageBoxIcon.Error);
    else if(e.Error != null)
        MessageBox.Show("Error: " + e.Error.ToString(),
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        MessageBox.Show("Message sent successfully.",
            "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    sendButton.Text = "Send";
}
```

7. Now, build and run your code. Verify that you can successfully transmit a message.
8. Next, send a message, but immediately click Cancel. If your SMTP server is extremely responsive, the Cancel button might disappear very quickly. Verify that your application correctly handles the cancellation and informs the user.
9. Finally, verify that your application correctly responds to incorrect server names, invalid users, and invalid credentials. Note whether an invalid server name is handled by the *SmtpClient.SendCompleted* event or the *sendButton_Click* exception handling.

Lesson Summary

- To send a message, create an instance of *SmtpClient*. Configure the SMTP server host name, and then call *SmtpClient.Send*.

- If you call *SmtpClient.Send* without defining the server hostname, the runtime will throw an *InvalidOperationException*. If the hostname is defined but the server cannot be found, the runtime will throw an *SmtpException* with an inner *WebException*. If the SMTP server reports that the recipient is invalid, the runtime will throw an *SmtpFailedRecipientException*. All other problems sending e-mail result in an *SmtpException*.
- To use the default network credentials, set *SmtpClient.UseDefaultCredentials* to True. To specify the username and password, create an instance of the *System.Net.NetworkCredential* class, and use it to define *SmtpClient.Credentials*.
- To enable SSL encryption for the SMTP connection, set *SmtpClient.EnableSsl* to True. Not all SMTP servers support SSL.
- To send a message asynchronously, first create a method to respond to the *SmtpClient.SendCompleted* event. Then add your event handler to *SmtpClient.SendCompleted*, and call *SmtpClient.SendAsync*. You can call *SmtpClient.SendAsyncCancel* to cancel an asynchronous message transmission before it has completed.

Lesson Review

You can use the following questions to test your knowledge of the information in Lesson 2, "Sending Mail." The questions are also available on the companion CD if you prefer to review them in electronic form.

NOTE Answers

Answers to these questions and explanations of why each answer choice is right or wrong are located in the "Answers" section at the end of the book.

---

1. Which method would you call to send an e-mail message and wait for the transmission to complete before proceeding?
   - **A.** *MailMessage.Send*
   - **B.** *SmtpClient.Send*
   - **C.** *SmtpClient.SendAsync*
   - **D.** *MailMessage.SendAsync*
2. You need to send e-mail messages from your assembly. The computer that will run your assembly also hosts an SMTP server. Which of the following are valid values for *SmtpClient.Host*? (Choose all that apply.)
   - **A.** self
   - **B.** 10.1.1.1

C. localhost

D. 127.0.0.1

3. What type of exception will the runtime throw if the SMTP server rejects a recipient e-mail address?

A. *SmtpFailedRecipientException*

B. *SmtpFailedRecipientsException*

C. *SmtpException*

D. *SmtpClientException*

4. You want to send an e-mail message to an SMTP server while encrypting the network communications. Which property do you need to modify from its default settings?

A. *SmtpClient.Credentials*

B. *SmtpClient.DeliveryMethod*

C. *SmtpClient.Port*

D. *SmtpClient.EnableSsl*

Chapter Review

To further practice and reinforce the skills you learned in this chapter, you can perform the following tasks:

- Review the chapter summary.
- Review the list of key terms introduced in this chapter.
- Complete the case scenarios. These scenarios set up real-world situations involving the topics of this chapter and ask you to create a solution.
- Complete the suggested practices.
- Take a practice test.

Chapter Summary

- To create an e-mail message, use the *System.Net.MailMessage* class. This class supports simple text e-mails, HTML e-mails, e-mails with multiple views and different encoding standards, and attachments.
- To send an e-mail message, use the *System.Net.SmtpClient* class. This class supports SMTP servers that accept anonymous connections, servers that require authentication, and servers that support SSL encryption. Additionally, you can send messages asynchronously to allow users to cancel a message before the transmission has completed.

Key Terms

Do you know what these key terms mean? You can check your answers by looking up the terms in the glossary at the end of the book.

- Multipurpose Internet Mail Extensions (MIME)
- Secure Sockets Layer (SSL)
- Simple Message Transfer Protocol (SMTP)

Case Scenario: Add E-mail Capabilities to an Existing Application

In the following case scenario, you will apply what you've learned about how to send e-mails. You can find answers to these questions in the "Answers" section at the end of this book.

You are an application developer for Contoso, Inc. For the last two years, you have been developing and maintaining an internal customer relationship management system. Recently, the Contoso Sales group has asked for a way to send confirmation e-mails to customers before making changes to their account information. Your manager asks you to interview key people and then come to her office to answer her questions about your design choices.

Interviews

Following is a list of company personnel interviewed and their statements:

- **Sales Manager** "It's critical to customer satisfaction that our client database is accurate. We've had several situations where someone mistyped an address or a phone number, and this resulted in missed deliveries, unsatisfied customers, and lost sales. What we'd like is to automatically send a confirmation e-mail to customers when we make a change to their contact information. If the change is incorrect, the customer should be able to reject the change. I suppose they could either reply to the e-mail, click a link, or call us, based on how hard it is for you to develop the solution. The rejections shouldn't be that common."
- **Chief Security Officer** "I understand the desire to confirm changes to contact information; however, we need to manage our risk. Sending confidential information through e-mail is never good. You can, and should, encrypt the connection to the SMTP server, but you lose any control over the security of the data after you hand it off. Messages often bounce through several different servers, and most connections are not encrypted. Being that this is just contact information, I'm okay with sending the messages, but please do use security when possible."

Questions

Answer the following questions for your manager.

1. What .NET Framework classes will you use to send the e-mails?
2. How could you process e-mail responses from customers?
3. How can you protect the data in the e-mail messages?

Suggested Practices

To help you successfully master the "Implementing interoperability, reflection, and mailing functionality in a .NET Framework application" exam objective, complete the following tasks.

Send Electronic Mail to a Simple Mail Transfer Protocol (SMTP) Server for Delivery from a .NET Framework Application

For this task, you should complete at least Practices 1 and 2. If you want a better understanding of how to work with file attachments, complete Practices 3 through 5. For a detailed understanding of the importance of using SSL, complete Practice 6 as well.

- **Practice 1** Use the application you created in the labs in this chapter, and attempt to send messages through different SMTP servers. Note how different servers respond. To look up the SMTP server for any domain, open a command prompt and run the command *nslookup -type=mx domain* (for example, *nslookup –type=mx contoso.com*).
- **Practice 2** Expand the application you created in the labs in this chapter to enable the user to attach files.
- **Practice 3** Attempt to send increasingly larger file attachments until the SMTP server rejects your message. Test several different SMTP servers. Note the maximum file size.
- **Practice 4** Using a text file as an attachment, change the file extension to .txt, .jpg, .bat, .cmd, .dll, and .exe. Note which file extensions different mail servers allow, and which of the attachments can be viewed using e-mail clients such as Microsoft Outlook.
- **Practice 5** In Chapter 6, "Graphics," you created an application that generates a chart. Add a button to the application that sends the chart as an e-mail attachment.
- **Practice 6** Using a protocol analyzer (also known as a *sniffer*) such as Microsoft Network Monitor, capture the network communications created when sending a message using standard SMTP, standard SMTP with user credentials, and SMTP protected by SSL. Use the protocol analyzer to view the raw packets and determine whether an attacker with access to the network could view your message and user credentials.

Take a Practice Test

The practice tests on this book's companion CD offer many options. For example, you can test yourself on just one exam objective, or you can test yourself on all the 70-536 certification exam content. You can set up the test so that it closely

simulates the experience of taking a certification exam, or you can set it up in study mode so that you can look at the correct answers and explanations after you answer each question.

MORE INFO Practice tests

For details about all the practice test options available, see the "How to Use the Practice Tests" section in this book's Introduction.