WEEK NO 14, 15: SMTP

- Overview
- E-mail Basics
- SMTP Mail Class
    - Class Method & Class properties
    - Using SMTP Mail Class
- Mail Message
    - Mail Format
    - Mail Message Class Properties
    - Using Mail Message Class
- Mail Attachment
    - Uuencode
    - MIME
- Introduction to Mail Attachment Class
    - Using Mail Attachment Class

SMTP

Overview

The ***System.Web.Mail*** namespace contains classes that are used for creating and sending e-mail messages to remote hosts. This is done using either the default Windows Simple Mail Transfer Protocol (SMTP) service on Windows 2000 and XP machines, or an external mail server.

E-mail Basics

Almost all Internet e-mail packages use the Unix e-mail model to implement electronic messaging with remote hosts. This model has become the most popular and widely implemented technique of delivering messages both to local customers and to remote host customers.

The Unix e-mail model divides the e-mail functions into three parts:

- The Message Transfer Agent (MTA)
- The Message Delivery Agent (MDA)
- The Message User Agent (MUA)

Each of these parts performs a specific function in the e-mail process of sending, delivering, and displaying messages.

The MTA Process

The MTA process handles both incoming and outgoing mail messages. This obviously includes two separate functions:

- Determining where and how to deliver outgoing mail
- Determining where to deliver incoming mail

Each of these functions requires different processing capabilities within the MTA process.

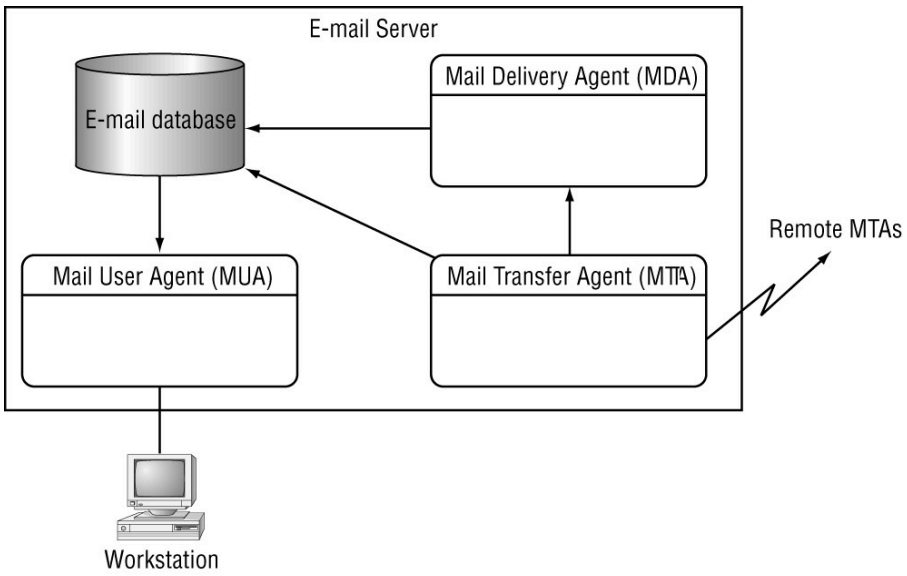


Figure: The Unix e-mail model

Sending Outgoing Mail

For each outgoing mail message, the MTA must determine the destination of the recipient addresses. If the destination is the local system, the MTA process can either deliver the message directly to the local mailbox system, or pass the message off to the MDA process for delivery.

However, if the destination is a remote domain, the MTA process must perform two functions:

- Determine the responsible mail server for the domain using the MX entry in DNS
- Establish a communication link with the remote mail server and transfer the message

The communication link that is established with the remote mail server almost always uses the Simple Mail Transfer Protocol (SMTP). This standard protocol provides a common communication technique to move messages between various types of systems on the Internet.

Receiving Incoming Mail

The MTA process is responsible for accepting incoming messages from both local system users and remote host users. Many variables are associated with this seemingly simple function. Destination addresses in the mail message must be examined closely, and a decision must be made as to whether the message can in fact be delivered by the local system.

There are three categories of destination addresses that can be used in a mail message: local system accounts, local alias accounts, and remote user accounts.

Local system accounts Every mail system, whether Windows, Unix, or even Macintosh, has its own set of configured local user accounts that have access to the system. The MTA must recognize messages destined for the user accounts and forward them either directly to the user's mailbox, or to a separate MDA process for delivery.

Local alias accounts Many MTA programs allow for user alias names to be created. The alias name itself cannot store mail messages. Instead, it is used as a pointer to one or more actual system user accounts where the messages are stored. Once the MTA determines that the alias name is valid, it converts the

destination address to the actual system user account name and either delivers the message or passes it to an MDA process.

Remote user accounts Many MTA processes also accept incoming messages destined for user accounts not on the local system. This is the trickiest of the delivery categories. There have been many debates and arguments over how MTA programs handle mail destined for remote user accounts.

This technique is called mail *relaying*. A mail server accepts a message destined for a remote host user and automatically forwards it to the destination remote host. For many ISPs, this feature is a necessity because dial-up customers do not have the capability to send Internet mail messages directly to remote hosts. Instead, they must send messages to the ISP mail server, which in turn passes them off (relays them) to the remote destination.

Unfortunately, mail relaying can be exploited. Unscrupulous individuals can use relay mail servers to help hide their original host addresses when forwarding mass mail advertisements to remote e-mail users. Whether it's called Unsolicited Commercial E-mail (UCE), Unsolicited Bulk E-mail (UBE), or just plain old spam, this mail is annoying and must be stopped by the mail administrator.

To help stop spam, most ISPs now use selective relaying, allowing only a set of IP addresses or authenticated users associated with their dial-up customers to forward mail through their mail servers and blocking any other host from forwarding mail through the server.

The MDA Process

The primary function of the MDA process is to deliver mail messages destined for local system user accounts. To do this, the MDA must know the type and location of the individual user mailboxes. Most e-mail systems use some type of database system to track the e-mail messages stored for local users. The MDA process must have access to each user's mailbox to insert incoming messages.

Many MDA processes also perform advanced techniques in addition to just delivering mail messages:

Automatic mail filtering Possibly the most helpful and popular feature of the MDA process is enabling the filtering of incoming mail messages. For users who get a lot of e-mail messages, this feature can be a lifesaver. Messages can be automatically sorted into separate e-mail folders based on a subject header value, or even just one word contained in the subject header.

Most MDA programs also allow the mail administrator to configure a global filter that can help block spam and well-known virus mail messages. This is done using standard text wildcard expressions. Each expression can be matched against incoming message subject headers, and the messages that match the expression are dropped and not delivered.

Automatic mail replying Many MDA programs allow the customer to configure an auto-reply message for e-mail messages. Some auto-reply messages can be configured to reply to all mail messages received by a user, such as an out-of-office message indicator. Other auto-reply messages can be set up to reply to specific text found in the message subject header, much like the mail filter. When a message matches the text expression, a specific auto-reply message is automatically sent.

Automatic Program Initialization The ability to start system programs based on incoming mail messages is another feature found in MDA programs. Certainly there must be controls in place to prevent misuse, but when used safely this feature is a handy tool. Administrators can use it to start server processes remotely, or to change configuration values from a machine other than the server.

The MUA Process

The MUA process allows customers who are located remotely from their mail server to access mailboxes. One common misconception about MUA programs is that they send mail messages. The MUA process itself is only responsible for *reading* mailbox messages; it does not receive or send new messages. The misconception comes from the fact that many MUA programs include a small MTA process that offers the convenience of *relaying* new messages to the mail server.

There are two philosophies underlying MUA programs, both dealing with how incoming messages are read and stored.

Storing Messages on the Client

Many ISPs prefer to have customers download their messages directly to their workstations. Once the message is downloaded, it is removed from the mail server. This helps the mail administrator maintain the disk space on the server and prevent messages from clogging up the disk system.

The access protocol used for this type of message access is the Post Office Protocol (POP, often called POP3 for version 3). The POP3 MUA program allows a customer to connect to the server and download all the messages in the mailbox to the workstation.

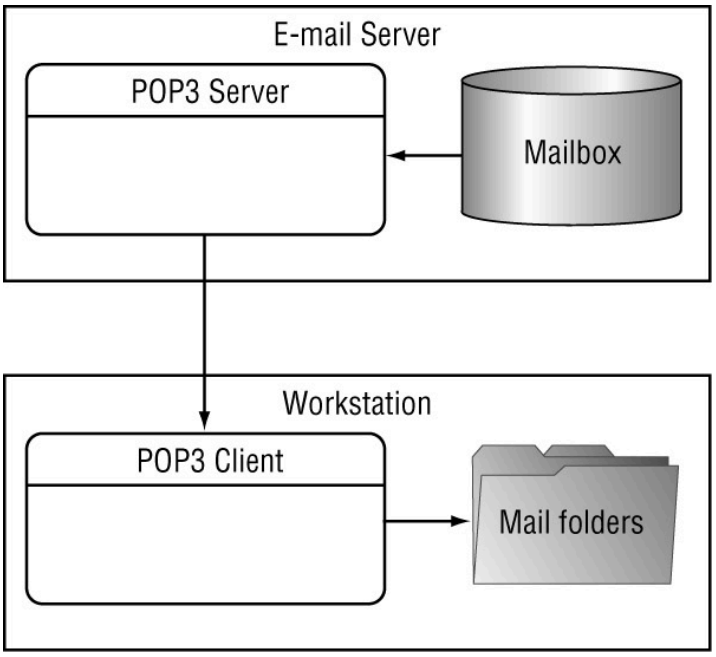


Figure: Downloading messages using POP3 software

The drawback to using POP3 is that the messages are stored on the workstation with which the customer connected to the mail server. If the customer connects using multiple workstations, the messages may end up being split between workstations, making it harder for the customer to access messages after they are downloaded.

Storing Messages on the Server

As an alternative to the POP3 access method, the Interactive Mail Access Protocol (IMAP, or IMAPrev4 for version 4) was created. IMAP allows the customer to build folders on the mail server and store messages in the folders instead of downloading them to the workstation. This is demonstrated in Figure 13.3. Because the

messages are stored on the server, the customer can connect from as many workstations as necessary and still see the same mail messages in the folders.

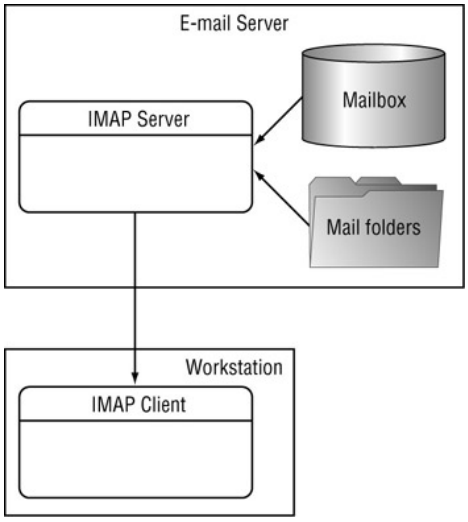


Figure: Storing messages in server folders using IMAP

The drawback to this system is that the mail administrator has the added concern of disk space consumption on the server. It does not take long for today's mail messages and attachments to fill up system disks.

SMTP Mail Class (SmtpMail)

The SmtpMail class, found in the System.Web.Mail namespace, allows you to send SMTP messages in your C# network programs. This section discusses the methods and properties of the SmtpMail class and shows some examples of typical uses in C# programs.

Class Methods and Properties

The SmtpMail class provides a .NET interface to the CDOSYS mail library on Windows 2000 and XP systems. It does not use a constructor to create an instance of the class. Instead, you must use the static class methods and properties to pass information to the CDOSYS library.

The SmtpMail class does contain the standard Equals(), GetHashCode(), and ToString() methods, but they're rarely (if ever) used. The Send() method, on the other hand, gets plenty of use. The Send() method is overloaded, using two separate formats:

```
Send(MailMessage message)


Send(string from, string to, string subject, string body)
```

- The first format allows you to send a MailMessage object. The MailMessage class is a self-contained e-mail message, created by populating the properties of the class with information related to the message, and the destination address(es). This process will be covered a little later in the section on the MailMessage class.

- The second format allows you to send a raw message, manually specifying the typical e-mail message header fields:
  - ***From*** specifies the e-mail address of the sender.
  - ***To*** specify the e-mail address of one or more recipients, separated by commas.
  - ***Subject*** specifies the topic of the e-mail message.
  - The final parameter of the Send() method is the actual ***body*** of the message. The body can be in either a plain text or HTML format.

The sole property of the SmtpMail class is SmtpServer. It's a static property that specifies the address of a mail relay server to use for forwarding outgoing mail messages. By default, the SmtpServer property is set to the IIS SMTP service if it is installed. If IIS is not installed, the SmtpServer property is set to a null value and will produce an error when you attempt to send a message.

If you are using a relay mail server, you must set the SmtpServer property before you attempt to send messages:

```
SmtpMail.SmtpServer = "mailsrvr.myisp.net";
```

When this value is set, all outgoing mail messages will be relayed through this mail server. Of course, you must ensure that you are allowed to relay mail messages through the server, or they will be rejected.

Using SmtpMail Class

The MailTest.cs program in Listing 13.1, demonstrates how to create a simple mail message and send it through a remote mail relay to the recipient.

Listing: The MailTest.cs program

```
using System;
using System.Net;
using System.Web.Mail;
class MailTest
{
  public static void Main()
  {
   string from = "jessica@myisp.net";
```

```
    string to = "katie@anotherisp.net";
    string subject = "This is a test mail message";
    string body = "Hi Katie, I hope things are going well today.";
    SmtpMail.SmtpServer = "192.168.1.150";
    SmtpMail.Send(from, to, subject, body);
  }
}
```

The MailTest program is about as simple as they come. All it does is define the relay mail server using the SmtpServer property, define each of the required string fields, and use the Send() method to send the message. If all goes well, the intended recipient should receive the message in their mailbox.

If you try to run this on a Windows platform that does not use the CDO 2 library, you will get an Exception message. The exact message will vary, depending on which platform you run it on.

If you are writing code that will be run on more than one Windows platform, it is best to catch this Exception and produce your own informational message:

```
try
{
  SmtpMail.SmtpServer = "192.168.1.150";
  SmtpMail.Send(from, to, subject, body);
}
catch (System.Web.HttpException)
{
  Console.WriteLine("Sorry, this application only works on Windows
     2000and XP platforms.");
}
```

Mail Message

Mail Format

RFC 2822 defines the standard formatting of an e-mail message. The .NET MailMessage class offers a standard way for you to create RFC2882 messages to send to remote customers.

The RFC2822 Mail Format

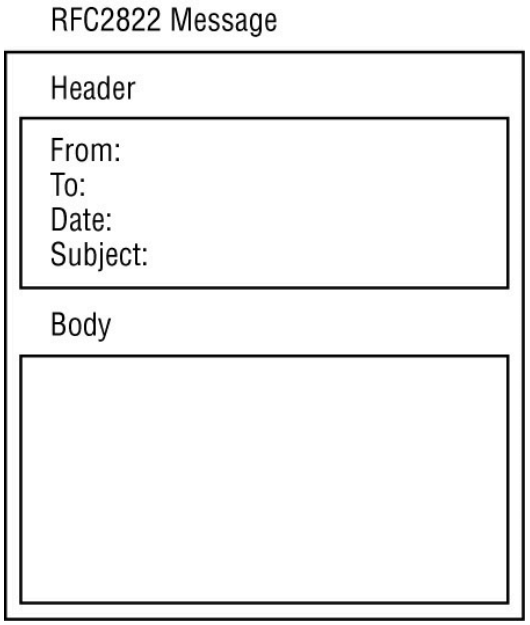
RFC 2822 specifies that each message should consist of two separate sections:

- A header that contains message information
- A body that contains the text of the message

The header of an RFC2822 standard mail message contains separate data fields that supply additional information in the message. Each header field is on a separate line in the message and contains data in text format. The format of each header line is as follows (note the colon separating the header tag from the data value):

```
header: text data
```

Individual header fields do not need to be specified in any particular order in the message. The header fields must also appear before the body of the message and must be separated by a single blank line.



The RFC2822 message format

The Origination Date Field

The origination date field identifies the date and time the message was originally sent. It consists of the keyword Date:, followed by the appropriate date and time value:

```
Date: Wed, 21 Aug 2002 18:30:00 -0500
```

It is important that the origination date identify the time zone of the sending host so the receiving customer can determine the actual time the message was sent.

The Originator Fields

Three separate originator fields identify where the message originated:

```
From: mailbox-list

Sender: mailbox

Reply-To: mailbox-list
```

At least one of the three originator fields must be present in a mail message. The mailbox-list value can contain either a single mail address or multiple mail addresses separated by commas.

The *From* field identifies the specific person who originated the message, while the Sender field identifies the person who actually sent the message (the two don't necessarily have to be the same). If the Reply-To field is present, it represents the return address where the message originator intends replies to be sent.

The Destination Address Fields

Three destination address fields identify where the message should be sent:

```
To: address-list

Cc: address-list

Bcc: address-list
```

The destination address fields can each specify one or more addresses in the address-list fields, using commas to separate multiple addresses.

The *To* field identifies primary recipients of the mail message, and the Cc field contains addresses of others who are to receive the message though it wasn't directed toward them.

The *Bcc* field contains addresses of recipients who are not disclosed in the mail message. The Bcc field is different from the other destination address fields; although the sending message may contain a Bcc field, it must be stripped off before sending to the intended destination addresses.

The Identification Fields

The identification fields help uniquely identify the message, both to the mail system and to the recipients. There are quite a few identification fields available, but these are the most common:

```
Message-ID: msg-id

In-Reply-To: msg-id

References: msg-id
```

The *Message-ID* field provides a unique tracking number to the mail message. Each message sent by an MTA should contain a Message-ID field, assigned by the MTA, that uniquely identifies the message from all others. The msg-id value can contain any combination of letters and numbers to uniquely identify the message.

The *In-Reply-To* and *References* fields are used to relate one message to a previous message. Often, a message sent in reply to a previous message will contain the Message-ID field of the previous message in the In-Reply-To field.

The Informational Fields

The informational fields are optional; they help recipients identify and possibly filter mail messages. The informational fields are as follows:

```
Subject: subject-text

Comments: comment-text

Keywords: phrase-text
```

The Subject field is the most common field. It identifies the message with a short string describing the topic of the message. The Comments field can present a longer string further describing the body of the message. The Keywords field specifies short phrases, separated by commas that identify the message content. The Keywords field can be used in MDA filtering software to sort incoming messages.

The MailMessage Class Properties

The .NET MailMessage class easily creates RFC2822-formatted messages to send using the SmtpMail class. This allows you to add more useful header field information than what the generic SmtpMail.Send() method allows. You construct the e-mail message piece by piece and specify formatting options for the message.

The default constructor for the MailMessage class is not very difficult:

```
MailMessage newmessage = new MailMessage();
```

Following table lists the properties that can be used.

Property	Description
Attachments	Specifies a list of file attachments to add to the message
Bcc	Gets or sets a semicolon-delimited list of addresses to use in the Bcc: header field
Body	Gets or sets the body of the e-mail message
BodyEncoding	Gets or sets the encoding type of the message body
BodyFormat	Gets or sets the content type of the message body
Cc	Gets or sets a semicolon-delimited list of addresses to use in the Cc: header field
From	Gets or sets the address to use in the From: header field
Headers	Sets custom header fields and values for the message
Priority	Gets or sets the priority of the message
Subject	Gets or sets the text string used in the Subject: header field

To	Gets or sets the semicolon-delimited list of addresses used in the To: header field
UrlContentBase	Gets or sets the Content-Base HTTP header value
UrlContentLocation	Gets or sets the Content-Location HTTP header value

The easy properties to use are the typical To, From, Cc, and Bcc values. You can specify additional header fields using the Header property, although this is done a little differently.

The Header Property

The Header property uses the IDictionary class to store the header field and value pair. The easiest way to do this is to use the Add() method:

```
newmessage.Header.Add("Reply-To", "testing@myisp.net");
```

You can specify any valid RFC2822 header field using the Header property and it will be included in the message header fields. You can easily add a Date field to your messages using the format:

```
DateTime mydate = DateTime.Now;

newmessage.Header.Add("Date", mydate.ToString());
```

The Body Properties

The Body property, the BodyEncoding, and BodyFormat properties specify both the content and the format of the message body. The text of the message is assigned to the Body property. How it looks to the remote mail server is controlled by the BodyEncoding and BodyFormat properties.

The BodyEncoding property uses the System.Text.Encoding classes to specify the type of text used for the message. The possible values are ASCII, Unicode, UTF-7, and UTF-8. The default is the default system encoding type (which is ASCII in the United States).

In contrast to the text encoding, the BodyFormat property specifies how the text will be presented in the mail message. The possible values are defined in the MailFormat enumeration:

- **MailFormat.Html** Uses hypertext markup language formatting
- **MailFormat.Text** Uses plain text formatting

By default, the Text value is used. This ensures that any e-mail client can read the message. Alternatively, you can use the Html value to incorporate special text formatting within the message.

The Priority Property

The Priority property allows you to set the Priority header field. Though it's not a standard header field, many mail clients recognize the Priority field as indicating the importance of a mail message. The System.Web.Mail namespace includes the MailPriority enumeration that specifies this value. The possible values are as follows:

- **MailPriority.High** Important messages that require immediate attention

- **MailPriority.Normal** Regular mail messages
- **MailPriority.Low** Extraneous mail messages, such as advertisements

To set the priority of a message, just assign the appropriate value to the Priority property:

```
newmessage.Priority = MailPriority.High;
```

The Attachment Property

The Attachment property specifies files you want to include as attachments to the mail message. You can specify one or more MailAttachment class objects as attachments using the following format:

```
MailAttachment ma = new MailAttachment("c:\\tempfile.bmp");

newmessage.Attachment.Add(ma);
```

The file specified in the MailAttachment object is encoded and added to the message.

Using the MailMessage Class

The FancyMailTest.cs program uses the MailMessage class to produce a fancier mail message to send to the recipient.

Listing: The FancyMailTest.cs program

```
using System;
using System.Web.Mail;
class FancyMailTest
{
  public static void Main()
  {
   MailMessage mm = new MailMessage();
   mm.From = "haley@myisp.net";
   mm.To = "riley@yourisp.net;rich@shadrach.ispnet1.net";
   mm.Cc = "matthew@anotherisp.net;chris@hisisp.net
   mm.Bcc = "katie@herisp.net;jessica@herisp.net";
   mm.Subject = "This is a fancy test message";
   mm.Headers.Add("Reply-To", "haley@myisp.net");
   mm.Headers.Add("Comments", "This is a test HTML message");
```

```
    mm.Priority = MailPriority.High;

    mm.BodyFormat = MailFormat.Html;

    mm.Body = "<html><body><h1>This is a test message</h1><h2>This
       message Â should have HTML-type formatting</h2>Please use an
       HTML-capable viewer.";

    try

    {

      SmtpMail.Send(mm);

    }

    catch (System.Web.HttpException)

    {

      Console.WriteLine("This device is unable to send Internet
       messages");

    }

   }

}
```

The FancyMailtest program creates a MailMessage object and assigns values to the various properties. Multiple recipients are added to the To, Cc, and Bcc properties, each one separated by a semicolon. Message priority is set to High, and two additional header lines are added to the mail message.

Because the BodyFormat type is set to Html, you can use any valid HTML syntax in the body text to format the message. Following figure demonstrates how this message looks when viewed with a Microsoft Outlook Express mail client.

Figure: Reading the FancyMailTest mail message using Outlook Express

Mail Attachment

The Body property allows you only to enter text, and a separate property is used to include file attachments. This arrangement is the result of how SMTP works and how mail servers must handle binary data.

SMTP transfers only text messages between systems on the Internet. To compensate for this, mail clients must convert any binary data (such as file attachments) into an ASCII text message before passing the message to the mail server. Then, of course, the recipient's mail client must be able to convert the text message back into the original binary data.

There are two popular techniques that can convert binary data into text:

- the uuencode format and
- the Multipurpose Internet Mail Extensions (MIME) format

Most mail systems allow you to use either technique, although there are still some that require a particular format.

uuencode

Many years before the Internet became popular, Unix administrators were sending binary data across modem lines by converting it to ASCII text and embedding it in mail messages. The program they used to convert binary data into ASCII text was called *uuencode*. The uu stands for Unix-to-Unix, part of the Unix-to-Unix Copy Protocol (UUCP) that's used to send messages between Unix hosts via modem.

The uuencode program uses a 3-to-4 encoding scheme, in which 3 bytes of binary data are converted to 4 bytes of ASCII characters. This scheme significantly increases the size of the converted file but ensures that the encoded file can be safely decoded back into the original binary data.

Because of its popularity, many mail systems still support the uuencode method of encoding binary data. However, a newer Internet standard for encoding binary data has been developed: MIME.

MIME

The MIME format (Multipurpose Internet Mail Extensions) is defined in RFCs 2045 and 2046. MIME is more versatile than uuencode in that it includes additional information about the binary file within the converted file. The decoder can thus automatically detect and decode various types of binary files.

The MIME standard also provides a way for the MIME-encoded binary data to be directly incorporated into a standard RFC2822 message. Five new header fields were defined to identify binary data types embedded in the mail message. E-mail clients that can handle MIME messages must be able to process these five new header fields. The fields are added immediately after the RFC2822 header fields, and before the message body. Any MIME attachments are added after the message body.

The MIME Message Header Fields

Field	Description
MIME-Version	Specifies the version of MIME used in the encoding

Content-Transfer-Encoding	Specifies the encoding scheme used to encode the binary data into ASCII
Content-ID	Specifies a unique identifier for the message section
Content-Description	A short description identifying the message section
Content-Type	Specifies the type of content contained in the encoded data

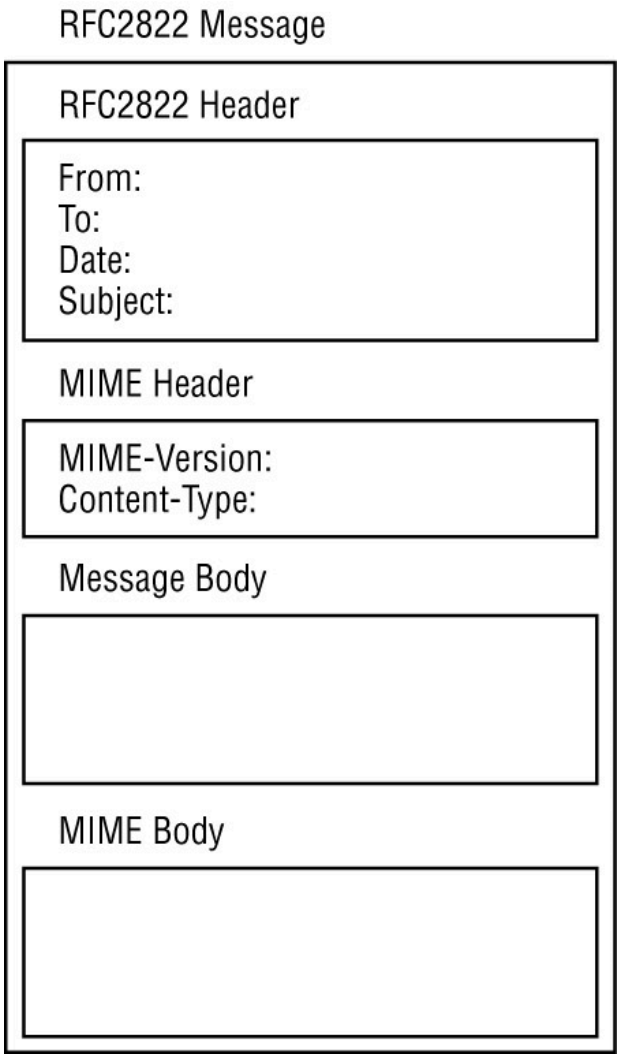


Figure: The MIME message format

The MIME-Version Field

The MIME-Version field identifies the MIME encoding version that the sender used to encode the message:

```
MIME-Version: 1.0
```

Alternatively, some software packages add text after the version number to identify additional vendor version information:

```
MIME-Version: 1.0 (software test 2.3a)
```

The MIME decoding software of the receiving mail server ignores the additional text.

The Content-Transfer-Encoding Field

The Content-Transfer-Encoding field identifies how the binary data in the message is encoded. There are currently seven methods defined for MIME. The first three methods define no encoding of the data. The 7-bit encoding method assumes that the encoded data is already 7-bit ASCII text characters, not binary data. This is the default used if no Content-Transfer-Encoding field is present.

MIME Encoding Methods

Method	Description
7-bit	Standard 7-bit ASCII text
8-bit	Standard 8-bit ASCII text
binary	Raw binary data
quoted-printable	Encodes binary data to printable characters in the U.S.-ASCII character set
base64	Encodes 6 bits of binary data into an 8-bit printable character
ietf-token	Extension token encoding defined in RFC 2045
x-token	Two characters, X- or x-, followed (with no intervening space) by any token

Base64 is the most common method used for encoding binary data. This scheme encodes binary data by mapping 6-bit blocks of binary data to an 8-bit byte of ASCII text. There is less “wasted space” in the encoded file than with the uuencode method, and it often results in a smaller encoded file.

The Content-ID Field

The Content-ID field identifies MIME sections with a unique identification code. One MIME content section can refer to another MIME message by using this unique field value.

The Content-Description Field

The Content-Description field is an ASCII text description of the data to help identify it in the e-mail message. The text can be any ASCII text of any length.

The Content-Type Field

The Content-Type field is where all the action is. This field identifies the data enclosed in the MIME message. Two separate values, a type and a subtype, identify the data. Here’s the field format:

```
Content-Type: type/subtype
```

Following are descriptions of the seven basic types of Content-Type identified in MIME:

- **text** The text Content-Type identifies data that is in ASCII text format. The subtypes for the text Content-Type can be in one of three formats:
- **plain** For unformatted ASCII text
- **html** For text formatted with HTML tags
- **enriched** For text formatted with rich text format (RTF) tags

The text Content-Type also specifies the character set used to encode the data with the charset parameter:

```
Content-Type: text/plain; charset=us-ascii
```

This line identifies the MIME section as being plain text, using the U.S. ASCII encoding system.

- **message** The message Content-Type identifies multiple RFC2822-formatted messages contained within a single message. It has three subtypes:
- **rfc822** Specifies a normal embedded RFC 822-formatted message
- **partial** Specifies one section of a long message that was broken up into separate sections
- **external-body** Specifies a pointer to an object that is not within the e-mail message
- **image** The image Content-Type defines embedded binary data that represents a graphic image. Currently two subtypes are defined: the JPEG format and the GIF format.
- **video** The video Content-Type defines embedded binary data that represents video data. The only subtype defined is the MPEG format.
- **audio** The audio Content-Type defines embedded binary data that represents audio data. The only subtype for this is the basic format, which defines a single-channel Integrated Services Digital Network (ISDN) mu-law encoding at an 8KHz sample rate.
- **application** The application Content-Type identifies embedded binary data that represents application data, such as spreadsheets, word processing documents, and other applications. There are two formal subtypes defined: the postscript format for Postscript-formatted print documents, and the octet-stream format, which defines messages containing arbitrary binary data. The octet-stream subtype represents most application-specific data, such as Microsoft Word documents and Microsoft Excel spreadsheets.
- **multipart** The multipart Content-Type is a special type. It identifies messages that contain multiple data content types combined into one message. This format is common in e-mail packages that can present a message in a variety of ways, such as plain ASCII text and HTML, or in messages that contain multiple attachments. There are four subtypes used:
- **Mixed** Specifies that each of the separate parts are independent of one another and should all be presented to the end customer in the order they are received.

- **Parallel** Specifies that each of the separate parts are independent of one another but can be presented to the end customer in any order.
- **Alternative** Specifies that each of the separate parts represents different ways of presenting the same information. Only one part should be presented to the end customer.
- **Digest** Identifies the same method as the mixed subtype but specifies that the body of the message is always in RFC822-format.

Introduction to Mail Attachment Class

To simplify e-mail attachments, the .NET library includes the MailAttachment class. This class specifies the content and format of any attachments that are included in the mail message.

There are two constructors that can create a new mail attachment:

```
MailAttachment(string filename);

MailAttachment(string filename, MailEncoding encodetype);
```

The first constructor format creates a new mail attachment using the file specified in the filename path and encodes it using the ***UUEncode*** encoding type. The second constructor format allows you to specify an alternative encoding type, such as the Base64 encoding type used for MIME messages.

The *System.Web.Mail* namespace includes the *MailEncoding* enumeration that defines the two separate encoding types:

- **MailEncoding.Base64** For base64 MIME encoding
- **MailEncoding.UUEncode** For uuencode encoding

You can query the encoding and filename properties of the MailAttachment object using the Encoding and Filename properties:

```
MailAttachment newfile = new
MailAttachment("c:\\testfile.bmp", MailEncoding.Base64);

string filename = newfile.Filename;

if (newfile.Encoding == MailEncoding.Base64)

   Console.WriteLine("{0} is a MIME encoded file", filename);
```

When the MailAttachment object is created, it can be included in a MailMessage object using the Attachments property:

```
MailMessage newmessage = new MailMessage();

MailAttachment newfile = new
MailAttachment("c:\\testfile.bmp");

newmessage.Attachments.Add(newfile);

newmessage.From = "frank@myisp.net";

newmessage.To = "melanie@myisp.net;nicholas@myisp.net";

newmessage.Body = "Here's my picture!!";

SmtpMail.Send(newmessage);
```

Using Mail Attachment Class

The MailAttachTest.cs program demonstrates how to include file attachments with your e-mail messages.

Listing: The MailAttachTest.cs program

```
using System;

using System.Web.Mail;

class MailAttachTest

{

  public static void Main()

  {

   MailAttachment myattach = new MailAttachment
("c:\\temp\\MailAttachTest.exe", MailEncoding.Base64);

   MailMessage newmessage = new MailMessage();

   newmessage.From = "barbara@shadrach.ispnet1.net";

   newmessage.To = "rich@shadrach.ispnet1.net";

   newmessage.Subject = "A test mail attachment message";

   newmessage.Priority = MailPriority.High;

   newmessage.Headers.Add("Comments",

         "This message attempts to send a binary attachment");

   newmessage.Attachments.Add(myattach);
```

```
    newmessage.Body = "Here's a test file for you to try";
    try
    {
      SmtpMail.SmtpServer = "192.168.1.100";
      SmtpMail.Send(newmessage);
    }
    catch (System.Web.HttpException)
    {
      Console.WriteLine("This device cannot send Internet
messages");
    }
   }
}
```

The MailAttachTest program creates a MailAttachment object, using the Base64 encoding scheme. Next a normal MailMessage object is created, and the MailAttachment object is added as an attachment:

```
newmessage.Attachments.Add(myattach);
```

This example also uses a remote SMTP mail relay server to forward the message to the intended recipients.

SMTP and Windows

SMTP Mail Service

Both Windows 2000 and XP provide a basic SMTP server service that supports both sending and receiving mail messages using the SMTP protocol. This functionality is included as part of the Internet Information Services (IIS) package. The .NET mail classes can utilize the IIS SMTP server to send outgoing messages directly to remote mail servers.

Windows XP systems include IIS version 5.1, which among other things includes the SMTP service.

Before you can use the SMTP service, it must be configured to work properly in your Internet mail environment. Configuration is done from the Computer Management window.

Here are the steps to access the Default SMTP Virtual Server Properties window:

1. ***Right-click the My Computer item in the Start menu, and select the Manage menu item.***
2. ***When the Computer Management window appears, expand Services and Applications and then Internet Information Services.***
3. ***Right-click Default SMTP Virtual Server and select Properties.***

The Properties window is where you'll configure the settings for the SMTP service.

In the General tab, select the network interface(s) that should allow incoming SMTP connections and the number of concurrent SMTP connections allowed. Also, you can enable/disable logging for the SMTP service.

In the Delivery tab, configure delivery retry times. The Advanced button reveals settings for a smart host for the mail server. This is important if your ISP does not allow you to directly communicate via SMTP to remote hosts. Often you must use your ISP mail server as a mail relay to reach remote mail servers. To do this, you must enter the ISP mail server address as the smart host, and the SMTP service will forward all outgoing messages through that mail server.

Figure: The Default SMTP Virtual Server Properties dialog box is where you configure SMTP settings.

References:

- ✔ C# Network Programming, Richard Blum, ISBN:0782141765, Sybex, 2003
  - Chapter 13