Catalyst
internet mail

Developer's Guide and Technical Reference
Version 4.0

Catalyst Internet Mail License Agreement

This License Agreement is a legal agreement between you, either as an individual or a single entity ("Developer"), and Catalyst Development Corporation ("Catalyst") for the software product identified as "Catalyst Internet Mail". The Software Product includes executable programs, redistributable modules, controls, and dynamic link libraries ("Components" or "Software Components"), electronic documentation, and may include associated media and printed materials.

Installing this Software Product onto a hard disk or any other storage device of a computer, or loading any of the Components into the memory of any computer, constitutes use of the Software and shall acknowledge your acceptance of the terms and conditions of this License Agreement and your agreement to be bound thereby.

1. GRANT OF LICENSE

Catalyst Development grants you as an individual, a personal, non-exclusive, non-transferable license to install the Software Product using an authorized serial number. If you are an entity, Catalyst grants you the right to appoint an individual within your organization to use and administer the Software Product subject to the same restrictions enforced on individual users. You may not network the Software or otherwise use it on more than one workstation or computer at the same time. Contact Catalyst for more information regarding multi-developer site licensing.

You may install the Software Product on one or more workstations or computers expressly for the purposes of evaluating the performance of the Software for a period of no more than thirty (30) days. If continued use of the Software is desired after the evaluation period has expired, then the Software Product must be purchased and/or registered with Catalyst Development for each developer. The Software Product must be removed from all unregistered workstation(s) or computer(s) after the evaluation period has expired. You acknowledge that there is no product evaluation period available for Software source code.

2. COPYRIGHT

Except for the licenses granted by this agreement, all right, title, and interest in and to the Software Product (including, but not limited to, all copyrights in any executable programs, modules, controls, libraries, electronic documentation, text and example programs), any printed materials and copies of the Software Product are owned by Catalyst Development. The Software Product is protected by copyright laws and international treaty provisions. Therefore you must treat the Software Product like any other copyrighted material except that you may (a) make one copy of the Software solely for backup or archival purposes, or (b) transfer the Software to a single hard disk, provided you keep the original solely for backup or archival purposes. You may not copy any printed materials that may accompany the Software Product.

3. REDISTRIBUTION

a) In addition to the rights granted in Section 1, Catalyst Development grants you the right to use and modify the source code version of those portions of the Software designated as "sample code" for the sole purposes of designing, developing, and testing your software product(s), and to reproduce and distribute the sample code, along with any modifications thereof, only in object code form, provided that you comply with Section 3.c.

b) In addition to the rights granted in Section 1, Catalyst Development grants you a non-exclusive, royalty-free right to reproduce and distribute the object code version of any portion of the Software Product, along with any modifications thereof, in accordance with the above stated conditions.

c) If you redistribute the sample code or redistributable components, you agree to: (i) distribute the redistributables in object code only, in conjunction with and as a part of a software application product developed by you which adds significant and primary functionality to the Software; (ii) not use Catalyst Development's name, logo, or trademarks to market your software application product; (iii) include a valid copyright notice on your software product ; (iv) indemnify, hold harmless, and defend Catalyst Development from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software application product; (v) not permit further distribution of the redistributables by your end user.

4. SOURCE CODE

If you obtain a license to any of the Software’s source code (also referred to as "Software" for the purposes of this entire agreement) neither the source code nor modified source code may be distributed by you to any third party under any circumstances. The Software source code must be protected as you would your own and you expressly and unequivocally agree to be bound by the acts of your employees and agents and the terms of the TRADE SECRETS AND CONFIDENTIALITY section below.

a) Catalyst Development retains all right, title and interest in and to the Software Product source code. Licensing of the Software source code does not constitute a transfer of ownership under the terms of this agreement, and the Software remains owned and copyrighted by Catalyst Development.

b) The Software source code may only be installed on a single computer for use by a single developer. If you are an entity, Catalyst Development grants you the right to appoint an individual within your organization to use and administer the Software source code subject to the same restrictions enforced on individual users.

c) You may modify the Software source code, however such modifications do not constitute ownership of the source code. Modifications to the Software source code may not be sold, transferred or published in any manner whatsoever.

d) You may redistribute object code only, in conjunction with and as a part of a software application product developed by you which adds significant and primary functionality to the Software. You may not use the Software source code to create a library, control or component, or a collection of components, for distribution as a product. You may not redistribute the original or modified Software source code.

e) You acknowledge that the Software source code is licensed "AS-IS", without warranty of any kind, and agree that Catalyst Development is under no obligation to provide technical support for any original or modified Software source code. You further acknowledge that Catalyst Development may modify the Software source code in the future and is not required to provide you with those modifications under the terms of this agreement.

5. TRADE SECRETS AND CONFIDENTIALITY

a) The Software contains information or material which is proprietary to Catalyst Development ("Confidential Information"), which is not generally known other than by Catalyst, and which you may obtain knowledge of through, or as a result of the relationship established hereunder with Catalyst. Without limiting the generality of the foregoing, Confidential Information includes, but is not limited to, the following types of information, and other information of a similar nature (whether or not reduced to writing or still in development): designs, concepts, ideas, inventions, specifications, techniques, discoveries, models, data, source code, object code, documentation, diagrams, flow charts, research, development, methodology, processes, procedures, know-how, new product or new technology information, strategies and development plans (including prospective trade names or trademarks).

b) Such Confidential Information has been developed and obtained by Catalyst by the investment of significant time, effort and expense, and provides Catalyst with a significant competitive advantage in its business.

c) You agree that you shall not make use of the Confidential Information for your own benefit or for the benefit of any person or entity other than Catalyst, except for the expressed purposes described in the paragraph hereof entitled "REDISTRIBUTION", in accordance with the provisions of this Agreement, and not for any other purpose.

d) You agree to hold in confidence, and not to disclose or reveal to any person or entity, the Software, other related documentation, your product Serial Number or any other Confidential Information concerning the Software other than to such persons as Catalyst shall have specifically agreed in writing to utilize the Software for the furtherance of the expressed purposes described in the paragraph hereof entitled "REDISTRIBUTION", in accordance with the provisions of this Agreement, and not for any other purpose.

e) You acknowledge the purpose of this section entitled "TRADE SECRETS AND CONFIDENTIALITY" is to protect Catalyst Development's ability to limit the use of the data and the Software generally to licensees, and to prevent use of Confidential Information concerning the Software by other developers or vendors of software.

6. OTHER RESTRICTIONS

You may not rent, lease or transfer the Software. You may not reverse engineer, decompile or disassemble the Software, except to the extent applicable law expressly prohibits the foregoing restriction. Without prejudice to any other rights, Catalyst Development may terminate this License Agreement if you fail to comply with the terms and conditions of the agreement. In such event, you must destroy all copies of the Software Product.

7. LIMITED WARRANTY

If within 30 days of your purchase of this software product, you become dissatisfied with the Software for any reason, you may return the software to Catalyst Development (or your dealer, if you did not purchase it directly from Catalyst) for a refund of your purchase price. To return the Software, you must contact Catalyst Development and obtain a return material authorization (RMA) number. Catalyst will not accept returns of opened or installed software without an RMA number. Returns are subject to the deduction from your purchase price of a 20% restocking fee and all shipping costs. You agree that this limited warranty does not apply to a Software source code license as specified in section 4.e.

8. NO OTHER WARRANTIES

CATALYST DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE ACCOMPANYING WRITTEN MATERIALS, AND ANY ACCOMPANYING HARDWARE.

9. LIMITATION OF LIABILITY

IN NO EVENT SHALL CATALYST OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITH LIMITATION, INCIDENTAL, CONSEQUENTIAL, SPECIAL, OR EXEMPLARY DAMAGES OR LOST PROFITS, BUSINESS INTERRUPTION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OR INABILITY OF THIS CATALYST PRODUCT, EVEN IF CATALYST HAS BEEN ADVISED OF SUCH DAMAGES.

APART FROM THE FOREGOING LIMITED WARRANTY, THE SOFTWARE PROGRAMS ARE PROVIDED "AS-IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. THE ENTIRE RISK AS TO THE PERFORMANCE OF THE PROGRAMS IS WITH THE PURCHASER. CATALYST DOES NOT WARRANT THAT THE OPERATION OF THE PROGRAMS WILL BE UNINTERRUPTED OR ERROR-FREE. CATALYST ASSUMES NO RESPONSIBILITY OR LIABILITY OF ANY KIND FOR ERRORS IN THE PROGRAMS OR DOCUMENTATION, OF/FOR THE CONSEQUENCES OF ANY SUCH ERRORS. THE LAWS OF THE STATE OF CALIFORNIA GOVERN THIS AGREEMENT.

10. GOVERNMENT-RESTRICTED RIGHTS

U.S. Government Restricted Rights. The Software and related documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer for such purposes is Catalyst Development Corporation, 56925 Yucca Trail, PMB254, Yucca Valley, CA 92284

11. EXPORT CONTROLS

If the Software is for use outside the United States of America, you agree to comply with all relevant regulations, including but not limited to those, of the United States Department of Commerce and with the United States Export Administration Act to insure that the Software is not exported in violation of United States law.

12. GOVERNING LAW

This License is governed by the laws of the State of California, without reference to conflict of laws principles. The exclusive jurisdiction and venue for any action arising from or relating to this agreement shall be in a court of competent jurisdiction in San Bernardino County, California. In addition to any other relief granted the prevailing party shall be entitled to recover its attorney’s fees and costs. THE PARTIES EXPRESSLY WAIVE THE RIGHT TO A TRIAL BY JURY. The parties acknowledge that any breach of this agreement may result in irreparable harm to Catalyst Development Corporation, thereby entitling Catalyst to injunctive relief for any such breach in addition to any other rights or remedies that Catalyst may have. This Agreement is the entire agreement between you and Catalyst and supersedes any other communications or advertising with respect to the Software and documentation. If any provision of this License is held invalid, the remainder of this License shall continue in full force and effect.

—Introduction—

Electronic mail is the most prevalent application in computer networking and its use has evolved beyond the simple exchange of text messages between two people. For the developer, e-mail provides a reliable means for sending and receiving messages where the protocols are based on well-known and widely used standards. The Catalyst Internet Mail control provides an interface to e-mail services, allowing developers to easily implement this functionality in their own software without requiring general knowledge of network programming or specific application protocols.

The Internet Mail control provides a single interface for composing, sending and retrieving e-mail messages. Instead of using separate controls to format, retrieve and send messages, the developer can use a single Internet Mail control for much of the same functionality without the inherent complexity and coding. For most applications, this is the only control that will be needed to process e-mail messages. However, in some cases a program may require the advanced features of a specific SocketTools control, such as sending extended authentication commands or server-specific options. In this situation, the Internet Mail control can be seamlessly integrated with the other SocketTools controls to build a more complex solution that requires a greater degree of customization.

The control is implemented as a standard COM object and is designed to be used in visual development tools as well as various scripting environments. Any programming language which can host ActiveX controls or create instances of a COM object should be capable of using the Internet Mail control, such as Visual Basic, Visual C++, Visual FoxPro and Delphi. Server and client-side scripting is also supported using languages such as VBScript and JScript. The control is completely self-contained and does not require developers to redistribute the Microsoft Foundation Classes (MFC) or Visual C runtime libraries, nor any other third-party library.

Protocol Standards

There are five core standards which form the foundation for sending and receiving e-mail messages over the Internet and corporate intranets. These standards are defined in documents called RFCs (Request For Comments) which describe how the various protocols should be implemented. The following standards were used when implementing the Internet Mail control:

RFC 822 documents the basic structure of e-mail messages, including how messages should be formatted and what the standard message header fields are. RFC 2045 documents Multipurpose Internet Mail Extensions (MIME), which details how more complicated messages are structured. File attachments, HTML formatted messages and other more complex aspects of message composition are covered by the MIME standard. The Internet Mail control supports both RFC 822 and MIME formatted e-mail messages, including multipart messages which contain alternate text and file attachments.

RFC 1939 documents the Post Office Protocol (POP3) which is used to retrieve messages from a user's mailbox on a server. The Internet Mail control uses this protocol to enable applications to list, retrieve and delete messages.

RFC 821 documents the Simple Mail Transfer Protocol (SMTP) which is used to deliver messages to one or more recipients. RFC 1869 documents extensions to the protocol which provide additional services such as delivery status notification and authentication. The Internet Mail control implements both the standard and extended SMTP protocols.

Licensing and Redistribution

The Catalyst Internet Mail license permits the use of the control to build application software and redistribute that software to end-users. There are no restrictions on the number of products in which the control may be used. However, if it has been installed with an evaluation license, any products built using it cannot be redistributed to another system until a licensed copy of the toolkit has been purchased and registered.

System Requirements

The Catalyst Internet Mail control is designed for the 32-bit Windows platform, and is supported on Windows XP, Windows 2000, Windows NT 4.0, Windows ME and Windows 98. For Windows 98 and Windows ME, it is recommended that the system have at least 32 MB of physical memory. For Windows NT 4.0, it is recommended that the system have at least 64 MB of physical memory and it is required that the system have Service Pack 6 (SP6) installed. For Windows 2000 and Windows XP, it is recommended that the system have at least 128 MB of physical memory. It is recommended that Windows 2000 have at least Service Pack 2 (SP2) installed, but this is not a requirement.

Control Redistribution

For those applications created using the Internet Mail control, the CSIMXCTL.OCX file must be distributed along with the application and the control must be registered by the installation program. The process of registration means that specific entries must be created in the system registry which provide information about the control such as the location of the OCX file. Fortunately, ActiveX controls are self-registering which means that the control has the ability to create or update those registry entries itself.

To take advantage of this, the installation program must be capable of loading the control and calling those functions inside the control which update the registry. Most modern installation tools are capable of registering ActiveX controls. For in-house setup programs, refer to the technical article on ActiveX Control Registration on the Microsoft Developers Network CD.

It is possible to register ActiveX controls manually without the use of an installation program. This may be desirable in those situations where an application is being deployed internally or the developer does not want to create a setup program for a limited distribution. The tool used to manually register a control is named **regsvr32** and can be obtained from a number of places including the Visual Basic or Visual C++ CD-ROM. This utility accepts a command line argument which specifies the name of the control to register. For example:

```
C:>REGSVR32 C:\WINDOWS\SYSTEM\CSIMXCTL.OCX
```

A message box would be displayed indicating that the control was registered successfully. To prevent the message box from being displayed, use the **/S** option which tells the utility to function silently. If an error is reported, typically the reason is that a required system DLL is missing or out of date. Note that the above command should reference the system32 directory, instead of system, if being used under Windows NT.

Version Information

The Internet Mail control has embedded information which provides version information to an installation utility. This information called the version resource, specifies the control's version number among other things. If you are using a third-party or in-house installation program, it is extremely important that the program knows how to use this information.

For example, if you are deploying an application which uses the control, the setup program must determine if it has already been installed on the target system. If it has, it must compare the version resource information in the two files. It should only overwrite the control if the version that you have included with your application is later than the one installed on the system. An installation program which overwrites the file without checking the version number may cause other programs to fail unexpectedly on the end-user's system, which is obviously not desirable.

Installation Directory

The Internet Mail control should be installed in the system directory on the local machine. Note that under Windows NT/2000/XP, the system32 directory should be used instead, since that is where 32-bit controls and support libraries are installed. Some developers may prefer to install the library along with their application in a private directory. It is not recommended that developers take this approach because the full pathname of the control file is stored in the system registry when it's registered. If multiple applications install the same control in different directories, the actual control that will be used is the one that was last registered. This means that it is possible that an application will load an earlier version of the control than it was built with, which may result in unexpected or fatal errors.

—Internet Mail Overview—

The Internet Mail control has properties and methods which are generally used to perform three basic functions: composing messages, retrieving messages from a mail server, and sending messages to one or more recipients. The interface is designed to be simple and intuitive, yet flexible enough to handle a wide variety of development needs. Everything is centered around the concept of a *current message*, with various attributes accessible through the control's properties. The structure of the message can be examined or changed dynamically using various properties and methods. The current message can be changed either by composing a new message, importing a message from a file, or by retrieving a message from a mail server. The message can then be sent to one or more recipients, either based on a specific list of addresses or using the addresses that have been included in the message itself.

Composing Messages

Many of the control properties are designed to give the developer access to the internal structure of the current message. To understand how these properties can be used, it's useful to understand how a message is actually formatted. Here is an example of a simple, plain text e-mail message:

```
From: John Doe <johndoe@company.com>
To: Jane Doe <janedoe@company.com>
Date: Mon, 1 Jul 2002 12:00:00 -0800 (PST)
Subject: Meeting scheduled for next week
Message-ID: <20020601200000.15637@mail01.company.com>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii

I wanted to confirm that you would be able to attend the meeting.
If there are any scheduling conflicts, please let me know.
```

The first thing that is apparent is that the message has two discrete sections. The first section consists of one or more header fields, followed by a colon and then a value. The second section contains the body of the message, with the headers and body separated by a single blank line.

Therefore, using this example message, reading the control's **From** property would return the string "John Doe <johndoe@company.com>", which is the address of the person who sent the message. To change the From header field, simply set the **From** property to a new string value.

The following is a complete list of properties that can be used to read, create or modify a message:

Property	**Description**
Attachment	The name of a file attachment in the current message part.
Bcc	One or more message recipients (blind carbon copy).
Cc	One or more message recipients (carbon copy).
Content-ID	The content identifier for the current message part.
ContentLength	The size of the current message part in bytes.
ContentType	The content type for the current message part.
Date	The date for the current message.
Encoding	The encoding type for the current message part.
From	The sender of the message.
Localize	Enable or disable message localization.
Mailer	The name of the application that generated the message.
Message	The complete message, including headers and body.
MessageID	A unique identifier string from the current message.
MessagePart	The current message part in a multipart message.
MessageParts	The number of parts in a multipart message.
MessageSize	The size of the current message in bytes.
MessageText	The text in the current message part.
Organization	The name of the sender's organization or company.
Priority	The current message priority.
Recipient	The address of one of the message recipients.
Recipients	The number of recipients for the current message.
ReplyTo	The address to which replies should be sent.
ReturnReceipt	The address to which a return-receipt message should be sent.
Subject	The subject of the current message.
TimeZone	The current timezone offset for the local system.
To	One or more message recipients.

Most of the message-related properties correspond to specific header fields, such as To, From and Subject. Reading those properties return their respective header values while setting them changes their value in the current message.

For more complex message processing such as attaching files or creating multipart messages, there are a number of additional methods which can be used to manage the current message:

Method	**Description**
AppendMessage	Append text to the current message.
AttachFile	Attach a file to the current message.
ComposeMessage	Compose a new message.
ClearMessage	Clear the contents of the current message.
CreatePart	Create a new message part in a multipart message.
DeleteHeader	Delete a header from the message.
DeletePart	Delete a message part.
ExportMessage	Export the complete message to a text file.
ExtractFile	Extract a file attachment.
GetFirstHeader	Return the first header in the current message part.
GetHeader	Return the value of a specified header field.
GetNextHeader	Return the next header in the current message part.
ImportMessage	Import a message from a text file.

ParseMessage	Parse a string, adding the contents to the current message.
SetHeader	Set the value of the specified header field.

The header related methods such as **GetHeader** and **SetHeader**, enable an application to read, create or modify any header field regardless of whether or not there is a predefined property value for it. Because there can be a potentially unlimited number of header fields in a message, these methods give the developer more control over the header portion of the message.

New messages can be created by setting properties which comprise the message. Here is an example of some Visual Basic code which would create a short message:

```
InternetMail1.From = "johndoe@company.com"
InternetMail1.To = "janedoe@company.com"
InternetMail1.Date = Date
InternetMail1.Subject = "This is the message subject"
InternetMail1.MessageText = "This is an example of a new message"
```

The resulting message would look like this:

```
From: johndoe@company.com
To: janedoe@company.com
Date: Fri, 01 Nov 2002 12:00:00 -0800
Subject: This is the message subject
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

This is an example of a new message
```

Note that in addition to those properties that were set, there were a number of additional header fields such as MIME-Version and Content-Type that were automatically created.

Although setting properties is one way to create a new message, it involves writing a fair amount of code. There is a simpler way to do it using a single method called **ComposeMessage**. The equivalent code would look like this:

```
InternetMail1.ComposeMessage "johndoe@company.com", _
      "janedoe@company.com", , , "This is the message subject", _
      "This is an example of a new message"
```

Once the message has been created, it can be further modified by setting properties or calling methods such as **SetHeader**.

Now that a simple message has been created, let's attach a file to the message. This can be easily done using the **AttachFile** method:

```
InternetMail1.AttachFile "c:\temp\image.gif"
```

Although this is a simple operation, it makes some significant changes to the message (some portions of the attachment data has been omitted):

```
From: johndoe@company.com
To: janedoe@company.com
Date: Fri, 01 Nov 2002 12:00:00 -0800
Subject: This is the message subject
MIME-Version: 1.0
Content-Type: multipart/mixed;
  boundary="----=_ST4020_0001_0BCF2D17_179E5A2E"
Content-Transfer-Encoding: 7bit
```

```
This is a multi-part message in MIME format.

------=_ST4020_0001_0BCF2D17_179E5A2E
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

This is an example of a new message
------=_ST4020_0001_0BCF2D17_179E5A2E
Content-Type: image/gif
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="image.gif"
Content-Length: 6434

R0lGODlhRgEyAPcAAJaWqqKitp6esqamuqamtrKywq6uvsrK176+yrq6xra2wsbG0sLCztLS
287O18LCytvb49fX38rK0sbGzt/f5/f3+/Pz9+/v8+vr7+fn6+Pj59/f49vb311hil1hhmFl
uXvrtV8rFWQ7l2qbuZJrt3rLFomrFovruRRhq4bbuZwbu1hLubBbuZF7tn7WuE2RurLWu7DL
u3nrt7Kbr3LbM+Pnu6hrFC2atcfburarucL7uY/Lurl7uVpLu5CLvJyLu9obvZO7u900W7jA
K7jkW72maxHGC72kWxbAu7xuq77DS7ip67qgl77va73fqxGryxDMK7/m67zgixEBAQA7

------=_ST4020_0001_0BCF2D17_179E5A2E--
```

The message has now become a multipart message that contains both human-readable text as well as data for the file attachment. Rather than having a single group of headers followed by a message body, the message is now broken into sections, each with its own group of headers and body. Each of these sections are called a message part, and can be accessed individually using the **MessagePart** property. Each message part is identified by a part number which starts at zero and increases for each subsequent part. Part 0 of this message consists of the following:

```
From: johndoe@company.com
To: janedoe@company.com
Date: Fri, 01 Nov 2002 12:00:00 -0800
Subject: This is the message subject
MIME-Version: 1.0
Content-Type: multipart/mixed;
  boundary="----=_ST4020_0001_0BCF2D17_179E5A2E"
Content-Transfer-Encoding: 7bit

This is a multi-part message in MIME format.
```

Part 0 of any message always refers to the headers and body of the main message. In the previous message, part 0 contains the entire message. Here, part 0 consists primarily of headers and a brief message that this is now a multi-part message. This is automatically done for the benefit of older mail clients which may not understand a MIME formatted message, so the user has a message that at least identifies what the message is. Another thing that has changed is the value of the Content-Type header. In the previous message it had a value of "text/plain; charset=us-ascii" which tells the mail client that this is a plain text message. With the file attachment, this has changed to a type called "multipart/mixed" which indicates that the message contains multiple parts with mixed types of information. The boundary value is what is used to actually designate the different parts of the message. As the message is being processed, the mail client knows that it has found a new message part when the boundary string is encountered.

The next part of the message, part 1, contains the message that was in the original version of the message:

```
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

This is an example of a new message
```

Note that the content type is back to plain text, just as it was with the original. When a mail client processes a message, it scans the message for plain text message parts which contain information to be displayed to the user.

The last part of the message, part 2, contains the actual file data that was attached to the message:

```
Content-Type: image/gif
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="image.gif"
Content-Length: 6434
```

```
R0lGODlhRgEyAPcAAJaWqqKitp6esqamuqamtrKywq6uvsrK176+yrq6xra2wsbG0sLCztLS
287O18LCytvb49fX38rK0sbGzt/f5/f3+/Pz9+/v8+vr7+fn6+Pj59/f49vb311hil1hhmFl
uXvrtV8rFWQ7l2qbuZJrt3rLFomrFovruRRhq4bbuZwbu1hLubBbuZF7tn7WuE2RurLWu7DL
u3nrt7Kbr3LbM+Pnu6hrFC2atcfburarucL7uY/Lurl7uVpLu5CLvJyLu9obvZO7u900W7jA
K7jkW72maxHGC72kWxbAu7xuq77DS7ip67qgl77va73fqxGryxDMK7/m67zgixEBAQA7
```

Here the Content-Type header tells the mail client that this is an image file in the GIF format. The other header fields in this message part are used by applications to extract the file attachment once it has been delivered to the recipient. Because e-mail messages must be sent over systems which may not be able to handle binary data, the image file data has been *encoded* using a standard algorithm called base64. This algorithm converts binary data into plain 7-bit text data that can be safely exchanged with other mail servers. The process of encoding and decoding attachments is automatically handled by the control when the file is attached. The **ExtractFile** method is essentially the reverse of the **AttachFile** method, automatically decoding and storing a file attachment on the local system.

Sending Messages

Sending an e-mail message is a simple process using the Internet Mail control, and typically involves calling a single method, **SendMessage**. Using the message that was created in the previous section, let's send the message:

```
InternetMail1.SendMessage
```

In most cases, that is all that’s required to send the message. The control will automatically extract the e-mail addresses from the **To**, **Cc**, and **Bcc** properties and deliver the message to those recipients. However, there may be circumstances where you wish to send a message to a different set of addresses. The **SendMessage** method has several optional arguments that enable you to control the sender, recipient and even the contents of the message itself.

Mail Exchanges

When a message is delivered to a user, the Internet Mail control must determine what mail server is responsible for accepting messages for that user. This is accomplished using the Domain Name Service (DNS), a protocol that is most commonly used to resolve host names such as www.microsoft.com into Internet addresses. This is typically accomplished by sending a request to a nameserver, a computer system that provides domain name services. In addition to resolving host names, nameservers can also provide information about those servers which are responsible for accepting mail for a given domain. There can be multiple servers which process mail for a domain with each server assigned a priority as part of their mail exchange (MX) record. If there is no mail exchange record for a domain, then the domain name itself is used.

The **SendMessage** method examines each recipient address and queries the nameserver for the domains which are responsible for accepting mail for users in that domain. If there are multiple mail servers, they are sorted in order of their listed priority. The control then attempts to establish a connection with the server and to deliver the message. If a connection cannot be established, the next mail exchange server is selected from the list. If the message cannot be delivered, then an error is generated and the next recipient is processed.

The ability to query a nameserver to determine the name of the server responsible for accepting mail for a given domain is handled internally by the control, and in most cases, is completely transparent. However, there may be situations in which a specific set of nameservers must be used, most commonly with corporate intranets. In this case, the **NameServer** property array can be used to control what nameservers are used to perform MX record queries. By default, this property contains the nameservers that were assigned to the local host. Changing these values changes the nameservers that are used. For example, if a company had two nameservers for their intranet with the IP addresses of 192.168.31.1 and 192.168.32.1, the control could be instructed to use those nameservers with the following statements:

```
InternetMail1.NameServer(0) = "192.168.31.1"
InternetMail1.NameServer(1) = "192.168.32.1"
```

Note that only Internet (IP) addresses may be assigned to the **NameServer** property array. Attempting to specify a domain name will result in an error. To restore the original default nameservers used by the control, either save their original value before modifying the **NameServer** property or call the **Reset** method.

Relay Servers

In some situations it may not be possible to send mail directly to the server that accepts mail for a given domain. The two most common situations are corporate networks which have centralized servers that are responsible for delivering and forwarding messages, or an Internet Service Provider (ISP) which specifically blocks access to all mail servers other than their own. This is usually done as either a security measure or as a means to inhibit users from sending unsolicited commercial e-mail messages. If the standard SMTP port is being blocked, then any connection attempts will either fail immediately with an error that the server is unreachable, or the connection will simply time-out. In either case, a relay server must be specified in order to send e-mail messages.

A relay server is a system which will accept messages addressed to users which may be in a different domain, and will relay those messages to the appropriate server that does accept mail for the domain. The **RelayServer** property specifies the host name or address of the mail server that will be used by the control to deliver a message. If this property is not set, then the **SendMessage** method will perform the normal MX record queries and attempt to make a direct connection to each mail server. If the **RelayServer** property is set, then the control always establishes a connection to that server and does not perform any MX record queries. The **RelayPort** property can be used to specify an alternate port number on which the relay mail server is accepting connections. If this property is set to a value of zero, then the default port number will be used.

It is important to note that using a mail server as a relay without the permission of the organization or individual who owns that server may violate Acceptable Use Policies and/or Terms of Service agreements with your service provider. Systems which relay messages from anyone, regardless of whether the message is coming from a recognized domain, are called *open relays*. Because open relays are often used to send unsolicited e-mail, many administrators block mail that comes from one. It is recommended that users check with their network administrators or Internet service providers to determine if access to external mail servers is restricted and what is the acceptable use policy for relaying messages through their mail servers.

Retrieving Messages

In addition to composing and sending e-mail messages, the Internet Mail control has the ability to retrieve a specific user's messages from a mail server. There are a number of properties which are used to access and manage a user's mailbox:

Properties	Description
LastMessage	The last message number available in the mailbox.
MailboxSize	The size of the current mailbox.
MessageCount	The number of messages available in the mailbox.
MessageIndex	The current message number.
MessageUID	The unique identifier for this message on the server.
Password	The password used to authenticate access to the mailbox.
ServerName	The host name or address of the mail server.
ServerPort	The port number for the mail server.
Timeout	The timeout period for retrieving messages.
UserName	The username used to authenticate access to the mailbox.

The **ServerName**, **ServerPort**, **UserName** and **Password** properties are used in establishing a connection with the mail server and authenticating the specified user. The **Timeout** property determines the amount of time the control spends waiting for an operation to complete before returning an error. The remaining properties are used to manage the mailbox or return information about the mailbox, such as the number of messages that are available to be retrieved.

There are also several methods available which can be used to access the user's mailbox:

Methods	Description
Connect	Connect to the specified mail server.
DeleteMessage	Delete a message from the current mailbox.
Disconnect	Disconnect from the mail server.
GetMessage	Get a complete message from the current mailbox.
StoreMessage	Get a complete message and store it as a text file.

The **Connect** method establishes a connection with the mail server and has a number of optional arguments. In its simplest form, a connection can be established like this:

```
InternetMail1.ServerName = strMailServer
InternetMail1.UserName = strUserName
InternetMail1.Password = strPassword

nError = InternetMail1.Connect()
If nError <> 0 Then
    MsgBox "Unable to connect to mail server" & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
    Exit Sub
End If

nMessages = InternetMail1.MessageCount
InternetMail1.Disconnect

MsgBox "There are " & nMessages & " messages " & _
       "in this mailbox", vbInformation
```

Once a connection has been established, the **MessageCount** property will return the number of messages that are currently available and the **LastMessage** property will return the total number of messages that were available when the session began. This is an important distinction because as messages are deleted from the mailbox, the **MessageCount** value will decrease while the **LastMessage** value remains unchanged for the current session.

The **MessageIndex** property is used to select the current message in the mailbox. Once a message has been selected, the header for the message is retrieved, but not the entire contents. This allows an application to use the various header-related properties and methods without incurring the overhead of downloading the complete message. For example, the following code would add the subject of each message in the mailbox to a list control:

```
For nMessage = 1 To InternetMail1.LastMessage
    InternetMail1.MessageIndex = nMessage
    List1.AddItem InternetMail1.Subject
Next
```

The **GetMessage** method is used to retrieve a complete message including the headers, body and any attachments. As the message is being downloaded, the **OnProgress** event will fire, notifying the application of how much of the message has been retrieved. Once the complete message has been downloaded, the **GetMessage** method will return and the program can continue processing the message. The following code retrieves all of the messages from a mailbox, and then scans each message to determine if it contains any file attachments:

```
For nMessage = 1 To InternetMail1.LastMessage
    ' Retrieve the message
    InternetMail1.GetMessage nMessage

    ' For each part of the message, check the value of the
    ' Attachment property; if it has a value, then this is
    ' the name of a file attachment in the current message part

    For nPart = 0 To InternetMail1.MessageParts - 1
        InternetMail1.MessagePart = nPart
        strFileName = InternetMail1.Attachment
        If Len(strFileName) > 0 Then
            ' This message part is an attachment, it could be
            ' extracted, the name added to a listbox, etc.
        End If
    Next
Next
```

The **StoreMessage** method is similar to **GetMessage** in that it retrieves a message from the server, but instead of loading the message into the control, it stores the message as a text file. This will not change the current message, nor will it cause the message to be processed in any way by the control; the data is stored in the file exactly as it is downloaded from the server. This can be useful if the application wants to store messages locally for processing later. The following example demonstrates how **StoreMessage** can be used:

```
For nMessage = 1 To InternetMail1.LastMessage
    strFileName = "c:\temp\msg" & Format(nMessage, "00000") & ".txt"
    nError = InternetMail1.StoreMessage(nMessage, strFileName)
    If nError <> 0 Then
        MsgBox "Unable to store message " & nMessage & vbCrLf & _
               InternetMail1.LastErrorString, vbExclamation
        Exit For
    End If
Next
```

Once a message has been retrieved, either for processing or storage, the application may wish to remove the message from the mailbox by calling the **DeleteMessage** method. This method marks the specified message for deletion, and when the session is closed (by calling the **Disconnect** method), the message is permanently deleted from the mailbox. Once a message has been marked for deletion, attempts to access the message will generate an error. To prevent the message from actually being deleted, the application must call the **Reset** method instead of **Disconnect**. This will reset the state of the mailbox, preventing any messages that were marked for deletion from actually being removed.

Regardless of whether messages are being searched, downloaded, or removed, keep in mind that in order to access those messages an active connection to the mail server must be established. A consequence of this is that most servers will disconnect sessions that become inactive for too long. In this case, an error would be returned indicating that the connection has been closed. It is generally recommended that applications allow minimal user interaction while retrieving messages. Instead, process each message in the mailbox and then allow the user to review or modify those messages locally, once they have all been downloaded.

Consider a simple routine which retrieves each mail message in a user's mailbox, saves it to a file if the subject contains the word 'failure', and then displays a message box warning the user:

```
For nMessage = 1 To InternetMail1.LastMessage
    InternetMail1.GetMessage nMessage
    If InStr(1, InternetMail1.Subject, "Failure", vbTextCompare) > 0 Then
       strFileName = "c:\temp\err" & Format(nMessage, "00000") & ".txt"
       InternetMail1.ExportMessage strFileName, mailOptionAllHeaders
       MsgBox "Stored message " & nMessage & " in " & strFileName
    End If
Next
```

Although this code would appear to execute correctly, the problem is that if the user waits too long to press the OK button on the message box, the session can time-out and the server will drop the connection. When the next iteration of the loop calls the **GetMessage** method, an error will be returned and the program will fail.

A better implementation would be to download all of the messages, store them, disconnect from the server, and then begin processing them. The simplest approach would be to use a string array in which to store each message as follows:

```
nMessages = InternetMail1.LastMessage
ReDim strMessage(nMessages)

For nMessage = 1 To nMessages
    InternetMail1.GetMessage nMessage
    strMessage(nMessage) = InternetMail1.Message
Next

InternetMail1.Disconnect
```

Now that all of the messages are stored in the *strMessage* array, they can be reloaded into the control by setting the **Message** property:

```
For nMessage = 1 To nMessages
    InternetMail1.Message = strMessage(nMessage)
    If InStr(1, InternetMail1.Subject, "Failure", vbTextCompare) > 0 Then
       strFileName = "c:\temp\err" & Format(nMessage, "00000") & ".txt"
       InternetMail1.ExportMessage strFileName, mailOptionAllHeaders
       MsgBox "Stored message " & nMessage & " in " & strFileName
    End If
Next
```

The one limitation of this approach is that the contents of each message is stored entirely in memory. For very large messages, a better approach would be to store the message in a temporary file and then use the **ImportMessage** method to load the message into the control, deleting the file when it's no longer needed.

—QuickStart Guide—

This section is provided as a means to quickly getting started with the Internet Mail control. The examples provided in this section presume some familiarity with the Visual Basic programming language, however the basic concepts are the same regardless of what language is used. Please refer to the technical reference for complete information on all of the properties, methods and constants used by the control. Before performing any of the steps in this guide, you should have installed the Internet Mail control on your development system.

To include the control in your project in Visual Basic, simply select the **Project|Controls** menu option and select the Catalyst Internet Mail Control. In other languages, follow the normal steps that are taken to include an ActiveX control in your development project.

Sending Text Messages

To compose and send a text e-mail message to one or more recipients, you only need to use two methods, **ComposeMessage** and **SendMessage**. Here is an example of how they could be used to send a message using the information entered by the user on a form:

```
Private Sub cmdSend_Click()
    Dim nError As Long

    cmdSend.Enabled = False
    nError = InternetMail1.ComposeMessage(editFrom.Text, _
                                          editTo.Text, _
                                          editCc.Text, _
                                          editBcc.Text, _
                                          editSubject.Text, _
                                          editMessage.Text)
    If nError Then
        MsgBox "Unable to compose a new message" & vbCrLf & _
               InternetMail1.LastErrorString, vbExclamation
        Exit Sub
    End If

    nError = InternetMail1.SendMessage()

    If nError Then
        MsgBox "Unable to send the message" & vbCrLf & _
               InternetMail1.LastErrorString, vbExclamation
        Exit Sub
    End If

    cmdSend.Enabled = True
End Sub
```

The **ComposeMessage** method creates a new message and has the following arguments:

Property	**Description**
From	A string value which specifies the e-mail address of the person sending the message. This argument is required.
To	A string value which specifies one or more e-mail addresses of those who will receive the message. Multiple addresses may be separated by a comma (such as "john@company.com, jane@company.com"). This argument is required.
Cc	An optional string value which specifies recipients who should receive a copy of the message. Multiple recipients may be separated by a comma and the addresses are included in the header of the message.

Bcc	An optional string value which specifies recipients who should receive a copy of the message, however, these addresses are not included in the header of the message. Multiple recipients may be separated by a comma.
Subject	An optional string value which specifies the subject of the message. If the argument is not specified then the message is created without a subject.
MessageText	An optional string value which specifies the body of the message. If the argument is not specified, then the message is created without a body.
MessageHTML	An optional string value which specifies an HTML version of the message. If this argument is provided along with the MessageText argument, then a multipart message is created which contains both plain text and HTML versions of the message. If the MessageText argument has not been specified, then only an HTML message is created. If this argument is omitted, then the message is sent with only a plain text body.
CharacterSet	An optional integer value which specifies a character set to use when composing the message. This typically only needs to be set for languages which use extended characters. For more information on the available character sets, consult the technical reference.
EncodingType	An optional integer value which specifies an encoding type to be used with the character set that was selected. For more information about the encoding types available, consult the technical reference.

The **SendMessage** method has four optional arguments:

Property	Description
Sender	An optional string value which specifies the sender of the message. If this argument is provided, then the message will be identified as being sent from this address. If the argument is omitted, then the value of the From property is used. Note that specifying a different sender address does not change the contents of the message.
Recipient	An optional string value which specifies one or more recipient e-mail addresses. Multiple addresses may be separated by a comma. If this argument is provided, then the message will be delivered only to those recipients, regardless of the addresses listed in the message itself. If the argument is omitted, then the addresses listed in the To, Cc and Bcc properties are used. Note that specifying a different recipient does not change the contents of the message.
Message	An optional string value which specifies a complete, formatted e-mail message including all headers and the body. If this argument is provided, then the message is used instead of the current message; otherwise the current message will be sent. If this argument has been specified, but the Sender or Recipient arguments have been omitted, the message is parsed and the values of the From, To and Cc header fields are used as appropriate. It is expected that this string contain a complete, correctly formatted message which conforms to the RFC 822 and MIME e-mail standards.
Options	A long integer value which specifies one or more options to be used when sending the message. If this argument is omitted, the value of the Options property is used. The most common option that is used is mailOptionNotify which enables delivery status notification (DSN) if it is supported by the server. This tells the server to return a message back to the sender indicating the message has been delivered or if there were any errors encountered during delivery. Note that if the server does not support DSN, this option is ignored and will not affect the delivery of the message itself.

In most cases, the optional arguments to the **SendMessage** method will not be used. They are primarily provided for advanced applications which need to control the sender, recipient or message contents that are being sent to the SMTP server.

Message Delivery Events

Once the **SendMessage** method is called, three different events will fire in sequence for each recipient. The first event is **OnRecipient** which has two arguments. The first is the recipient address, and the second is a boolean variant which can be used to prevent delivery of the message to that address. For example, to prevent a message from being sent to anyone with a hotmail.com address, the following code could be used:

```
Private Sub InternetMail1_OnRecipient(ByVal Address As Variant, _
                                      Cancel As Variant)

    If InStr(1, Address, "@hotmail.com", vbTextCompare) Then
        Cancel = True
    End If

End Sub
```

If delivery to an address is prevented by setting the *Cancel* argument to True, the message will continue to be delivered to any remaining recipients. Preventing delivery will not change the contents of the message so if a recipient address is included in the message headers, it will still be visible to any other recipients even though the message was not actually delivered to that address. To cancel delivery of the message and all subsequent recipients, use the **Cancel** method instead.

Once a connection with the mail server has been established, the **OnProgress** event will begin firing, indicating how much of the message has been delivered. It should be noted that this does not specify the overall progress for multiple recipients but rather the progress in delivering the message for that specific address. To display an overall progress, an application would have to use the **OnProgress** information in conjunction with the total number of recipients to whom the message is being sent. The **Recipients** property returns the number of recipients for the current message and the **Recipient** property array allows an application to enumerate all of the recipient e-mail addresses.

After the message has been delivered to the mail server, the **OnDelivered** event is fired which specifies the recipient address and the size of the message that was sent. It should be noted that there are still circumstances in which a message can be accepted by a mail server but not actually delivered to the user. A server may decide to reject or re-route a message based on its own internal configuration, content filters or message routing rules. To confirm that a message has actually been delivered, set the **Options** property to *mailOptionNotify* to enable delivery status notification. Setting the **ReturnReceipt** property to the sender's e-mail address is another option although this depends on the recipient's software automatically generating the return-receipt message after it has been read.

Sending HTML Messages

To send an HTML formatted e-mail message, the **ComposeMessage** method can be used, similar to how plain text messages are sent. For example, consider the following HTML text:

```
<html>
<head></head>
<body>
<font face="Arial">
<h3>Test HTML Message</h3>
This is a test message which uses HTML to format the text. This
message was created using the <b>Internet Mail</b> control from
<a href="http://www.catalyst.com/">Catalyst Development</a>.
</font>
</body>
```

```
</html>
```

You could either assign this text to a string, or you could read the message from a file using code like this:

```
hFile = FreeFile()
Open strMessageFile For Input As hFile
strMessageHTML = Input(LOF(hFile), hFile)
Close hFile
```

Where *strMessageFile* contains the HTML message you wish to send. To compose the HTML formatted e-mail, simply call the **ComposeMessage** method as you did with the plain text message, except that instead of passing the message to the *MessageText* argument, you pass it to the *MessageHTML* argument:

```
nError = InternetMail1.ComposeMessage(editFrom.Text, _
                                      editTo.Text, _
                                      editCc.Text, _
                                      editBcc.Text, _
                                      editSubject.Text, _
                                      "", _
                                      strMessageHTML)
```

The message that will be sent will now be displayed to the recipient using HTML and will include the formatting (such as font and text size) as well as the hyperlink. However, not all mail clients are capable of displaying HTML e-mail. This poses a problem because the message that they'll receive will be the largely unreadable HTML source. To resolve this problem, create both a plain text version of the message along with the HTML version. Ideally it would contain similar content, although you could provide a simple message which says that this is an HTML e-mail and they should request a plain-text version if they can't display HTML messages. In either case, simply provide both the *MessageText* and *MessageHTML* arguments:

```
nError = InternetMail1.ComposeMessage(editFrom.Text, _
                                      editTo.Text, _
                                      editCc.Text, _
                                      editBcc.Text, _
                                      editSubject.Text, _
                                      strMessageText, _
                                      strMessageHTML)
```

This will create what is called a multipart/alternative MIME message which contains both plain text and HTML versions of the message. Mail clients which are capable of displaying the HTML message will use that version, while those that cannot will display the plain text version.

It should be noted that there are still some mail clients which do not understand multipart/alternative messages and therefore will display both the plain text and the HTML source text. While confusing, the plain text version will ensure that the message is still readable. For the most part, e-mail is still a plain text medium so if you consider readability and compatibility with older mail software to be more important than formatted text, it is recommended that you use only plain text messages. However, if you know that the recipients have mail clients that are capable of displaying HTML, the Internet Mail control makes this easy to do.

Attaching Files

In addition to sending text messages, e-mail is commonly used as a means to exchange files. This can be easily done using the **AttachFile** method. Let's modify the previous example, presuming that an edit control has been included on the form which allows a user to input the name of the file they wish to attach. Add this after the call to the **ComposeMessage** method:

```
' If a file name has been entered, then attach it to
' the message that was composed

If Len(editFileName.Text) > 0 Then
     nError = InternetMail1.AttachFile(editFileName.Text)

     If nError Then
         MsgBox "Unable to attach file " & editFileName.Text & _
                vbCrLf & InternetMail1.LastErrorString, vbExclamation
     Exit Sub
 End If
```

If the file does not exist or cannot be accessed, then the **AttachFile** method will return an error. Otherwise, the file data will be encoded and attached to the message. The **AttachFile** method has two arguments:

Property	Description
FileName	A string which specifies the name of the file to attach. This argument is required, and must be a file which the current user has permission to open and access for reading.
Options	An optional long integer which specifies how the file is to be attached to the current message. If this argument is omitted, the method used is based on the file type and the contents of the file.

In most cases, it is not necessary to specify the *Options* argument since the **AttachFile** method will automatically determine the correct encoding method based on the contents of the file. However, there are some situations in which you may wish to use some specific encoding method. For example, you may want to force the control to use base64 encoding even though the attachment is a plain text file. To do this, you can use one of the following values:

Constant	Description
mailAttachBase64	The base64 algorithm is used to encode the file data. This is the default encoding type used for binary data such as executables, image or audio files.
mailAttachUucode	The uuencode algorithm is used to encode the file data. This is an older encoding type that was commonly used before the MIME standard was developed. It is not recommended that you use this encoding method unless specifically required by an application.
mailAttachQuoted	The quoted-printable algorithm is used to encode the file data. This should only be used to encode text files which may contain non-printable or extended ANSI characters. Using this format on binary files may cause them to become corrupted when extracted by the recipient.

For example, if you want to always have the attached file encoded using the base64 algorithm, the code would be changed to look like this:

```
' If a file name has been entered, then attach it to
' the message that was composed

If Len(editFileName.Text) > 0 Then
     nError = InternetMail1.AttachFile(editFileName.Text, _
                                     mailAttachBase64)
     If nError Then
         MsgBox "Unable to attach file " & editFileName.Text & _
                vbCrLf & InternetMail1.LastErrorString, vbExclamation
     Exit Sub
 End If
```

When attaching a file, keep in mind that the size of the attachment in the message will typically be about 33% larger than the size of the file itself. This is an important consideration because most mail servers restrict the size of the messages they will accept and will reject messages that exceed that limit. For example, if a mail server restricts messages to 5 megabytes, the maximum size of a file that can be attached to the message is about 3.5 megabytes.

Another consideration with file attachments is compatibility with third-party mail client software. If the current message contains alternative messages (i.e., both plain text and HTML text) then **AttachFile** will change the message structure, creating a more complex multipart message which has mixed content types. Mail software which does not fully conform to the MIME standard may not be able to correctly display this type of message, either being unable to display the body of the message, or, displaying the complete message including the alternate text and the encoded file attachment. To ensure that your message is readable by most recipients, it's recommended that files are attached to plain text messages.

Importing Messages

In addition to using the **ComposeMessage** method to create a new message, it is possible to import existing messages into the control. These messages may exist either as text files already stored on the local system, records in a database, or may even be created dynamically by the application using data from other sources. The most important thing to keep in mind is that any message which is imported into the control must adhere to the basic structure outlined previously in the section discussing message composition. The control is tolerant of malformed messages, however, importing a corrupted message will often produce unexpected results. If the source of the message is unknown, it is recommended that an application perform checks to ensure that it contains reasonable values. For example, check to make sure the From and To header fields contain e-mail addresses, the message has a valid Date, and a message body is present.

The simplest method of importing a message into the control is using the **ImportMessage** method. Here is a Visual Basic example which uses the Common Dialog control to select a file and then calls ImportMessage to import the file:

```
Dim nError As Long

On Error GoTo ImportCanceled
CommonDialog1.CancelError = True
CommonDialog1.DefaultExt = ".txt"
CommonDialog1.DialogTitle = "Import Message"
CommonDialog1.FilterIndex = 1
CommonDialog1.Flags = cdlOFNFileMustExist + cdlOFNLongNames
CommonDialog1.Filter = "Text Files (*.txt)|*.txt|" & _
                       "E-Mail Message Files (*.eml)|*.eml|" & _
                       "All Files (*.*)|*.*"
```

```
CommonDialog1.ShowOpen
On Error GoTo 0

nError = InternetMail1.ImportMessage(CommonDialog1.FileName)
If nError Then
    MsgBox "Unable to import message from " & _
           CommonDialog1.FileTitle & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
    Exit Sub
End If

MsgBox "Imported message from " & InternetMail1.From & vbCrLf & _
       "regarding " & Chr(34) & InternetMail1.Subject & Chr(34), _
       vbInformation

Exit Sub

ImportCanceled:
Exit Sub
```

Once the message has been imported successfully, the various message related properties can be accessed just as if the message had been composed. Note that the current message, if any, will be completely replaced by the message that has been imported.

Another method of importing a message is from a string. This is useful if a message has been stored in something other than a text file such as a record in a database. To do this, simply set the control's Message property to the string which contains the message:

```
On Error Resume Next: Err.Clear
InternetControl1.Message = strMessage
If Err.Number Then
    MsgBox Err.Description, vbExclamation
    Exit Sub
End If
```

Unlike the **ImportMessage** method, which returns an error code if it fails, setting the **Message** property will result in an error being raised if there is a problem. Because of this, any application which sets the **Message** property should use an error handler. In this example, it simply executes the next statement and uses the *Err* object to obtain the error code and description.

A third method of importing a message into the control is to use the **ParseMessage** method. Unlike the **ImportMessage** method or the **Message** property, it is not required that the complete message be available at once. Instead, **ParseMessage** enables an application to import a message in pieces, dynamically parsing the data and adding to the contents of the current message. The following example opens a file which contains a message, reads it in 1,024 byte blocks and then passes it to the **ParseMessage** method:

```
hFile = FreeFile()
Open strFileName For Input As hFile
nFileLength = LOF(hFile)

InternetMail1.ClearMessage
```

```
Do While nFileLength > 0
    cbBuffer = nFileLength
    If cbBuffer > 1024 Then cbBuffer = 1024
    nFileLength = nFileLength - cbBuffer
    strBuffer = Input(cbBuffer, hFile)
    nError = InternetMail1.ParseMessage(strBuffer)
    If nError <> 0 Then
        MsgBox InternetMail1.LastErrorString, vbExclamation
        Exit Do
    End If
Loop

Close hFile
```

The initial call to the **ClearMessage** method ensures that if there is a current message, the contents are cleared first. Then the **ParseMessage** method is repeatedly called until the end-of-file is reached. This approach could be used when a message is being created by some external data source or the application needs to make some sort of change to the message contents dynamically. Note that the purpose of the above example is to demonstrate how to use the **ParseMessage** method and is not the recommended procedure for importing a message from a text file. Refer to the technical reference documentation for the **ImportMessage** method for more information on importing messages from text files.

Exporting Messages

The Internet Mail control has the ability to export the current message contents as a string or as a text file on the local system. When a message is exported, the complete message including headers, message body and any file attachments are included. The following example uses the CommonDialog control to choose a file name to export the current message to:

```
Dim nError As Long

On Error GoTo ExportCanceled
CommonDialog1.CancelError = True
CommonDialog1.DefaultExt = ".txt"
CommonDialog1.DialogTitle = "Export Message"
CommonDialog1.FilterIndex = 1
CommonDialog1.Flags = cdlOFNLongNames + cdlOFNOverwritePrompt
CommonDialog1.Filter = "Text Files (*.txt)|*.txt|" & _
                       "E-Mail Message Files (*.eml)|*.eml|" & _
                       "All Files (*.*)|*.*"

CommonDialog1.ShowSave
On Error GoTo 0

nError = InternetMail1.ExportMessage(CommonDialog1.FileName)
If nError Then
    MsgBox "Unable to export message to " & _
           CommonDialog1.FileTitle & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
    Exit Sub
End If

MsgBox "Exported message to " & CommonDialog1.FileName & vbCrLf & _
       "regarding " & Chr(34) & InternetMail1.Subject & Chr(34), _
       vbInformation

ExportCanceled:
Exit Sub
```

When a message is exported, headers may be re-ordered and certain headers which contain routing information (such as Received and Return-Path) are omitted by default.

These headers are not normally needed when composing or delivering a message, however, there may be situations in which an application needs to preserve these headers or the order in which they were originally received. The **ExportMessage** method has an optional argument which can be used to specify one or more export options:

Constant	Description
mailOptionAllHeaders	Preserve all headers in the message when it is exported, including those headers which provide routing information such as Received and Return-Path.
mailOptionKeepOrder	Preserve the original order of the message headers. This is only useful if the message was retrieved from a mail server or imported from a file.

These two values may be combined if both options are required. For example, the following code would export a message with all of the headers, preserving their original order:

```
nOptions = mailOptionAllHeaders Or mailOptionKeepOrder
nError = InternetMail1.ExportMessage(strFileName, nOptions)
```

Note that the option values are actually bit flags, so a bitwise OR operation is used to combine them. This method is preferred over using simple addition which can produce unexpected results in some cases. Note that if the **ExportMessage** method is called without specifying the optional argument, then the value of the **Options** property is used as a default, such as:

```
InternetMail1.Options = mailOptionAllHeaders Or mailOptionKeepOrder
nError = InternetMail1.ExportMessage(strFileName)
```

This would also cause the message to be exported with all headers and preserve their original order. A general rule of thumb is that if there is an optional argument to a method which corresponds to a property in the control, if that argument is not specified, the property value will be used as a default.

If an application needs to do some processing on the message but doesn't want the overhead of exporting the message to a file, then the message contents can be read using the control's **Message** property. This property returns a string which contains the complete message, including all headers. The **Options** property determines whether or not all headers are exported and if the original header order is preserved, just as with the **ExportMessage** method. It should be noted that this is different than the **MessageText** property, which returns only the body of the current message part, not the complete message.

Extracting Attachments

To determine if a message contains one or more file attachments, the simplest method is to check the value of the **Attachment** property for each message part. This property returns the name of the file attachment in the current message part or returns an empty string if there is no file attachment. The control checks for standard MIME attachments as well as non-standard uuencoded files which are embedded in the body of the message. The following code demonstrates how this could be done:

```
bHasAttachments = False

For nPart = 0 To InternetMail1.MessageParts - 1
    InternetMail1.MessagePart = nPart
    If Len(InternetMail1.Attachment) > 0 Then
        bHasAttachments = True
        Exit For
    End If
Next
```

To extract the attached file, use the **ExtractFile** method and specify the name of the attachment on the local system. For example:

```
For nPart = 0 To InternetMail1.MessageParts - 1
    InternetMail1.MessagePart = nPart
    If Len(InternetMail1.Attachment) > 0 Then
        strFileName = "c:\temp\" & InternetMail1.Attachment
        nError = InternetMail1.ExtractFile(strFileName)
        If nError Then
            MsgBox "Unable to extract " & InternetMail1.Attachment, _
                   vbExclamation
        End If
    End If
Next
```

If the file is extracted from the message successfully, the **ExtractFile** method will return a value of zero, otherwise an error code will be returned.

Message Headers

Each message contains additional information about the message. The information is organized in header blocks at the beginning of the message and at the beginning of each section of a multipart message. Each header block contains one or more header fields, along with their values. For example, the sender of a message is specified in the "From" header field, with the value being the sender's e-mail address.

As discussed in the section on composing messages, there are a number of properties in the control which represent header fields in the current message. The **From**, **To** and **Subject** properties each correspond to their respective header field in the message. However, messages can have many more headers than those which are represented by properties in the control. To access these headers, there is the **GetHeader** method which can be used to obtain the value of a particular field. For example, one way to determine if a message was generated by Microsoft Outlook is to check for the presence of the X-MimeOLE header:

```
If InternetMail1.GetHeader("X-MimeOLE", strHeaderValue) Then
    MsgBox strHeaderValue, vbInformation
End If
```

It should be noted that unlike most of the other methods, the **GetHeader** method returns a boolean value, not an error code. The method returns True if the header field is present in the message and the second argument, a string variant, will contain the header value. If the header field does not exist, the method will return False.

There are a number of standard headers which are recognized by all mail clients and are typically those which are represented as properties in the control. However, implementers are free to create their own custom headers as long as they prefix them with "X-" (which indicates that they are a non-standard extension). While a mail client can check for the presence of extended headers, it should not depend on them being in the message. General purpose applications must be able to handle situations where a non-standard header is either missing or contains an unexpected value.

To change the value of a header, an application can use the **SetHeader** method. If the header field exists, its value will be replaced by the new value specified as an argument to the method. If the header field does not exist, it will be added to the current message. For example, to create a custom header field which contains a customer number, the following code could be used:

```
InternetMail1.SetHeader "X-Customer-Number", nCustomerNumber
```

Because the second argument to the **SetHeader** method is a variant, a variety of data types such as strings, integers, etc., may be passed as values. Note that data types which are affected by localization, such as date values, will be represented differently based on how the local system has been configured.

Remember that the **GetHeader** and **SetHeader** methods use the current message part, so if your message is multipart (such as having a file attachment) then make sure that you first set the **MessagePart** property to a value of 0 to ensure you are in the main header block for the message.

In some circumstances you may need to list all of the header values in a message without knowing what header fields are present. You can do this by enumerating the header fields using the **GetFirstHeader** and **GetNextHeader** methods. The following example stores each header field and value in a pair of listboxes:

```
If InternetMail1.GetFirstHeader(strHeaderField, strHeaderValue) Then
    Do
        List1.AddItem strHeaderField
        List2.AddItem strHeaderValue
    Loop While InternetMail1.GetNextHeader(strHeaderField, strHeaderValue)
End If
```

To delete a message header, call the **DeleteHeader** method with the header field as the argument. If the header field exists, it will be deleted and the method will return a value of True. If the header does not exist, the method will return False. To delete all of the headers in a message, you could use code like this:

```
Do While InternetMail1.GetFirstHeader(strHeaderField, strHeaderValue)
    InternetMail1.DeleteHeader strHeaderField
Loop
```

Remember that a valid mail message requires a header block, so once the headers have been deleted, new values should be set for fields such as From, To, Date, Subject and so on. Also, keep in mind that changing or deleting header fields in a message should only be done when the effect of that change is well understood. In other words, simply deleting all of the headers to "clean up" a message is not recommended since it may damage the internal structure of the message, making it unreadable to other mail software.

Retrieving Messages

Mail messages are stored on a mail server in a mailbox, typically a folder or single file which contains all of the messages for a specific user. As new messages arrive for the user they are added to the mailbox and they are removed as they are read and deleted. The first step in retrieving messages from a user's mailbox is establishing a network connection with the mail server and authenticating that connection with a user name and password. This can be done using the **Connect** method, which has a number of optional arguments:

Constant	Description
ServerName	An optional string value which specifies the domain name or IP address of the mail server. If this argument is not specified, the value of the **ServerName** property is used instead.
ServerPort	An optional integer value which specifies the port number to use when establishing a connection with the server. If this argument is not specified, the value of the **ServerPort** property is used instead. A value of zero means that the default port for the server should be used.
UserName	An optional string value which specifies the user name is required to authenticate access to the mailbox. If this argument is not specified, the value of the **UserName** property is used instead.
Password	An optional string value which specifies the password which is required to authenticate access to the mailbox. If this argument is not specified, the value of the **Password** property is used instead.
Timeout	An optional integer value which specifies the amount of time in seconds that the control will wait for an operation to complete before returning an error. If this argument is not specified, the value of the **Timeout** property is used. The default timeout period is sixty (60) seconds.

Options	An optional integer value which specifies one or more options as bit flags. If this argument is not specified, the value of the **Options** property is used instead. The most common option that is used is *mailOptionAPOP* which indicates that APOP authentication should be used instead of standard user authentication when connecting to the mail server. Refer to the technical reference for information about the other options that are available.

The following code demonstrates connecting to a mail server:

```
Dim nError As Long
Dim nMessages As Long

nError = InternetMail1.Connect(editServerName.Text, , _
                               editUserName.Text, _
                               editPassword.Text)

If nError Then
    MsgBox "Unable to connect to " & editServerName.Text & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
    Exit Sub
End If
```

As with most of the other control methods, **Connect** returns an error status if the connection has failed or a value of zero if it was successful. Once a connection with the mail server has been established, you can list the available messages by setting the **MessageIndex** property. For example, the following code would check to make sure that there are messages in the mailbox, and if so, add the subject of each message to a listbox:

```
If InternetMail1.LastMessage = 0 Then
    MsgBox "The mailbox is currently empty", vbInformation
    InternetMail1.Disconnect
    Exit Sub
End If

For nMessage = 1 To InternetMail1.LastMessage
    InternetMail1.MessageIndex = nMessage
    List1.AddItem InternetMail1.Subject
Next
```

When the **MessageIndex** property is set, the headers for the specified message are retrieved from the mail server and can be accessed using the control's properties. However, the entire message contents are not retrieved unless the **GetMessage** method is called. This allows applications to retrieve message headers without the requirement that the complete message be downloaded. For example, the following code would check to see if a message has attachments by checking the Content-Type header:

```
For nMessage = 1 To InternetMail1.LastMessage
    InternetMail1.MessageIndex = nMessage
    If InStr(InternetMail1.ContentType, "multipart/mixed") Then
        ' This is a multipart/mixed message, so it is likely
        ' that it contains one or more file attachments
        List1.AddItem InternetMail1.Subject
    End If
Next
```

The **GetMessage** method will retrieve the complete message including the headers, message body and any attachments. Immediately after the **GetMessage** method is called, the **OnProgress** event will begin to fire, enabling an application to update the user interface.

As with most of the other methods, once the message has been retrieved, **GetMessage** will return a value of zero if it was successful or an error code if a problem was encountered. The following example demonstrates how **GetMessage** and **ExportMessage** can be used to save messages to the local system:

```
For nMessage =  1 To InternetMail1.LastMessage
    strFileName = "c:\temp\msg" & Format(nMessage, "00000") & ".txt"

    ' Retrieve the message from the server
    nError = InternetMail1.GetMessage(nMessage)
    If nError Then Exit For

    ' Export the message to a file on the local system
    nError = InternetMail1.ExportMessage(strFileName)
    If nError Then Exit For
Next

If nError Then
    MsgBox "Unable to retrieve or store message " & nMessage & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
     Exit Sub
End If
```

An alternate method of storing messages in files on the local system is to use **StoreMessage** instead of **GetMessage** and **ExportMessage**. The most significant difference between the two approaches is that the **StoreMessage** method does not change the contents of the current message. Instead, it simply retrieves the message from the server and stores it in the specified file. The previous example could be re-written as:

```
For nMessage =  1 To InternetMail1.LastMessage
    strFileName = "c:\temp\msg" & Format(nMessage, "00000") & ".txt"
    ' Store the message from the server
    nError = InternetMail1.StoreMessage(nMessage, strFileName)
    If nError Then Exit For
Next

If nError Then
    MsgBox "Unable to store message " & nMessage & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
     Exit Sub
End If
```

It is important to keep in mind that while the application is connected to the mail server, the mailbox is in a locked state which prevents any other clients from accessing it. For this reason, it is recommended that any significant message processing be done on local copies of the messages.

Deleting Messages

Because mailboxes are frequently limited in size by the mail system administrator, it is common practice to delete those messages which have been stored on the local system or have been marked for deletion by the client. To delete a message from the mailbox, use the **DeleteMessage** method. This will mark the message for deletion, removing it from the mailbox when the client session is terminated. Once a message has been marked for deletion it is no longer available to the client and attempting to access the message (for example, by setting the **MessageIndex** property) will result in an error.

As messages are marked for deletion from the mailbox, the **MessageCount** property will be decremented to reflect the current number of messages. Because of this, the **MessageCount** property should generally not be used in constructs like For..Next or Do..While loops where each message is being processed. Instead, use the **LastMessage** property which remains constant for the duration of the client session. To delete all of the messages in the mailbox, use code like this:

```
For nMessage = 1 To InternetMail1.LastMessage
    nError = InternetMail1.DeleteMessage(nMessage)
    If nError Then
        MsgBox "Unable to delete message " & nMessage & vbCrLf & _
               InternetMail1.LastErrorString, vbExclamation
        Exit Sub
    End If
Next
```

To prevent those messages from actually being removed from the mailbox, call the **Reset** method which resets both the state of the control and the current mailbox (if a connection to a mail server has been established). Keep in mind that once the **Disconnect** method is called, any messages that have been marked for deletion are permanently removed from the mailbox and cannot be recovered.

—Programming with Visual C++—

The Internet Mail control can be used in Visual C++ in several ways, depending on the type of program being developed and the way in which the control will be utilized. Although much of the complexity of COM can be hidden through the use of wrapper classes and smart pointers, development using COM objects is still more complex than simply using the MFC classes with which most Windows developers are familiar.

One of the first things that a C++ developer will encounter when programming with the control is that all of the methods use a data type called a *variant*. Most developers aren't familiar with what a variant is or how it should be used unless they have experience with COM programming, so this is a frequent point of confusion. The simplest definition is that a variant is a structure which contains type information and a union of intrinsic data types such as characters, integers and so on. The variant essentially serves as a generic data type, and the function being called has the responsibility of using or converting that data as necessary. In addition to the VARIANT structure itself, there are several classes which encapsulate variants, such as **_variant_t**, **COleVariant** and **CComVariant**. These classes make it easier for C++ programmers to use variants, and for the most part allows them to be used just as if the variant was an intrinsic type.

Another data type that may be unfamiliar is the BSTR, which is used for string data. Similar to C strings, the BSTR is a pointer to a null terminated array of characters which make up a string. However, there are some significant differences between the two. First, a BSTR always uses the Unicode character set, even if the program itself does not use Unicode. That means that each character in the BSTR is actually 16 bits, so special care must be taken to not assume that a character in the string is equivalent to a single byte. Second, although BSTR strings are null terminated, they may actually contain embedded nulls. This is because the BSTR also has information about the length of the string, so standard string functions (even the Unicode versions of them) should not be used if there is a chance that the string contains embedded nulls. Part of the Automation API is a collection of functions which manage BSTR strings, such as **SysAllocString** and **SysStringLen**.

However, most programmers prefer to use one of the classes which encapsulate BSTRs, such as **_bstr_t** and **CComBSTR**.

In addition to the COM data types, another aspect of using COM objects is that most COM related functions return HRESULT values. The HRESULT is a 32-bit unsigned integer which contains status information about an error or warning returned by a function. Two macros which are commonly used are FAILED and SUCCEEDED which are used to determine whether or not the HRESULT value indicates that the function failed or was successful. All COM object methods and property accessor functions return HRESULT values which must be checked by the caller to ensure that the function was called correctly.

Microsoft Foundation Classes

The Internet Mail control can be used with MFC based applications by including the control in the project that is being developed. This is done through the Visual C++ IDE by selecting the menu option **Project | Add to Project | Components and Controls**. This will display a dialog which is used to select the component to add. First select the Registered ActiveX Controls folder, scroll over to the Catalyst Internet Mail Control, and press the Insert button. A dialog is then displayed which determines the class name and files which will be generated to "wrap" the ActiveX control. In this case, the default class name will be CInternetMail. A new source file will be added to the project named **InternetMail.cpp** which contains the methods for the wrapper class that was created. A header file named **InternetMail.h** will also be created and included in the header file for the dialog class.

To create an instance of the control, the simplest approach is to create a dialog-based application, in which case the control can be selected from the dialog component palette, similar to how controls are placed on forms in Visual Basic. The control is included as a resource and assigned a resource ID.

Then, using the MFC Class Wizard, a member variable for the dialog class is assigned to that instance of the control. This means that a declaration similar to this will be added to the dialog class:

```
CInternetMail m_ctlInternetMail;
```

And in the **DoDataExchange** method, a line will be added which initializes the control when the dialog is created:

```
DDX_Control(pDX, IDC_INTERNETMAIL1, m_ctlInternetMail);
```

Now, any of the control's properties or methods may be accessed through the member variable for the CInternetMail class. For example:

```
COleVariant varServerName(m_strServerName);
COleVariant varServerPort(m_nServerPort);
COleVariant varUserName(m_strUserName);
COleVariant varPassword(m_strPassword);
COleVariant varTimeout(m_nTimeout);
COleVariant varOptions;
COleVariant varError;

varError = m_ctlInternetMail.Connect(varServerName,
                                     varServerPort,
                                     varUserName,
                                     varPassword,
                                     varTimeout,
                                     varOptions);

varError.ChangeType(VT_I4);

if (V_I4(&varError) != 0)
{
    CString strError;
    strError.Format(_T("Unable to connect to %s\n%s"),
                    m_strServerName,
                    m_ctlInternetMail.GetLastErrorString());

    AfxMessageBox(strError, MB_ICONEXCLAMATION, 0);
}
else
{
    CString strMessage;
    LONG nMessages;

    nMessages = m_ctlInternetMail.GetMessageCount();
    m_ctlInternetMail.Disconnect();

    strMessage.Format(_T("This mailbox has %ld messages"), nMessages);
    AfxMessageBox(strMessage, MB_ICONINFORMATION, 0);
}
```

In this example, the arguments are converted to variants by initializing **COleVariant** variables which are then passed to the **Connect** method. There are two important things to note here. First, even though the documentation lists some of the arguments as optional, when using the control this way in C++, you must specify all of them. This is because optional parameters really aren't omitted from the method; they are still passed as variants, but instead of having a value, they are initialized to tell the control that they were not specified. This is accomplished here by passing an empty (uninitialized) **COleVariant**, as with the **varOptions** variable. The second important point is that although the method is documented as returning a long integer, the actual return type is a variant that contains a long integer value.

You'll notice that in this code, there are also some macros being used with the variant types. The first one is used when checking the return value from the method:

```
varError.ChangeType(VT_I4);

if (V_I4(&varError) != 0)
{
    .
    .
    .
}
```

The **ChangeType** method for the **COleVariant** class changes the type of variant, in this case to a long integer, specified by the value **VT_I4**. What this does is coerce the variant data into a long integer if it already isn't one. If the variant already represents a long integer, then the call to **ChangeType** doesn't have any effect. Next, the **V_I4** macro is used to obtain the actual value from the long integer. Note that it expects a pointer to a variant, not the variant itself.

Instantiating CWnd Based Controls

To create an instance of the control in an MFC application without using a dialog, add the control to the project using the same method described previously. However, instead of placing the control on a dialog using the resource editor, declare a CInternetMail member variable in the class that will be using the control. Then, call the **Create** function to create an instance of the control for that class. For example:

```
CRect rcNull;
BSTR bstrLicKey;
BOOL bCreated;
USES_CONVERSION;

bstrLicKey = SysAllocString(T2OLE(CSIMXCTL_LICENSE_KEY));
```

```
bCreated = m_ctlInternetMail.Create(NULL,          // window name
                                    0,             // window style
                                    rcNull,        // window rect
                                    this,          // parent window
                                    IDC_CONTROL,   // control ID
                                    NULL,          // persistent storage
                                    FALSE,         // IStorage
                                    bstrLicKey);   // license key

if (bCreated == FALSE)
{
    AfxMessageBox(_T("Control creation failed"), MB_ICONEXCLAMATION);
    EndDialog(0);
}
```

Because the control is not part of the program's resources as in the previous example, an instance of the control must be explicitly created by calling the **Create** method. Because the Internet Mail control is not visible at runtime, most of the window arguments are null, however it is still required that a parent window be specified; in this case, the *this* pointer is used. If the class that is using the control is not derived from CWnd, a hidden window can be created and specified as the parent instead.

Another issue is that to create an instance of the control, the application must pass it a runtime license key. This is a BSTR string which is used by the control to determine if it can be used in an application. If this string is NULL, then the control will only load if the current system has a valid development license. If it is not NULL, then the license key is validated and an instance of the control is created. The license key for the Internet Mail control defined in the csimxkey.h header file is found in the Include folder where the product was installed. Note that the key value will be NULL for evaluation versions of the control, which means that the application cannot be redistributed until a license has been purchased and registered.

The **SysAllocString** function is used to create the license key BSTR and this requires that the license key be converted to Unicode. In **afxpriv.h** there are several string conversion macros that are useful for converting between ANSI and Unicode. One is OLE2T which converts a Unicode string to an LPTSTR, and the other is T2OLE which converts an LPTSTR to a Unicode string. The **afxpriv.h** header file is not usually included in MFC applications, so it will need to be added to **StdAfx.h** manually.

The USES_CONVERSION macro is required and must be included in the function prior to using on the conversion macros. In this case, T2OLE is used to convert the ANSI license key string to Unicode, and then that is passed to **SysAllocString** to create a BSTR. It should be noted that OLE2T and T2OLE allocate memory from the stack to do the conversion, so they should not be used with very large amounts of data.

Importing ActiveX Controls

In Visual C++ 5.0, the **#import** compiler directive was included as an alternative to adding the control to a project through the IDE. Similar to how header files are included in an application, this directive incorporates information from a type library, automatically creating wrapper classes for its interfaces. These classes use smart pointers which handle things like reference counting automatically, and makes actually using the control's interface much simpler.

To use this method of referencing a control, the first thing that needs to be done is to import the control into the module where it will be used. This is done using the **#import** directive which can be placed in an appropriate header file:

```
#import "csimxctl.ocx" no_namespace named_guids
```

The **no_namespace** attribute specifies that the interface classes should not be defined in a namespace. Normally, a namespace is created which is based on the name of the library. The **named_guids** attribute tells the compiler to initialize the GUID variables using the standard naming convention.

The next step is to declare a variable that is used to reference an instance of the control. For example, the following could be included in the definition of a class:

```
IInternetMailPtr m_pIInternetMail;
```

Note that the member variable is declared as type **IInternetMailPtr**, which is a specialization of the smart pointer **_com_ptr_t** template class. If errors are encountered when compiling the application indicating that the compiler cannot instantiate an abstract class (because the class contains pure virtual functions) then most likely the member variable was declared as type **IInternetMail**, which is incorrect. Don't forget the "**Ptr**" on the end of the name.

To use the control, an instance of the control must be created using the **CreateInstance** function. However, before that can be done, the COM subsystem must be initialized by the application. For MFC based applications, this is accomplished by calling the function **AfxOleInit** which is essentially a wrapper around **CoInitializeEx**. This should be done fairly early in the application, typically in the **InitInstance** function of the CWinApp derived application class. Next, the control's **CreateInstance** member function must be called before it is used:

```
HRESULT hr;

hr = m_pIInternetMail.CreateInstance(CLSID_InternetMail);
if (FAILED(hr))
{
    AfxMessageBox(_T("Control creation failed"), MB_ICONEXCLAMATION);
    return;
}
```

The HRESULT return value should be 0, which indicates that an instance of the control was created successfully. If an error is returned, this typically means that **AfxOleInit** (or **CoInitializeEx**) was not called first, or the control has not been registered on the system.

Unlike the previous examples where the initialization of the control was performed automatically or by calling the **Create** function, this instance of the control should be explicitly initialized by calling the Initialize method:

```
_variant_t varModule;
_variant_t varLicKey;
_variant_t varError;
USES_CONVERSION;

// Create the runtime license key defined in csimxkey.h
varLicKey = SysAllocString(T2OLE(CSIMXCTL_LICENSE_KEY));

// Initialize the control
varError = m_pIInternetMail->Initialize(varModule, varLicKey);
if (V_I4(&varError) != 0)
{
   AfxMessageBox(_T("Control initialization failed"), MB_ICONEXCLAMATION);
   return;
}
```

Just as in the previous example using the **Create** method, the runtime license key is created by converting it to Unicode and then calling **SysAllocString** to create a BSTR string. Because the control methods use variants, this key is assigned to a variant. Note that the **_variant_t** type is used, which is a COM support class which encapsulates a variant. The **Initialize** method returns a long integer variant which specifies an error code. A value of zero indicates that the control was successfully initialized, while a non-zero value is an error code.

Once the control has been created and initialized, it can be used in a fashion similar to how the previous examples were written:

```
_variant_t varServerName(m_strServerName);
_variant_t varServerPort(m_nServerPort);
_variant_t varUserName(m_strUserName);
_variant_t varPassword(m_strPassword);
_variant_t varTimeout(m_nTimeout);
_variant_t varOptions;
_variant_t varError;

varError = m_pIInternetMail->Connect(varServerName,
                                     varServerPort,
                                     varUserName,
                                     varPassword,
                                     varTimeout,
                                     varOptions);

if (V_I4(&varError) != 0)
{
    CString strError;
    USES_CONVERSION;

    strError.Format(_T("Unable to connect to %s\n%s"),
                    m_strServerName,
                    OLE2T(m_pIInternetMail->GetLastErrorString()));

    AfxMessageBox(strError, MB_ICONEXCLAMATION, 0);
}
else
{
    CString strMessage;
    LONG nMessages;

    nMessages = m_pIInternetMail->MessageCount;
    m_pIInternetMail->Disconnect();

    strMessage.Format(_T("This mailbox has %ld messages"), nMessages);
    AfxMessageBox(strMessage, MB_ICONINFORMATION, 0);
}
```

There are two significant differences between the previous examples which uses the control as a CWnd derived class and this class which is based on COM smart pointers. The first is that methods are accessed through the **m_pIInternetMail** object as a pointer to the interface, so the **->** operator is used. The second is that the control's properties, such as **MessageCount**, can be accessed as if they are member variables of the class rather than using accessor functions like **GetMessageCount**. This is a bit of slight-of-hand being performed by the interface class using the **__declspec**(property) extension. For example, the **MessageCount** member is declared as:

```
__declspec(property(get=GetMessageCount)) long MessageCount;
```

This tells the compiler whenever the **MessageCount** member is read, it should call the **GetMessageCount** function to return the value. So, in effect the above code is changed by the compiler into:

```
nMessages = m_pIInternetMail->GetMessageCount();
```

Either method may be used, so it is generally up to the personal preferences of the developer as to which is used.

Component Object Model API

The fourth approach that can be used to create an instance of an ActiveX control in your C++ program is to use the COM API directly. Generally speaking this option should only be used if absolutely necessary; it is a more complex process and involves more coding than either using CWnd derived controls or the **#import** directive.

The first step is to create the header file for the interface defined in the control's type library. This will require two tools that are included with Visual C++ and the Microsoft Platform SDK, the COM Object Viewer, and the Microsoft Interface Definition Language (MIDL) compiler. These tools can also be downloaded from Microsoft from their MSDN resources section of the website.

To create the interface definition (IDL) file, start the COM Object Viewer, select the Control folder and then the Catalyst Internet Mail control from the list of available controls. Right click on the control and select View Type Information. This will open the ITypeLib Viewer window which contains the interface definition. Select **File | Save As** and save it as csimxctl.idl in the project directory. Close the viewer window and exit the COM Object Viewer.

Once the IDL file has been created, open the IDL file in the editor and look for a series of enum typedefs which define the constants for the control:

```
typedef [public]
_mailOptionConstants mailOptionConstants;

typedef enum {
mailOptionNoStartTLS = 1,
mailOptionAPOP = 2,
…
} _mailOptionConstants;
```

These declarations are a side-effect of how the COM Object Viewer generates the IDL file and it needs to be cleaned up a bit so that the MIDL compiler generates the correct header file. Remove the **typedef** [public] section before each typedef enum section in the file. In other words, you would want to remove each section that looks like:

```
typedef [public]
_mailOptionConstants mailOptionConstants;
```

The actual enum typedefs can stay in the IDL so that they're included in the header file and can be used by the application. Note that if these extraneous typedefs aren't removed, the MIDL compiler will generate duplicate enums in the header file and will cause compiler errors.

Save the IDL file and then use the MIDL compiler to generate the header file which will be included with your project. From the command line, enter:

```
midl /Oicf /W1 /Zp8 /h csimxctl.h /iid csimxctl_i.c csimxctl.idl
```

This will create three files: **csimxctl.h**, **csimxctl_i.c** and **csimxctl.tlb**. The TLB is the compiled type library and isn't needed for this example. The **csimxctl.h** header file contains the interface definition for the control, and the **csimxctl_i.c** file is a C source file which defines the GUIDs used by the control. Both of these files should be included in your project, typically in the source module where the control will be used. Note that the MIDL compiler may emit a warning that there are too many methods in the interface. This is a warning that applies only to early versions of Windows NT 4.0 and doesn't affect current versions of Windows NT 4.0 or other Windows platforms such as Windows 98 or Windows 2000.

Now that the header file for the control interface has been created, the next step is to create an instance of the control. Define a member variable that is a pointer to the interface which looks like this:

```
IInternetMail *m_pIInternetMail;
```

Safe programming practices would also ensure that the pointer is initialized to NULL in the constructor to avoid potential errors when referencing the variable. As with the previous examples, the COM subsystem must be initialized. For MFC based applications, this can be done by calling **AfxOleInit** in the **InitInstance** function for the CWinApp derived application class. For other applications, **CoInitializeEx** should be called as:

```
HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
if (FAILED(hr))
{
   // Unable to initialize COM subsystem
   return;
}
```

Then the following code can be used to create an instance of the control:

```
HRESULT hr;
BSTR bstrLicKey;
IClassFactory2 *pFactory = NULL;
IUnknown *pUnknown = NULL;
USES_CONVERSION;

m_pIInternetMail = NULL;

hr = CoGetClassObject(CLSID_InternetMail,
                      CLSCTX_INPROC_SERVER,
                      NULL,
                      IID_IClassFactory2,
                        (LPVOID *)&pFactory);

if (FAILED(hr))
{
    // Unable to get the class factory interface for the
    // control, probably because it isn't registered
    return;
}

// Create the runtime license key defined in csimxkey.h
bstrLicKey = SysAllocString(T2OLE(CSIMXCTL_LICENSE_KEY));

// Create an instance of the control
hr = pFactory->CreateInstanceLic(NULL, NULL, IID_IUnknown,
                                 bstrLicKey,
                                 (LPVOID *)&pUnknown);
pFactory->Release();

if (FAILED(hr))
{
    // Unable to create an instance of the control using
    // the specified license key
    return;
}

hr = pUnknown->QueryInterface(IID_IInternetMail,
                              (LPVOID *)&m_pIInternetMail);

if (FAILED(hr))
{
    // Unable to get the interface to the control
```

```
        return;
    }
```

The **CoGetClassObject** function is used to get an interface pointer to the control's class factory, which actually does the work of creating an instance of the class. The **CreateInstanceLic** member function passes the runtime license key to the control, and an instance is created if the key is valid. Note that if a NULL value is passed as the license key, then the control will only be created if the system has a development license installed. The interface to the class factory is released and then **QueryInterface** is called on the returned pointer to obtain the interface to the control's properties and methods.

The code to use the interface is similar to the previous examples, however there are several significant differences:

```
    COleVariant varServerName(m_strServerName);
    COleVariant varServerPort(m_nServerPort);
    COleVariant varUserName(m_strUserName);
    COleVariant varPassword(m_strPassword);
    COleVariant varTimeout(m_nTimeout);
    COleVariant varOptions;
    COleVariant varError;
    HRESULT hr;

    hr = m_pIInternetMail->Connect(varServerName,
                                   varServerPort,
                                   varUserName,
                                   varPassword,
                                   varTimeout,
                                   varOptions,
                                   &varError);

    if (V_I4(&varError) != 0)
    {
        CString strError;
        BSTR bstrError;
        USES_CONVERSION;

        hr = m_pIInternetMail->get_LastErrorString(&bstrError);
        if (FAILED(hr))
            return;

        strError.Format(_T("Unable to connect to %s\n%s"),
                        m_strServerName,
                        OLE2T(bstrError));

        AfxMessageBox(strError, MB_ICONEXCLAMATION, 0);
    }
    else
    {
        CString strMessage;
        LONG nMessages = 0;

        hr = m_pIInternetMail->get_MessageCount(&nMessages);
        if (FAILED(hr))
            return;

        m_pIInternetMail->Disconnect(&varError);

        strMessage.Format(_T("This mailbox has %ld messages"), nMessages);
        AfxMessageBox(strMessage, MB_ICONINFORMATION, 0);
    }
```

As with the version of the code using the COM smart pointer, **p_IInternetMail** is a pointer to the interface, which requires that the **->** operator be used to access its member functions. Property values are read using accessor functions that are prefixed with "**get_**", while those which set properties are prefixed with "**put_**". For example, to get the value of the **MessageCount** property, the function name would be **get_MessageCount**. Methods in the control are called using the same name.

Another difference is that all of the functions return HRESULT values, with the actual property value or return value from the method specified as a function parameter that is passed by reference. This is why the **varError** variable is passed as the last argument to the **Connect** method. If the HRESULT return value is non-zero, this typically will indicate an error. The error may be specific to the control, or it may be a general error coming from the COM subsystem.

Once the application is done using the control, the interface must be released with code like this:

```
if (m_pIInternetMail)
    m_pIInternetMail->Release();
```

Each control that is created has a reference count which is used to keep track of how many times one of its interfaces has been requested. When the reference count drops to zero, the control destroys itself and releases the memory that was allocated. Failing to release the interface will prevent the control from ever being destroyed and will result in a memory leak.

Control Event Handling

In languages like Visual Basic, using the events for a control simply involves adding code for the desired event. Many of the details, such as connecting the control's event interface to the container, is largely invisible to the programmer. However, when using a control in Visual C++, some extra work does need to be done. This section will cover two basic methods, one specific to CWnd derived controls which are placed in a dialog and another approach which uses a CCmdTarget derived class to handle event notifications.

To create an event handler for a control that has been placed on a dialog form, open the form in the resource editor, right click the control and select Events. This will open a dialog that lists the available events for the control. Selecting one of the events adds the event to the dialog class with a name like **OnProgressInternetMail1**. In the implementation for the dialog class, a section of code will be added that looks like this:

```
BEGIN_EVENTSINK_MAP(CImail1Dlg, CDialog)
  //{{AFX_EVENTSINK_MAP(CImail1Dlg)
  ON_EVENT(CImail1Dlg, IDC_INTERNETMAIL1, 4, OnProgressInternetMail1,
           VTS_VARIANT VTS_VARIANT VTS_VARIANT)
  //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()
```

This is the event sink map which is used to map a function in the dialog class to the control's event dispatch interface. The ON_EVENT macro defines the event sink with the dialog class name, the control ID, the dispatch ID for the event, the event handler function and then the parameters that are passed to the event. The three VTS_VARIANT macros specify that the event handler has three VARIANT arguments. All of this code is automatically generated with one ON_EVENT for each control event that was selected. Of the two approaches, this is the simplest but it depends on the fact that the control has been placed on a dialog.

A more general purpose way to implement event handling is to derive a class from the CCmdTarget class which will act as the event sink for the control. First, edit the **StdAfx.h** header file to include **afxctl.h**. Next, create a new class for the project called CEventSink. It should be derived from CCmdTarget with Automation support enabled (however, do not make it creatable by type ID). A dialog may be displayed that it was unable to edit the object definition (ODL) file for the product. Since your project may not have one, this is only a warning and can be ignored.

Open the **EventSink.cpp** implementation file and look towards the end of the file where there is a section that looks something like this:

```
BEGIN_INTERFACE_MAP(CEventSink, CCmdTarget)
    INTERFACE_PART(CEventSink, IID_IEventSink, Dispatch)
END_INTERFACE_MAP()
```

This maps the CEventSink class to the event interface. This needs to be changed so that it is mapped to the control's **IInternetMailEvents** interface, so change the second argument of the INTERFACE_PART macro to the value **DIID__IInternetMailEvents**. This section should now look like:

```
BEGIN_INTERFACE_MAP(CEventSink, CCmdTarget)
    INTERFACE_PART(CEventSink, DIID__IInternetMailEvents, Dispatch)
END_INTERFACE_MAP()
```

Next, decide what events handlers should be implemented for the control. This example will implement all of them, but it isn't necessary if they aren't actually going to be used by the application. The event handlers will be protected member functions of the CEventSink class and defined in **EventSink.h** and implemented in **EventSink.cpp**:

```
void OnCancel();
void OnDelivered(VARIANT& varAddress, VARIANT& varMessageSize);
void OnError(VARIANT& varError, VARIANT& varDescription);
void OnProgress(VARIANT& varMessageSize, VARIANT& varMessageCopied,
                VARIANT& varPercent);
void OnRecipient(VARIANT& varAddress, VARIANT* pvarCancel);
void OnTimeout();
```

Once the event handler functions have been implemented, they need to be added to the dispatch map. Look for the BEGIN_DISPATCH_MAP section in **EventSink.cpp** and add the definitions for the events:

```
DISP_FUNCTION_ID(CEventSink,"OnCancel",1,OnCancel,VT_EMPTY,VTS_NONE)
DISP_FUNCTION_ID(CEventSink,"OnDelivered",2,OnDelivered,
                 VT_EMPTY,VTS_VARIANT VTS_VARIANT)
DISP_FUNCTION_ID(CEventSink,"OnError",3,OnError,
                 VT_EMPTY,VTS_VARIANT VTS_VARIANT)
DISP_FUNCTION_ID(CEventSink,"OnProgress",4,OnProgress,
                 VT_EMPTY,VTS_VARIANT VTS_VARIANT VTS_VARIANT)
DISP_FUNCTION_ID(CEventSink,"OnRecipient",5,OnRecipient,
                 VT_EMPTY,VTS_VARIANT VTS_PVARIANT)
DISP_FUNCTION_ID(CEventSink,"OnTimeout",6,OnTimeout,VT_EMPTY,VTS_NONE)
```

These declarations are similar to those used with the ON_EVENT macros in the previous example. The VT_EMPTY type specifies that the event handler does not return a value. VTS_VARIANT specifies a VARIANT argument, and VTS_PVARIANT specifies a pointer to a variant. VTS_NONE specifies that the event doesn't have any arguments.

With the event handlers implemented, the next step is to connect them to the control. Include the **EventSink.h** header file in the module where the control is being used and create two new member variables for the class:

```
DWORD m_dwEventSink;
CEventSink* m_pEventSink;
```

Next, an instance of the CEventSink class needs to be created and then the sink dispatch interface needs to be connected to the control using the **AfxConnectionAdvise** function:

```
// Create an instance of the event sink class
m_pEventSink = new CEventSink();

// Get a pointer to the sink IDispatch interface
LPUNKNOWN pUnknownSink = m_pEventSink->GetIDispatch(FALSE);

// Connect the event source to the sink
AfxConnectionAdvise(m_pIInternetMail,
                    DIID__IInternetMailEvents,
                    pUnknownSink,
                    FALSE,
                    &m_dwEventSink);
```

The last thing that needs to be done is to disconnect the event sink from the control when it is no longer needed. This is done by calling **AfxConnectionUnadvise**, typically right before an instance of the control is deleted:

```
if (m_pEventSink)
{
    LPUNKNOWN pUnknownSink = m_pEventSink->GetIDispatch(FALSE);

    AfxConnectionUnadvise(m_pIInternetMail,
                          DIID__IInternetMailEvents,
                          pUnknownSink,
                          FALSE,
                          m_dwEventSink);

    delete m_pEventSink;
    m_pEventSink = NULL;
    m_dwEventSink = 0;
}
```

With this code, the application is now wired to receive event notifications from the control. Keep in mind that because the instance of the CEventSink class was created on the heap, failure to destroy the sink will cause a memory leak in the application.

Catalyst Internet Mail 4.0 Technical Information

Component Information

Filename	CSIMXCTL.OCX
Version	4.00.0023
ProgID	SocketTools.InternetMail
ClassID	9E27D964-DFF7-44EB-91CB-CA251B6449C7
Threading Model	Apartment
Help Filename	CSIMXCTL.CHM
Dependencies	None

Platform Compatibility

The Internet Mail control is a 32-bit ActiveX component compatible with Windows 95, Windows 98, Windows ME, Windows NT 4.0 SP6, Windows 2000 and Windows XP. Note that Windows 95, Windows NT 3.51 and versions of Windows NT 4.0 prior to Service Pack 6 are not supported, however the control may function on those platforms.

Language Compatibility

The Internet Mail control is compatible with most languages which provide support for ActiveX or Component Object Model (COM) objects. Supported languages include Microsoft Visual Basic 5.0 or later, Visual FoxPro 5.0 or later and Visual C++ 6.0 or later. Other languages such as Delphi and PowerBuilder also provide support for the use of ActiveX controls with their development tools. Scripting languages which support referencing or creating instances of COM objects can also use the control. Refer to the Initialize method in the technical reference for more information about using the control in a scripting environment.

Redistribution Requirements

It is recommended that applications which redistribute CSIMXCTL.OCX install the control in the Windows system directory, either \Windows\System or \Windows\System32 depending on the platform. The control should be installed as a shared component which could be potentially locked (in use by another application), and the installation software should ensure that it does not overwrite a later version of the control.

If the control is being installed manually on a system, it will need to be registered using the RegSvr32.exe utility. If the application returns an error that it was unable to load or create an instance of the control, the most likely cause is that it was not registered after it was installed.

Windows XP, Windows 2000, Windows ME and Windows 98 SE support COM redirection, which enables an application to isolate the components that it uses, ensuring that the same version of the component which was used to build the application is loaded when the program is executed. To activate COM redirection, create an empty file named after the executable with a .local extension. For example, if the program is named MyApp.exe then an empty file named MyApp.exe.local should be created in the same directory as MyApp.exe. This binds the application to the local version of any components or libraries which are installed in the same directory as the application. When the component or library is loaded, Windows will first search the application's directory, and then uses the standard search rules for locating the file. Note that COM redirection is not supported on Windows 95 or Windows 98.

Catalyst Internet Mail 4.0 Technical Reference

Description

The Catalyst InternetMail control enables a developer to create, send and retrieve e-mail messages. The control implements the Simple Mail Transfer Protocol (SMTP) for sending messages, the Post Office Protocol (POP3) for retrieving messages from a mail server and the Multipurpose Internet Mail Extensions (MIME) standard for composing messages.

Reference

- Properties
- Methods
- Events
- Errors

File Name

CSIMXCTL.OCX

Object Type

InternetMail

Requirements

This components can be used with any Microsoft Windows development language or scripting tool that supports ActiveX controls. It is designed for the 32-bit Windows platform, and is supported on Windows 98, Windows ME, Windows NT 4.0, Windows 2000 and Windows XP.

Distribution

When you redistribute your application that uses the control, it is recommended that you install the control in the Windows system directory. ActiveX controls must be registered on the target system, either by the installation program or using the RegSvr32.exe utility.

Copyright

—Properties—

Attachment Property

Return the name of the attached file in the current message part.

Syntax

object.**Attachment**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **Attachment** property returns the name of the file attachment in the current part of a multipart message. When a new part is selected that contains an attached file, the **Attachment** property is updated to reflect the attached file's name. If the current message part does not contain a file attachment, this property will return an empty string.

Data Type

String

Bcc Property

Return or set the list of addresses that should receive a blind copy of the current message.

Syntax

object.**Bcc** [= *value*]

The **Bcc** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies one or more addresses that a copy of the message will be delivered to.

Remarks

The **Bcc** property is used to specify one or more addresses that a copy of the message will be delivered to. Note that these addresses are not included in the message header and cannot be viewed by the recipient.

Multiple addresses may be specified by separating them with a comma. Each address must conform to the standard Internet address format.

Data Type

String

See Also

Cc Property, From Property, ReplyTo Property, To Property

Cc Property

Return or set the list of addresses that should receive a copy of the current message.

Syntax

object.**Cc** [= *value*]

The **Cc** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies one or more addresses that a copy of the message will be delivered to.

Remarks

The **Cc** property returns the list of addresses that received a copy of the current message. If there is no current message, or the Cc header field is not defined, then this property will return an empty string.

Setting the **Cc** property creates or changes the value of the Cc header field in the message and specifies additional recipients of the message. Multiple addresses may be specified by separating them with a comma. Each address must conform to the standard Internet address format.

Data Type

String

See Also

Bcc Property, From Property, ReplyTo Property, To Property

CertificateExpires Property

Return the date and time that the server certificate expires.

Syntax

object.**CertificateExpires**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **CertificateExpires** property returns the date and time that the server certificate expires. This property will return an empty string if a secure connection has not been established with the server.

Data Type

String

See Also

CertificateIssued Property, CertificateIssuer Property, CertificateStatus Property, CertificateSubject Property, Secure Property

CertificateIssued Property

Return the date and time that the server certificate was issued.

Syntax

object.**CertificateIssued**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **CertificateIssued** property returns the date and time that the server certificate was issued. This property will return an empty string if a secure connection has not been established with the server.

Data Type

String

See Also

CertificateExpires Property, CertificateIssuer Property, CertificateStatus Property, CertificateSubject Property, Secure Property

CertificateIssuer Property

Returns information about the organization that issued the server certificate.

Syntax

object.**CertificateIssuer**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **CertificateIssuer** property returns a string that contains information about the organization that issued the server certificate. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the issuer's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification
Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Name	Desciption
C	The ISO standard two character country code
S	The name of the state or province
L	The name of the city or locality
O	The name of the company or organization
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for

This property will return an empty string if a secure connection has not been established with the server.

Example

The following example demonstrates how to extract the value of a relative distinguished name token:

```
Function GetCertNameValue(ByVal strValue As String, ByVal strFieldName As String)
As String
     Dim strFieldValue As String
     Dim cchValue As Long Integer, cchFieldName As Long Integer
     Dim nOffset As Long Integer

     GetCertNameValue = ""
     cchValue = Len(strValue)
     cchFieldName = Len(strFieldName)

     If cchValue = 0 Or cchFieldName = 0 Then
          Exit Function
     End If

     nOffset = InStr(strValue, strFieldName & "=")

     If nOffset > 0 Then
          '
          ' If the field name was found in the string, then
          ' remove everything to the left of the token from
          ' the string
          '
          strFieldValue = Right(strValue, cchValue - (nOffset + cchFieldName))
          '
          ' If the value is quoted, then strip off the leading
          ' quote and look for the ending quote in the string;
          ' otherwise look for the comma that marks the end of
          ' the field name/value pair
          '
          If Left(strFieldValue, 1) = Chr(34) Then
               strFieldValue = Right(strFieldValue, Len(strFieldValue) - 1)
               nOffset = InStr(strFieldValue, Chr(34))
          Else
               nOffset = InStr(strFieldValue, ",")
          End If
          '
          ' If the offset is 0, then the name/value pair is
          ' the last token in the string; otherwise, remove
          ' everything to the right of that position
          '
          If nOffset > 0 Then
               strFieldValue = Left(strFieldValue, nOffset - 1)
          End If

          GetCertNameValue = strFieldValue
     End If

End Function
```

This function could then be used to return the name of the company who issued the server certificate:

```
Dim strIssuer As String
Dim strCompanyName As String

strIssuer = InternetMail1.CertificateIssuer
If Len(strIssuer) = 0 Then
     MsgBox "A secure connection has not been established"
Else
     strCompanyName = GetCertNameValue(strIssuer, "O")
     MsgBox "This certificate was issued by " & strCompanyName
End If
```

Data Type

String

See Also

CertificateExpires Property, CertificateIssued Property, CertificateStatus Property, CertificateSubject Property, Secure Property

CertificateName Property

Return or set the location of the certificate name.

Syntax

object.**CertificateName** [= *value*]

The **CertificateName** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the name of the client certificate which should be provided to the server if requested.

Remarks

The **CertificateName** property specifies the name of a client certificate on the local host which is used with a secure connection. If the property is set to an empty string, then no client certificate will be provided to the server.

Data Type

String

See Also

Secure Property

CertificateStatus Property

Return the status of the server certificate.

Syntax

object.**CertificateStatus**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **CertificateStatus** property may return one of the following values:

Constant	Value	Description
mailCertificateNone	0	No certificate information is available. A secure connection was not established with the server.
mailCertificateValid	1	The certificate is valid.
mailCertificateNoMatch	2	The certificate is valid, however the domain name specified in the certificate does not match the domain name of the site that the client has connected to. This is typically the case if the **ServerName** property is set to an IP address rather than a host name. It is recommended that the client examine the **CertificateSubject** property to determine the domain name of the site that the certificate was issued for.
mailCertificateExpired	3	The certificate has expired and is no longer valid. The client can examine the **CertificateExpires** property to determine when the certificate expired.
mailCertificateRevoked	4	The certificate has been revoked and is no longer valid. It is recommended that the client application immediately terminate the connection if this status is returned.
mailCertificateUntrusted	5	The certificate has not been issued by a trusted authority, or the certificate is not trusted on the local host. It is recommended that the client application immediately terminate the connection if this status is returned.
mailCertificateInvalid	6	The certificate is invalid. This typically indicates that the internal structure of the certificate is damaged. It is recommended that the client application immediately terminate the connection if this status is returned.

This property value should be checked after the connection to the server has completed, but prior to beginning a transaction. If a secure connection has not been established, this property will return a value of zero.

Data Type

String

See Also

CertificateExpires Property, CertificateIssued Property, CertificateIssuer Property, CertificateSubject Property, Secure Property

CertificateStore Property

Return or set the location of the certificate store.

Syntax

object.**CertificateStore** [= *value*]

The **CertificateStore** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the name of the certificate store which contains the client certificate.

Remarks

The **CertificateStore** property specifies the name of the certificate store on the local host which contains the client certificate. If the property is set to an empty string, then the default certificate store will be used.

This property is only used when a secure connection has been established and a client certificate name has been specified.

Data Type

String

See Also

Secure Property

CertificateSubject Property

Returns information about the organization that the server certificate was issued to.

Syntax

object.**CertificateSubject**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **CertificateSubject** property returns a string that contains information about the organization that the server certificate was issued for. The string value is a comma separated list of tagged name and value pairs. In the nomenclature of the X.500 standard, each of these pairs are called a relative distinguished name (RDN), and when concatenated together, forms the subject's distinguished name (DN). For example:

```
C=US, O="RSA Data Security, Inc.", OU=Secure Server Certification
Authority
```

To obtain a specific value, such as the name of the issuer or the issuer's country, the application must parse the string returned by this property. Some of the common tokens used in the distinguished name are:

Name	Description
C	The ISO standard two character country code
S	The name of the state or province
L	The name of the city or locality
O	The name of the company or organization
OU	The name of the department or organizational unit
CN	The common name; with X.509 certificates, this is the domain name of the site the certificate was issued for

This property will return an empty string if a secure connection has not been established with the server.

Data Type

String

See Also

CertificateExpires Property, CertificateIssued Property, CertificateIssuer Property, CertificateStatus Property, Secure Property

CipherStrength Property

Return the length of the key used by the encryption algorithm.

Syntax

object.**CipherStrength**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **CipherStrength** property returns the number of bits in the key used to encrypt the secure data stream. Common values returned by this property are 40, 56 and 128. A key length of 40-bits is considered to be relatively insecure, while a key length of 56-bit is considered moderate and 128-bit keys are considered to be very secure. If this property returns a value of 0, this means that a secure connection has not been established with the server.

Data Type

Long Integer

See Also

HashStrength Property, Secure Property, SecureCipher Property, SecureHash Property, SecureKeyExchange Property, SecureProtocol Property

ContentID Property

Return or set the content identifier for the selected message part.

Syntax

object.**ContentID** [= *value*]

The **ContentID** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which uniquely identifies the content in the current message part.

Remarks

The **ContentID** property returns the unique content identifier string for the current message part. This multipart header field is not commonly used, and if undefined, will return an empty string. If set, this will change the value of the Content-ID header field in the current message part.

Data Type

String

See Also

ContentLength Property, ContentType Property

ContentLength Property

Returns the size of the data stored in the selected message part.

Syntax

object.**ContentLength**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **ContentLength** property returns the size of the current message part in bytes. This property is read-only, and is automatically updated when the current message part changes.

Data Type

Long Integer

See Also

ContentID Property, ContentType Property

ContentType Property

Return or set the content type of the selected message part.

Syntax

object.**ContentType** [= *value*]

The **ContentType** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which identifies the content contained in the current message part.

Remarks

The **ContentType** property returns the MIME type for the currently selected message part. The type string consists of a primary type and secondary sub-type separated by a slash, followed by one or more optional parameters delimited by semi-colons. For example:

```
text/plain; charset=us-ascii
```

The **text** designation indicates that this message part contains readable text, and the **plain** sub-type indicates that the text does not contain any special encoding. The optional parameter which follows the content type provides additional information about the content. In this example, it specifies which character set should be used to display the text. The two common character sets used are ISO-8859 and US-ASCII (which is the default character set that is used if none is specified).

There are seven predefined, standard content types, each with their own sub-types. The following table lists these types, along with some common sub-types that are found in messages:

Type	Sub-Types	Description
text	plain, richtext, html	Indicates that the message part contains text. This is the most common type found in mail messages; if no content type is explicitly defined, then it is assumed to be plain text.
image	gif, jpeg	Indicates that the message part contains a graphics image.
audio	basic, aiff, wav	Indicates that the message part contains audio data; the basic sub-type is 8-bit PCM encoded audio (commonly found with the .au filename extension).
video	mpeg, avi	Indicates that the message part contains a video clip in the specified format.
application	octet-stream	Indicates that the message part contains application specific data, typically used with the octet-stream sub-type to indicate binary file attachments for executable programs, compressed file archives, etc.
message	rfc822	Indicates that the message part contains a complete RFC 822 compliant message, complete with headers.
multipart	mixed, alternative	Indicates that this is part of a message that contains multiple parts with different content types.

The three most common content types that are used in applications are text/plain for the mail message body, application/octet-stream for binary file attachments and multipart/mixed for messages that contain both text and attached files. For more information about the different content types, refer to the Multipurpose Internet Mail Extensions (MIME) standards document RFC 1521.

Data Type

String

See Also

ContentID Property, ContentLength Property

Date Property

Return or set the date for the current message.

Syntax

object.**Date** [= *value*]

The **Date** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the date and time.

Remarks

The **Date** property returns the value of the Date header field in the current message. Setting this property causes the Date field to be updated with the specified value. When setting the date, any one of the following formats may be used:

Format	Example
mm/dd/yy[yy] hh:mm[:ss]	03/01/98 12:00
yy[yy]/mm/dd hh:mm[:ss]	98/03/01 12:00
dd mmm yy[yy] hh:mm[:ss]	01 Mar 1998 12:00:00
mmm dd yy[yy] hh:mm[:ss]	Mar 01 1998 12:00:00

Any extraneous information that may be included in the date string, such as the day of the week, is ignored. In addition to the date and time, the string may also include a time zone specification at the end. If no time zone is specified, the current time zone is used.

When specifying a time zone, the value should either be prefixed by a plus sign (+) to indicate that the time zone is to the east of GMT, or a minus sign (-) to indicates that it's to the west. Four digits follow, with the first two indicating the number of hours east or west of GMT, and the last two digits indicating the number of minutes. Therefore, a value of -0800 means that the time zone is eight hours to the west of GMT, or in other words, the Pacific time zone. Regardless of the format of the string assigned to the property, it always returns the date in the same standard format.

Note that the **Localize** property affects how dates are processed by the control. If enabled, dates are automatically adjusted for the local time zone. By default, localization is disabled.

Data Type

String

See Also

Localize Property

Domain Property

Return or set the local domain name.

Syntax

object.**Domain** [= *value*]

The **Domain** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the local domain name.

Remarks

The **Domain** property specifies the domain name of the local host, and is used to identify the current system when sending messages. If this property is not defined, then the local host name will be used.

Note that explicitly setting the **Domain** property to a value that does not match your local host name may cause some mail servers to reject any messages that you attempt to send.

Data Type

String

Encoding Property

Return or set the content encoding for the current message part.

Syntax

object.**Encoding** [= *value*]

The **Encoding** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies type of encoding used.

Remarks

The **Encoding** property returns a string which specifies the method used for encoding the current message part. Setting this property causes the Content-Transfer-Encoding header value to be updated. The following values are commonly used:

Type	Description
7bit	The default transfer encoding type, which consists of printable ASCII characters.
8bit	Printable ASCII characters, including those characters with the high-bit set (as is common with the ISO Latin-1 character set); this encoding type is not commonly used.
base64	The transfer encoding type commonly used to convert binary data into 7-bit ASCII characters so that it may be transported safely through the mail system.
binary	All characters; binary transfer encoding is rarely used.

quoted-printable	Printable ASCII characters, with non-printable or extended characters represented using their hexadecimal equivalents.
x-uuencode	A transfer encoding type similar in function to the base64 encoding method.

Note that setting this property only updates the Content-Transfer-Encoding header value. To control the actual encoding method used when attaching a file, see the **AttachFile** method.

Data Type

String

See Also

ContentLength Property, ContentType Property, ExtractFile Method, AttachFile Method

From Property

Return or set the address of the person who sent the message.

Syntax

object.**From** [= *value*]

The **From** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the address of the sender.

Remarks

The **From** property returns the address of the person who sent the message. Setting the property causes the From header field in the current message to be updated with the new value.

Data Type

String

See Also

Bcc Property, Cc Property, ReplyTo Property, To Property

HashStrength Property

Return the length of the message digest that was selected.

Syntax

object.**HashStrength**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **HashStrength** property returns the number of bits used in the message digest (hash) that was selected. Common values returned by this property are 128 and 160. If this property returns a value of 0, this means that a secure connection has not been established with the server.

Data Type

Long Integer

See Also

CipherStrength Property, Secure Property, SecureCipher Property, SecureHash Property, SecureKeyExchange Property, SecureProtocol Property

LastError Property

Return or set the last error code.

Syntax

object.**LastError** [= *value*]

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A integer which specifies an error code.

Remarks

The **LastError** property can be read to determine the last error that occurred for this instance of the object. If a value is assigned to this property, it must either be zero (to clear the error) or a valid error code, which will cause the object to raise the specified error.

Data Type

Long Integer

See Also

OnError Event

LastErrorString Property

Return a description of the last error that occurred.

Syntax

object.**LastErrorString**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **LastErrorString** property returns a string that contains a description of the last error that occurred.

Data Type

String

See Also

LastError Property

LastMessage Property

Return the number of the last message available on the server.

Syntax

object.**LastMessage**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **LastMessage** property returns the last message available on the server. Note that unlike the **MessageCount** property, this value remains constant even when a message is deleted.

Data Type

Long Integer

See Also

Message Property, MessageCount Property, MessageSize Property

Library Property

Return or set the file name of the Windows Sockets networking library.

Syntax

object.**Library** [= *value*]

The **Library** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
dllname	A string which specifies the name of the Windows Sockets library.

Remarks

The **Library** property returns the full pathname of the Windows Sockets library loaded by the control. Setting the property to the file name of an alternate DLL causes that library to be loaded. By default, the library WSOCK32.DLL is automatically loaded when any network function is called.

This should be the first property set by the control when it is loaded. Attempting to set this property after a library has already been loaded will generate an error.

Data Type

String

LocalAddress Property

Return the Internet address of the local host.

Syntax

object.**LocalAddress**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **LocalAddress** property returns the Internet address of the local host as a string in dotted notation. If there is an active connection to a server, then the return value will depend on the network interface that was used to establish the connection. If there isn't a connection, then the default address for the local host will be returned.

Data Type

Long Integer

See Also

LocalName Property

Localize Property

Enable or disable message localization.

Syntax

object.**Localize** [= { **True** | **False** }]

The object is an expression that evaluates to an InternetMail object. The property returns a boolean value.

Remarks

The **Localize** property is used to enable or disable localization features of the object. Currently this only affects the way in which dates are processed. If set to True, the date and time will be adjusted for the local time zone when setting and reading the **Date** property. The default value for this property is False.

Data Type

Boolean

See Also

Date Property

LocalName Property

Return the Internet domain name of the local host.

Syntax

object.**LocalName**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **LocalName** property returns the Internet domain name for the local host. If there is an active connection to a server, then the domain name will depend on the network interface that was used to establish the connection. If there isn't a connection, then the default domain name for the local host will be returned.

Data Type

String

See Also

LocalAddress Property, Resolve Method

MailboxSize Property

Return the size of the current mailbox in bytes.

Syntax

object.**MailboxSize**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **MailboxSize** property returns the combined size of all of the available messages in the current mailbox. Note that as messages are deleted from the mailbox, this property value will decrease.

Data Type

Long Integer

Mailer Property

Return or set the name of the mailer application.

Syntax

object.**Mailer** [= *value*]

The **Mailer** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the name of application.

Remarks

The **Mailer** property returns the value of the X-Mailer header field in the current message. This is typically used to identify the application that created the message, however it is not required that this be specified. If the header field is not present in the message, this property will return an empty string. Setting this property will change the value of the X-Mailer header. If the property is set to an empty string, the header will be removed from the message.

Data Type

String

See Also

GetHeader Method, SetHeader Method

Message Property

Return or set the current message headers and text.

Syntax

object.**Message** [= *value*]

The **Message** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the complete message.

Remarks

The **Message** property returns the current message, including the headers and all message parts, as a string. Setting this property will cause the current message to be cleared and replaced by the new value. The string contents must follow the standard specifications for a message. If the property is set to an empty string, the current message is cleared.

Note that setting the **Message** property will cause the value of the **Bcc** property to be reset to an empty string.

Data Type

String

MessageCount Property

Return the number of messages available on the server.

Syntax

object.**MessageCount**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **MessageCount** property returns the number of messages available to be retrieved from the mail server. When a message is deleted from the mailbox, this value will decrease. To determine the highest valid message number, regardless of any deleted messages, use the **LastMessage** property.

Data Type

Long Integer

See Also

LastMessage Property, Message Property, MessageSize Property

MessageID Property

Return a unique identifier for the current message.

Syntax

object.**MessageID**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **MessageID** property returns the value of the Message-ID header field, a string which is assigned by the mail server to uniquely identify the current message.

Data Type

String

See Also

GetHeader Method

MessageIndex Property

Return or set the current message number on the server.

Syntax

object.**MessageIndex** [= *value*]

The **MessageIndex** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies the message number on the server.

Remarks

The **MessageIndex** property sets or returns the current message number on the server. Message numbers range from 1 through the number of messages available on the server, as returned by the **LastMessage** property. Setting the **MessageIndex** property to an invalid message number will generate an error.

Data Type

Long Integer

See Also

MessageCount Property, MessageSize Property

MessagePart Property

Return or set the current part in a multipart message.

Syntax

object.**MessagePart** [= *part*]

Part	Description
object	An object expression that evaluates to an InternetMail object.
part	A long integer which specifies the message part.

Remarks

The **MessagePart** property returns the current message part index. All messages have at least one part, which consists of one or more header fields, followed by the body of the message. The default part, part 0, refers to the main message header and body. If the message contains multiple parts (as with a message that contains one or more attached files), the **MessagePart** property can be set to refer to that specific part of the message.

Messages with file attachments typically consist of a message part which describes the contents of the attachment, followed by the attachment itself. For a message with one attached file, there would be a total of three parts. Part 0 would refer to the main message part, which contains the headers such as From, To, Subject, Date and so on. For multipart messages, part 0 typically does not have a message body, since any text is usually created as a seperate part (for those messages that do not contain multiple parts, the part 0 body contains the text message). Part 1 would contain the text describing the attachment, and part 2 would contain the attachment itself. If the attached file is binary, then the transfer encoding type would usually be base64.

Data Type

Long Integer

See Also

ContentType Property, ContentLength Property, Encoding Property, MessageParts Property

MessageParts Property

Return the number of parts in the current message.

Syntax

object.**MessageParts**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **MessageParts** property returns the number of parts in the current message. All messages have at least one part, referenced as part 0. Multipart messages will consist of additional parts which may be accessed by setting the **MessageParts** property.

Data Type

Long Integer

See Also

MessagePart Property, AttachFile Method, ExtractFile Method, ExportMessage Method

MessageSize Property

Return the size of the current message in bytes.

Syntax

object.**MessageSize**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **MessageSize** property returns the size of the current message in bytes. The size includes the header and body portion of the message.

Data Type

Long Integer

See Also

Message Property, MessageCount Property

MessageText Property

Return or change the text in the current message part.

Syntax

object.**MessageText** [= *value*]

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the new message text.

Remarks

The **MessageText** property returns the body of the current message part. Setting this property replaces the entire message body with the new text. Note that setting the property to an empty string deletes the body of the current message part, but does not delete the message part itself.

Data Type

String

MessageUID Property

Return the UID for the current message on the mail server.

Syntax

object.**MessageUID**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **MessageUID** property returns a string which uniquely identifies the message on the server. The identifier is assigned by the mail server, and retains the same value across multiple client sessions. This value is typically used when the client wants to leave a message on the mail server, but does not wish to retrieve the message contents multiple times. For example, the client can store the UID for each message that it retrieves, but does not delete from the server. The next time that it connects to the mail server, it compares the UID of a message against the stored values. If there is a match, the client knows that the message has already been retrieved, and does not need to do so again.

This property requires that the mail server support the optional UIDL command. If the command is not supported, this property will always return an empty string. Note that the UID for the message comes from the mail server and is not the same as the Message-ID header field in the message itself.

Data Type

String

NameServer Property

Return or set the Internet address for a nameserver.

Syntax

object.**NameServer**(*index*) [= *value*]

The **NameServer** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
index	An integer which specifies the nameserver.
value	A string which specifies the new server address.

Remarks

The **NameServer** property array is used to specify one or more nameservers. The address value must be an Internet address in dot notation. The *index* specifies which nameserver to set or return a value for. There may be up to four nameservers defined for any single instance of the object.

A nameserver is a computer which converts a domain name, such as microsoft.com, into an IP address which can be used to establish a connection to a remote server. In addition to mapping domain names, nameservers also can return information about what servers are responsible for handling mail messages for a given domain. These servers are called "mail exchanges" and there may be more than one mail exchange for a domain, each with its own assigned priority. This information is used by the **SendMessage** method to determine the address of the appropriate SMTP server in order to deliver the message to the specified recipient.

If no nameservers are specified, then the default nameservers for the local host will be used. For those systems which use dial-up connections to the Internet, this requires that the system have an active connection established before this object is initialized.

Data Type

String

Options Property

Return or set the options for the current object.

Syntax

object.**Options** [= *value*]

The **Options** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies one or more options.

Remarks

The **Options** property returns or modifies the options used for retrieving and sending messages. The value is represented as one or more bitflags which may be combined using the logical or operator. The following options are defined:

Constant	Description
mailOptionNoStartTLS	For secure POP3 and SMTP connections only; prevents the use of the STARTTLS command which is used to negotiate a secure session. This option should only be used if required by the server.
mailOptionAPOP	Causes the APOP authentication method to be used when connecting to a POP3 mail server. The default is to use standard password authentication.
mailOptionAllHeaders	Preserves all headers in the message when it is exported, including the Received and Return-Path headers which are normally excluded.
mailOptionKeepOrder	Preserves the order of the headers when it is exported; by default, some headers may be re-ordered.
mailOptionNotify	Notify the sender of the delivery status of the message, if the server supports delivery status notification. This option is a combination of the *mailNotifySuccess*, *mailNotifyFailure*, *mailNotifyDelay* and *mailReturnHeaders* options.
mailNotifySuccess	If the mail server supports delivery status notification, this causes a message to be returned to the sender once it has been successfully delivered.
mailNotifyFailure	If the mail server supports delivery status notification, this causes a message to be returned to the sender if it could not be delivered.
mailNotifyDelay	If the mail server supports delivery status notification, this causes a message to be returned to the sender if delivery has been delayed.
mailReturnHeaders	If the mail server supports delivery status notification, this causes a message to be returned which contains the headers of the message that was sent.
mailReturnMessage	If the mail server supports delivery status notification, this causes a message to be returned which contains the complete message that was sent.

Data Type

Long Integer

Organization Property

Return or change the text in the current message part.

Syntax

object.**Organization** [= *name*]

The **Organization** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
name	A string which specifies the organization name.

Remarks

The **Organization** property returns the name of the organization that sent the current message. Setting this property updates the specified header value. Note that many mailers do not generate an Organization header field, in which case the property value will be an empty string.

Data Type

String

See Also

GetHeader Method, SetHeader Method

Password Property

Return or set the password for the current user.

Syntax

object.**Password** [= *value*]

The **Password** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the current user password.

Remarks

The **Password** property specifies the password used to authenticate the user. If the property is not explicity set, then an application must provide the password to the **Connect** method. Once the connection has been established, this property will be updated with the appropriate value.

Data Type

String

See Also

UserName Property, Connect Method

Priority Property

Return or set the current message priority.

Syntax

object.**Priority** [= *name*]

The **Priority** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
name	A string which specifies the message priority.

Remarks

The **Priority** property returns the current priority for the message. Setting this property value causes the X-Priority header to be updated with the specified value.

There is no strict standard for specifying message priority. The convention is to use a number from 1-5, with 1 indicating the highest priority, 3 as normal priority and 5 as the lowest priority. Some mailers follow the number with a space and then text that describes the priority level.

Data Type

String

See Also

GetHeader Method, SetHeader Method

Recipient Property

Return the address of a message recipient.

Syntax

object.**Recipient**(*index*)

The **Recipient** property array syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
index	An integer which specifies the recipient.

Remarks

The **Recipient** property array returns the e-mail address of one of the recipients of the current message, as specified by the *index* argument. This property enables an application to enumerate all of the recipient addresses for the current message without having to parse the individual **To**, **Cc** and **Bcc** property values. Note that this property array is read-only; to change the recipients for the current message you must set the **To**, **Cc** or **Bcc** properties.

The *index* argument specifies which address to return, with a base value of zero up to the number of recipients.

The string returned by the **Recipient** property contains only the actual e-mail address and does not include the name of the recipient or any comments that may have been included with the address. For example, if the **To** property is set to "John Doe <jdoe@company.com>" then the **Recipient** property would return a value of "jdoe@company.com" for that address.

Example

The following example enumerates all of the recipients for the current message and adds them to a listbox:

```
For nIndex = 0 To InternetMail1.Recipients
    List1.AddItem InternetMail1.Recipient(nIndex)
Next
```

Data Type

String

See Also

Bcc Property, Cc Property, Recipients Property, To Property

Recipients Property

Return the number of recipients for the current message.

Syntax

object.**Recipients**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **Recipients** property returns the number of recipients for the current message. This value can be used in conjunction with the **Recipient** property array to enumerate the recipient e-mail addresses for the current message.

Example

The following example enumerates all of the recipients for the current message and adds them to a listbox:

```
For nIndex = 0 To InternetMail1.Recipients
    List1.AddItem InternetMail1.Recipient(nIndex)
Next
```

Data Type

Long Integer

See Also

Recipient Property

RelayServer Property

Return or set the host name or address of a relay server.

Syntax

object.**RelayServer** [= *value*]

The **RelayServer** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies a host name or IP address.

Remarks

The **RelayServer** property is used to specify an alternate mail server which will deliver messages for the current user.

Normally, when the **SendMessage** method is used, the recipient address is used to determine what mail server is responsible for accepting messages for that user. However, under some circumstances this may not be desirable or even possible. For example, many Internet Service Providers (ISPs) require that customers send all messages through their servers and block any attempt to establish a direct connection with another mail server. Setting the **RelayServer** property to the host name or address of the ISP mail server will cause all messages to be relayed through that server rather than directly to the recipient.

Note that using a mail server as a relay without the permission of the organization or individual who owns that server may violate Acceptable Use Policies and/or Terms of Service agreements with your service provider.

Data Type

String

See Also

LocalName Property, Resolve Method

RelayPort Property

Return or set the port number for the specified relay server.

Syntax

object.**RelayPort** [= *value*]

The **RelayPort** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies the port number.

Remarks

The **RelayPort** property defines the port number which is used to establish a connection with the mail server. This property is used in conjunction with the **RelayServer** property to specify an alternate mail server which is responsible for delivering messages for the current user.

If this property is not set, the default SMTP port will be used when connecting to a relay mail server.

Data Type

Long Integer

See Also

RelayServer Property

ReplyTo Property

Return or set the address of the person who should receive replies to this message.

Syntax

object.**ReplyTo** [= *value*]

The **ReplyTo** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies a return address.

Remarks

The **ReplyTo** property returns the address of the user who should receive replies to the current message. Setting this property updates the Reply-To header field with the specified value.

Data Type

String

See Also

Bcc Property, Cc Property, From Property, To Property

ReturnReceipt Property

Return or set the address of the person who should receive a message indicating that the message has been read.

Syntax

object.**ReturnReceipt** [= *value*]

The **ReturnReceipt** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies an address.

Remarks

The **ReturnReceipt** property returns the address of the person who should receive a message indicating that the current message has been read. Setting this property updates the Disposition-Notification-To header field with the specified value.

Setting the **ReturnReceipt** property does not automatically cause an acknowledgement to be returned to the sender. An application is responsible for checking to make sure the header field contains a valid address and then generating the return receipt message.

Data Type

String

Secure Property

Specify if a connection to the server is secure.

Syntax

object.**Secure** [= { **True** | **False** }]

The object is an expression that evaluates to an InternetMail object. The property returns a boolean value.

Remarks

The **Secure** property determines if a secure connection is established to the server. The default value for this property is **False**, which specifies that a standard connection to the server is used. To establish a secure connection, the application must set this property value to **True** prior to calling the **Connect** method. Once the connection has been established, the client may retrieve messages from the server as with standard connections.

It is strongly recommended that any application that sets this property to **True** use error handling to trap any errors that may occur.

If the control is unable to initialize the security libraries, or otherwise cannot create a secure session for the client, an error will be generated when this property value is set.

Data Type

Boolean

Example

The following example establishes a secure connection to a server and retrieves a message:

```
  '
  ' Set the Secure property to True, which tells the
  ' component to initialize the security interface and
  ' attempt to establish a secure connection
  '
  On Error Resume Next: Err.Clear
  InternetMail1.Secure = True

  If Err.Number Then
      MsgBox "Unable to initialize the security interface"
      Exit Sub
  End If

  On Error GoTo 0

  nError = InternetMail1.Connect(strServerName, nServerPort, strUserName,
strPassword)
  If nError <> 0 Then
      MsgBox "Unable to connect to server " & strServerName, vbExclamation
      Exit Sub
  End If

  If InternetMail1.CertificateStatus <> mailCertificateValid Then
       nResult = MsgBox("The server certificate could not be validated" &
vbCrLf & _
                        "Are you sure you wish to continue?", vbYesNo)

       If nResult = vbNo Then
            InternetMail1.Disconnect
            Exit Sub
       End If
  End If

  nError = InternetMail1.GetMessage(1)
  If nError <> 0 Then
       InternetMail1.Disconnect
       MsgBox "Unable to retrieve message from server " & strServerName
       Exit Sub
  End If

  InternetMail1.Disconnect
```

See Also

CertificateExpires Property, CertificateIssued Property, CertificateIssuer Property, CertificateStatus Property, CertificateSubject Property, CipherStrength Property, HashStrength Property, SecureCipher Property, SecureHash Property, SecureKeyExchange Property, SecureProtocol Property, Connect Method

SecureCipher Property

Return the encryption algorithm used to establish the secure connection with the server.

Syntax

object.**SecureCipher**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **SecureCipher** property returns an integer value which identifies the algorithm used to encrypt the data stream. This property may return one of the following values:

Constant	Value	Description
mailCipherNone	0	No cipher has been selected. This is not a secure connection with the server.
mailCipherRC2	1	The RC2 block cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.
mailCipherRC4	2	The RC4 stream cipher was selected. This is a variable key length cipher which supports keys between 40- and 128-bits in length, in 8-bit increments.
mailCipherDES	4	The DES (Data Encryption Standard) block cipher was selected. This is a fixed key length cipher using 56-bit keys.
mailCipherDES3	8	The Triple DES block cipher was selected. This cipher encrypts the data three times using different keys, effectively using a 168-bit key length.
mailCipherSkipjack	16	The Skipjack block cipher was selected. This is a fixed key length cipher, using 80-bit keys.

If a secure connection has not been established, this property will return a value of zero.

Data Type

Long Integer

See Also

CipherStrength Property, HashStrength Property, Secure Property, SecureHash Property, SecureKeyExchange Property, SecureProtocol Property

SecureHash Property

Return the message digest selected when establishing the secure connection with the server.

Syntax

object.**SecureHash**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **SecureHash** property returns an integer value which identifies the message digest algorithm that was selected when a secure connection is established. This property may return one of the following values:

Constant	Value	Description
mailHashNone	0	No message digest algorithm has been selected. This is not a secure connection with the server.
mailHashMD5	1	The MD5 algorithm was selected. This algorithm takes a message of arbitrary length and produces a 128-bit message digest.
mailHashSHA	2	The SHA algorithm was selected. This algorithm produces a 160-bit message digest.

If a secure connection has not been established, this property will return a value of zero.

Data Type

Long Integer

See Also

CipherStrength Property, HashStrength Property, Secure Property, SecureCipher Property, SecureKeyExchange Property, SecureProtocol Property

SecureKeyExchange Property

Return the key exchange algorithm used to establish the secure connection with the server.

Syntax

object.**SecureKeyExchange**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **SecureKeyExchange** property returns an integer value which identifies the key-exchange algorithm used when establishing a secure connection. This property may return one of the following values:

Constant	Value	Description
mailKeyExchangeNone	0	No key exchange algorithm has been selected. This is not a secure connection with the server.
mailKeyExchangeRSA	1	The RSA public key exchange algorithm has been selected.
mailKeyExchangeKEA	2	The KEA public key exchange algorithm has been selected. This is an improved version of the Diffie-Hellman public key algorithm.
mailKeyExchangeDH	4	The Diffie-Hellman public key exchange algorithm has been selected.

If a secure connection has not been established, this property will return a value of zero.

Data Type

Long Integer

See Also

CipherStrength Property, HashStrength Property, Secure Property, SecureCipher Property, SecureHash Property, SecureProtocol Property

SecureProtocol Property

Return the security protocol used to establish the secure connection with the server.

Syntax

object.**SecureProtocol**

The object is an expression that evaluates to an InternetMail object. The property returns a long integer value.

Remarks

The **SecureProtocol** property returns an integer value which identifies the protocol used to establish the secure connection. This property may return one of the following values:

Constant	Value	Description
mailProtocolNone	0	No protocol has been selected. This is not a secure connection with the server.
mailProtocolSSL2	1	The Secure Sockets Layer (SSL) version 2.0 protocol has been selected.
mailProtocolSS3	2	The Secure Sockets Layer (SSL) version 3.0 protocol has been selected.
mailProtocolPCT1	4	The Private Communication Technology (PCT) version 1.0 protocol has been selected.
mailProtocolTLS1	8	The Transport Layer Security (TLS) version 1.0 protocol has been selected.

If a secure connection has not been established, this property will return a value of zero.

Data Type

Long Integer

See Also

CipherStrength Property, HashStrength Property, Secure Property, SecureCipher Property, SecureHash Property, SecureKeyExchange Property

ServerName Property

Return or set the host name of the current mail server.

Syntax

object.**ServerName** [= *value*]

The **ServerName** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies a host name or address.

Remarks

The **ServerName** property returns the name of the mail server that the client is connected to. Setting this property specifies the host name or Internet address for a subsequent connection.

If the **ServerName** property is not explicity set, then an application must provide the host name or address to the **Connect** method. Once the connection has been established, this property will be updated with the appropriate value. If the server uses a non-standard port number, it can be specified using the **ServerPort** property.

The mail server must support Post Office Protocol v3 (POP3) to retrieve messages. Setting this property does not affect what server is used to deliver messages. See the **RelayServer** and **RelayPort** properties to specify a mail server that is responsible for relaying messages.

Data Type

String

See Also

LocalName Property, Resolve Method

ServerPort Property

Return or set the port number for the current mail server.

Syntax

object.**ServerPort** [= *value*]

The **ServerPort** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies a TCP port number.

Remarks

The **ServerPort** property returns the port number that was used to establish a connection with the mail server. Setting this property specifies an alternate port number to use for a subsequent connection. A value of zero specifies that the default port should be used for the connection.

The mail server must support Post Office Protocol v3 (POP3) to retrieve messages. Setting this property does not affect what server is used to deliver messages. See the **RelayServer** and **RelayPort** properties to specify a mail server that is responsible for relaying messages.

Data Type

Long Integer

See Also

RelayServer Property, RelayPort Property

Subject Property

Return or set the subject of the current message.

Syntax

object.**Subject** [= *value*]

The **Subject** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the subject of the message.

Remarks

The **Subject** property returns the subject of the current message. Setting this property updates the Subject header with the specified value. Note that not all messages have subjects, in which case this property will return an empty string.

Data Type

String

See Also

GetHeader Method, SetHeader Method

ThrowError Property

Enable or disable error handling by the container of the control.

Syntax

object.**ThrowError** [= { **True** | **False** }]

The object is an expression that evaluates to an InternetMail object. The property returns a boolean value.

Remarks

Error handling for methods of the control can be done in either of two different styles, according to the value of this property.

If the **ThrowError** property is set to False, methods will not raise an exception if an error occurs. Instead, the application should check the return value of the method and report any errors based on that value. It is the responsibility of the application to interpret the error code and take an appropriate action. This is the default value for the property.

If the **ThrowError** property is set to True, any method which generates an error will cause the component to raise an exception which must be handled or the application will terminate.

Note that if an error occurs while a property is being read or written, an error will be raised regardless of the value of this property. This property only controls how errors are handled when calling methods.

Data Type

Boolean

See Also

LastError Property, OnError Event

Timeout Property

Return or set the amount of time until a blocking network operation is aborted.

Syntax

object.**Timeout** [= *value*]

The **Timeout** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies the time in seconds.

Remarks

The **Timeout** property controls the amount of time that the component will wait for a network operation to complete before aborting the operation and returning an error. A value of zero specifies that the component will wait an indefinite period of time for the operation to complete. The default value for this property is sixty (60) seconds. It may be required to increase this value if a slow or unreliable network connection is being used.

Data Type

Long

TimeZone Property

Return or set the current timezone offset in seconds.

Syntax

object.**TimeZone** [= *value*]

The **TimeZone** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies timezone offset in seconds.

Remarks

The **TimeZone** property returns the current offset from UCT in seconds. Setting the property changes the current timezone offset to the specified value. The value of this property is initially determined by the date and time settings on the local system.

The **TimeZone** property value is used in conjunction with the **Localize** property to control how message date and time localization is handled.

Data Type

Long

Example

The following code enables localization and changes the current timezone to Eastern Standard, which is five hours (18,000 seconds) west of UCT:

```
InternetMail1.Localize = True
InternetMail1.TimeZone = (5 * 60 * 60)
```

See Also

Localize Property

To Property

Return or set the recipient of the current message.

Syntax

object.**To** [= *value*]

The **To** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies one or more recipient addresses.

Remarks

The **To** property returns the list of addresses that received a copy of the current message. If there is no current message, or the To header field is not defined, then this property will return an empty string.

Setting the **To** property creates or changes the value of the To header field in the message and specifies additional recipients of the message. Multiple addresses may be specified by separating them with a comma. Each address must conform to the standard Internet address format.

Data Type

String

See Also

Bcc Property, Cc Property, From Property, ReplyTo Property

Trace Property

Enable or disable network function level tracing.

Syntax

object.**Trace** [= { **True** | **False** }]

The object is an expression that evaluates to an InternetMail object. The property returns a boolean value.

Remarks

The **Trace** property is used to enable or disable the tracing of network function calls and is primarily used as a debugging tool. When enabled, each function call is logged to a file, including the function parameters, return value and error code if applicable. This facility can be enabled and disabled at run time, and the trace log file can be specified by setting the **TraceFile** property. All function calls that are being logged are appended to the trace file, if it exists. If no trace file exists when tracing is enabled, the trace file is created.

The tracing facility is enabled or disabled for an entire process. This means that once tracing is enabled for a given instance of the object, all of the function calls made by the process will be logged.

If tracing is not enabled, there is no negative impact on performance or throughput. Once enabled, application performance can degrade, especially in those situations in which multiple processes are being traced or the trace file is fairly large. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

Data Type

Boolean

See Also

TraceFile Property, TraceFlags Property

TraceFile Property

Return or specify the network function trace output file.

Syntax

object.**TraceFile** [= *value*]

The **TraceFile** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies a valid file name.

Remarks

The **TraceFile** property is used to specify the name of the trace file that is created when network function tracing is enabled. If this property is set to an empty string, then a file named CSTRACE.LOG is created in the system's temporary directory. If no temporary directory exists, then the file is created in the current working directory.

If the file exists, the trace output is appended to the file, otherwise the file is created. Since function tracing is enabled per-process, the trace file is shared by all instances of the object being used. Since trace files can grow very quickly, even with modest applications, it is recommended that you delete the file when it is no longer needed.

The trace file has the following format:

```
VB6 INF: WSAAsyncSelect(46, 0xcc4, 0x7e9, 0x27) returned 0
VB6 WRN: connect(46, 192.0.0.1:1234, 16) returned -1 [10035]
VB6 ERR: accept(46, NULL, 0x0) returned -1 [10038]
```

The first column contains the name of the process that is being traced; in this case, it is Visual Basic 6.0. The second column identifies if the trace record is reporting information, a warning, or an error. What follows is the name of the function being called, the arguments passed to the function and the function's return value. If a warning or error is reported, the error code is included in brackets.

If parameters are passed as integer values, they are recorded in decimal. If the parameter or return value is a memory address, it is recorded as a hexadecimal value preceded with "0x". Those functions which expect Internet addresses are displayed in the following format:

```
aa.bb.cc.dd:nnnn
```

The first four numbers separated by periods represent the IP address, and the number following the colon represents the port number in host byte order. Note that in the second line of the above example, the control is attempting to connect to a system with the IP address 192.0.0.1 on port 1234.

Data Type

String

See Also

Trace Property, TraceFlags Property

TraceFlags Property

Return or set the current network function tracing flags.

Syntax

object.**TraceFlags** [= *value*]

The **TraceFlags** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A long integer which specifies one or more trace flags.

Remarks

The **TraceFlags** property is used to specify the type of information written to the trace file when network function tracing is enabled. The following values may be used:

Value	Constant	Description
0	mailTraceInfo	All function calls are written to the trace file. This is the default value.
1	mailTraceError	Only those function calls which fail are recorded in the trace file.
2	mailTraceWarning	Only those function calls which fail, or return values which indicate a warning, are recorded in the trace file.
4	mailTraceHexDump	All function calls are written to the trace file, plus all the data that is sent or received is logged, in both ASCII and hexadecimal format.

Since network function tracing is enabled per-process, the trace flags are shared by all instances of the object being used.

Warnings are generated when a non-fatal error is returned by a network function. For example, if data is being sent to the server and the error 10035 is returned, a warning is generated since the application simply needs to attempt to write the data at a later time.

Data Type

Long Integer

See Also

Trace Property, TraceFile Property

UserName Property

Return or set the current user name.

Syntax

object.**UserName** [= *value*]

The **UserName** property syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
value	A string which specifies the current user name.

Remarks

The **UserName** property specifies the name used to authenticate the user. If the property is not explicity set, then an application must provide the user name to the **Connect** method. Once the connection has been established, this property will be updated with the appropriate value.

Data Type

String

See Also

Password Property, Connect Method

Version Property

Return the current version of the object.

Syntax

object.**Version**

The object is an expression that evaluates to an InternetMail object. The property returns a string value.

Remarks

The **Version** property returns the current version of the object. This can be used by an application for validation purposes.

Data Type

String

—Methods—

AppendMessage Method

Append text to the current message part.

Syntax

object.**AppendMessage**(*msgtext*)

The **AppendMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
msgtext	A string which specifies the text to append.

Return Type

Long Integer

Remarks

The **AppendMessage** method appends the specified string to the end of the body of text in the current message part. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

AttachFile Method

Attach the specified file to the current message.

Syntax

object.**AttachFile**(*filename* [, *options*])

The **AttachFile** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
filename	A string which specifies the text to append.
options	A long integer which specifies one or more options.

Return Type

Long Integer

Settings

The settings for *options* is:

Constant	Description
mailAttachDefault	The file attachment encoding is based on the file content type. Text files are not encoded, and binary files are encoded using the standard base64 algorithm. This is the default option for file attachments.
mailAttachBase64	The file attachment is always encoded using the standard base64 algorithm, even if the attached file is a plain text file.
mailAttachUucode	The file attachment is always encoded using the standard uuencode algorithm, even if the attached file is a plain text file.

mailAttachQuoted	The file attachment is always encoded using quoted-printable encoding. Note that this encoding method is only recommended for text content, typically either as HTML or RTF.

Remarks

The **AttachFile** method attaches the specified file to the current message. If the message already contains one or more file attachments, then it is added to the end of the message. If the message does not contain any attached files, then it is converted to a multipart message and the file is appended to the message.

The *filename* argument specifies the name of the file to be attached to the message. If the file is empty or does not exist, an error will be returned.

The *options* argument specifies the type of encoding that will be applied to the attachment. If this argument is not specified, then text files will not be encoded and binary files will be encoded using the standard base64 algorithm.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

ContentType Property, ExtractFile Method

Cancel Method

Cancel the current operation.

Syntax

object.**Cancel**

The object placeholder represents an expression that evaluates to an InternetMail object.

Return Type

Long Integer

Remarks

The **Cancel** method cancels the current operation, returning control to the application. This method is typically used to either cancel the retrieval of a message from the mail server, or the delivery of a message. Once the operation has been canceled, the **OnCancel** event will fire.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Reset Method, OnCancel Event

ChangePassword Method

Change the mailbox password for the specified user.

Syntax

object.**ChangePassword**(*username, oldpass, newpass*)

The **ChangePassword** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
username	A string which specifies the username.
oldpass	A string which specifies the old password.
newpass	A string which specifies the new password.

Return Type

Long Integer

Remarks

The **ChangePassword** method changes the password that will be used to authenticate the specified user.

The *username* argument specifies the username for the mailbox. If this is the same value as the **UserName** property, then successfully changing the password will cause the **Password** property to be updated with the new password.

The *oldpass* argument specifies the current password for the user's mailbox. An error will be returned if this is an empty string.

The *newpass* argument specifies the new password for the user's mailbox. An error will be returned if this is an empty string, or if the old and new password are the same value.

Note that in order to change the user's mailbox password, the server must be running the poppass service on port 106, on the same server. Because passwords are transmitted as clear text (unencrypted), this service is not considered secure and may not be available.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Password Property, UserName Property

ComposeMessage Method

Compose a new mail message.

Syntax

object.**ComposeMessage**(*from*, *to* [, *cc*] [, *bcc*] [, *subject*] [, *msgtext*] [, *msghtml*] [, *charset*] [, *enctype*])

The **ComposeMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
from	A string which specifies the sender address.
to	A string which specifies one or more recipient addresses.
cc	A string which specifies one or more recipient addresses.
bcc	A string which specifies one or more recipient addresses.
subject	A string which specifies the subject of the message.
msgtext	A string which specifies the text of the message.
msghtml	A string which specifies optional HTML formatted text.
charset	An integer value which specifies the message character set.
enctype	An integer value which specifies the message encoding type.

Return Type

Long Integer

Settings

The settings for *charset* are:

Constant	Description
mailCharsetUSASCII	The default character set using US-ASCII which defines 7-bit printable characters with values ranging from 20h to 7Eh.
mailCharsetISO8859_1	An 8-bit character set for most western European languages such as English, French, Spanish and German. This character set is also commonly referred to as Latin1.
mailCharsetISO8859_2	An 8-bit character set for most central and eastern European languages such as Czech, Hungarian, Polish and Romanian. This character set is also commonly referred to as Latin2.
mailCharsetISO8859_5	An 8-bit character set for Cyrillic languages such as Russian, Bulgarian and Serbian.
mailCharsetISO8859_6	An 8-bit character set for Arabic languages. Note that the application is responsible for displaying text that uses this character set. In particular, any display engine needs to be able to handle the reverse writing direction and analyze the context of the message to correctly combine the glyphs.
mailCharsetISO8859_7	An 8-bit character set for the Greek langauge.
mailCharsetISO8859_8	An 8-bit character set for the Hebrew language. Note that similar to Arabic, Hebrew uses a reverse writing direction. An application which displays this character should be capable of processing bi-directional text where a single message may include both right-to-left and left-to-right languages, such as Hebrew and English.
mailCharsetISO8859_9	An 8-bit character set for the Turkish language. This character set is also commonly referred to as Latin5.

The settings for *enctype* are:

Constant	**Description**
mailEncoding7Bit	Each character is encoded in one or more bytes, with each byte being 8 bits long, with the first bit cleared. This encoding is most commonly used with plain text using the US-ASCII character set, where each character is represented by a single byte in the range of 20h to 7Eh. Most e-mail messages are composed using 7-bit ASCII.
mailEncoding8Bit	Each character is encoded in one or more bytes, with each byte being 8 bits long and all bits are used. 8-bit encoding may be used with multi-byte character sets, although this encoding type is uncommon in e-mail messages. It is recommended that quoted-printable encoding be used for 8-bit character sets.
mailEncodingBinary	Binary encoding is essentially the absence of any encoding performed on the message data, and there is no presumption that the data contains textual information. No character set localization or conversion is performed on binary encoded data. This encoding type is not recommended. Instead, binary data should be encoded using the standard base64 algorithm.
mailEncodingQuoted	Quoted-printable encoding is designed for textual messages where most of the characters are represented by the ASCII character set and is generally human-readable. Non-printable characters or 8-bit characters with the high bit set are encoded as hexadecimal values and represented as 7-bit text. Quoted-printable encoding is typically used for messages which use character sets such as ISO-8859-1, as well as those which use HTML.
mailEncodingBase64	Base64 encoding is designed to represent binary data in a form that is not human readable but which can be safely exchanged with servers that only accept 7-bit data. Base64 encoding is typically used with file attachments.
mailEncodingUucode	Uuencoding and uudecoding is a legacy encoding format that was used before the MIME standard was established. This encoding method has largely been replaced by base64 encoding, although it is still commonly used for binary newsgroup postings on USENET. Although this encoding format is supported, it is not officially part of the MIME standard and its use in e-mail messages is discouraged.

Remarks

The **ComposeMessage** method creates a new mail message, or replaces the current message if one already exists.

The *from* argument is required and specifies the sender's e-mail address. Only a single address should be specified. After the message has been composed, the **From** property will be updated with this value.

The *to* argument is required and specifies one or more recipient e-mail addresses. Multiple e-mail addresses may be specified by separating them with commas. After the message has been composed, the **To** property will be updated with this value.

The *cc* argument is optional and specifies one or more additional recipient addresses that will receive a copy of the message. If this argument is not specified, then no Cc header field will be created for this message. After the message has been composed, the **Cc** property will be updated with this value.

The *bcc* argument is optional and specifies one or more additional recipient addresses that will receive a copy of the message. Unlike the *cc* argument, these recipients will not be included in the header of the message. If this argument is not specified, then no blind carbon copies of the message will be sent. After the message has been composed, the **Bcc** property will be updated with this value.

The *subject* argument is optional and specifies the subject for the message. If the argument is not specified, then no Subject header field will be created for this message. After the message has been composed, the **Subject** property will be updated with this value.

The *msgtext* argument is optional and specifies the body of the message. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line. If the argument is not specified, then the message will have an empty body unless the *msghtml* argument has been specified.

The *msghtml* argument is optional and specifies an alternate HTML formatted message. If the *msgtext* argument has been specified, then a multipart message will be created with both plain text and HTML text as the alternative. This allows mail clients to select which message body they wish to display. If the *msgtext* argument is not specified or is an empty string, then the message will only contain HTML. Although this is supported, it is not recommended because older mail clients may be unable to display the message correctly.

The *charset* and *enctype* arguments are optional and specify the character set and encoding type for the message text. The default is for the message to use the standard US-ASCII character set and 7-bit encoding. Note that if an 8-bit character set is selected, the default encoding type will be set to quoted-printable.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

Example

The following example composes a message and sends it to the specified recipients:

```
Dim nError As Long

nError = InternetMail1.ComposeMessage(comboFrom.Text, _
                                      editTo.Text, _
                                      editCc.Text, _
                                      editBcc.Text, _
                                      editSubject.Text, _
                                      editMessage.Text)
If nError <> 0 Then
    MessageBox "Unable to compose message", vbExclamation
    Exit Sub
End If

nError = InternetMail1.SendMessage()

If nError <> 0 Then
    MsgBox "Unable to send message", vbExclamation
    Exit Sub
End If
```

See Also

Bcc Property, Cc Property, Encoding Property, From Property, MessageText Property, Subject Property, To Property

Connect Method

Establish a connection with the specified mail server.

Syntax

object.**Connect**([*server*] [, *port*] [, *username*] [, *password*] [, *timeout*] [, *options*])

The **Connect** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
server	A string which specifies the host name or IP address of the mail server.
port	A long integer which specifies the mail server port number.
username	A string which specifies the username.
password	A string which specifies the password.
timeout	A long integer which specifies the timeout period in seconds.
options	A long integer which specifies one or more options.

Return Type

Long Integer

Settings

The settings for *options* are:

Constant	Description
mailOptionNoStartTLS	For secure POP3 and SMTP connections only; prevents the use of the STARTTLS command which is used to negotiate a secure session. This option should only be used if required by the server.
mailOptionAPOP	Causes the APOP authentication method to be used when connecting to a POP3 mail server. The default is to use standard password authentication.

Remarks

The **Connect** method is used to establish a connection with the specified mail server. This is the first method that must be called prior to the application retrieving mail messages using the **GetMessage** method.

If the **Connect** method is called when a connection already exists, the current connection will be closed. This has the side-effect of causing any messages which have been marked for deletion to be removed by the mail server.

Note that it is not required to call the **Connect** method to send messages since this is handled internally by the component. For more information about sending messages, see the **SendMessage** method and the **RelayServer** and **RelayPort** properties.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

Example

The following example connects to a mail server and retrieves each of the mail messages, storing them in a file on the local system:

```
    Dim strFileName As String
    Dim nMessage As Long, nError As Long

    nError = InternetMail1.Connect(strServerName, , strUserName, strPassword)

    If nError <> 0 Then
        MsgBox "Unable to connect to " & strServerName & vbCrLf & _
               InternetMail1.LastErrorString, vbExclamation
        Exit Sub
    End If

    If InternetMail1.LastMessage = 0 Then
        MsgBox "The mailbox is currently empty", vbInformation
        InternetMail1.Disconnect
        Exit Sub
    End If

    For nMessage = 1 To InternetMail1.LastMessage
        strFileName = "c:\temp\msg" & Format(nMessage, "00000") & ".txt"
        nError = InternetMail1.StoreMessage(nMessage, strFileName)
        If nError <> 0 Then
            MsgBox "Unable to store message " & nMessage & vbCrLf & _
                   InternetMail1.LastErrorString, vbExclamation
            Exit For
        End If
    Next

    If nError = 0 Then
        MsgBox "Stored " & InternetMail1.LastMessage & " messages",
vbInformation
    End If

    InternetMail1.Disconnect
```

See Also

Disconnect Method, GetMessage Method, SendMessage Method, RelayPort Property, RelayServer Property, Secure Property

ClearMessage Method

Clear the current message.

Syntax

object.**ClearMessage**

The object placeholder represents an expression that evaluates to an InternetMail object.

Return Type

Long Integer

Remarks

The **ClearMessage** method clears the current message, releasing the memory allocated for the message and any attachments. This will also reset the value of the **Bcc** property back to an empty string.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Bcc Property

CreatePart Method

Create a new message part in a multipart message.

Syntax

object.**CreatePart**([*msgtext*] [, *charset*] [, *enctype*])

The **CreatePart** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
msgtext	A string which specifies the message part text.
charset	An integer value which specifies the message part character set.
enctype	An integer value which specifies the message part encoding type.

Return Type

Long Integer

Settings

The settings for *charset* are:

Constant	Description
mailCharsetUSASCII	The default character set using US-ASCII which defines 7-bit printable characters with values ranging from 20h to 7Eh.
mailCharsetISO8859_1	An 8-bit character set for most western European languages such as English, French, Spanish and German. This character set is also commonly referred to as Latin1.
mailCharsetISO8859_2	An 8-bit character set for most central and eastern European languages such as Czech, Hungarian, Polish and Romanian. This character set is also commonly referred to as Latin2.
mailCharsetISO8859_5	An 8-bit character set for Cyrillic languages such as Russian, Bulgarian and Serbian.
mailCharsetISO8859_6	An 8-bit character set for Arabic languages. Note that the application is responsible for displaying text that uses this character set. In particular, any display engine needs to be able to handle the reverse writing direction and analyze the context of the message to correctly combine the glyphs.
mailCharsetISO8859_7	An 8-bit character set for the Greek language.
mailCharsetISO8859_8	An 8-bit character set for the Hebrew language. Note that similar to Arabic, Hebrew uses a reverse writing direction. An application which displays this character should be capable of processing bi-directional text where a single message may include both right-to-left and left-to-right languages, such as Hebrew and English.
mailCharsetISO8859_9	An 8-bit character set for the Turkish langauge. This character set is also commonly referred to as Latin5.

The settings for *enctype* are:

Constant	Description
mailEncoding7Bit	Each character is encoded in one or more bytes, with each byte being 8 bits long, with the first bit cleared. This encoding is most commonly used with plain text using the US-ASCII character set, where each character is represented by a single byte in the range of 20h to 7Eh. Most e-mail messages are composed using 7-bit ASCII.
mailEncoding8Bit	Each character is encoded in one or more bytes, with each byte being 8 bits long and all bits are used. 8-bit encoding may be used with multi-byte character sets, although this encoding type is uncommon in e-mail messages. It is recommended that quoted-printable encoding be used for 8-bit character sets.
mailEncodingBinary	Binary encoding is essentially the absence of any encoding performed on the message data, and there is no presumption that the data contains textual information. No character set localization or conversion is performed on binary encoded data. This encoding type is not recommended. Instead, binary data should be encoded using the standard base64 algorithm.
mailEncodingQuoted	Quoted-printable encoding is designed for textual messages where most of the characters are represented by the ASCII character set and is generally human-readable. Non-printable characters or 8-bit characters with the high bit set are encoded as hexadecimal values and represented as 7-bit text. Quoted-printable encoding is typically used for messages which use character sets such as ISO-8859-1, as well as those which use HTML.
mailEncodingBase64	Base64 encoding is designed to represent binary data in a form that is not human readable but which can be safely exchanged with servers that only accept 7-bit data. Base64 encoding is typically used with file attachments.
mailEncodingUucode	Uuencoding and uudecoding is a legacy encoding format that was used before the MIME standard was established. This encoding method has largely been replaced by base64 encoding, although it is still commonly used for binary newsgroup postings on USENET. Although this encoding format is supported, it is not officially part of the MIME standard and its use in e-mail messages is discouraged.

Remarks

The **CreatePart** method creates a new message part. If the current message is a simple RFC822 message, then this method converts it to a MIME multipart message. The current message part will be set to the new part that was just created.

The *msgtext* argument is optional and specifies the body of the new message part. Each line of text contained in the string should be terminated with a carriage-return/linefeed (CRLF) pair, which is recognized as the end-of-line. If the argument is not specified, then the message part will have an empty body.

The *charset* and *enctype* arguments are optional and specify the character set and encoding type for the message text. The default is for the message to use the standard US-ASCII character set and 7-bit encoding.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

AttachFile Method, DeletePart Method

DeleteHeader Method

Delete a header field from the current message part.

Syntax

object.**DeleteHeader**(*header*)

The **DeleteHeader** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
header	A string which specifies the message header to delete.

Return Type

Boolean

Remarks

The **DeleteHeader** method deletes the specified header field value from the current message part.

DeleteMessage Method

Delete the specified message from the mail server.

Syntax

object.**DeleteMessage**([*number*])

The **DeleteMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
number	A long integer which specifies the message to delete.

Return Type

Long Integer

Remarks

The **DeleteMessage** method marks the specified message for deletion. If the optional message number is not specified, then the current message is deleted. Once a message has been marked as deleted, any attempt to access it will result in an error.

The message will not actually be removed from the server until the **Disconnect** method is called or the InternetMail object is unloaded. To prevent messages which have been marked for deletion from actually being removed from the mailbox, call the **Reset** method.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

DeleteMessage Method, Disconnect Method, GetHeader Method, GetMessage Method, Reset Method

DeletePart Method

Delete the specified message part in the current message.

Syntax

object.**DeletePart**([*part*])

The **DeletePart** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
part	A long integer which specifies the message part to delete.

Return Type

Long Integer

Remarks

The **DeletePart** method deletes the specified message part in the current message. If the optional message part is not specified, then the current message part is deleted.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

AttachFile Method, CreatePart Method

Disconnect Method

Disconnect from the mail server.

Syntax

object.**Disconnect**

The object placeholder represents an expression that evaluates to an InternetMail object.

Return Type

Long Integer

Remarks

The **Disconnect** method causes all messages that have marked for deletion to be removed by the server and the network connection is closed.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Connect Method

ExportMessage Method

Export the current message to a text file.

Syntax

object.**ExportMessage**(*filename* [, *options*])

The **ExportMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
filename	A string which specifies the name of the file to contain the message.
options	A long integer which specifies one or more options.

Return Type

Long Integer

Settings

The settings for *options* are:

Constant	Description
mailOptionAllHeaders	Preserves all headers in the message when it is exported, including the Received and Return-Path headers which are normally excluded.
mailOptionKeepOrder	Preserves the order of the headers when it is exported; by default, some headers may be re-ordered.

Remarks

The **ExportMessage** copies the current message to the specified file. If the file does not exist it will be created, otherwise it will be overwritten with the contents of the message.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

ExtractFile Method, ImportMessage Method

ExtractFile Method

Extract an attached file from the current message.

Syntax

object.**ExtractFile**(*filename* [, *part*])

The **ExtractFile** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
filename	A string which specifies the attached file name.
part	A long integer which specifies the message part.

Return Type

Long Integer

Remarks

The **ExtractFile** method extracts the attached file stored in the specified message part, storing it in a file. If the optional message *part* argument is not specified, the current message part is used. To determine if the current message part contains an attachment and to determine its file name, check the value of the **Attachment** property. An error will be returned if the specified message part does not contain a file attachment.

This method will return value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Attachment Property, ExportMessage Method

GetFirstHeader Method

Return the first header in the current message part.

Syntax

object.**GetFirstHeader**(*header*, *value*)

The **GetFirstHeader** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
header	A string which will contain the name of the header field.
value	A string which will contain the value of the header field.

Return Type

Boolean

Remarks

The **GetFirstHeader** method allows an application to enumerate all of the headers in the current message. If the current message part does not contain any header fields, this method will return False.

Example

The following example enumerates all of the headers in the main part of the current message and adds them to a listbox:

```
Dim strHeader As String, strValue As String
Dim bResult As Boolean

bResult = InternetMail1.GetFirstHeader(strHeader, strValue)
Do While bResult
    List1.AddItem strHeader & ": " & strValue
    bResult = InternetMail1.GetNextHeader(strHeader, strValue)
Loop
```

See Also

MessagePart Property, GetHeader Method, GetNextHeader Method, SetHeader Method

GetHeader Method

Return the value for the specified header in the current message part.

Syntax

object.**GetHeader**(*header*, *value*)

The **GetHeader** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
header	A string which specifies the name of the header field.
value	A string which will contain the value of the header field.

Return Type

Boolean

Remarks

The **GetHeader** method is used to retrieve the value for a specific header in the current message part. If the header field exists, the method will return True and the *value* argument will contain the header value. If the header does not exist, the method will return False.

If there are multiple headers with the same name, the first value will be returned. To enumerate all of the headers in a message, including duplicate header fields, use the **GetFirstHeader** and **GetNextHeader** methods.

See Also

MessagePart Property, GetFirstHeader Method, GetNextHeader Method, SetHeader Method

GetNextHeader Method

Return the next header in the current message part.

Syntax

object.**GetNextHeader**(*header*, *value*)

The **GetNextHeader** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
header	A string which will contain the name of the header field.
value	A string which will contain the value of the header field.

Return Type

Boolean

Remarks

The **GetNextHeader** method allows an application to enumerate all of the headers in the current message. When all of the headers in the current message part have been returned, this method will return False.

Example

The following example enumerates all of the headers in the main part of the current message and adds them to a listbox:

```
Dim strHeader As String, strValue As String
Dim bResult As Boolean

bResult = InternetMail1.GetFirstHeader(strHeader, strValue)
Do While bResult
    List1.AddItem strHeader & ": " & strValue
    bResult = InternetMail1.GetNextHeader(strHeader, strValue)
Loop
```

See Also

MessagePart Property, GetFirstHeader Method, GetHeader Method, SetHeader Method

GetMessage Method

Retrieve the specified message from the mail server.

Syntax

object.**GetMessage**([*number*])

The **GetMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
number	A long integer which specifies the message to retrieve.

Remarks

The **GetMessage** method retrieves the specified message from the mail server. This method will cause the current message to be replaced with the new message, and the **MessageIndex** property will be updated with the new message number. If the optional message *number* argument is not specified, then the value of the **MessageIndex** property is used instead.

Note that unlike setting the **MessageIndex** property, which only causes the headers for the specified message to be retrieved, the **GetMessage** method downloads the complete message. The **OnProgress** event will fire periodically as the message is retrieved, allowing an application to update its user interface if desired.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

Example

The following example connects to a mail server and retrieves the first message:

```
Dim nError As Long

nError = InternetMail1.Connect(strServerName, , strUserName, strPassword)

If nError <> 0 Then
    MsgBox "Unable to connect to " & strServerName & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
    Exit Sub
End If
```

```
If InternetMail1.LastMessage = 0 Then
    MsgBox "The mailbox is currently empty", vbInformation
    InternetMail1.Disconnect
    Exit Sub
End If

  nError = InternetMail1.GetMessage(1)

If nError <> 0 Then
    MsgBox "Unable to retrieve the message" & vbCrLf & _
           InternetMail1.LastErrorString, vbExclamation
Next

InternetMail1.Disconnect
```

See Also

GetHeader Method, MessageIndex Method, OnProgress Event

ImportMessage Method

Import a new message from the specified text file.

Syntax

object.**ImportMessage**(*filename*)

The **ImportMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
filename	A string which specifies the name of the file to import from.

Return Type

Long Integer

Remarks

The **ImportMessage** method replaces the current message with the message contained in the specified text file. Note that calling this method will result in the **Bcc** property value being cleared.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Bcc Property, AttachFile Method, ExportMessage Method, ExtractFile Method

Initialize Method

Initialize the component and load the networking library.

Syntax

object.**Initialize**([*module*] [, *license*] [, *options*] [, *reserved*])

The **Initialize** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
module	A string which specifies the name of the network module.
license	A string which specifies the runtime license key.
options	A reserved argument; this argument is currently unused.
reserved	A reserved argument; this argument is currently unused.

Return Type

Long Integer

Remarks

The **Initialize** method explicitly loads and initializes the Windows Sockets networking library. Typically this is not required because it's done automatically when any control function is taken which requires network access, such as attempting a connection to a remote host.

The *module* argument is the full pathname of the Windows Sockets library loaded by the control. Setting the argument to the name of an appropriate library causes it to be loaded. If this argument is not specified, or an empty string is passed as the value, the default system Windows Sockets library is loaded.

The *license* argument specifies a runtime license key used to initialize the control. Normally this argument is not needed, since the appropriate license key is used when an instance of the control is created. However, if an instance of the control is created using the *CreateObject* function, the **Initialize** method must be called with a valid runtime license key. If the license key is omitted or passed as an empty string, a development license must be installed on the local system.

The *options* and *reserved* arguments are unused and should not be specified.

A value of zero is returned if the library was initialized successfully. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

Example

```
'
' Create an instance of the InternetMail object
'
Set objMail = WScript.CreateObject("Catalyst.InternetMail")
'
' Initialize the object, using the default Windows Sockets
' library and the specified runtime license key; if the key
' is not specified, the development license will be used
'
nError = objMail.Initialize("", CSTOOLS4_LICENSE_KEY)
If nError <> 0 Then
    WScript.Echo "Unable to initialize the InternetMail object"
    WScript.Quit(1)
End If
```

See Also

Uninitialize Method

ParseAddress Method

Parse an Internet e-mail address.

Syntax

object.**ParseAddress**(*address*)

The **ParseAddress** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
address	A string which specifies the address to be parsed.

Return Type

String

Remarks

The **ParseAddress** method parses a string which contains an e-mail address and returns only the address portion, excluding any comments. An address may contain comments enclosed in parenthesis, or may specify a name along with the address in which case the address is enclosed in angle brackets. For example, consider the following header field value:

```
"User Name" <user@domain.com> (This is a comment)
```

The **ParseAddress** method would return "user@domain.com" if passed the above string, removing the name and any comments.

Note that the **ParseAddress** method will only parse a single address. If multiple addresses are specified, they must be comma delimited and split prior to calling this method.

Example

The following example parses all of the recipient e-mail addresses in the current message, storing them in the *strAddresses* string array.

```
Dim strAddresses() As String, strAddress As String
Dim nIndex As Integer, nAddresses As Integer

nAddresses = 0
strAddresses = Split(InternetMail1.To & "," & _
                     InternetMail1.Cc & "," & _
                     InternetMail1.Bcc, ",")

For nIndex = 0 To UBound(strAddresses)
    If Len(Trim(strAddresses(nIndex))) > 0 Then
        '
        ' The ParseAddress method will return an e-mail address
        ' or an empty string if the argument could not be parsed
        '
        strAddress = InternetMail1.ParseAddress(strAddresses(nIndex))
        If Len(strAddress) > 0 Then
                    strAddresses(nAddresses) = strAddress
                    nAddresses = nAddresses + 1
                End If
    End If
Next
```

ParseMessage Method

Parse the specified string, adding the contents to the current message.

Syntax

object.**ParseMessage**(*msgtext*)

The **ParseMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
msgtext	A string which specifies the message text to be parsed.

Return Type

Long Integer

Remarks

The **ParseMessage** method parses a string which contains message data, adding it to the current message. This method is useful when the application needs to parse an arbitrary block of text and add it to the current message. If the string contains header fields, the values will be added to the message header. Once the end of the header block is detected, all subsequent text is added to the body of the message.

Note that unlike the **ImportMessage** method, the **ParseMessage** method does not clear the contents of the current message and may be called multiple times. Use the **ClearMessage** method to clear the current message before calling **ParseMessage** if necessary.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

Example

The following example demonstrates the use of **ParseMessage** to parse multiple blocks of data from a file. This example effectively does the same thing as calling the **ImportMessage** method:

```
InternetMail1.ClearMessage

hFile = FreeFile()
Open strFileName For Input As hFile
nFileLength = LOF(hFile)

Do While nFileLength > 0
    '
    ' Read the contents of the file in 1K blocks; note that
    ' this is intentionally inefficient to demonstrate
    ' multiple calls to the ParseMessage method.
    '
    cbBuffer = nFileLength: If cbBuffer > 1024 Then cbBuffer = 1024
    nFileLength = nFileLength - cbBuffer
    strBuffer = Input(cbBuffer, hFile)
    '
    ' Parse the string, adding to the current message
    '
    nError = InternetMail1.ParseMessage(strBuffer)
    If nError <> 0 Then
        MsgBox InternetMail1.LastErrorString, vbExclamation
        Exit Do
    End If
Loop

Close hFile
```

See Also

ClearMessage Method, ImportMessage Method

Reset Method

Reset the state of the component.

Syntax

object.**Reset**

The object placeholder represents an expression that evaluates to an InternetMail object.

Return Type

None

Remarks

The **Reset** method resets the internal state of the component, releasing any memory allocated for messages and/or network connections. If the application is connected to a mail server, the connection will be terminated. If any messages were marked for pending deletion, those messages will not be deleted.

See Also

Cancel Method

SendMessage Method

Send an e-mail message to one or more recipients.

Syntax

object.**SendMessage**([*sender*] [, *recipient*] [, *message*] [, *options*])

The **SendMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
sender	A string which specifies the e-mail address of the sender.
recipient	A string which specifies one or more recipient e-mail addresses.
message	A string which specifies a complete message, including headers.
options	A long integer which specifies one or more options.

Return Type

Long Integer

Settings

The settings for *options* are:

Constant	Description
mailOptionNoStartTLS	For secure SMTP connections only; prevents the use of the STARTTLS command which is used to negotiate a secure session. This option should only be used if required by the server.
mailOptionNotify	Notify the sender of the delivery status of the message, if the server supports delivery status notification. This option is a combination of the *mailNotifySuccess*, *mailNotifyFailure*, *mailNotifyDelay* and *mailReturnHeaders* options.
mailNotifySuccess	If the mail server supports delivery status notification, this causes a message to be returned to the sender once it has been successfully delivered.
mailNotifyFailure	If the mail server supports delivery status notification, this causes a message to be returned to the sender if it could not be delivered.
mailNotifyDelay	If the mail server supports delivery status notification, this causes a message to be returned to the sender if delivery has been delayed.
mailReturnHeaders	If the mail server supports delivery status notification, this causes a message to be returned which contains the headers of the message that was sent.
mailReturnMessage	If the mail server supports delivery status notification, this causes a message to be returned which contains the complete message that was sent.

Remarks

The **SendMessage** method sends the specified message to one or more recipients. This method can be used in a number of different ways, depending on the arguments specified by the caller.

The optional *sender* argument identifies the sender of the message and must be a standard Internet e-mail address. If this argument is ommitted, then the address specified by the **From** property will be used.

The optional *recipient* argument specifies one or more recipients of the message. If this argument is ommitted, then the addresses listed in the **Bcc**, **Cc** and **To** properties will be used to determine the recipients of the message.

The optional *message* argument specifies a complete e-mail message that will be delivered. This must be a properly formatted message that conforms to the standards for Internet e-mail. If this argument is ommitted, then the current message is sent. If this argument is specified, but the *sender* and *recipient* properties are ommitted, then the message will be parsed and the addresses will be automatically determined by the values of the From, Cc and To header fields. Note that specifying a *message* argument does not change the current message.

The *options* argument specifies one or more options for sending the message. If this argument is ommitted, the value of the **Options** property will be used instead.

For each recipient listed, either as an argument to the method or in the message itself, the **SendMessage** method will determine the appropriate mail exchange server and deliver the message to that user. If the **RelayServer** and **RelayPort** properties are defined, then all messages will be relayed through that specific server, regardless of the recipient address. Note that the **Secure** property and related options only affects connections to relay mail servers. See the **RelayServer** and **RelayPort** properties for additional information.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

See Also

Bcc Property, Cc Property, From Property, Options Property, RelayPort Property, RelayServer Property, Secure Property, To Property

SetHeader Method

Set the value of a header field in the current message part.

Syntax

object.**SetHeader**(*header*, *value*)

The **SetHeader** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
header	A string which specifies the name of the header field.
value	A string which specifies the value of the header field.

Return Type

Boolean

Remarks

The **SetHeader** method creates or changes the value of the specified header field in the current message part. If the header does not exist, it will be created with the new value. If the header does exist, its current value will be replaced by the new value.

See Also

MessagePart Property, GetFirstHeader Method, GetHeader Method, GetNextHeader Method

StoreMessage Method

Store the specified message in a file.

Syntax

object.**StoreMessage**(*number*, *filename*)

The **StoreMessage** method syntax has the following parts:

Part	Description
object	An object expression that evaluates to an InternetMail object.
number	A long integer which specifies the message to store.
filename	A string which specifies the name of the file to store the message in.

Return Type

Long Integer

Remarks

The **StoreMessage** method retrieves the specified message from the mail server and stores it in a file. The *number* argument specifies the message to retrieve and the *filename* argument specifies the name of the file that the message will be stored in.

For applications which need to store messages on the local system, the **StoreMessage** method is somewhat more efficient than using the **GetMessage** and **ExportMessage** methods to load and store the message. **StoreMessage** does not attempt to analyze the message or change the current message contents.

This method will return a value of zero if the action was successful. Otherwise, a non-zero error code is returned which indicates the cause of the failure.

Example

The following example connects to a mail server and retrieves each of the mail messages, storing them in a file on the local system:

```
    Dim strFileName As String
    Dim nMessage As Long, nError As Long

    nError = InternetMail1.Connect(strServerName, , strUserName, strPassword)

    If nError <> 0 Then
        MsgBox "Unable to connect to " & strServerName & vbCrLf & _
               InternetMail1.LastErrorString, vbExclamation
        Exit Sub
    End If

    If InternetMail1.LastMessage = 0 Then
        MsgBox "The mailbox is currently empty", vbInformation
        InternetMail1.Disconnect
        Exit Sub
    End If

    For nMessage = 1 To InternetMail1.LastMessage
        strFileName = "c:\temp\msg" & Format(nMessage, "00000") & ".txt"
        nError = InternetMail1.StoreMessage(nMessage, strFileName)
        If nError <> 0 Then
            MsgBox "Unable to store message " & nMessage & vbCrLf & _
                   InternetMail1.LastErrorString, vbExclamation
            Exit For
        End If
    Next

    If nError = 0 Then
        MsgBox "Stored " & InternetMail1.LastMessage & " messages",
vbInformation
    End If

    InternetMail1.Disconnect
```

See Also

GetMessage Method, ExportMessage Method

Uninitialize Method

Uninitialize the component and unload the networking library.

Syntax

object.**Uninitialize**

The object placeholder represents an expression that evaluates to an InternetMail object.

Return Type

None

Remarks

The **Uninitialize** method terminates any connection established by the component and unloads the networking library. This method is not typically used since this is done automatically when the component is unloaded.

See Also

Initialize Method

—Events—

OnCancel Event

The **OnCancel** event is generated when an operation is canceled.

Syntax

Private Sub *object*_**OnCancel** ([*Index* **As Integer**])

Remarks

The **OnCancel** event is generated after an operation is canceled by calling the **Cancel** method.

See Also

OnError Event, Cancel Method

OnDelivered Event

The **OnDelivered** event is generated after a message has been delivered.

Syntax

Private Sub *object*_**OnDelivered**([*Index* **As Integer**,] *ByVal Address* **As Variant**, *ByVal MessageSize* **As Variant**)

Remarks

The **OnDelivered** event is generated after a message has been successfully submitted to the mail server for delivery. When used in conjunction with the **OnRecipient** and **OnProgress** events, this event can be used to track the delivery of a message to multiple recipients. If the message was not delivered, either because delivery was canceled in the **OnRecipient** event or because of an error, the **OnDelivered** event will not fire.

The *Address* argument is a string which specifies the recipient email address.

The *MessageSize* argument is a long integer which specifies the size of the message that was delivered.

Note that even though a message has been successfully delivered to the mail server, it may not actually be delivered to the recipient. The server may accept the message and then subsequently decide to reject or re-route the message based on its own internal configuration. To confirm message delivery to the actual user, use the delivery status notification options and/or set the **ReturnReceipt** property to an address which will be notified when the message has been read.

See Also

OnProgress Event, OnRecipient Event, ReturnReceipt Property

OnError Event

The **OnError** event is generated when an error occurs.

Syntax

Private Sub *object*_**OnError** ([*Index* **As Integer**,] *ByVal Error* **As Variant**, *ByVal Description* **As Variant**)

Remarks

The **OnError** event is generated when an error occurs while the component is performing an operation. Visual Basic errors do not generate this event.

The *Error* argument specifies the last control error that has occurred. This corresponds to the **LastError** property.

The *Description* argument is a string that describes the error. This corresponds to the **LastErrorString** property.

See Also

LastError Property, LastErrorString Property

OnProgress Event

The **OnProgress** event is generated when retrieving or sending messages.

Syntax

Private Sub *object*_**OnProgress** ([*Index* **As Integer**,] *ByVal MessageSize* **As Variant**, *ByVal MessageCopied* **As Variant**, *ByVal Percent* **As Variant**)

Remarks

The **OnProgress** event is generated when a message is being retrieved or sent. This event can be used to update the user interface, such as displaying a progress bar during the transaction. To cancel the current operation, the application can call the **Cancel** method from within this event.

The *MessageSize* argument is a long integer which specifies the size of the message in bytes that is currently being sent or received.

The *MessageCopied* argument is a long integer which specifies the number of bytes that have been sent or received for the current message.

The *Percent* argument is an integer which specifies the completion percentage between a value of 0 and 100.

See Also

Cancel Method

OnRecipient Event

The **OnRecipient** event is generated before a message is sent.

Syntax

Private Sub *object*_**OnRecipient** ([*Index* **As Integer**,] *ByVal Address* **As Variant**, *ByRef Cancel* **As Variant**)

Remarks

The **OnRecipient** event is generated immediately before a message is sent to a recipient. When used in conjunction with the **OnProgress** event, this event can be used to track the delivery of a message to multiple recipients. If an error occurs during the delivery of the message, the **OnError** event will fire.

The *Address* argument is a string which specifies the recipient email address.

The *Cancel* argument determines whether or not the message delivery is canceled for the specified recipient. Setting this argument to a value of True causes the **SendMessage** method to not deliver the message and continue on to the next recipient. The default value for this argument is False, which indicates that the message should be delivered.

Note that setting the *Cancel* argument to True is different from using the **Cancel** method, which would cancel delivery of the message to all subsequent recipients as well as the current recipient specified by the *Address* argument.

See Also

Cancel Method, OnProgress Event, OnError Event, SendMessage Method

OnTimeout Event

The **OnTimeout** event is generated when an operation is canceled.

Syntax

Private Sub *object*_**OnTimeout** ([*Index* **As Integer**])

Remarks

The **OnTimeout** event is generated after an operation times out. The amount of time that the component will wait for an operation to complete can be controlled by the **Timeout** property.

See Also

Timeout Property, OnCancel Event

—Error Codes—

Value	Constant	Description
10001	mailErrorNotHandleOwner	Handle not owned by the current thread or process
10002	mailErrorFileNotFound	The specified file or directory does not exist
10003	mailErrorFileNotCreated	The specified file could not be created
10004	mailErrorOperationCanceled	The blocking operation has been canceled
10005	mailErrorInvalidFileType	The specified file is a block or character device
10006	mailErrorInvalidDevice	The specified device or address does not exist
10007	mailErrorTooManyParameters	The maximum number of function parameters has been exceeded
10008	mailErrorInvalidFileName	The specified file name is too long or contains invalid characters
10009	mailErrorInvalidFileHandle	Invalid file handle passed to function
10010	mailErrorFileReadFailed	Unable to read data from the specified file
10011	mailErrorFileWriteFailed	Unable to write data to the specified file
10012	mailErrorOutOfMemory	Out of memory
10013	mailErrorAccessDenied	Access denied
10014	mailErrorInvalidParameter	Invalid parameter
10015	mailErrorClipboardUnavailable	The system clipboard is currently unavailable
10016	mailErrorClipboardEmpty	The system clipboard is empty or does not contain any text data
10017	mailErrorFileEmpty	The specified file does not contain any data
10018	mailErrorFileExists	The specified file already exists
10019	mailErrorEndOfFile	End of file
10020	mailErrorDeviceNotFound	The specified device could not be found
10021	mailErrorDirectoryNotFound	The specified directory could not be found
10022	mailErrorInvalidBuffer	Invalid memory address passed to function
10023	mailErrorBufferTooSmall	The specified buffer is too small
10024	mailErrorNoHandles	No more handles available to this process
10035	mailErrorOperationWouldBlock	The specified operation would block
10036	mailErrorOperationInProgress	A blocking operation is currently in progress
10037	mailErrorAlreadyInProgress	The specified operation is already in progress
10038	mailErrorInvalidHandle	The specified handle is invalid
10039	mailErrorInvalidAddress	Invalid network address specified
10040	mailErrorInvalidSize	Invalid message size, message is too long
10041	mailErrorInvalidProtocol	Invalid network protocol specified
10042	mailErrorProtocolNotAvailable	The specified network protocol is not available
10043	mailErrorProtocolNotSupported	The specified protocol is not supported
10044	mailErrorSocketNotSupported	The specified socket type is not supported
10045	mailErrorInvalidOption	The specified option is invalid
10046	mailErrorProtocolFamily	The specified protocol family is not supported
10047	mailErrorProtocolAddress	The specified address is invalid for this protocol family
10048	mailErrorAddressInUse	The specified address is in use by another process
10049	mailErrorAddressUnavailable	The specified address cannot be assigned
10050	mailErrorNetworkUnavailable	The networking subsytem is unavailable
10051	mailErrorNetworkUnreachable	The specified network is unreachable
10052	mailErrorNetworkReset	Network dropped connection on remote reset
10053	mailErrorConnectionAborted	Network connection aborted by local host
10054	mailErrorConnectionReset	Network connection aborted by remote host
10055	mailErrorOutOfBuffers	No buffer space available
10056	mailErrorAlreadyConnected	Connection already established with remote

		host
10057	mailErrorNotConnected	No connection established with remote host
10058	mailErrorConnectionShutdown	Unable to send or receive data after connection shutdown
10060	mailErrorOperationTimeout	The specified operation has timed out
10061	mailErrorConnectionRefused	The connection has been refused by the remote host
10064	mailErrorHostUnavailable	The specified host is unavailable
10065	mailErrorHostUnreachable	The specified host is unreachable
10067	mailErrorTooManyProcesses	Too many processes are using the networking subsystem
10091	mailErrorNetworkNotReady	The networking subsystem is not available
10092	mailErrorInvalidVersion	The specified version is invalid or not supported
10093	mailErrorNetworkNotInitialized	The network subsystem has not been initialized
10101	mailErrorRemoteShutdown	The remote host has initiated a graceful shutdown sequence
11001	mailErrorInvalidHostname	The specified hostname is invalid or could not be resolved
11002	mailErrorHostnameNotFound	Non-authoritative hostname not found, retry operation
11003	mailErrorHostnameRefused	Unable to resolve hostname, request refused
11004	mailErrorHostnameNotResolved	Unable to resolve hostname, no address for specified host
12001	mailErrorInvalidLicense	The license for this product is invalid
12002	mailErrorProductNotLicensed	This product is not licensed to perform this operation
12003	mailErrorNotImplemented	This function has not been implemented on this platform
12004	mailErrorUnknownLocalhost	Unable to determine local host name
12005	mailErrorInvalidHostaddress	Invalid host address specified
12006	mailErrorInvalidServicePort	Invalid service port number specified
12007	mailErrorInvalidServiceName	Invalid or unknown service name specified
12008	mailErrorInvalidEventId	Invalid event identifier specified
12009	mailErrorOperationNotBlocking	No blocking operation in progress on this socket
12101	mailErrorSecurityNotInitialized	Unable to initialize security interface for this process
12102	mailErrorSecurityContext	Unable to establish security context for this session
12103	mailErrorSecurityCredentials	Unable to open specified certificate store or establish credentials
12104	mailErrorSecurityCertificate	Unable to validate specified certificate
12105	mailErrorSecurityDecryption	Unable to decrypt data stream
12106	mailErrorSecurityEncryption	Unable to encrypt data stream
12201	mailErrorOperationNotSupported	The specified operation is not supported
12202	mailErrorInvalidProtocolVersion	Invalid application protocol version specified
12203	mailErrorNoServerResponse	No data returned from server
12204	mailErrorInvalidServerResponse	Invalid data returned from server
12205	mailErrorUnexpectedServerResponse	Unexpected response code returned from server
12206	mailErrorServerTransactionFailed	Server transaction failed
12207	mailErrorServiceUnavailable	The service is currently unavailable
12208	mailErrorServiceNotReady	The service is not ready, try again later
12209	mailErrorServerResyncFailed	Unable to resynchronize with server
12210	mailErrorInvalidProxyType	Invalid proxy server type specified

12211	mailErrorProxyRequired	Resource must be accessed through specified proxy
12212	mailErrorInvalidProxyLogin	Unable to login to proxy server using specified credentials
12213	mailErrorProxyResyncFailed	Unable to resynchronize with proxy server
12214	mailErrorInvalidCommand	Invalid command specified
12215	mailErrorInvalidCommandParameter	Invalid command parameter specified
12216	mailErrorInvalidCommandSequence	Invalid command sequence specified
12217	mailErrorCommandNotImplemented	Specified command not implemented on this server
12218	mailErrorCommandNotAuthorized	Specified command not authorized for the current user
12219	mailErrorCommandAborted	Specified command was aborted by the remote host
12220	mailErrorOptionNotSupported	The specified option is not supported on this server
12221	mailErrorRequestNotCompleted	The current client request has not been completed
12222	mailErrorInvalidUsername	The specified username is invalid
12223	mailErrorInvalidPassword	The specified password is invalid
12224	mailErrorInvalidAccount	The specified account name is invalid
12225	mailErrorAccountRequired	Account name has not been specified
12226	mailErrorInvalidAuthenticationType	Invalid authentication protocol specified
12227	mailErrorAuthenticationRequired	User has already been authenticated
12228	mailErrorProxyAuthenticationRequired	Proxy authentication required
12229	mailErrorAlreadyAuthenticated	User has already been authenticated
12230	mailErrorAuthenticationFailed	User has already been authenticated
12251	mailErrorNetworkAdapter	Unable to determine network adapter configuration
12252	mailErrorInvalidRecordType	Invalid record type specified
12253	mailErrorInvalidRecordName	Invalid record name specified
12254	mailErrorInvalidRecordData	Invalid record data specified
12282	mailErrorInvalidMessagePart	Message is not multi-part or an invalid message part was specified
12283	mailErrorInvalidMessageHeader	The specified message header is invalid or has not been defined
12284	mailErrorInvalidMessageBoundary	The multipart message boundary has not been defined
12285	mailErrorNoFileAttachment	The current message part does not contain a file attachment
12286	mailErrorUnknownFileType	The specified file type could not be determined
12287	mailErrorDataNotEncoded	The specified data block could not be encoded
12288	mailErrorDataNotDecoded	The specified data block could not be decoded
12289	mailErrorFileNotEncoded	The specified file could not be encoded
12290	mailErrorFileNotDecoded	The specified file could not be decoded
12291	mailErrorNoMessageText	No message text
12292	mailErrorInvalidCharacterSet	Invalid character set specified
12293	mailErrorInvalidEncodingType	Invalid encoding type specified
12294	mailErrorInvalidMessageNumber	Invalid message number specified
12295	mailErrorNoReturnAddress	No valid return address specified
12296	mailErrorNoValidRecipients	No valid recipients specified
12297	mailErrorInvalidRecipient	The specified recipient address is invalid
12298	mailErrorNoMessageRelay	Server will not relay messages
12299	mailErrorMailboxUnavailable	Specified mailbox is currently unavailable
12300	mailErrorInvalidMailbox	Specified mailbox is invalid