# IDSMail™ OLE Server Help Contents 

# Whats *New* in IDSMail™ v4.0

**Address Books**

v4.0 includes support for Address Books, which was something our customers were asking for. We listened and included that in this version. Here are some references for getting you started quickly with these new capabilities.

AddressBook Object Overview

AddressBook Object properties and methods

**Faster Internet Mail Download**

The LoadMessageIdentifiers method is new in v4.0. It provides a faster alternative to LoadMessageHeaders in applications which do not need a great deal of header information prior to message download.

**Distinguishing between cc:Mail and Notes**

While IDSMail handles the differences between cc:Mail and Notes automatically, some customers wanted the ability to tell which system was present at runtime. The QueryMailSystem method has been extended to allow for this.

**Return Path Item Now Universal**

The IDSM_ITM_RETURN_PATH selector now returns a fully qualified return address for any E-mail system (GetMessageItems and GetMessageItemString methods). This information was previously only available for Internet mail.

# Introduction

The IDSMail™ OLE Server provides a universal client interface for sending and receiving network E-mail (*Professional Edition* required for receive). The server supports the following E-mail interfaces and protocols:

**SMTP/POP** - The Internet mail protocols. SMTP (Simple Mail Transport Protocol) is used for mail send. POP (Post Office Protocol) is used for mail receive. IDSMail™ supports v3.0 of POP. IDSMail is fully MIME (Multipart Internet Mail Extension) compliant on send and receive.

**VIM** - Vendor Independent Messaging used by products such as Lotus cc:Mail and Lotus Notes.

**MAPI** - Messaging Application Program Interface supported by Microsoft products.

**MHS** - Message Handling Service, used extensively in messaging products designed for Novell networks.

**VINES Intelligent Messaging** - Messaging protocol used by the Banyan VINES network operating system.

**Compuserve** - Via Compuserve MS Mail Driver

Enabling an application for E-mail is extremely easy. No knowledge of the of the underlying mail service is necessary. The IDSMail™ OLE Server automatically senses which E-mail system is installed on the workstation and processes the message appropriately.

*Quick Start* Programming Examples

Receiving E-mail

Properties & Methods

The AddressBook Object

## Properties

ApplicationName**
AttachmentEncoding
Certify
DateFormat**
DateSent**
DateSort**
DialogCaption
DialogErrorOnCancel
ErrorDisplay
Folder**
ForceDownload
From
HeadersInMessage**
HWnd
LoginDialog
LoginName
MailSystem
MaximumHeaders**
Message
MessageCount**
MessageIndex**
MessageType
NameResolution
NativeMailDialog
ObjectKey
Password
PeekOnly**
POPPort**
PostOffice
ProgressBar
ProgressBarThreshold
RecipientDelimiter
ServerPath
ServerVersion
SessionHandle**
SMTPPort
SMTPServer
SuppressAttachments**
SuppressAlternatives**
Subject
TimeoutConnect
TimeoutProtocol
UnreadCount**
UnreadOnly**
UTCOffset
XMailer

## Methods

AddAlternative
AddAttachment
AddRecipientBcc
AddRecipientCc
AddRecipientTo
AddXHeader
AsynchAbort
AsynchObjectRegister
AsynchObjectUnregister
ClearMessageItems
DeleteMessage**
GetHeaderItems**
GetHeaderItemString**
GetHeaderValueByName**
GetMessageItemCount
GetMessageItems
GetMessageItemString
GetWebFormValue**
LoadMessage**
LoadMessageHeaders**
LoadMessageIdentifiers**
Login
Logout
MoveMessage**
NewMessage
QueryMailSystem
ResolveName
ResolveNameEx
Send
SetNativeSystem
ShowMailDialog
ShowMailSystemDialog

**** = Functional in *Professional Edition* Only**

Address Book properties and methods are on a separate page.

# *Quick Start* Properties and Methods

## Properties

Message

ObjectKey

Subject

## Methods

AddAttachment

AddRecipientCc

AddRecipientTo

NewMessage

Send

See Also:

*Quick Start* Programming Examples

Complete Properties and Methods

# *Quick Start* Programming Examples

This sample code shows how to send a message with an atttachment. Two versions are provided, one for MAPI/VIM/MHS/VINES and another for SMTP.

Be sure you have included the file IDSMCONS.TXT in your project prior to running the example. This file defines various IDSMail constants.

Visual Basic

Access

Visual FoxPro

Excel

Delphi 2.0

PowerBuilder

# Visual Basic/Access/Excel *QuickStart* Code

**For MAPI/VIM/MHS/VINES:**

```
Dim idsMail as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
idsMail.ObjectKey = "ABC123"
idsMail.NewMessage
idsMail.AddRecipientTo "Jim Smith"
idsMail.AddRecipientCc "Mary Brown, Doug Williams"
idsMail.Subject = "Meeting Agenda"
idsMail.Message = "Here is the agenda for the weekly meeting."
idsMail.AddAttachment "C:\MEETINGS\AGENDA.DOC"
idsMail.Send
```

**For SMTP/POP:**

```
Dim idsMail as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
idsMail.ObjectKey = "ABC123"
idsMail.MailSystem =IDSM_SYS_SMTP_POP
idsMail.SMTPServer = "myprovider.com"
idsMail.NewMessage
idsMail.From = "MyName <me@mycompany.com>"
idsMail.AddRecipientTo "Jim Smith <jsmith@acme.com>"
idsMail.AddRecipientCc "Mary Brown <mary@congress.gov>, estone@acme.com"
idsMail.Subject = "Meeting Agenda"
idsMail.Message = "Here is the agenda for the weekly meeting."
idsMail.AddAttachment "C:\MEETINGS\AGENDA.DOC"
idsMail.Send
```

# Visual FoxPro *QuickStart* Code

**For MAPI/VIM/MHS/VINES:**

```
Local idsMail
idsMail = CreateObject("IDSMailInterface.Server")
idsMail.ObjectKey = "ABC123"
idsMail.NewMessage
idsMail.AddRecipientTo ("Jim Smith")
idsMail.AddRecipientCc ("Mary Brown, Doug Williams")
idsMail.Subject = "Meeting Agenda"
idsMail.Message = "Here is the agenda for the weekly meeting."
idsMail.AddAttachment ("C:\MEETINGS\AGENDA.DOC")
idsMail.Send
```

**For SMTP/POP:**

```
Local idsMail
idsMail = CreateObject("IDSMailInterface.Server")
idsMail.ObjectKey = "ABC123"
idsMail.MailSystem =IDSM_SYS_SMTP_POP
idsMail.SMTPServer = "myprovider.com"
idsMail.NewMessage
idsMail.From = "MyName <me@mycompany.com>"
idsMail.AddRecipientTo ("Jim Smith <jsmith@acme.com>")
idsMail.AddRecipientCc ("Mary Brown <mary@congress.gov>, estone@acme.com ")
idsMail.Subject = "Meeting Agenda"
idsMail.Message = "Here is the agenda for the weekly meeting."
idsMail.AddAttachment ("C:\MEETINGS\AGENDA.DOC")
idsMail.Send
```

# Delphi 2.0 *QuickStart* Code

**For MAPI/VIM/MHS/VINES:**

```
public
        idsMail: Variant;

idsMail := CreateOLEObject('IDSMailInterface.Server');
idsMail.ObjectKey := 'ABC123';
idsMail.NewMessage;
idsMail.AddRecipientTo ('Jim Smith');
idsMail.AddRecipientCc ('Mary Brown, Doug Williams');
idsMail.Subject := 'Meeting Agenda';
idsMail.Message := 'Here is the agenda for this weeks meeting';
idsMail.AddAttachment ('C:\MEETINGS\AGENDA.DOC');
idsMail.Send;
```

**For SMTP/POP:**

```
public
        idsMail: Variant;

idsMail := CreateOLEObject('IDSMailInterface.Server');
idsMail.ObjectKey := 'ABC123';
idsMail.MailSystem := IDSM_SYS_SMTP_POP;
idsMail.SMTPServer := 'myprovider.com';
idsMail.NewMessage;
idsMail.From := MyName <me@mycompany.com>;
idsMail.AddRecipientTo ('Jim Smith <jsmith@acme.com>');
idsMail.AddRecipientCc (' Mary Brown <mary@congress.gov>, estone@acme.com');
idsMail.Subject := 'Meeting Agenda';
idsMail.Message := 'Here is the agenda for this weeks meeting';
idsMail.AddAttachment ('C:\MEETINGS\AGENDA.DOC');
idsMail.Send;
```

# PowerBuilder *QuickStart* Code

**For MAPI/VIM/MHS/VINES:**

```
idsMail = Create OLEObject
idsMail.ConnectToNewObject("IDSMailInterface.Server")
idsMail.ObjectKey = "ABC123"
idsMail.NewMessage
idsMail.AddRecipientTo ("Jim Stevens")
idsMail.AddRecipientCc ("Mary Brown, Doug Williams")
idsMail.AddAttachment ("C:\MEETINGS\AGENDA.DOC")
idsMail.Subject = "Meeting Agenda"
idsMail.Message = "Here is the agenda for the weekly meeting."
idsMail.Send
```

**For SMTP/POP:**

```
idsMail = Create OLEObject
idsMail.ConnectToNewObject("IDSMailInterface.Server")
idsMail.ObjectKey = "ABC123"
idsMail.MailSystem = IDSM_SYS_SMTP_POP
idsMail.SMTPServer = "mail.wust.edu"
idsMail.NewMessage
idsMail.From = "Janet Stevens <jstevens@acme.com>"
idsMail.AddRecipientTo ("support@intuitive-data.com")
idsMail.AddRecipientCc ("Bill Clinton <president@whitehouse.gov>")
idsMail.AddAttachment ("C:\MEETINGS\AGENDA.DOC")
idsMail.Subject = "Meeting Agenda"
idsMail.Message = "Here is the agenda for the weekly meeting."
idsMail.Send
```

# **Receiving E-mail** (*Professional Edition Only*)

Receiving E-mail is almost as easy as sending it. Think for a moment about how your favorite E-mail program works. When you first run it, it loads message headers from your InBox and shows them to you in a summary list. To view a message, you double-click to load it into a window.

IDSMail™ works much the same way. You start by executing LoadMessageHeaders. This gives you the summary information such as the subject, sender, read status, and date. Use GetHeaderItems to retrieve these.

Select the specific message to read by setting the MessageIndex property. Execute LoadMessage which makes the message, recipients, file attachments, etc. immediately available. Use properties such as Message, Subject, etc. to get the message items, or call GetMessageItems to get a list of them.

If you plan to display the received E-mail in a receive form/window, use GetMessageItemString to retrieve each of the items in a convenient comma-delimited list suitable for direct display.

If you are like me, you probably have hundreds of old messages languishing in your InBox. Use the UnreadOnly & MaximumHeaders properties to limit your search to the most recent messages. When you are ready to do some housekeeping, try the DeleteMessage and MoveMessage methods. Oh, and if the Folder you move your messages to doesnt exist, IDSMail™ creates it for you automatically.

Want to just check up on things? The UnreadCount property will query your mail system to see if there are any new messages for you. And, if you want to sneak a peek without marking the message as read, the PeekOnly property is for you.

If you receive E-mail responses from World Wide Web forms, be sure to have a look at our GetWebFormValue method. It really makes things easy.

# AddAlternative (method)

## Description

This method adds an alternative representation of the message text to the mail message. It is applicable to Internet mail only. Alternative representations will be ignored for all other mail systems.

Please read this overview of MIME Alternative Text Parts if you have not already.

## Parameters

*theAlternative$* - String specifying the full path to the file containing the alternative text (e.g. "C:\TEMP\MYHTML.HTM"). Leading/trailing spaces will be ignored.

*contentType$* - String specifying the MIME content type information for the alternative. Content type strings for the most commonly used alternative types should be specified as follows:

| | |
|---|---|
| Rich Text Format (RTF) | "text/richtext" |
| HTML | "text/html" |
| Enriched (used by Eudora) | "text/enriched" |

*reserved%* - A short integer that is reserved for future expansion. Applications should pass zero for this parameter.

## Example

```
idsMail.NewMessage
idsMail.Message = "This is the plain-text message.  HTML and RTF alternatives are also present."
theAlternative$ = "C:\TEMP\MYHTML.HTM
contentType$ = "text/html"
reserved% = 0
idsMail.AddAlternative theAlternative$, contentType$, reserved%
theAlternative$ = "C:\TEMP\MYHTML.HTM
contentType$ = "text/richtext"
idsMail.AddAlternative theAlternative$, contentType$, reserved%
idsMail.AddRecipientTo "billg@microsoft.com"
idsMail.Send
```

# AddAttachment (method)

## Description

This method adds file attachments to the mail message. Attachments accumulate in a message with each call of this method.

## Parameters

*theAttachment$* - String specifying one or more file attachments, with multiple attachments separated by commas. Attachments must be specified with a full path (e.g. "C:\PROJECTS\MYFILE.TXT"). Leading/trailing spaces will be ignored.

## Example

The following two code fragments are equivalent:

```
idsMail.AddAttachment "C:\AUTOEXEC.BAT,C:\CONFIG.SYS"
idsMail.AddAttachment "C:\WINDOWS\WIN.INI"
idsMail.AddRecipientTo "Bob Linden"
idsMail.Send
```

---

```
idsMail.AddAttachment "C:\AUTOEXEC.BAT, C:\CONFIG.SYS,     C:\WINDOWS\WIN.INI"
idsMail.AddRecipientTo "Bob Linden"
idsMail.Send
```

# AddRecipientBcc (method)

## Description

This method adds a "blind" recipient to the mail message. The blind recipient will receive the message, but other recipients will be unaware of this. Bcc recipients accumulate in a message with each call of this method.

## Parameters

*theRecipient$* - String specifying one or more "blind" recipients, with multiple recipients separated by commas (or the character specified by the RecipientDelimiter property). Leading/trailing spaces will be ignored.

## Example

The following two code fragments are equivalent:

```
idsMail.AddRecipientBcc "JohnP@Accounting@Acme,FrankS@Manufacturing@Acme"
idsMail.AddRecipientBcc "MelindaB@Accounting@Acme"
idsMail.AddRecipientTo "WilmaR@Finance@Acme"
idsMail.Send
```

---

```
idsMail.AddRecipientBcc "JohnP@Accounting@Acme,      FrankS@Manufacturing@Acme,
MelindaB@Accounting@Acme"
idsMail.AddRecipientTo "WilmaR@Finance@Acme"
idsMail.Send
```

## See Also:

Address Formats

# AddRecipientCc (method)

## Description

This method adds a "carbon copy" recipient to the mail message.

## Parameters

*theRecipient$* - String specifying one or more "carbon copy" recipients, with multiple recipients separated by commas (or the character specified by the RecipientDelimiter property). Leading/trailing spaces will be ignored.

## Example

The following two code fragments are equivalent:

```
idsMail.AddRecipientCc "John Powers,Frank Short"
idsMail.AddRecipientCc "Melinda Brennan"
idsMail.AddRecipientTo "Wilma Meyers"
idsMail.Send
```

---

```
idsMail.AddRecipientCc "John Powers,     Frank Short, Melinda Brennan"
idsMail.AddRecipientTo "Wilma Meyers"
idsMail.Send
```

## See Also:

Address Formats

# AddRecipientTo (method)

## Description

This method adds a primary recipient to the mail message. Primary recipients accumulate in a message with each call of this method. At least one primary recipient must be designated or an error will occur when the message is sent.

## Parameters

*theRecipient$* - String specifying one or more primary recipients, with multiple recipients separated by commas (or the character specified by the RecipientDelimiter property). Leading/trailing spaces will be ignored.

## Example

The following two code fragments are equivalent:

```
idsMail.AddRecipientTo "JohnP@Accounting, FrankS@Manufacturing"
idsMail.AddRecipientTo "MelindaB@Accounting"
idsMail.AddRecipientCc "WilmaR@Finance"
idsMail.Send
```

---

```
idsMail.AddRecipientTo "JohnP@Accounting,  FrankS@Manufacturing, MelindaB@Accounting"
idsMail.AddRecipientCc "WilmaR@Finance"
idsMail.Send
```

## See Also:

Address Formats

# AddXHeader (method)

## Description

This method adds a header with the "X-" prefix to an Internet mail message. Values added with this method are ignored for all other mail systems.

Headers that begin with "X are designated experimental and may be used for any purpose. Internet mail client programs are not required to process these in any way. The headers added with this method are cleared each time the NewMessage method is executed.

## Parameters

*theName$* - String specifying the name of the header, without the "X-" prefix. The "X-" prefix will be automatically prepended to the name. There are two special-case values for this variable:

"Mailer" - Specifying this value for the name will also change the IDSMail XMailer property value.

"Interface" - This value is reserved by IDSMail and will trigger an error if used.

*theValue$* - String specifying the value for the header.

## Example

Add an XHeader designating company name

```
theName$ = Company
theValue$ = Acme Manufacturing
idsMail.AddXHeader theName$, theCompany$
```

## See Also:

White Paper: Using X-Headers to your Advantage

# ApplicationName (property)

## Description

This property is used only for MHS. It specifies the name of the subdirectory under the users inbox where a Message Transfer Agent (MTA) deposits incoming mail messages. It is also used during execution of the MoveMessage methods for MHS messages.

The application name must be a valid DOS directory descriptor, with a maximum of 8 characters.

If you do not know your MHS application name, contact your network administrator. If none is specified, IDSMail™ will look for messages in the top-level user inbox.

## Usage

Read/Write, Default = empty string

## Example

```
idsMail.ApplicationName = "OUR_MAIL"
```

# AsynchAbort (method)

## Description

This method aborts an asynchronous operation, and will return an error if a notification object has not first been registered via the AsynchObjectRegister method.

The method may take a few seconds to return. After executing the method, two notifications will be performed:

> 1) The running process will issue a notification with the *event* parameter set to IDSM_EVENT_COMPLETE, and the *status* flag set to IDSM_ERR_ASYNCH_ABORT.
>
> 2) The abort process will issue a notification with the *method* parameter set to IDSM_PROCESS_ABORT, the *event* parameter set to IDSM_EVENT_COMPLETE, and the *status* flag set to True (-1).

NOTE: This method is safe to use during the LoadMessageHeaders and LoadMessageIdentifiers methods. Executing it during the LoadMessage, method could leave the network in an error state, resulting in IDSMail errors during processes following the AsynchAbort.

Executing the method during Send, and ShowMailDialog methods is generally safe, especially after the message headers have been sent successfully. Executing it during transmission of headers may result in a corrupted message being received by the recipient. Which part of the process that IDSMail is in can be derived from the notification event parameters.

## Parameters

{NONE}

## Example

idsMail.AsynchAbort

## See Also:

> The Notification Object
>
> AsynchObjectRegister

# AsynchObjectRegister (method)

## Description

This method enables the client application to register an asynchronous notification object with IDSMail™. The AsynchNotification method of that object will then be called with progress updates of the mail operation in process. This enables mail operations to be performed entirely in the background. The client application if free to perform other tasks during this time.

The AsynchNotification method of the registered object has a specific parameter list that should not be modified. For a source code example of this object, refer to the Visual Basic 4.0 class module IDSMNTFY.CLS that was installed in the same directory as IDSMail™. Refer also to The Notification Object.

Once an object is registered, the following IDSMail™ methods will return control to the client application shortly after executing them, even though their processes may not yet be complete:

| | | |
|---|---|---|
| DeleteMessage | LoadMessage | LoadMessageHeaders |
| LoadMessageIdentifiers | Send | ShowMailDialog |

Asynchronous notifications are only supported for Internet mail (SMTP/POP protocols). Other mail systems will function normally.

**NOTE1: It is critical that the object be unregistered prior to termination of the client application via the AsynchObjectUnregister method.**

**NOTE2: This method is not supported for Visual Basic v3.0, Access v2.0, and other environments which cannot create and pass object references.**

## Parameters

*asynchObject* - Object variable that receives the notification object passed in by the client application.

*objectType* - Long integer variable that specifies the type of object passed in. In the current version of IDSMail™, this should always be the constant IDSM_OBJTYPE_NOTIFY (value = 0). Other values are reserved for future expansion.

*config* - Long integer variable that specifies the types of notifications made. One of the following values:

IDSM_ASYNCH_COMPLETION_ONLY
IDSM_ASYNCH_PROGRESS

*reserved* - Variant variable reserved for future expansion.

## Example

Declarations section: Declare notification object to be global

```
Global notify As Object
```

In procedure/function, create object and register it

```
Set notify = New IDSMailNotify
idsMail.AsynchObjectRegister(notify, IDSM_OBJTYPE_NOTIFY, IDSM_ASYNCH_PROGRESS,
reserved)
```

## See Also:

The Notification Object

AsynchObjectUnregister

**IDSM_ASYNCH_COMPLETION_ONLY (value = 0)**

This flag specifies that notifications will only be provided when the event completes, whether successfully or in an error state.

**IDSM_ASYNCH_PROGRESS (value = 1)**

This flag specifies that notifications will be provided throughout the event, giving progress updates. These updates are suitable for driving a progress bar, etc.

# AsynchObjectUnregister (method)

## Description

This method unregisters an asynchronous object. All IDSMail methods will then revert to their normal non-asynchronous mode of operation. If an asynchronous object has been registered, it is critical that it be unregistered prior to termination of the client application.

**NOTE: This method is not supported for Visual Basic v3.0, Access v2.0, and other environments which cannot create and pass object references.**

## Parameters

*asynchObject* - Object variable that was previously registered via the AsynchObjectRegister method.

*objectType* - Long integer variable that specifies the type of object. In the current version of IDSMail™, this should always be the constant IDSM_OBJTYPE_NOTIFY (value = 0). Other values are reserved for future expansion.

## Example

Declarations section: Declare notification object to be global

```
Global notify As Object
```

In procedure/function, unregister object

```
idsMail.AsynchObjectUnregister(notify, IDSM_OBJTYPE_NOTIFY)
Set notify = Nothing
```

## See Also:

The Notification Object

AsynchObjectRegister

# AttachmentEncoding (property)

## Description

This property specifies how file attachments are compressed and/or encoded for mail send. The property value is a combination of flag values which are added together (logical OR) to form the desired encoding method.

The default value of the property specifies no encoding at all for MAPI, MHS, VIM, or VINES. Internet mail messages (SMTP) are specified to use MIME encoding. This property has no effect on attachment decoding: IDSMail™ will automatically determine the proper decoding method to use based on the contents of the received message.

The following flag values are possible for this property:

IDSM_ENC_NONE
IDSM_ENC_UUENCODE
IDSM_ENC_MIME
IDSM_ENC_ZIP
IDSM_ENC_ZIP_TOGETHER
IDSM_ENC_ZIP_ALL_SYSTEMS
IDSM_ENC_ENC_ALL_SYSTEMS

## Usage

Read/Write, Default = IDSM_ENC_MIME

## Examples

ZIP Internet attachments prior to encoding, and use UUENCODE.

```
idsMail.AttachmentEncoding = IDSM_ENC_ZIP + IDSM_ENC_UUENCODE
```

ZIP all file attachments for any mail system, and use MIME encoding for the Internet only.

```
idsMail.AttachmentEncoding = IDSM_ENC_ZIP + IDSM_ENC_ZIP_ALL_SYSTEMS + IDSM_ENC_MIME
```

**IDSM_ENC_NONE (value = 0)**

This flag specifies that no encoding or compression should be applied for most mail systems. For Internet mail (SMTP), encoding is necessary and MIME encoding will automatically be applied if this value is specified for the property.

**IDSM_ENC_UUENCODE (value = 1)**

This flag specifies that UUENCODE is used for Internet mail (SMTP). UUENCODE is an older, less standardized method than MIME. No encoding is performed for other systems unless the IDSM_ENC_ENC_ALL_SYSTEMS flag is also set.

**IDSM_ENC_MIME (value = 2)**

This flag specifies that Internet mail messages should be sent in MIME format whenever file attachments are present. MIME stands for **M**ultipart **I**nternet **M**ail **E**xtensions and is a newer, more robust method as compared to UUENCODE. Each file attachment will be encoded using the Base64 algorithm and inserted into the MIME message.

No encoding is performed for other systems unless the IDSM_ENC_ENC_ALL_SYSTEMS flag is also set.

**IDSM_ENC_ZIP (value = 16)**

This flag specifies that Internet mail file attachments should first be zipped (compressed) prior to encoding. This speeds the mail send operation significantly if large non-compressed files are being sent. The compression algorithm is fully compatible with PKZIP v2.04g. Attachments are not zipped for the other mail systems unless the IDSM_ENC_ZIP_ALL_SYSTEMS flag is also set.

**IDSM_ENC_ZIP_TOGETHER (value = 32)**

This flag specifies that multiple file attachments will all be combined into a single .ZIP file prior to encoding and send. This flag should be set together with IDSM_ENC_ZIP. The ZIP operation will only apply to Internet mail messages (SMTP) unless the IDSM_ENC_ZIP_ALL_SYSTEMS flag is also set.

**IDSM_ENC_ZIP_ALL_SYSTEMS (value = 64)**

This flag specifies that ZIP file compression will be applied to all mail systems and not just Internet mail. For this flag to have an effect, IDSM_ENC_ZIP must also be set.

**IDSM_ENC_ENC_ALL_SYSTEMS (value = 128)**

This flag specifies that UUENCODE or Base64 encoding (used within MIME messages) will be applied to all mail systems and not just to Internet mail.

# Certify (property)

## Description

Setting the Certify property to True (-1) causes the sender to receive an automatic confirmation message (return receipt) when the mail is read by each recipient.

## Usage

Read/Write, True (-1) =Certify, False (0) =Dont certify, Default = False

## Example

```
idsMail.AddRecipientTo "Rachel Administrator"
idsMail.Message = "My system is down, please send a repair tech. "
idsMail.Subject = "Urgent! "
idsMail.Certify = True
idsMail.Send
```

# ClearMessageItems (method)

## Description

A utility method which may be used to clear specific types of items from a message. This may be used (for example) in place of NewMessage if several similar messages are being composed and sent.

## Parameters

*itemType%* - Integer specifying the type of item to retrieve from the message. One of the following constants:

IDSM_ITM_TO
IDSM_ITM_CC
IDSM_ITM_BCC
IDSM_ITM_ATTACHMENT
IDSM_ITM_SUBJECT
IDSM_ITM_MESSAGE

## Example

```
idsMail.ClearMessageItems IDSM_ITM_TO
```

# DateFormat (property)

## Description

This property specifies how dates are formatted during mail receive and has no impact on mail send. All of the formats except for IDSM_DATEFORMAT_FIXED will be dependent on the international date settings in Windows Control Panel and may display differently than the examples below. IDSM_DATEFORMAT_FIXED will always display in the same format, regardless of the Control Panel settings.

| Value | Resulting Format (for US) |
|---|---|
| IDSM_DATEFORMAT_GENERAL | 3/11/96 11:44:23 AM |
| IDSM_DATEFORMAT_SHORT | 3/11/96 |
| IDSM_DATEFORMAT_MEDIUM | 11-Mar-96 |
| IDSM_DATEFORMAT_LONG | Sunday, March 03, 1996 |
| IDSM_DATEFORMAT_FIXED | Oct 02, 1996 14:24:03 |
| IDSM_DATEFORMAT_NATIVE | Varies depending on mail system |

## Usage

Read/Write, Default = IDSM_DATEFORMAT_GENERAL

## Example

```
idsMail.DateFormat = IDSM_DATEFORMAT_SHORT
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 2
idsMail.LoadMessage
```

# DateSent (property)

## Description

The DateSent property is read-only and returns the date that the current message was sent.

## Usage

Read-only

## Example

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 2
idsMail.LoadMessage
theDate$ = idsMail.DateSent
MsgBox Message sent on    + theDate$
```

# DateSort (property)

## Description

If True (-1), message headers are sorted by date with the latest message first. This sorting happens upon the next execution of LoadMessageHeaders. The default for this property is True.

This property is ignored for VIM and Banyan VINES since these mail systems do the sorting automatically. Setting the property to False (0) will provide a slight performance increase during header loading.

## Usage

Read/Write, Default=True (-1)

## Example

```
idsMail.DateSort = True
idsMail.LoadMessageHeaders
```

# DeleteMessage (method)

## Description

The DeleteMessage method is used to delete the message pointed at by the MessageIndex property. Note that this message is not necessarily the loaded message.

## Parameters

{NONE}

## Example

Delete the second message in this folder

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 1
idsMail.DeleteMessage
```

# DialogCaption (property)

## Description

The DialogCaption property is used for the Send Mail dialog (launched by the ShowMailDialog method. DialogCaption is ignored for native mail system dialogs (i.e. when NativeMailDialog = False).

## Usage

Read/Write, Default = "Send Mail Message"

## Example

Show a mail send dialog using the IDSMail server dialog rather than the native dialog.

```
idsMail.NativeMailDialog = False
idsMail.DialogCaption = "My Send Mail Dialog"
idsMail.ShowMailDialog
```

# DialogErrorOnCancel (property)

## Description

The DialogErrorOnCancel property determines whether or not an error is triggered when the user cancels a mail send from the user interface.

If the user cancels when this property is True (-1), the server generates an error (error code = IDSM_ERR_USER_ABORT)  If the user cancels when this property is False (0), no error is generated by the server.

## Usage

Read/Write, Default = False (0)

## Example

Show a mail send dialog and trigger an error if user cancels

```
idsMail.NativeMailDialog = False
idsMail.DialogErrorOnCancel = True
idsMail.ShowMailDialog
```

# ErrorDisplay (property)

## Description

Setting the ErrorDisplay property to True (-1) causes the server to display error messages in message boxes as they occur. IDS recommends using it only during test and debugging.

Some errors will be displayed from low-level mail handling routines. The error codes displayed will not have the usual offset number pertaining to the specific mail system ( see Error Reporting for more information). Instead, they will be the actual value returned from the mail system API function. Normally the function name is included in the descriptive error message.

For example, a VIM mail system may return a value of 11 which indicates an invalid password. This corresponds to the IDSMail error code of 4011. The mail system specific error codes are defined in the file IDSM_ERR.TXT which was installed in the same directory as IDSMail.

## Usage

Read/Write, Default = False (0)

## Example

```
idsMail.NativeMailDialog = False
idsMail.ErrorDisplay = True
```

# Folder (property)

## Description

The Folder property is used to specify the folder that will be loaded on the next execution of the LoadMessageHeaders method. If not specified, the InBox (or "General" folder in Banyan VINES) will be used as a default.

Note that the concept of folders is absent from MAPI and POP. For MAPI and POP, this property will be ignored and messages will always be loaded from the InBox.

Under MHS, directories are used for folders. If set, Folder must represent a valid DOS directory, including path. (Ex: C:\MYMAIL\URGENT). If not set, the default user inbox directory (in MHS directory structure) will be used.

If Lotus Notes is the underlying VIM mail system, folders are known as "Categories". If this property is specified, it must be a valid Notes database pathname (e.g. "c:\file.nsf, "myserver!!mail\file.nsf", etc.)

A special-case folder for cc:Mail is the string value "IDSM_MESSAGE_LOG" which causes the contents of the Message Log folder (i.e sent or saved messages) to be listed.

## Usage

Read/Write

## Example

```
idsMail.MailSystem = IDSM_SYS_VINES
idsMail.Folder = "MyStuff"
idsMail.LoadMessageHeaders
```

# ForceDownload (property)

## Description

This property is used by MAPI only and is ignored for other mail systems.  If True (-1), MAPI will download mail from the server during login.  In remote installations, this property will also trigger a dial-in if required.  Login may take substantially longer if this property is set True.

## Usage

Read/Write, Default = False (0)

## Example

```
idsMail.ForceDownload = True
idsMail.LoadMessageHeaders
```

# From (property)

## Description

During mail receive, the From property identifies the sender of the currently loaded message.

During mail send, From is only used with Internet mail (SMTP). In other mail systems the property is ignored as messages receive their identity from the mail system login information.

## Usage

Read/Write, Default = empty string

## Examples

Mail receive example

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 4
idsMail.LoadMessage
whoFrom$ = idsMail.From
MsgBox "This message is from " + whoFrom$
```

Internet mail send example

```
idsMail.MailSystem = IDSM_SYS_SMTP_POP
idsMail.NewMessage
idsMail.From = "Joe Riley <joe@acme.com>"
idsMail.Message = "The meeting was cancelled"
```

All of the following formats are acceptable for Internet mail

```
idsMail.From = "Joe Riley <joe@acm0e.com>"
idsMail.From = "joe@acme.com"
idsMail.From = "<joe@acme.com>"
```

# GetHeaderItems (method)

## Description

This method fills an array with the specified type of items from all currently loaded message headers. This enables the message headers to be searched for a particular message (for example) without having to load each message individually. The first item will be placed at array index=0.

**NOTE: This method is not supported for Visual Basic v3.0, Access v2.0, and other environments which implemented an older version of OLE support. In these cases, use GetHeaderItemString instead.**

## Parameters

*itemType%* - Integer specifying the type of item to retrieve from the headers. One of the following constants:

IDSM_ITM_FROM
IDSM_ITM_SUBJECT
IDSM_ITM_DATE_SENT
IDSM_ITM_STATE (i.e. unread, read, etc.)
IDSM_ITM_CERTIFIED
IDSM_ITM_ATTACHMENT_COUNT
IDSM_ITM_MESSAGE_ID
IDSM_ITM_RETURN_PATH

The following constants only return information for Internet mail messages (POP).

IDSM_ITM_RAW_HEADERS
IDSM_ITM_X_MAILER
IDSM_ITM_CONTENT_LENGTH
IDSM_ITM_MESSAGE_LENGTH
IDSM_WEB_FORM_ENCODED
IDSM_ITM_ALTERNATIVE_COUNT
IDSM_ITM_UIDL

NOTE: For IDSM_ITM_STATE, IDSM_ITM_CERTIFIED, IDSM_ITM_ATTACHMENT_COUNT, and IDSM_ITM_ALTERNATIVE_COUNT the numeric values will be returned as strings.

*itemArray$()* - An array of strings which will be filled with all of the available items. The array must be dimensioned large enough to hold all of the items, or an error will occur.

*itemCount%* - Integer designating the number of items retrieved. This will normally be the same as the MessageCount property.

## Example

Load messages and get the subjects

```
idsMail.LoadMessageHeaders
numMessages% = idsMail.MessageCount
ReDim items$(numMessages%)
idsMail.GetHeaderItems IDSM_ITM_SUBJECT, items$(), itemCount%
```

```
    Search for important messages

For i% = 0 to numMessages& -1
        If items$(i%) = "IMPORTANT!" Then
                idsMail.MessageIndex = i%
                idsMail.LoadMessage
                Exit For
        End If
Next I%
```

**IDSM_ITM_TO (value = 1)**

The To recipients of the message.

**IDSM_ITM_BCC (value = 3)**

The Bcc recipients of the message.

**IDSM_ITM_CC (value = 2)**

The Cc recipients of the message.

**IDSM_ITM_FROM (value = 9)**

Who the message was from.

**IDSM_ITM_SUBJECT (value = 4)**

The subject of the mail message.

**IDSM_ITM_DATE_SENT (value = 10)**

The date that the message was sent.

**IDSM_ITM_STATE (value = 11)**

The read/unread status of the message. One of the following values:

IDSM_STATE_UNKNOWN
IDSM_STATE_UNREAD
IDSM_STATE_READ
IDSM_STATE_UNSENT
IDSM_STATE_SENT

NOTE: Many mail systems only indicate the state if it is unread. The Read, Sent, and Unsent states may not be available. Because of this, the preferred program logic is:

```
If theState = IDSM_STATE_UNREAD Then
    MsgBox "The message is unread"
Else
    MsgBox "The message is NOT unread"
End If
```

**IDSM_ITM_CERTIFIED(value = 12)**

True (-1) if the received message was certified, or False (0) if not.

**IDSM_ITM_ATTACHMENT (value = 6)**

The filename(s) including path(s) of the attached files on the local workstation. See the description for the LoadMessage method to see how these are determined.

**IDSM_ITM_ATTACHMENT_NAME (value = 7)**

The original filename(s) of the attached files, not including path (Ex: CONFIG.SYS). To get the filename as stored on the local workstation including path, use the IDSM_ITM_ATTACHMENT selector.

**IDSM_ITM_ATTACHMENT_COUNT (value = 8)**

The count of the attached files. For most mail systems, this count is accurate immediately after the LoadMessageHeaders method is executed.

For some mail systems, this parameter is only accurate after executing the LoadMessage method. These exceptions are described below:

**Internet Mail (POP)**

The Internet mail protocols do not support an attachment count in the message headers. Therefore, this parameter will only be correct after loading the actual message via the LoadMessage method.

Note that Internet mail messages containing file attachments can be sent either as MIME messages or as non-MIME messages with the files UUENCODED into the message body. IDSMail™ recognizes the two formats immediately after loading the headers and sets the attachment count as follows:

MIME Message - 1
non-MIME Message (UUENCODE) - 0

This allows you to present an attachment designator in your user interface if a MIME message is received. The attachment count will be updated to the actual value after the message is loaded.

**Novell GroupWise (MAPI)**

If GroupWise is the underlying MAPI mail system, the attachment count will only be correct after executing the LoadMessage method. If only the headers have been loaded, this parameter will always be zero. Note that for Microsoft MAPI systems this parameter will always be correct, even if the message has not yet been loaded..

**IDSM_ITM_ALTERNATIVE_COUNT (value = 30)**

The count of the MIME alternative message parts in a received Internet mail message. The count is only accurate after executing the LoadMessage method.

**IDSM_ITM_RAW_HEADERS (value = 21)**

**Internet mail only**. Returns the raw message headers as a string, with the individual headers/values separated by a CRLF pair. Here is an example of the raw headers:

-------- <RAW HEADERS EXAMPLE>-------
Return-Path: <Peter.Vogel@odyssey.on.ca>
Received: from mur.odyssey.on.ca by kudonet.com (5.x/SMI-SVR4) id AA27597; Wed, 31 Jul 1996 18:55:42 -0700
Received: from mur.odyssey.on.ca (ts2-3.odyssey.on.ca [206.47.131.103]) by mur.odyssey.on.ca (8.7.5/8.7.3) with SMTP id VAA20739 for <ids@kudonet.com>; Wed, 31 Jul 1996 21:52:36 -0400 (EDT)
Message-Id: <1.5.4.32.19960801013934.006812e0@mail.odyssey.on.ca>
X-Sender: peter.vogel@mail.odyssey.on.ca
X-Mailer: Windows Eudora Light Version 1.5.4 (32)
Mime-Version: 1.0
Date: Wed, 31 Jul 1996 21:39:34 -0400
To: ids@kudonet.com
From: Peter Vogel <Peter.Vogel@odyssey.on.ca>
Subject: Re: IDSMail Product Review
Content-Type: text/plain; charset="us-ascii"
Content-Length: 3651
Status: RO
-------- <END RAW HEADERS EXAMPLE>-------

**IDSM_ITM_X_MAILER (value = 22)**

**Internet mail only**. An optional header field that indicates the type of mail application program that was used to send the message.

**IDSM_ITM_RETURN_PATH (value = 20)**

The address to be used for a reply message, which may be different than that given by the From property. The format of the returned string is "DisplayName <FullAddress>". This string may be passed to the IDSMail AddRecipient methods to specify a fully address.

Note that in MAPI & VIM, the From property returns only the name of the sender (e.g. "Joe Smith"). If the sender is a member of the same post office as the receiver, a reply may successfully be addressed to this name by itself. But, if the sender is a member of a different post office, specifying just that name may be insufficient. The string IDSM_ITM_RETURN_PATH is a better choice.

As an example, assume a cc:Mail (VIM) system is in use and a message is received from Joe Smith who is a member of a different post office. The From property will read "Joe Smith" while the IDSM_ITM_RETURN_PATH string will be "Joe Smith <Joe Smith at JoesPostOffice>".

**In general: Always try to use the IDSM_ITM_RETURN_PATH variable for reply addressing as it contains the most specific addressing information.**

**IDSM_ITM_MESSAGE_ID (value = 23)**

The globally unique message ID for the mail message. The message ID is supported by all mail systems except for VIM. Format of the message ID varies widely between mail systems. The message ID is not required by IDSMail™ for any processing operations.

**IDSM_ITM_UIDL (value = 31)**

The unique identifier provided by the POP server.  Not all POP servers support this field.

**IDSM_ITM_CONTENT_LENGTH (value = 24)**

**Internet mail only**. The length (in bytes) of the message excluding headers and other extraneous information. Note that this is based on a header field in the message that is not guaranteed to be present. If this header field is absent, 0 is returned. IDSM_ITM_MESSAGE_LENGTH will return a size measurement that will always be present but includes extraneous items.

**IDSM_ITM_MESSAGE_LENGTH (value = 25)**

**Internet mail only**. The length (in bytes) of the entire message as reported by the POP mail server. This will include the size of extraneous items such as headers, MIME boundaries, etc. IDSM_ITM_CONTENT_LENGTH provides the size of the message and attachments only (without extraneous information) but may not be available for every message.

**IDSM_ITM_WEB_FORM_ENCODED (value = 26)**

**Internet mail only**. Returns True in string form ("-1") if the message is a web form response, or False ("0") if not. IDSMail determines this from the Content-Type field in the message header. Not all web browsers correctly mark this field (AOL browsers are a specific example) so it is possible for False to be returned when in fact it IS a web form response.

**IDSM_ITM_WEB_FORM_FIELD (value = 27)**

**Internet mail only**. If the received message is a web form response, this selector causes the web form field names to be returned. In the pair "Name=Jerry", "Name" will be returned.

**IDSM_ITM_WEB_FORM_VALUE (value = 28)**

**Internet mail only**. If the received message is a web form response, this selector causes the web form field names to be returned. In the pair "Name=Jerry", "Jerry" will be returned.

See Also:

GetWebFormValue

**IDSM_ITM_BAD_RECIPIENT (value = 29)**

**Internet mail only**. If the sent message had any bad recipients, this selector can be used to retrieve information about the bad recipients. Note that if the error reported by the SMTP server was a severe one (i.e. a command error vs. a bad recipient) then the bad recipient(s) may not be available.

# GetHeaderItemString (method)

## Description

This method returns a string with the specified type of item from the message header pointed at by MessageIndex. For Internet mail only, raw header values can also be retrieved via the GetHeaderValueByName method.

## Parameters

*itemType%* - Integer specifying the type of item to retrieve from the headers. One of the following constants:

IDSM_ITM_FROM
IDSM_ITM_SUBJECT
IDSM_ITM_DATE_SENT
IDSM_ITM_STATE (i.e. unread, read, etc.)
IDSM_ITM_CERTIFIED
IDSM_ITM_ATTACHMENT_COUNT
IDSM_ITM_MESSAGE_ID
IDSM_ITM_RETURN_PATH

The following constants only return information for Internet mail messages (POP).

IDSM_ITM_RAW_HEADERS
IDSM_ITM_X_MAILER
IDSM_ITM_CONTENT_LENGTH
IDSM_ITM_MESSAGE_LENGTH
IDSM_WEB_FORM_ENCODED
IDSM_ITM_ALTERNATIVE_COUNT
IDSM_ITM_UIDL

NOTE: For IDSM_ITM_STATE, IDSM_ITM_CERTIFIED, IDSM_ITM_ATTACHMENT_COUNT, and IDSM_ITM_ALTERNATIVE_COUNT the numeric values will be returned as strings.

## Example

Get the originator of the fifth message

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 4
theOriginator$ = idsMail.GetHeaderItemString(IDSM_ITM_FROM)
```

# GetHeaderValueByName (method)

## Description

This method returns a string containing the raw value of the specified Internet mail header element. It is only used for Internet mail and will return an empty string for other mail systems. The message for which the header element is desired must be pointed to by the MessageIndex property. Header values are available immediately after the LoadMessageHeaders method completes.

This method can obtain any arbitrary header value. The GetHeaderItemString method is a higher performance means of returning selected elements.

## Parameters

*theField$* - String which specifies the POP mail header element to get the value for. Case insensitive.

## Example

Get the X-mailer field from the third message

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 2
theField$ = "X-Mailer"
theMailer$ = idsMail.GetHeaderValueByName (theField$)
```

# GetMessageItemCount (method)

## Description

A utility method which may be used to retrieve the count of specific types of items in the currently loaded message.

## Parameters

*itemType%* - Integer specifying the type of item to count. One of the following constants:

IDSM_ITM_TO
IDSM_ITM_CC
IDSM_ITM_BCC
IDSM_ITM_ATTACHMENT
IDSM_ITM_ATTACHMENT_COUNT
IDSM_ITM_ATTACHMENT_NAME
IDSM_ITM_SUBJECT (always 1)
IDSM_ITM_MESSAGE (always 1)
IDSM_ITM_FROM (always 1)
IDSM_ITM_WEB_FORM_ENCODED
IDSM_ITM_WEB_FORM_FIELD
IDSM_ITM_WEB_FORM_VALUE
IDSM_ITM_BAD_RECIPIENT
IDSM_ITM_ALTERNATIVE_COUNT

NOTE: All three ATTACHMENT constants return the count of the file attachments.

## Example

Count the cc recipients in the third message

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 2
idsMail.LoadMessage
theCcCount% = idsMail.GetMessageItemCount(IDSM_ITM_CC)
```

# GetMessageItems (method)

## Description

A utility method which may be used to retrieve all of the items of a specific type in the current mail message. The first item will be placed at array index=0.

**NOTE: This method is not supported for Visual Basic v3.0, Access v2.0, and other environments which implemented an older version of OLE support. In these cases, use GetMessageItemString instead.**

## Parameters

*itemType%* - Integer specifying the type of item to retrieve from the message. One of the following constants:

IDSM_ITM_TO
IDSM_ITM_CC
IDSM_ITM_BCC
IDSM_ITM_FROM
IDSM_ITM_SUBJECT
IDSM_ITM_DATE_SENT
IDSM_ITM_STATE (i.e. unread, read, etc.)
IDSM_ITM_CERTIFIED
IDSM_ITM_ATTACHMENT
IDSM_ITM_ATTACHMENT_NAME
IDSM_ITM_ATTACHMENT_COUNT
IDSM_ITM_MESSAGE_ID
IDSM_ITM_RETURN_PATH

The following constants only return information for Internet mail messages (POP).

IDSM_ITM_RAW_HEADERS
IDSM_ITM_X_MAILER
IDSM_ITM_CONTENT_LENGTH
IDSM_ITM_MESSAGE_LENGTH
IDSM_ITM_WEB_FORM_ENCODED
IDSM_ITM_WEB_FORM_FIELD
IDSM_ITM_WEB_FORM_VALUE
IDSM_ITM_BAD_RECIPIENT
IDSM_ITM_ALTERNATIVE_COUNT
IDSM_ITM_UIDL

NOTE: For IDSM_ITM_STATE, IDSM_ITM_CERTIFIED, IDSM_ITM_ATTACHMENT_COUNT, and IDSM_ITM_ALTERNATIVE_COUNT the numeric values will be returned as strings.

*itemArray$()* - An array of strings which will be filled with all of the available items. The array must be dimensioned large enough to hold all of the items, or an error will occur.

*itemCount%* - Integer designating the number of items retrieved.

## Example

```
Dim items$(50)
idsMail.GetMessageItems IDSM_ITM_TO, items$(), itemCount%
```

# GetMessageItemString (method)

## Description

This method returns a string containing the specified type of item from the currently loaded message. If more than one item is present, they will be separated by a delimiter and a single space. The method is intended to allow the programmer to easily display message recipients in a text box, in the same format as most E-mail packages.

IDSM_ITM_TO, IDSM_ITM_CC, and IDSM_ITM_BCC strings will be delimited by the character specified by the RecipientDelimiter property, which defaults to a comma. All other item types use a comma as the delimiter.

Here is a typical returned string: "Bob Jones, Mary Williams, Fred McMurray"

## Parameters

*itemType%* - Integer specifying the type of item to retrieve from the message. One of the following constants:

IDSM_ITM_TO
IDSM_ITM_CC
IDSM_ITM_BCC
IDSM_ITM_FROM
IDSM_ITM_SUBJECT
IDSM_ITM_DATE_SENT
IDSM_ITM_STATE (i.e. unread, read, etc.)
IDSM_ITM_CERTIFIED
IDSM_ITM_ATTACHMENT
IDSM_ITM_ATTACHMENT_NAME
IDSM_ITM_ATTACHMENT_COUNT
IDSM_ITM_MESSAGE_ID
IDSM_ITM_RETURN_PATH

The following constants only return information for Internet mail messages (POP).

IDSM_ITM_RAW_HEADERS
IDSM_ITM_X_MAILER
IDSM_ITM_CONTENT_LENGTH
IDSM_ITM_MESSAGE_LENGTH
IDSM_ITM_WEB_FORM_ENCODED
IDSM_ITM_WEB_FORM_FIELD
IDSM_ITM_WEB_FORM_VALUE
IDSM_ITM_BAD_RECIPIENT
IDSM_ITM_ALTERNATIVE_COUNT
IDSM_ITM_UIDL

NOTE: For IDSM_ITM_STATE, IDSM_ITM_CERTIFIED, IDSM_ITM_ATTACHMENT_COUNT, and IDSM_ITM_ALTERNATIVE_COUNT the numeric values will be returned as strings.

## Example

Display the first message in our Received Message window.

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 0
idsMail.LoadMessage
txtFrom.Text = idsMail.GetMessageItemString (IDSM_ITM_FROM)
txtTo.Text = idsMail.GetMessageItemString (IDSM_ITM_TO)
txtCc.Text = idsMail.GetMessageItemString (IDSM_ITM_CC)
txtBcc.Text = idsMail.GetMessageItemString (IDSM_ITM_BCC)
txtSubj.Text = idsMail.GetMessageItemString (IDSM_ITM_SUBJECT)
txtDate.Text = idsMail.GetMessageItemString (IDSM_ITM_DATE_SENT)
txtMsg.Text = idsMail.GetMessageItemString (IDSM_ITM_MESSAGE)
```

# GetWebFormValue (method)

## Description

This method is used to retrieve individual fields from a "mail-to" web form response message. The method returns the field as a string. The field name is passed as a parameter.

A typical web form response might appear like this:

> &FirstName=Jeanne&LastName=Holmes&Title=Quality+Data+Systems+Analyst&Company=Allied+Telesyn+International&Address1=950+Kifer+Rd.&ISODate=Recommended+for+Certification+3%2F1%2F96&Info=YES

This method decodes the data and extracts individual fields. For example, passing the method the field name "FirstName" would return the value "Jeanne". The field names are not case-sensitive.

The GetHeaderItems and GetHeaderItemString methods can be used to determine if a received message is actually a web form response. The GetMessageItems and GetMessageItemString methods can be used to get all of the field names and/or values in arrays or delimited strings.

## Parameters

*fieldName$* - The web form field name to retrieve the value for.

## Example

Retrieve and process a Web Form response.

```
idsMail.LoadMessage
firstName$ = idsMail.GetWebFormValue ("FirstName")
lastName$ = idsMail.GetWebFormValue ("LastName")
address$ = idsMail.GetWebFormValue ("Address")
```

# HeadersInMessage (property)

## Description

For Internet mail only (POP) this property specifies whether or not the raw message headers are included in the received message text (i.e. the Message property). If True (-1) the headers are included. If False (0) the headers are not included. This property has no effect on mail send operations.

Note that the raw headers can be retrieved separately via the GetMessageItemString or GetHeaderItemString methods by using the IDSM_ITM_RAW_HEADERS flag.

## Usage

Read/Write, Default = False (0)

## Example

```
idsMail.HeadersInMessage = True
idsMail.LoadMessage
```

# HWnd (property)

## Description

This property may be used to set the parent window handle for a native Send Mail dialog. (See ShowMailDialog and NativeMailDialog). If the property is not set, IDSMail will use its own internal hidden window as the parent window for the dialogs.

If this property is not set and the end user clicks off of the native send mail dialog, a "Server Busy" message may be displayed. This message is not harmful but may be distracting or unexpected. Setting this property avoids the possibility of the "Server Busy" message.

## Usage

Read/Write, Default = 0

## Example

```
idsMail.NativeMailDialog = True
idsMail.HWnd = myForm.hWnd
idsMail.ShowMailDialog
```

# LoadMessage (method)

## Description

This method loads the message pointed at by the MessageIndex property. Prior to using this method, message headers must have been loaded by executing LoadMessageHeaders.

File attachments will be automatically saved to temporary files on the local disk. The files will be stored to the directory pointed to by the TEMP environment variable, or stored in the WINDOWS directory if TEMP is not specified. To load the message without attachments, set SuppressAttachments = True.

MHS is an exception. For MHS, the message files and any attachment files will be left in their current location. The MoveMessage method can be used to move the messages and attachments to a different directory for filing.

## Parameters

{NONE}

## Example

Show all the messages in my InBox

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.LoadMessageHeaders
numMessages% = idsMail.MessageCount
For i% = 0 to numMessages% - 1
    idsMail.MessageIndex = i%
    idsMail.LoadMessage
    theMessage$ = idsMail.Message
    MsgBox theMessage$
Next i%
```

# LoadMessageHeaders (method)

## Description

This method loads the headers for all the messages in the current folder. This is the first method you must use when receiving mail. **It is only available in the Professional Edition**.

To load just the unread messages, set UnreadOnly = True (-1) prior to executing the method. To limit the number of headers loaded, set the MaximumHeaders property.

If messages have been deleted in a session, or new messages have arrived since logging in, it may be necessary to Logout then Login again to see the changes. This is especially applicable to Internet mail.

## Parameters

{NONE}

## Example

Show all the messages in my InBox

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.LoadMessageHeaders
numMessages% = idsMail.MessageCount
For i% = 0 to numMessages% - 1
    idsMail.MessageIndex = i%
    idsMail.LoadMessage
    theMessage$ = idsMail.Message
    MsgBox theMessage$
Next I%
```

# LoadMessageIdentifiers (method)

## Description

This method behaves identically to LoadMessageHeaders for all mail systems except for SMTP/POP Internet mail. With SMTP/POP, this method provides a faster way of loading the headers if only minimal information is needed about each. The only information available is the UIDL field which may be retrieved from the GetHeaderItems or GetHeaderItemString methods with the IDSM_ITM_UIDL constant.

Not all POP servers support this method. Your application should be prepared to trap the error that will occur if the method is unsupported (INET_ERR_UIDL_FAILED).

Headers returned with this method are always in the order in which they exist on the POP server. The DateSort property has no effect.

If messages have been deleted in a session, or new messages have arrived since logging in, it may be necessary to Logout then Login again to see the changes. This is especially applicable to Internet mail.

## Parameters

{NONE}

## Example

Download all the messages in my InBox

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.LoadMessageIdentifiers
numMessages% = idsMail.MessageCount
For i% = 0 to numMessages% - 1
    idsMail.MessageIndex = i%
    idsMail.LoadMessage
    theMessage$ = idsMail.Message
    MsgBox theMessage$
Next I%
```

# Login (method)

## Description

The Login method is generally not required, since IDSMail™ handles this automatically. However, the user may be explicitly logged in by executing this method. The LoginName, Password, and PostOffice property values will be used. If required and not supplied, the server will prompt for the values via a dialog.

Note that Novell GroupWise will display a login dialog even if all the parameters are provided. This login dialog can be disabled with the LoginDialog property.

## Parameters

{NONE}

## Example

```
idsMail.ObjectKey = "ABCD1234"
idsMail.LoginName = "BSmith"
idsMail.Password = "MySecret4"
idsMail.Login
```

# LoginDialog (property)

## Description

This property controls whether or not a login dialog is shown to the user, if the dialog is required. If all the necessary login parameters (LoginName, Password, & PostOffice) are provided to IDSMail™ prior to the first mail operation, a dialog is normally not displayed anyway.

Setting this property to False (0) disables all mail login dialogs. This is appropriate for an application which must operate entirely without a user interface. Setting this property to True (-1) will cause a login dialog to be posted whenever any of the login parameters are missing.

## Usage

Read/Write, Default = True (-1)

## Example

```
idsMail.LoginDialog = False
```

# LoginName (property)

## Description

The LoginName property is used as the account name/user ID when it is necessary to log in to the underlying mail system. If not provided, it will be prompted for (if required). When interfacing to Microsoft Exchange, this property may also be used to set the profile name.

Note that for Banyan VINES it is never necessary to set this property. VINES uses the default network login name.

## Usage

Read/Write

## Example

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.Message = "Dont forget the staff meeting today at 2:00 PM. "
idsMail.Subject = "Staff Meeting"
idsMail.LoginName = "BSmith"
idsMail.Password = "MySecret4"
idsMail.Send
```

# Logout (method)

## Description

The Logout method is generally not required, since IDSMail™ handles this automatically.  However, the user may be explicitly logged out by executing this method.

## Parameters

{NONE}

## Example

idsMail.Logout

# MailSystem (property)

## Description

The MailSystem property is used to specify the underlying mail system type. If not specified, the mail system type will be auto-detected, and the property will be set accordingly. MailSystem may have the following values:

IDSM_SYS_UNKNOWN
IDSM_SYS_SMTP_POP
IDSM_SYS_MAPI
IDSM_SYS_VIM
IDSM_SYS_MHS
IDSM_SYS_CSERVE**
IDSM_SYS_VINES

To determine if a particular mail system is present on the workstation, use the QueryMailSystem method.

## Usage

Read/Write, Default = IDSM_SYS_UNKNOWN

## Example

```
idsMail.AddRecipientTo "JAdams@Staff@Acme"
idsMail.Message = "Dont forget the staff meeting today at 2:00 PM. "
idsMail.Subject = "Staff Meeting"

  Make sure we send this message through Banyan VINES mail

idsMail.MailSystem = IDSM_SYS_VINES
idsMail.Send
```

# MaximumHeaders (property)

## Description

This property may be used to restrict the number of headers loaded by the LoadMessageHeaders method.

## Usage

Read/Write

## Example

Show the first 5 unread messages in my InBox

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.MaximumHeaders = 5
idsMail.LoadMessageHeaders
numMessages% = idsMail.MessageCount
For i% = 0 to numMessages% - 1
    idsMail.MessageIndex = i%
    idsMail.LoadMessage
    theMessage$ = idsMail.Message
    MsgBox theMessage$
Next i%
```

# Message (property)

## Description

The Message property defines the textual message in the body of the mail.  In some implementations, this is referred to as a "note."

## Usage

Read/Write

## Example

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.AddAttachment "C:\AUTOEXEC.BAT"
idsMail.Message = "Here is my AUTOEXEC file, do you see anything wrong? "
idsMail.Subject = "My Configuration"
idsMail.Send
```

# MessageCount (property)

## Description

MessageCount returns the number of message headers currently loaded.

## Usage

Read-only

## Example

Show the first 5 unread messages in my InBox

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.MaximumHeaders = 5
idsMail.LoadMessageHeaders
numMessages% = idsMail.MessageCount
For i% = 0 to numMessages% - 1
    idsMail.MessageIndex = i%
    idsMail.LoadMessage
    theMessage$ = idsMail.Message
    MsgBox theMessage$
Next i%
```

# MessageIndex (property)

## Description

MessageIndex is used to designate the message header to be loaded or queried. This property is reset to 0 automatically after executing the LoadMessageHeaders method. Note that **the first message is at index= 0**.

## Usage

Read/Write

## Example

Show the first 5 unread messages in my InBox

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.MaximumHeaders = 5
idsMail.LoadMessageHeaders
numMessages% = idsMail.MessageCount
For i% = 0 to numMessages% - 1
    idsMail.MessageIndex = i%
    idsMail.LoadMessage
    theMessage$ = idsMail.Message
    MsgBox theMessage$
Next I%
```

# MessageType (property)

## Description

The MessageType property is available for MAPI only, and is ignored by other mail systems. It allows MAPI messages other than the normal interpersonal types to be sent and received.

On send, the MessageType is used for the sent message. On receive, it filters the message headers received by the LoadMessageHeaders method. Only messages of that type will be loaded. If the message type is not defined for the workstation, an error will occur on both send and receive.

Usage of the property requires a thorough understanding of the MAPI mail system via resources such as the Microsoft Mail SDK Technical Reference. Non-Microsoft mail systems may not support this functionality. IDS cannot provide technical support in this area.

## Usage

Read/Write, default is an empty string which specifies normal interpersonal MAPI messages.

## Example

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.MessageType = "IPM.Microsoft Schedule.MtgReq"
idsMail.Send
```

# MoveMessage (method)

## Description

This method moves the message pointed at by the MessageIndex property from its current folder to the specified folder.

Note that the concept of folders is absent from MAPI and Internet mail (SMTP/POP). For these mail systems, this method will trigger an error (IDSM_ERR_NOT_SUPPORTED).

## Parameters

*folderTo$* - The folder to move the message to. Under MHS, directories are used for folders. If set, Folder must represent a valid DOS directory, including path. (Ex: C:\MYMAIL\URGENT). If not set, the default user inbox directory (in MHS directory structure) will be used.

## Example

Move the second message in this folder

```
idsMail.LoadMessageHeaders
idsMail.MessageIndex = 1
idsMail.MoveMessage "OldStuff"
```

# NameResolution (property)

## Description

This property is used during mail send with MAPI only.

If True (-1), the name of each recipient is resolved. If the name exists and is unique, the send process proceeds normally. If the name is ambiguous, a dialog is displayed allowing the user to choose from similar names. If the name does not exist and is not similar to any in the mail system database, an error is triggered.

As an example, assume that the mail system contains entries for Bob Jones and Bob Smith. Setting the mail recipient to just Bob will require resolution by the user.

As a second example, assume that there is a user Zana Williams in the system, and Zana is the only user whose name (first or last) begins with the letter Z. If NameResolution=True, a mail message can be successfully addressed to Z. No dialog will be displayed since that is enough information to uniquely identify Zana Williams in the mail system.

When the name resolution dialog is displayed, the user may choose Cancel. An error will only be triggered if DialogErrorOnCancel =True.

## Usage

Read/Write, Default = False (0)

## Example

```
idsMail.MailSystem = IDSM_SYS_MAPI
idsMail.NameResolution = True
```

# NativeMailDialog (property)

## Description

This property is used in conjunction with the ShowMailDialog method. Some mail systems provide a standard (native) Send Mail dialog. If NativeMailDialog is True (-1), the native dialog will be used if available. If a native dialog is not available (or NativeMailDialog is False), the standard IDSMail™ dialog will be used instead.

Note that the DialogCaption property is ignored if a native dialog is used. If a consistent user interface is desired across systems, NativeMailDialog should be set to False (0). See also the HWnd property.

## Usage

Read/Write, Default = False (0)

## Example

Show a mail send dialog using the standard MAPI Send Mail dialog.

```
idsMail.MailSystem = IDSM_SYS_MAPI
idsMail.NativeMailDialog = True
idsMail.HWnd = myForm.hWnd
idsMail.ShowMailDialog
```

# NewMessage (method)

## Description

This method clears the current message, resetting the object for a new message.  After sending or receiving (loading) a message, the previous recipients, attachments, etc. are maintained until this method is executed.

## Parameters

{NONE}

## Example

idsMail.NewMessage

# ObjectKey (property)

## Description

The ObjectKey is a security mechanism which prevents unauthorized client applications from sending mail via the server. The object key string must be set for each object prior to executing methods such as Send, ShowMailDialog, or LoadMessageHeaders.

IDS provides a unique object key with each licensed copy of the IDSMail™ server. You may **NEVER** distribute your Object Key or otherwise make it available to other persons or entities.

## Usage

Write only

## Example

```
Dim idsMail As New IDSMailInterface.Server
idsMail.ObjectKey = "P8S62HIE4M"
idsMail.AddRecipientTo "Jim Adams"
idsMail.Message = "Dont forget the staff meeting today at 2:00 PM. "
idsMail.Subject = "Staff Meeting"
idsMail.Send
```

# Password (property)

## Description

Specifies the mail login password. Login occurs automatically when the first mail message is sent. If a valid mail session already exists, it will be used and the password is ignored. If a new session must be established, the password may be required.

If the password is required but not provided, it will be prompted for. Note that for Banyan VINES it is never necessary to set this property. VINES integrates the mail system login with the network login.

## Usage

Read-only

## Example

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.Message = "Dont forget the staff meeting today at 2:00 PM. "
idsMail.Subject = "Staff Meeting"
idsMail.LoginName = "BSmith"
idsMail.Password = "MySecret4"
idsMail.Send
```

# PeekOnly (property)

## Description

Normally messages are automatically marked as read whenever the LoadMessage method is executed. Setting this property to True (-1) will leave unread messages in their current state, even after reading them via IDSMail™.

Note that Banyan VINES and Internet mail (POP) always mark messages as read when they are first accessed. This property will be ignored for those mail systems.

## Usage

Read/Write, default = False (0)

## Example

Examine my first unread message without marking it as read

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.SuppressAttachments = True
idsMail.PeekOnly = True
idsMail.MaximumHeaders = 1
idsMail.LoadMessageHeaders
idsMail.LoadMessage
theMessage$ = idsMail.Message
MsgBox theMessage$
```

# POPPort (property)

## Description

The Internet server port that is designated for POP mail service. Normal convention is to use port 110 for POP mail, which is the default value of the property.

## Usage

Read/Write, Default = 110

## Example

idsMail.POPPort = 95

# PostOffice (property)

## Description

The PostOffice property refers to the users mail server and is only required by VIM (Lotus mail systems) and Internet mail (SMTP/POP).

For VIM, the PostOffice is only required for cc:Mail and should be the path to the post office, not the post office name.  For Notes, the PostOffice property is not required but will be used if specified.

For Internet mail receive, this property refers to the POP mail server and can be designated as a domain name (Ex:. "acme.com") or as an IP address (Ex: "192.56.204.78").  On Internet mail send (SMTP), the PostOffice property is also used to designate the SMTP server if the SMTPServer property is empty.

## Usage

Read/Write

## Examples

VIM Example (cc:Mail)

```
idsMail.LoginName = "BSmith"
idsMail.Password = "MySecret4"
idsMail.PostOffice = "M:\CCMAIL"
idsMail.AddRecipientTo "Jim Adams"
idsMail.Message = "Dont forget the staff meeting today at 2:00 PM. "
idsMail.Subject = "Staff Meeting"
idsMail.Send
```

Internet Mail Example

```
idsMail.LoginName = "BSmith"
idsMail.Password = "MySecret4"
idsMail.PostOffice = "mycompany.com"
idsMail.LoadMessageHeaders
```

# ProgressBar (property)

## Description

This property is used for Internet mail only and specifies when and how the progress bar is displayed. The property value is a combination of flag values which are added together (logical OR) to form the desired encoding method.

The default value of the property specifies that the progress bar should be displayed on both send and receive, with a cancel button. Display of the progress bar is also dependent on the value of the ProgressBarThreshold property.

The following flag values are possible for this property:

IDSM_PROGBAR_NONE
IDSM_PROGBAR_SEND
IDSM_PROGBAR_RECEIVE
IDSM_PROGBAR_CANCEL_BTN

## Usage

Read/Write, Default = IDSM_PROGBAR_SEND + IDSM_PROGBAR_RECEIVE + IDSM_PROGBAR_CANCEL_BTN

## Example

Show a progress bar on send and receive with messages greater than 5k in size. Dont show a cancel button.

```
idsMail.ProgressBar = IDSM_PROGBAR_SEND + IDSM_PROGBAR_RECEIVE

idsMail.ProgressBarThreshold = 5000
idsMail.LoadMessage
```

**IDSM_PROGBAR_NONE (value = 0)**

This flag specifies that the progress bar is never shown.

**IDSM_PROGBAR_SEND (value = 1)**

This flag specifies that the progress bar is to be shown during Internet mail send operations. Display of the progress bar is also dependent on the value of the ProgressBarThreshold property.

**IDSM_PROGBAR_RECEIVE (value = 2)**

This flag specifies that the progress bar is to be shown during Internet mail receive operations. Display of the progress bar is also dependent on the value of the ProgressBarThreshold property.

**IDSM_PROGBAR_CANCEL_BTN (value = 4)**

This flag specifies that a cancel button is shown with the progress bar. Note that a cancel button is never shown during a LoadMessage operation because the Internet mail protocols do not support cleanly aborting the download process.

# ProgressBarThreshold (property)

## Description

This property specifies how large an Internet message must be (in bytes) before a ProgressBar is displayed. This relates to the Send and LoadMessage processes and is not applicable to the LoadMessageHeaders or LoadMessageIdentifiers process. The message size is the sum of all of its parts, including the message, headers, and file attachments.

## Usage

Read/Write, Default = 10000

## Example

Only show a progress bar if the message is greater than 10k in size

```
idsMail.ProgressBar = IDSM_PROGBAR_RECEIVE + IDSM_PROGBAR_SEND
idsMail.ProgressBarThreshold = 10000
idsMail.LoadMessage
```

# QueryMailSystem (method)

## Description

This method queries the server to see if a particular mail system is installed on the workstation. Returns True (-1) if so, False (0) if not. Note that for Internet mail, IDSMail™ will only check for the presence of a WINSOCK.DLL. This does not guarantee that the user has access to a SMTP or POP mail server.

A special case usage of this method is to distinguish between Lotus Notes and Lotus cc:Mail as the underlying VIM mail system. While IDSMail does not require the application developer to distinguish between the systems, this information may still be of interest. For this special case, pass the IDSM_QUERY_NOTES_OR_CCMAIL (value = 25) constant in place of the mail system constant. The return value of the method will then be:

IDSM_ENUM_CCMAIL (value = 11) - cc:Mail is the VIM system present.
IDSM_ENUM_NOTES (value = 63) - Notes is the VIM system present.
IDSM_ENUM_NO_VIM (value = 0) - No VIM system is present.

## Parameters

*mailSystem%* - Integer specifying the type of mail system to check for. One of the following constants:

IDSM_SYS_SMTP_POP
IDSM_SYS_MAPI
IDSM_SYS_VIM
IDSM_SYS_MHS
IDSM_SYS_VINES
IDSM_SYS_CSERVE**

Special case: IDSM_QUERY_NOTES_OR_CCMAIL

## Example

```
If idsMail.QueryMailSystem(IDSM_SYS_MHS) = True Then
    MsgBox "MHS mail system is present. "
End If
```

See Also: SetNativeSystem

# RecipientDelimiter (property)

## Description

The RecipientDelimiter property designates the ASCII character that is used to separate individual recipients in a string. The default delimiter is a comma.

Though unusual, some mail system implementations may have a comma as part of the actual name or address. In these cases, the RecipientDelimiter must be changed to a semi-colon or other character that does not occur in the names or addresses.

## Usage

Read/Write, Default=comma

## Example

```
idsMail.RecipientDelimiter = ";"
idsMail.AddRecipientTo "Jim Adams; Mary Jones; Bob Engals"
idsMail.Message = "Dont forget the staff meeting today at 2:00 PM. "
idsMail.Subject = "Staff Meeting"
idsMail.Send
```

# ResolveName (method)

## Description

This method is available for MAPI and VIM only.

The possibly ambiguous name is passed to the method. If the name exists and is unique, the name is returned immediately. If the name is ambiguous (i.e. there are one or more names that can match) the following happens, depending on which mail system is in use:

> MAPI - A dialog is displayed allowing the user to choose from similar names. The chosen name is returned.
>
> VIM - The first matching name that is found is returned.

If the name does not exist and is not similar to any names in the mail system database, an empty string is returned.

As an example, assume that the mail system contains entries for Bob Jones and Bob Smith. Passing just the first name Bob will allow the user to choose between Bob Jones and Bob Smith (for MAPI). For VIM, either Bob Jones or Bob Smith will be returned. The VIM address book does not guarantee any particular order of search.

As a second example, assume that there is a user Zana Williams in the system, and Zana is the only user whose name (first or last) begins with the letter Z. Passing just the letter Z will result in Zana Williams being returned since there is enough information to uniquely identify Zana Williams in the mail system.

When the name resolution dialog is displayed, the user may choose Cancel. An error will only be triggered if DialogErrorOnCancel =True (-1). Otherwise, an empty string is returned.

See Also:

> ResolveNameEx
>
> AddressBook Object Overview

## Parameters

*initialName$* - The possibly ambiguous name to be resolved by the method.

## Example

```
initialName$ = "Bob"
realName$ = idsMail.ResolveName(initialName$)
If realName$ = "" then
    MsgBox "Bob could not be successfully resolved."
End If
```

# ResolveNameEx (method)

## Description

This method is available for MAPI and VIM only. It is similar to the ResolveName method, except that the resolved address is returned in angle brackets along with the name. This string may then be passed to one of the AddRecipient methods, thereby guaranteeing a unique addressee. An example of the returned string is:

Bob Jones <MS:ACME/BJONES/BJONES>

See Also:

ResolveName

AddressBook Object Overview

## Parameters

*initialName$* - The possibly ambiguous name to be resolved by the method.

## Example

```
initialName$ = "Bob"
realNameAddress$ = idsMail.ResolveNameEx(initialName$)
If realNameAddress$ = "" then
    MsgBox "Bob could not be successfully resolved."
End If
```

# Send (method)

## Description

This method sends the current mail message. Login occurs automatically. If a valid mail session already exists, it will be used.

For Internet mail only, the message will be sent to any valid recipients. If there were invalid recipients, the GetMessageItemCount, GetMessageItems, and GetMessageItemString methods can be used to retrieve information about them.

## Parameters

{NONE}

## Example

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.AddAttachment "C:\AUTOEXEC.BAT"
idsMail.Message = "Here is my AUTOEXEC file, do you see anything wrong?"
idsMail.Subject = "My Configuration"
idsMail.Send
```

# ServerPath (property)

## Description

The ServerPath property returns the directory that the IDSMail™ OLE Server is installed in.

## Usage

Read-only

## Example

```
theDirectory$ = idsMail.ServerPath
```

# ServerVersion (property)

## Description

The ServerVersion property returns the version number of the IDSMail™ OLE Server.

## Usage

Read-only

## Example

```
theVersion& = idsMail.ServerVersion
```

# SessionHandle (property)

## Description

This property returns the session handle from the underlying mail system.  It may then be used in advanced applications which need to make direct calls to the mail system DLLs.

The session handle is not valid in MHS or SMTP/POP, since these mail systems do not use a DLL interface.

## Usage

Read-only

## Example

```
hSession& = idsMail.SessionHandle
```

# SetNativeSystem (method)

## Description

This method causes the OLE Server to search the workstation for mail systems. When one is found, the MailSystem property is set to the appropriate value. It is executed automatically if a mail message is sent when the underlying mail system type is not known. The search sequence is:

VINES
VIM
MHS
MAPI
SMTP/POP

## Parameters

{NONE}

## Example

```
idsMail.SetNativeSystem
Select Case idsMail.MailSystem
    Case IDSM_SYS_MAPI
        MsgBox "MAPI is default mail system.
    Case IDSM_SYS_VIM
        MsgBox "VIM is default mail system."
    Case IDSM_SYS_MHS
        MsgBox "MHS is default mail system."
    Case IDSM_SYS_VINES
        MsgBox "VINES is default mail system."
    Case IDSM_SYS_SMTP_POP
        MsgBox "Internet is default mail system."
    Case IDSM_SYS_CSERVE**
        MsgBox "Compuserve is default mail system."
Case Else
        MsgBox "No mail system exists!!"
End Select
```

# ShowMailDialog (method)

## Description

The method launches a Send Mail dialog. The dialog can either be the standard IDSMail™ dialog (default) or the native mail system dialog if NativeMailDialog = True (if the mail system does not provide a dialog, the standard dialog will be used instead).

Mail items specified prior to showing the dialog will be preloaded, giving the user the opportunity to alter or augment the message prior to sending. Preloading may not be supported by some native dialogs, however.

If the default standard dialog is used, the caption may be specified in the DialogCaption property.

After launching the modal dialog, the user is in control of the message and can choose to send it or cancel the action. See also the HWnd property.

## Parameters

{NONE}

## Example

Preload a file attachment and addressee, then
let the user edit the message as required

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.AddAttachment "C:\AUTOEXEC.BAT"
idsMail.NativeMailDialog = True
idsMail.HWnd = myForm.hWnd
idsMail.DialogCaption = "File Delivery Message"
idsMail.Message = " --- Edit this text ---"
idsMail.ShowMailDialog
```

# ShowMailSystemDialog (method)

## Description

This method displays a mail system dialog allowing the user to select which mail system to use. The selected value may be read back via the MailSystem property. The initial value of the dialog will be the MailSystem property value prior to the call.

## Parameters

{NONE}

## Example

Let the user select a mail system (default = VIM)

```
idsMail.MailSystem = IDSM_SYS_VIM
idsMail.ShowMailSystemDialog
```

See what they chose

```
systemSelected% = idsMail.MailSystem
```

# SMTPPort (property)

## Description

The Internet server port that is designated for SMTP mail service.  Normal convention is to use port 25 for SMTP mail, which is the default value of the property.

## Usage

Read/Write, Default = 25

## Example

idsMail.SMTPPort = 30

# SMTPServer (property)

## Description

The SMTPServer property refers to the Internet server that is used by the SMTP protocol for mail send. It is required only for Internet mail send. If the property is not specified, IDSMail™ will attempt to use the PostOffice property instead. If neither are specified, a login dialog will be shown that will prompt for the SMTPServer.

The property can be designated as a domain name (Ex:. "acme.com") or as an IP address (Ex: "192.56.204.78").

Note that in general, a client may use different Internet servers for mail send (SMTP) and mail receive (POP). Many times the same domain name or address is used for both. In this special case, it is only necessary to specify the PostOffice property.

## Usage

Read/Write

## Example

```
idsMail.LoginName = "BSmith"
idsMail.SMTPServer = "mycompany.com"
idsMail.NewMessage
idsMail.AddRecipientTo "The President <pres@whitehouse.gov>"
idsMail.Subject = "Seasons Greetings"
idsMail.Message = "Best of the season to you and the first lady. "
idsMail.Send
```

# Subject (property)

## Description

Specifies the subject of the mail message.  Some mail systems limit the length allowed.  If the subject length exceeds the maximum, it will be truncated.

## Usage

Read/Write

## Example

```
idsMail.AddRecipientTo "Jim Adams"
idsMail.AddAttachment "C:\AUTOEXEC.BAT"
idsMail.Message = "Here is my AUTOEXEC file, do you see anything wrong? "
idsMail.Subject = "My Configuration"
idsMail.Send
```

# SuppressAlternatives (property)

## Description

This property applies to Internet mail only. Setting this property to True (-1) will cause IDSMail™ to NOT save MIME alternative text parts to disk when the LoadMessage method is executed. To later retrieve the MIME alternative text parts, set this property to False (0) and re-execute the LoadMessage method.

Please read this overview of MIME Alternative Text Parts if you have not already.

## Usage

Read/Write, Default = False (0)

## Example

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.SuppressAlternatives = True
idsMail.MaximumHeaders = 1
idsMail.LoadMessageHeaders
idsMail.LoadMessage
theMessage$ = idsMail.Message
MsgBox theMessage$
```

# SuppressAttachments (property)

## Description

Setting this property to True (-1) will cause IDSMail™ to NOT save attached files to disk when the LoadMessage method is executed. This enables the method to complete faster, and saves disk space. To later retrieve the attached files, set this property to False and re-execute the LoadMessage method.

Note that this property is ignored during Internet mail receive as the POP protocol does not support this.

## Usage

Read/Write, Default = False (0)

## Example

Examine my first unread message without marking it as read

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.SuppressAttachments = True
idsMail.PeekOnly = True
idsMail.MaximumHeaders = 1
idsMail.LoadMessageHeaders
idsMail.LoadMessage
theMessage$ = idsMail.Message
MsgBox theMessage$
```

# TimeoutConnect (property)

## Description

This property specifies the timeout in milliseconds to be used for SMTP or POP server connections. The connection period encompasses the time from the beginning of the first mail event until connection is established. In dial-up connections this includes the time spent dialing the number, waiting for modem negotiation/connection, etc.

The default value of 30000 (30 seconds) is appropriate for most situations. If a dedicated Internet connection is available, the timeout can be safely reduced. The minimum and maximum values for the property are 2000 and 60000 respectively.

## Usage

Read/Write, Default = 30000 (30 seconds) Minimum = 2000 (2 seconds) Maximum = 60000 (60 seconds)

## Example

Set connection timeout to 10 seconds

```
idsMail.TimeoutConnect = 10000
idsMail.TimeoutProtocol = 5000
idsMail.PostOffice = "provider.com"
idsMail.LoginName = "myself"
idsMail.Password = "TopSecret"
idsMail.LoadMessageHeaders
```

# TimeoutProtocol (property)

## Description

This property specifies the timeout in milliseconds to be used in SMTP or POP operations as a "watchdog" timer.  If the mail server does not respond to commands within this period, the applicable IDSMail™ method will exit with a timeout error.

The default value of 10000 (10 seconds) is appropriate for most situations.

## Usage

Read/Write, Default = 30000 (30 seconds)  Minimum = 2000 (2 seconds) Maximum = 60000 (60 seconds)

## Example

Set protocol timeout to 5 seconds

```
idsMail.TimeoutConnect = 10000
idsMail.TimeoutProtocol = 5000
idsMail.PostOffice = "provider.com"
idsMail.LoginName = "myself"
idsMail.Password = "TopSecret"
idsMail.LoadMessageHeaders
```

# UnreadCount (property)

## Description

Reading this property causes the underlying mail system to be queried for the unread message count, which is then returned. Since reading this property is actually triggering a call to the underlying mail system, the result will likely not be available immediately.

Depending on the mail system implementation, this property may not be suitable for constant polling. If performance problems arise, use a timer control to read this property less frequently.
**This property is only available in the Professional Edition**.

Note that for Internet mail the result is derived from loading all of the available message headers, then counting those that are unread. So, this method provides no efficiency savings as compared to the LoadMessageHeaders method.

## Usage

Read-only

## Example

Get the number of unread messages

```
idsMail.ObjectKey = "ABCDE12345"
uCount% = idsMail.UnreadCount
If uCount% > 0 then
    msg$ = You have  + Str$(uCount%) + unread messages.
End If
```

# UnreadOnly (property)

## Description

Setting this property to True (-1) will cause LoadMessageHeaders to only load unread messages.

Note that for Internet mail, the POP server cannot be instructed to provide only unread messages. IDSMail™ implements this property by downloading all headers first, then filtering out the ones that are already read. Therefore, this property will have a negligible effect on the processing time of the LoadMessageHeaders method.

## Usage

Read/Write, Default = False (0)

## Example

Examine my first unread message without marking it as read

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UnreadOnly = True
idsMail.SuppressAttachments = True
idsMail.PeekOnly = True
idsMail.MaximumHeaders = 1
idsMail.LoadMessageHeaders
idsMail.LoadMessage
theMessage$ = idsMail.Message
MsgBox theMessage$
```

# UTCOffset (property)

## Description

This single-precision floating point number is used for Internet mail send only. It allows the time offset in hours from UTC (GMT) to be specified for transmitted messages. The property value can range from -24 to +24, or have the special value of IDSM_NO_UTC_OFFSET (-50) which causes the UTC offset to be omitted from transmitted messages. This is also the default value/behavior.

Here are some examples for various time zones:

| | |
|---|---|
| US Pacific Standard Time | -7.0 |
| US Pacific Daylight Savings Time | -8.0 |
| US Eastern Standard Time | -4.0 |
| Greenwich, England | 0.0 |

Note that a value of 0 does not disable the offset. Note also that the NewMessage method does not clear this value. It remains set and applies to all composed mail messages until changed.

## Usage

Read/Write, Default = IDSM_NO_UTC_OFFSET (-50)

## Example

Set the main parameters for Internet mail send

```
idsMail.ObjectKey = "ABCDE12345"
idsMail.UTCOffset = -6.0
idsMail.SMTPServer = "myprovider.com"
```

# XMailer (property)

## Description

This property is used only with Internet mail send. It specifies the "X-Mailer" header field that is often used to designate the type and version of the mail client application that sent the message. To eliminate this header from the sent message, specify the property value as an empty string. Other X headers may be added by using the AddXHeader method.

## Usage

Read/Write, Default = "IDSMail vN.nn" where N.nn designates the IDSMail™ version number.

## Example

```
idsMail.LoginName = "BSmith"
idsMail.SMTPServer = "mycompany.com"
idsMail.NewMessage
idsMail.AddRecipientTo "The President <pres@whitehouse.gov>"
idsMail.Subject = "Seasons Greetings"
idsMail.Message = "Best of the season to you and the first lady. "
idsMail.XMailer = "Super Mail v2.0"
idsMail.Send
```

# Technical Topics

**VB3/MS Office 16-bit Considerations**
Minor problems and workarounds for VB3 programming.

**Frequently Asked Questions**
Common questions about IDSMail™ operation.

**Performance Considerations**
How to get the best performance from your mail-enabled application.

**Implementing truly *Universal* E-mail**
How to avoid minor incompatibilities between E-mail systems.

**Distributing your Applications**
How to distribute applications that utilize the IDSMail™ OLE Server.

**Error Reporting**
Trapping and interpreting server-generated errors.

**Internet Mail Special Topics**

**Developing & Debugging Internet Mail Applications**

**MIME Alternative Text Parts**

**Using X-Headers to your Advantage**

# Developing & Debugging Internet Mail Applications

Developing Internet applications can sometimes mean working via a dial-up network connection that is less reliable than a local LAN connection. Here are some techniques that can avoid or remedy problems when an unexpected loss of connection occurs.

## Connection Timeout

After successfully performing a mail operation the first time, you may wish to set the TimeoutConnect property to its minimum value. If you suddenly lose your connection, IDSMail™ will eventually timeout. If you then execute another method, IDSMail™ will attempt to reconnect and login again. Having a short timeout will allow you to regain control from this failed method in the shortest amount of time.

## Limited POP Server Sessions

Many POP servers will return an error during login if there are already sessions established with the same login name. Normally this is not a problem, and is even desirable in some applications.

During development work, this can be a nuisance. If you unexpectedly lose your Internet connection, or your application suddenly terminates or crashes, your POP session may still be running on the server. When you try to login again, the POP server may not allow it. In this situation IDSMail™ will return the error INET_ERR_MAILDROP_BUSY with a message similar to "The maildrop is busy (another client logged in?)."

If IDSMail™ terminates normally (i.e. you free all references to it) it will automatically logout of all active mail sessions. However, it is good programming practice to explicitly execute the Logout method.

Should you receive the INET_ERR_MAILDROP_BUSY error, you may need to manually kill the mail session running on the POP server. Your network administrator can provide guidance on how to do this, as the technique varies widely depending on the system. Some POP servers will kill the sessions automatically after a certain period of inactivity.

Here is a technique that we have used successfully on one remote server, though there is no guarantee that it will work for yours.

## Corrupted SMTP Message

Should a disconnection or system crash occur during the sending of an Internet mail message, it may become corrupted and unreadable by POP mail clients. This is often evidenced by the LoadMessageHeader process appearing to hang when the POP mailbox is accessed.

The solution is to delete the mail message using a non-POP mail client (i.e. a mail client running on the server). These are often available via a UNIX shell login, or via Telnet. One of the most common is Pine which can normally be accessed by typing "pine" at the UNIX prompt.

Using this remote mail client, you then identify and delete the offending message.

# MIME Alternative Text Parts

## Introduction

MIME-compliant messages in Internet mail can include special file attachments that represent alternative formats for the main message text. For example, a sophisticated mail program may use Rich Text Format (RTF) or HTML to represent the message text, thereby allowing for fonts, colors, indentations, and other advanced text formatting to be used.

But not all mail programs can interpret these formats. It is desirable to be able to send both formatted text and a plain (unformatted) text version of the same message. In this way, the mail message will be readable by all recipients. Sophisticated mail programs can choose to use the formatted text representation, while more basic mail programs can use the unformatted text and ignore the formatted version.

## Sending Alternative Text Parts

IDSMail handles these alternative text parts adeptly on both send and receive. On send, the plain unformatted text is passed to the Message property. The formatted text versions of the message are added via the AddAlternative method.

## Receiving Alternative Text Parts

On receive, IDSMail includes the alternative text parts in the file attachments list that is obtainable via the GetMessageItems or GetMessageItemString methods. The alternative text files will have extensions that indicate the type of formatted text contained in each. The following extensions are used:

| | |
|---|---|
| .RTF | Rich Text Format |
| .HTM | HTML |
| .TXT | Text |
| .ENR | Enriched text |
| .DAT | Unknown alternative type |

The GetMessageItemCount method can be used to determine the count of all the attached files (including alternative text attachments) by passing the constant IDSM_ITM_ATTACHMENT_COUNT. The count of the alternative text parts can be determined by passing the constant IDSM_ITM_ALTERNATIVE_COUNT. Note that the number of "normal" file attachments is the difference between these two counts.

If you wish to ignore the received alternative text parts, the SuppressAlternatives property can be set to True (-1). The alternative text attachments will then be automatically deleted and will not appear in the received attachments list.

# Killing POP Server Processes

If you receive the INET_ERR_MAILDROP_BUSY error with a message similar to "The maildrop is busy (another client logged in?)" you may need to manually kill the your mail process on the POP server. The technique below works for us, but **is not guaranteed to work with your system**. Consult your system administrator for more information.

1) Telnet to the machine that the POP mail server software is running on.

2) Issue the following command:

> **ps<SP>-ef<SP>|<SP>grep<SP>username**
>
> <SP> = Space character
> username = your user name

3) A list of the running processes will appear as shown in the example below.

```
chou 18770   3921   2 07:20:19 pts/67    0:00 grep username
username 12217    166 18 01:19:32 ?          0:00 popper
```

4) Look for any processes that contain the word "popper". You must kill each of them with a command like this:

> **kill<SP>-9<SP>12217**
>
> In this case 12217 is the process ID of the process we are killing, and is shown in the above process list.

# VB3/MS Office 16-bit Considerations

Because VB3 and 16-bit Microsoft Office applications support an older version of OLE, there are a few considerations that the programmer must be aware of. These are described below.

## OLE Server Startup Error

VB3 may generate an error during startup of the IDSMail™ OLE server. This usually happens the first time OLE is utilized during a session, and does not occur after that.

IDS has provided a helper function to avoid this error. The function can be found in the IDSMINIT.BAS file that was installed with IDSMail™ and is called StartIDSMailServer. Call this function in place of the following line:

```
Set idsMail = CreateObject("IDSMailInterface.Server")
```

Note that the object variable idsMail (or your equivalent) must be declared globally, since VB3 cannot set an uninitialized object variable that is passed as a parameter. For convenience, here is a listing of the function.

## Incompatible Methods

The following methods are incompatible with VB3 and MS Office 16-bit applications. Use the alternative methods instead.

| VB3 Incompatible Method | Use This Method Instead |
|---|---|
| GetHeaderItems | GetHeaderItemString |
| GetMessageItems | GetMessageItemString |
| AsynchAbort | <NO EQUIVALENT> |
| AsynchObjectRegister | <NO EQUIVALENT> |
| AsynchObjectUnregister | <NO EQUIVALENT> |
| **AddressBook Object Methods** | |
| GetAddressBookItems | Enumeration methods/properties such as MoveFirst, MoveNext, Name, etc. |
| GetDialogResultItems | GetDialogResultString |

## AddressBook Syntax Differences

VB3/Access 2.0 Syntax Note

# StartIDSMailServer Function

**DLL Declaration**

Declare Function GetTickCountIDS& Lib "User" Alias "GetTickCount" ()

**Function Listing**

```
Function StartIDSMailServer% ()

'    VB3 can be slow to load OLE DLL's, causing
'    the initial creation of OLE servers to fail.
'    This routine handles these errors by trying
'    to load the server again after a delay.

'    Note that idsMail must be declared as a
'    module-level or global variable (VB doesn't
'    let you pass object variables to be set in
'    a function).

On Error GoTo StartIDSMailServerHandler

Set idsMail = CreateObject("IDSMailInterface.Server")
StartIDSMailServer% = True

StartIDSMailServerExit:
    Exit Function

StartIDSMailServerHandler:
    errCount% = errCount% + 1
    If errCount% < 5 Then
        startTick& = GetTickCountIDS&()
        Do
            DoEvents
            deltaTicks& = GetTickCountIDS&() - startTick&
        Loop Until deltaTicks& > 500
        Resume
    Else
        StartIDSMailServer% = Err
        Resume StartIDSMailServerExit
    End If

End Function
```

# Frequently Asked Questions

# FAQ: VIM Autodetection/Operation

**Path Statement**

Lotus breaks Windows conventions by installing their DLL files in private directories, rather than in the WINDOWS\SYSTEM directory. This means that a path must exist to the private directory where VIM.DLL is located.

During cc:Mail installation, the private directory (usually CCMAIL directory) is added to the path statement. However, user reconfiguration of the workstation may have eliminated this path. In these cases, the path needs to be restored. IDSMail™ will then be able to autodetect and work with VIM.

This also explains why in some installations, IDSMail™ works fine while cc:Mail is running, but fails if cc:Mail is not running. While cc:Mail is running, the needed DLLs are already in memory so a path search is not required.

**32-bit Notes Client**

If only the 32-bit Notes client is installed, IDSMail will not be able to interface through VIM. The solution is to also install the Notes 16-bit client and assure that a path to that directory exists. This will install the 16-bit VIM files required by IDSMail.

Note that after installation, the majority of the 16-bit Notes client can be deleted if disk space is an issue. If you choose to do this, be sure to leave all of the Lotus DLLs in place.

**Multiple Copies of VIM.DLL**

If you have developed with VIM before, you may have multiple copies of the DLL on your system. This is rarely the case on client machines since the Lotus installation programs generally assure that only one copy exists.

You can use File Manager or Explorer to search for all copies of VIM.DLL. If you have multiple copies, delete all of them except for the one in the same directory as the cc:Mail or Notes client program. Lotus specifically cautions against having an extra copy of VIM.DLL in the WINDOWS or WINDOWS\SYSTEM directory, since unpredictable results can occur.

# FAQ: Exchange Profile

The Exchange Profile prompt can be avoided by specifying the users profile name in the LoginName property prior to logging in.

# FAQ: Undefined Dynalink Error

During the first Internet mail operation, a message box posted by the operating system which states "Undefined Dynalink" indicates that the WINSOCK.DLL that was loaded for Internet communications is not compatible with IDSMail. Different versions of this DLL are produced by a variety of companies and they do not all comply with the specification for the library.

If this error occurs, you should use File Manager to search your drive to find all the copies of this library. Adjust your path statement or physically move the libraries into the WINDOWS\SYSTEM directory one at a time until you can find one that will work correctly. It is necessary to restart Windows after each of these reconfigurations to assure that the previous copy of WINSOCK.DLL is unloaded from memory.

If you are familiar with the WPS.EXE program (shipped with VB3 and was installed in the VB\CDK directory, or on the VB4 CD-ROM under TOOLS\PSS) you can expedite this process. WPS will show you which DLL libraries are in memory, and where they loaded from. You can kill an undesired library instantly without having to restart Windows.

IDSMail was tested with a variety of 16-bit WINSOCK.DLL libraries including those by Trumpet, Netscape, Mosaic, and the standard Microsoft 16-bit WINSOCK.DLL on WindowsNT 3.51/4.0 and Windows95 (this library thunks to the 32-bit TCP/IP layer). Because of the wide variety of alternative Winsock libraries available, there may be some that are incompatible.

# FAQ: VB4 "Class Doesnt Support OLE Automation"

When instantiating IDSMail in VB4, this error indicates one of the following:

1) You have not set a reference to IDSMail in your project. Select Tools/References from the VB4 menu and check the box for IDSMail OLE Server.

2) IDSMail is not properly registered on your workstation. Execute IDSMail from the command line with the "/REGSERVER" command switch. For example:

```
c:\windows\system\olesvrs\idsmail.exe /REGSERVER
```

{Note that there is a space preceding the "/REGSERVER" command switch}

Another option is to change how IDSMail is instantiated. Replace this line:

```
Dim idsMail as New IDSMailInterface.Server
```

With:

```
Dim idsMail as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
```

# FAQ: Controlling the Dialer in Windows95 and WindowsNT

Your application may need to establish a dial-up connection to an external server prior to exchanging E-mail. While NT 4.0 provides the ability to automatically establish a connection as needed, that capability was missing from NT 3.51 and earlier versions of Windows95. To control the dialer in these cases a separate component is needed to interface to the Remote Access Service (also called Dial-Up Networking in Windows95 & NT4).

The best component we know of to do this is the shareware RAS ActiveX control which may be downloaded from **http://www.intr.net/coolstf**. The sample program included with the control demonstrates how to do this.

# FAQ: Determining if VIM System is cc:Mail or Notes

While IDSMail™ does not require the application developer to distinguish between cc:Mail and Notes as VIM systems, this information may still be of interest. The QueryMailSystem method may be used for this purpose.

# Error Reporting

The IDSMail™ OLE Server will generate errors under certain conditions, just like a traditional VBX or OCX control. Handling the errors can be done in a similar manner.

If your programming environment fully supports the OLE v2.02 specification (i.e. VB4), you will be able to get three pieces of information from each error:

> **Source** - The server that generated the error, always equal to IDSM_ERR_SOURCE ("IDSMailInterface.Server") for the IDSMail™ server.
>
> **Description** - A textual description of the error.
>
> **Number** - The error code.

If your programming environment does not fully support the OLE v2.02 specification, you will be limited to the error number only (VB3 & MS Access v2.0 are examples of environments that do not fully support the specification).

The error number reported to your application is generated by adding the internal Server Error Number to the fixed constant IDSM_BASE_ERROR (&H80040000). This constant is identical to vbObjectError in VB4. To arrive at the true Server Error Number, subtract IDSM_BASE_ERROR from the error value. Example:

```
MyErrorHandler:
    If Err.Source = IDSM_ERR_SOURCE then
        serverError& = Err.Number - IDSM_BASE_ERROR
        Select Case serverError&
            Case IDSM_ERR_INVALID_PROPERTY
                MsgBox "Bad property!"
            Case Else
                MsgBox Err.Description
        End Select
    Else
        MsgBox "Error not from server."
    End If
```

The Server Error Numbers are partitioned as follows:

| | |
|---|---|
| 0-1999 | General Server Error |
| 2000 - 2999 | MAPI Error |
| 3000 - 3999 | VINES Error |
| 4000 - 4999 | VIM Error |
| 5000 - 5999 | MHS Error |
| 6000 - 6999 | SMTP/POP Error |

Although not required, this partitioning allows programmers familiar with low-level mail programming to interpret errors in the context of the underlying mail system. The file IDSM_ERR.TXT (installed in same directory as IDSMail) includes definitions of the mail system specific error codes above 2000.

Note that setting the ErrorDisplay property to True (-1) will cause IDSMail to post descriptive low-level message boxes whenever an error condition is encountered. This is useful for debugging, especially if your development environment does not support full error detail reporting. IDS recommends setting this property to False (0, the default) prior to shipping a production application.

# Server Error Numbers

NOTE: *Subtract IDSM_BASE_ERROR (&H80040000) from the returned error code to arrive at these values.*

| Num | Constant | Meaning |
|---|---|---|
| 513 | IDSM_ERR_UNDETERMINED | Undetermined error. |
| 514 | IDSM_ERR_UNDETERMINED_UI | Undetermined error resulted while the Send Mail user interface was displayed. |
| 515 | IDSM_ERR_NOMAIL | No mail system is available on the workstation. |
| 516 | IDSM_ERR_NOADDRESSEES | There are no valid addressees. At least one TO recipient is required. |
| 517 | IDSM_ERR_TOOMANY_BCC | Too many BCC recipients. |
| 518 | IDSM_ERR_TOOMANY_CC | Too many CC recipients. |
| 519 | IDSM_ERR_TOOMANY_TO | Too many TO recipients. |
| 520 | IDSM_ERR_TOOMANY_ATTACHMENTS | Too many file attachments. |
| 521 | IDSM_ERR_INVALID_ATTACHMENT | The file attachment was not found or was otherwise invalid. |
| 522 | IDSM_ERR_USER_ABORT | The user aborted the action. |
| 523 | IDSM_ERR_NO_OBJECT_KEY | No valid Object Key has been specified. |
| 524 | IDSM_ERR_BAD_OBJECT_KEY | The Object Key provided is invalid. |
| 525 | IDSM_ERR_NO_LICENSE | No valid IDSMail™ license files were found. |
| 526 | IDSM_ERR_INVALID_LICENSE | The IDSMail™ license file is invalid. |
| 527 | IDSM_ERR_EXPIRED_LICENSE | The IDSMail™ license is expired (demos only) |
| 528 | IDSM_ERR_UNAUTHORIZED | The action is unauthorized. |
| 529 | IDSM_ERR_INVALID_PROPERTY | Invalid property setting. |
| 530 | IDSM_ERR_NOT_SUPPORTED | This action is not supported by the underlying mail system. |
| 531 | IDSM_ERR_NOT_PROFESSIONAL | The Professional Edition is required for this feature. |
| 532 | IDSM_ERR_NO_HEADERS | No message headers have been loaded. |

| | | |
|---|---|---|
| 533 | IDSM_BAD_MESSAGE_INDEX | The MessageIndex property is invalid. |
| 534 | IDSM_ERR_ZIP | Error in ZIP algorithm. |
| 535 | IDSM_ERR_ENCODE | Error in encoding algorithm. |
| 536 | IDSM_ERR_ASYNCH_REGISTER | Error registering notification object. |
| 537 | IDSM_ERR_ASYNCH_UNREGISTER | Error unregistering notification object. |
| 538 | IDSM_ERR_NO_ASYNCH_OBJECT | Asynchronous methods not available until you register a notification object. |
| 539 | IDSM_ERR_ASYNCH_ABORT | Could not abort asynchronous process. |
| 540 | IDSM_ERR_ASYNCH_IN_PROGRESS | You accessed an IDSMail method or property while an asynchronous method was still in process. |
| 541 | IDSM_ERR_ASYNCH_RECURSION | You must allow the notification object method to return back to IDSMail prior to starting another operation. |
| 542 | IDSM_ERR_BAD_OBJ_TYPE | Object type not supported. |
| 543 | IDSM_ERR_BAD_OBJ_CONFIG | Object configuration parameter not supported. |
| 544 | IDSM_ERR_NO_LOGIN_INFO | No login information was provided. |
| 545 | IDSM_ERR_RESERVED_HEADER | AddXHeader method attempted to add a reserved header. |
| 546 | IDSM_ERR_NO_POST_OFFICE | The VIM post office directory does not exist. |
| 750 | IDSM_ERR_AB_NO_MAIL_OBJECT | The AddressBook object MailObject property has not been set. |
| 751 | IDSM_ERR_AB_NOT_OPEN | The address book is not open. Execute OpenAddressBook to open it. |
| 752 | IDSM_ERR_AB_NO_MAIL_SYSTEM | No mail system was detected by the AdressBook object. |
| 753 | IDSM_ERR_AB_NO_SUPPORT_SYS | The AddressBook object contains no support for this mail system. |
| 754 | IDSM_ERR_AB_NO_SUPPORT_FUNC | The AddressBook method is not supported for the current mail system |
| 755 | IDSM_ERR_AB_NO_CURRENT_REC | The AddressBook object does not have a current record. |
| 2XXX | {MAPI Error} | Subtract 2000 to arrive at the MAPI error value. Refer to MAPI documentation or IDSM_ERR.TXT for interpretation. |
| 3XXX | {VINES Error} | VINES natively reports errors in the 3XXX range. Refer to VINES programming references or IDSM_ERR.TXT for |

| | | |
|---|---|---|
| | | interpretation. |
| 4XXX | {VIM Error} | Subtract 4000 to arrive at the VIM error value. Refer to VIM documentation or IDSM_ERR.TXT for interpretation. |
| 5XXX | {MHS Error} | Subtract 5000 to arrive at the MHS error value. Refer to MHS documentation or IDSM_ERR.TXT for interpretation. |
| 6XXX | {SMTP/POP Error} | Subtract 6000 to arrive at the SMTP/POP error value. Refer to IDSM_ERR.TXT for interpretation. |

# Performance Considerations

For best performance, IDS recommends dimensioning a single global instance of the IDSMailInterface.Server object. The reference to the object should not be released until just before your application terminates.

The largest performance impact from OLE Servers occurs when they are first loaded. Maintaining a global instance of the server means that this impact is only felt once - the first time any of the properties or methods are accessed.

A secondary performance impact is felt the first time a message is sent or message headers are loaded. By design, the IDSMail™ Server does not login to the underlying mail system until this point is reached. Some lag may occur as the mail system DLLs load and a mail session is established. However, the Server continues to maintain the session and use it for subsequent operations.

You may wish to use the Login method explicitly to force session establishment at a convenient point in your application.

# The Notification Object

## Overview

Asynchronous notifications are achieved by having the client application (your application) create an object and pass it to IDSMail™. That object must have a method called AsynchNotification in it, with a precisely defined list of parameters. For an overview of how asynchronous notification works, refer to the description for the AsynchObjectRegister method.

## VB4 Users

If you are using Visual Basic v4.0 or above, you should take a look at the IDSMNTFY.CLS class module that was installed in the same directory as IDSMail™. It has a pre-defined AsynchNotification method in it. Your task is simply to "fill in the blanks".

## Other Development Environments

If you are using another development environment that can create and pass objects, the method definition below should help you to create your own notification class. Here is the method prototype:

### Long AsynchNotification(Long method, Long event, Long min, Long max, Long value, Long status, Long reserved, LPSTR msg)

Long designates a long integer. The method (function) returns a long integer, and all parameters except for the last one are long integers. The last parameter (msg) is a pointer to a null-terminated string.

Because this method will be executed across the OLE layer, your development environment may require you to accept the parameters as Variants. Refer to your development environments documentation regarding OLE Automation for explanations of this. Several language-specific tutorials are also available at the IDS Web Site.

## Using Notifications

The process for using asynchronous notifications is straight-forward:

1) Create an instance of your notification object.

2) Register it with the IDSMail AsynchObjectRegister method.

3) Execute the asynchronous method. The AsynchNotification method of your notification object will be called periodically by IDSMail™, informing it of status information (including completion). You should refrain from accessing any of the IDSMail™ properties or methods until after the event completes, or an error may result. (Exception: AsynchAbort)

4) Prior to terminating your program or freeing your object reference to IDSMail™, be sure to unregister your notification object via the AsynchObjectUnregister method.

## Notification Parameters

**method** -Specifies which IDSMail™ method the notification is related to:

IDSM_METHOD_LOAD_HEADERS - LoadMessageHeaders

| | |
|---|---|
| IDSM_METHOD_LOAD_IDENTIFIERS - | LoadMessageIdentifiers |
| IDSM_METHOD_LOAD_MESSAGE - | LoadMessage |
| IDSM_METHOD_DELETE - | Delete |
| IDSM_METHOD_SEND - | Send, ShowMailDialog |
| IDSM_METHOD_ABORT - | AsynchAbort |

**event** - The type of event that this notification is. Either a Progress update event or a Completion event.

| | |
|---|---|
| IDSM_EVENT_PROGRESS - | Progress update event. |
| IDSM_EVENT_COMPLETE - | Completion event. |

**min** - The minimum value of the progress measurement.

**max** - The maximum value of the progress measurement. Note that during loading of headers, the max value reflects the number of messages available on the POP server, even if the MaximumHeaders property was set to a lower value.

**value** - The value of this progress update, normally between the min and max values (but not guaranteed to be).

**status** - During a completion event, this will be True if completed successfully, or an error code if not. This parameter should be ignored during progress update events.

**msg** - A short textual message related to the event.

IMPORTANT: The min, max, and value parameters should not be used to infer anything about the underlying mail system. They are provided solely as relative measures of progress. The final message count, message size, etc. may be quite different from these values.

# Distributing your Applications

There are four basic ways to distribute your application. These are listed below in order of increasing complexity. Prior to choosing a distribution strategy, **please review your license agreement to see how many copies you are allowed to distribute**, and to review the other terms.

**IMPORTANT**: You may **NEVER** distribute your Object Key or otherwise make it available to other persons or entities.

## Distributing the Original Disk Set

The easiest method of them all.

## Distributing via a Network

Simple and effective internal distribution.

## Using the Visual Basic v4.0 Setup Wizard

The Wizard only requires a little bit of guidance.

## Using Other Distribution Programs

Gain flexibility with these more difficult techniques.

# Distributing the Original Disk Set

**IMPORTANT**: You may **NEVER** distribute your Object Key or otherwise make it available to other persons or entities.

The simplest way to distribute the IDSMail™ OLE Server with your application is to simply copy the disks and send them with your application. IDS permits this because the server cannot be used without the calling program providing a valid Object Key (which only you know, right?).

If you choose this option, you must do the following:

1) Distribute the disks with all files intact and not modified in any way.

2) Include the following terminology on the disk label:

**IDSMail™ OLE Server**
Copyright 1995 by Intuitive Data Solutions

# Distributing via a Network

**IMPORTANT**: You may **NEVER** distribute your Object Key or otherwise make it available to other persons or entities.

Distributing via a network is just like distributing the original disk sets. Make two subdirectories on the network called "DISK1" and "DISK2". These subdirectories should be at the same level in the directory tree. Copy the contents of the distribution floppies into these directories.

Other users may then install IDSMail™ by running SETUP.EXE from the DISK1 directory. They will not be prompted for disk 2. The setup program will find the DISK2 directory and use it automatically.

# Using the VB 4.0 Setup Wizard

**IMPORTANT**: You may **NEVER** distribute your Object Key or otherwise make it available to other persons or entities.

The Setup Wizard provided with Microsoft Visual Basic v4.0 is an effective tool for distributing the IDSMail™ OLE Server with your VB4 application. It will automatically include and install all the OLE 2.0 DLLs needed for proper operation, and will make all the required entries in the target system registry. Refer to the Visual Basic Programmers Guide for more information.

You will need to instruct the wizard to include the following non-standard files in the installation set:

Install to the WINDOWS\SYSTEM directory:

DSSOCK.VBX
DZIP.DLL
IDSMHS.DLL
IDSVIM.DLL
VBAMAPI.DLL
VBZIP.VBX
VIMVBWRP.DLL

Install to the WINDOWS directory:

{XXXX}.ID$ (license file with a randomly generated name)

NOTE: The Visual Basic 3.0 Setup Wizard will not install the server properly and should not be used.

# Using Other Distribution Programs

**IMPORTANT**: You may **NEVER** distribute your Object Key or otherwise make it available to other persons or entities.

Several other installation system toolkits are available on the market. One of the best is InstallShield by Stirling Technologies. This is the setup method used by IDS. Whichever method you choose, there are a few critical steps that you must carry out. These are listed below.

**Install All the Required Files**

Here is a listing of all the required files and where to install them.

**Register OLE2**

Register OLE2 by using REGEDIT.EXE and the registration data file OLE2.REG. Do so by shelling out of your installation program and executing a command line statement such as this: "REGEDIT /S C:\WINDOWS\SYSTEM\OLE2.REG". Here is an InstallShield function we developed for doing this.

**Self-Register IDSMAIL.EXE**

IDSMAIL.EXE can be registered by executing it with the command line argument /REGSERVER. Do so by shelling out of your installation program and executing a command line statement such as this: "C:\WINDOWS\SYSTEM\OLESVR\IDSMAIL.EXE /REGSERVER" Here is an InstallShield function we developed for doing this.

This completes the necessary steps for properly installing and registering the IDSMail™ OLE Server.

# InstallShield Script Function for OLE2 Registration

NOTE: Pass this function the filespec for the OLE2.REG file (e.g. " C:\WINDOWS\SYSTEM\OLE2.REG")

```
function Regedit(szRegFilePath)
string szFileSpec;
string szCommand;
number nResults;
begin

szFileSpec = "REGEDIT";
szCommand = " /s " + szRegFilePath;
nResults = LaunchAppAndWait (szFileSpec, szCommand, NOWAIT);
Delay(2);

end;
```

# InstallShield Script Function for OLE Server Registration

NOTE: Pass this function the filespec for the IDSMAIL.EXE file (e.g. " C:
\WINDOWS\SYSTEM\OLESVR\IDSMAIL.EXE")

```
function EXESelfReg( szFileSpec )
string szCommand;
number nResults;
begin

szCommand = " /REGSERVER";
nResults = LaunchAppAndWait (szFileSpec, szCommand, NOWAIT);
Delay(2);

end;
```

# InstallShield

Information on InstallShield is available on the World Wide Web at:

**http://www.installshield.com**

# IDSMail™ File List

The following files are required by the IDSMail™ OLE Server:

| Location | File |
|---|---|
| OLE Server Directory (recommended location is WINDOWS\SYSTEM\OLESVR) | IDSMAIL.EXE<br>IDSMAIL.HLP (optional) |
| WINDOWS | {XXXX}.ID$ (license file, random filename) |
| WINDOWS\SYSTEM | COMPOBJ.DLL<br>CTL3DV2.DLL<br>DSSOCK.VBX<br>DZIP.DLL<br>IDSMHS.DLL<br>IDSVIM.DLL<br>OC25.DLL<br>OLE2.DLL<br>OLE2.REG<br>OLE2CONV.DLL<br>OLE2DISP.DLL<br>OLE2NLS.DLL<br>OLE2PROX.DLL<br>SCP.DLL<br>STDOLE.TLB<br>STORAGE.DLL<br>TYPELIB.DLL<br>VAEN21.OLB<br>VBZIP.VBX<br>VB40016.DLL<br>VBAMAPI.DLL<br>VIMVBWRP.DLL<br>VSHARE.386 |

# Implementing Truly *Universal* E-mail

IDSMail™ is designed to make all the various E-mail interfaces (MAPI, VIM, SMTP/POP, etc.) appear to be identical to the programmer. However, there are actually significant differences between the systems. Some of these differences can affect the portability of your code. This topic outlines the differences, allowing you to incorporate fully portable, truly universal E-mail features in your product.

## General Properties

The PostOffice property is only used by VIM and Internet mail. It is ignored for other mail systems.

## Standard Edition (E-mail Send functionality)

The send properties and methods are consistent for all mail systems. The only exceptions are the functionality of the Mail Send dialogs (ShowMailDialog method), and the MAPI-only MessageType property.

MAPI and VIM each provide their own Mail Send dialog which can be accessed by setting NativeMailDialog = True prior to executing ShowMailDialog. Native dialogs are not available for Internet mail, MHS, or Banyan VINES, so the IDSMail™ default dialog is used instead.

When used with MAPI or Banyan VINES mail systems, the IDSMail™ default dialog provides address book functionality for To, cc, and bcc entries.

If a consistent user interface is important across platforms, you must use the built-in IDSMail™ Send Dialog (NativeMailDialog = False). The disadvantage of this is that the address book functionality provided by the native dialogs will not be available for mail systems other than MAPI or VINES.

**Internet-only (SMTP) Properties/Methods** - The following properties and methods are only used for Internet mail. No error will be returned for other systems, so they are safe to use for universal code. However, they will only have an effect on Internet mail systems.

AddAlternative MIME alternative message parts are only supported for Internet mail. Alternative message parts added to a message will be ignored for other mail systems.

AddXHeader X-headers are only included in Internet mail messages and are ignored for other mail systems.

AttachmentEncoding While the built-in ZIP capabilities are useful across mail systems, the encoding method (UUENCODE vs. MIME) generally applies only to Internet mail. (Tip: Let your users choose UUENCODE or MIME in a Settings menu or dialog).

From For mail receive, this property is used for all mail systems. On send, it only applies to Internet mail. (Tip: Let your users default this once via a configuration dialog.)

ProgressBar & ProgressBarThreshold Progress bars are available for Internet mail only. You may wish to disable them entirely if your users are accessing the Internet via a high speed connection.

SMTPPort Internet mail send only. Almost all Internet mail systems use the default port assignments, but you may wish to provide this to your users in a configuration dialog.

TimeoutConnect & TimeoutProtocol Decide how you wish to configure timeouts. Some experimentation may be necessary. Note that the Windows environment will still be active during

a protocol error or connection error period, but your application will be waiting for the method to return. A short timeout is desirable, assuming a reliable and predictable connection.

UTCOffset Internet mail send only. This setting is ignored for all other mail systems.

XMailer Use this field to designate messages originating from your application. It can also be used to "categorize" your messages such that on receipt you can only load the ones that you wish.

## Professional Edition (E-mail Receive functionality)

Here are the significant differences between mail systems, and techniques for writing universal code:

**Folders** - MAPI and Internet mail do not use folders. The MoveMessage method will not work with MAPI or SMTP/POP.

**Internet-only (POP) Properties/Methods** - The following properties and methods are only used for Internet mail. No error will be returned for other systems, so they are safe to use for universal code. However, they will only have an effect on Internet mail systems.

GetHeaderValueByName Header values retrieved via this method are not available for other mail systems.

HeadersInMessage Decide if you will include the raw headers in your received messages, or let the user decide via a Settings menu or dialog.

POPPort Internet mail receive only. Almost all Internet mail systems use the default port assignments, but you may wish to provide this to your users in a configuration dialog.

ProgressBar & ProgressBarThreshold Progress bars are available for Internet mail only. You may wish to disable them entirely if your users are accessing the Internet via a high speed connection.

SuppressAlternatives MIME Alternative message parts are only available for Internet mail. This property setting will be ignored for other mail systems.

TimeoutConnect & TimeoutProtocol Decide how you wish to configure timeouts. Some experimentation may be necessary. Note that the Windows environment will still be active during a protocol error or connection error period, but your application will be waiting for the method to return. A short timeout is desirable, assuming a reliable and predictable connection.

**Internet-only (SMTP/POP) Header & Message Fields** - Note that some of the fields provided by the GetHeaderItems and GetMessageItems methods are only available for Internet mail.

**Internet-only: Different "Post Offices" for Send & Receive** - In general, a user may use different Internet servers for mail send and mail receive. Your application should allow the PostOffice and SMTPServer properties to be set separately. Normally this would be a one-time setting appropriate for a Configuration dialog.

**Internet-only: No SuppressAttachments Availability** - If your application makes use of the SuppressAttachments property to ignore file attachments, note that this is ineffective with Internet mail. Any attached files will always be downloaded. Your application may need to provide a means of deleting these files after receipt.

**MAPI-only Properties/Methods** - The following properties and methods are only used by MAPI. No error will be returned for other systems, so they are safe to use for universal code. However, they will only have an effect on MAPI mail systems.

ForceDownload (Tip: Set to True for programs that receive E-mail, False for those that only send E-mail)

NameResolution (Tip: Leave at default value of False)

ResolveName (Tip: Use only if MAPI is the only mail system you expect to encounter).

MessageType (Tip: Use only if your application is restricted to Microsoft MAPI systems.)

**MHS-only - ApplicationName** - The ApplicationName property is used only by MHS during mail receive. No error will occur if set for other mail systems, but it will be ineffectual. Depending on the MHS system configuration, users receiving messages with IDSMail™ may need this property set correctly before they can "see" the messages..

**Banyan VINES/Internet No PeekOnly** - Banyan VINES and Internet mail messages are always marked as read after loading. The PeekOnly property has no effect.

**Address Book Functions** - The address book functions are not universally available for every mail system. Check the DialogAvailable and EnumerationAvailable properties to allow your application to degrade gracefully if certain functionality is not present. Refer to The AddressBook Object topic for more information.

## Asynchronous Notifications

Asynchronous notifications are only supported for Internet mail (SMTP/POP). For further information on asynchronous notifications, refer to description for the AsynchObjectRegister method and the discussion of the Notification Object.

# Using X-Headers to your Advantage in Internet Mail

## Introduction

If you have ever poked about in the headers of an Internet mail message, you may have seen one or more headers that begin with the X character. Per the Internet mail specifications (RFCs) these headers are experimental. Implementors (programmers) may create any number of them, name them anything they wish, and use them for any reason. Commonly used X-headers include X-Mailer, X-UIDL, and X-Authentication-Warning.

In this paper well look at some practical uses of custom X-headers with IDSMail including:

- Tagging messages so that you can later identify their point of origin or type of originating process
- Automatically identifying bad addresses in a mailing list
- Retrieving critical information from a message without having to wait for the entire message to be downloaded
- Including passwords and other encryption data

## Message Tagging

Lets assume that weve designed an Order Wizard application that collects order information from our customers, then transmits the information back to us as a file attached to an E-mail message. We distribute the wizard in various ways (download, magazine inserts, direct mail, etc.) and we want to track the effectiveness of the distribution and promotion.

An X-Header is an ideal mechanism for this. Here is how our Visual Basic code might look:

```
Dim idsMail as New IDSMailInterface.Server
idsMail.ObjectKey = ABC123
idsMail.mailSystem = IDSM_SYS_SMTP_POP
idsMail.SMTPServer = mail.myprovider.com
idsMail.NewMessage
idsMail.AddRecipientTo orders@myCompany.com
idsMail.Subject = Customer Order
idsMail.Message = theCustomerComments$
idsMail.AddAttachment theOrderFile$
```

**Now add the X-Header for this distribution type (download)**

```
idsMail.AddXHeader DistributionMethod, Download
```

And send the message

```
idsMail.Send
```

What will now appear in the message headers is a line like this:

**X-DistributionMethod: Download**

## Receiving Tagged Messages

Now that we have tagged orders coming in from our Wizard, lets see how we can easily identify the distribution method using IDSMail. The code below receives the orders automatically, and counts how many resulted from each type of Wizard distribution:

```
Dim idsMail as New IDSMailInterface.Server
idsMail.ObjectKey = ABC123
idsMail.mailSystem = IDSM_SYS_SMTP_POP
idsMail.PostOffice = "mail.myprovider.com"
idsMail.LoginName = "orders"
idsMail.Password = "topSecret"

'    Load the message headers and get the message count

idsMail.LoadMessageHeaders
msgCount% = idsMail.MessageCount
If msgCount% > 0 Then

        Loop through the messages and count the orders from
        each type of Wizard distribution.

    For i% = 0 To msgCount% - 1
        idsMail.MessageIndex = i%
        theHeader$ = X-DistributionMethod
        distributionMethod$ = idsMail. GetHeaderValueByName (theHeader$)
        Select case distributionMethod$
            Case Download
                downloadCount% = downloadCount% + 1
            Case Magazine
                magazineCount% = magazineCount% + 1
            Case DirectMail
                directCount% = directCount% + 1
            Case Else

          If distributionMethod$ is an empty string, then the X-DistributionMethod header was not
        present in this message.

        End Select

            Here, we would go ahead and process the attached order

            idsMail.LoadMessage

    Next i%
End If
```

One thing to note is that when we sent the message from the Wizard, we omitted the X- when we specified the header to the AddXHeader method:

```
        idsMail.AddXHeader DistributionMethod, Download
```

But when we received the order, we included the X- in the header identifier passed to the GetHeaderValueByName method:

```
        theHeader$ = X-DistributionMethod
        distributionMethod$ = idsMail. GetHeaderValueByName (theHeader$)
```

The reason is that GetHeaderValueByName is a generic method that can be used to retrieve the value of *any* header, and not just the X-headers. We must include the X- to tell IDSMail exactly which header we are interested in.

## Identify Bad Mailing List Addresses Automatically

Lets assume for this example that you are an expert stock market prognosticator, and your friends wait impatiently for your predictions each week. You use IDSMail to send each person a customized message with predictions in their personal favorite areas of the market. Fortunes are made and lost based on your whims.

But each week the chore of maintaining your database of addresses gets more daunting. People move, change their E-mail addresses, etc. and you are bombarded with a slew of bounced mail messages shortly after your transmissions. Wouldnt it be nice if those bad addresses could be removed automatically from the database? With IDSMail and X-headers, they can be.

Your table of addresses likely has a key field which uniquely identifies each record. In many cases this will be a simple numeric counter field. Using this key field you can:

- Add the addressees key field to an X-header in the message.
- Receive the bounced messages automatically using IDSMail
- Search each bounced message for the X-header and extract the key field
- Seek the matching record (address) in your table and mark it as invalid

To make extraction of the key field easy on the receive end, well form a string from the numeric record counter, with leading 0s padding the string (e.g. 00382):

```
keyString$ = Format$(recordNumber&, 00000)
```

Then, we add this string as the value of a custom X-header called RecordNumber:

```
idsMail.AddXHeader RecordNumber, keyString$
```

And send the message.

If the message is bounced by the recipients mail server, the mail server will include all of the original headers in the return message. Note that in this case, ***the original headers are in the body of the bounced message***. This means we cant use the IDSMail GetHeaderValueByName method because it only looks in the current messages header, not in the body. Well have to be a bit more clever.

We also need a way to segment bounced messages from normal messages received in our mailbox. We wouldnt want to inadvertently purge an address just because a recipient replied with a Thanks for the Tip! message. Again, a bit of cleverness is in order. Most bounced messages originate from a Mailer-Daemon, so well look for this in the From line of the message.

Here is some Visual Basic/Access Basic code that should do the trick:

```
Dim idsMail as New IDSMailInterface.Server
idsMail.ObjectKey = ABC123
idsMail.mailSystem = IDSM_SYS_SMTP_POP
idsMail.PostOffice = " mail.myprovider.com"
idsMail.LoginName = "carnac_the_magnificent"
idsMail.Password = "Johnny"
```

```
'    Load the message headers and get the message count

idsMail.LoadMessageHeaders
msgCount% = idsMail.MessageCount
If msgCount% > 0 Then

        Loop through the messages and look for any from Mailer-Daemon

    For i% = 0 To msgCount% - 1
        idsMail.MessageIndex = i%
        whoFrom$ = Ucase$(idsMail.From)
        If Instr(whoFrom$, MAILER-DAEMON) > 0 Then

            We have a bounced message.   Download it and look for our X-RecordNumber header.

            idsMail.LoadMessage
            theMessage$ = idsMail.Message
            myXHeader$ = X-RecordNumber:
            lenHeader% = Len(myXHeader$)
            xPos% = Instr(theMessage$, myXHeader$)
            If xPos% > 0 then

                The header is present.   Parse out the record number

            recStr$ = Mid$(theMessage$, xPos% + lenHeader%, 5)
            recordNumber& = Val(recStr$)

                Seek for a matching record in our table and mark it as bad
                (using MS Access syntax and assuming table already open)

            tbl.Seek =, recordNumber&
            If tbl.NoMatch = False Then
                tbl.Edit
                tbl!badAddress = True
                tbl.Update
            End If

                Delete the message from our mail server

            idsMail.DeleteMessage

        End If

    Next i%
End If
```

## Other Uses for X-Headers

X-headers can be used for a variety of other tracking, identification, and informational applications. Perhaps your IDSMail application needs to send encrypted messages with a different password each time?  The password could itself be encrypted and passed along in a X-Password header field.

If your messages are typically large and you want to minimize a users Internet connect time, you may want to add several X-headers that provide content summaries (note that you can add any number of X-headers to a message with IDSMail).  The user will see these summaries in the headers without having to download the entire message.  They can then select only those messages of interest for immediate

download.

## Conclusion

X-headers bring powerful capabilities to your IDSMail Internet mail applications. As demonstrated in this white paper, they are simple to use on both the send and receive side. For further information, review the descriptions in the IDSMail Programming Reference for these properties and methods:

- AddXHeader
- GetHeaderValueByName
- XMailer

# Address Formats

Normally just specifying a name for the recipient is all that is necessary. The mail system will automatically look up the name and find the internal address that it must use to send the mail. This is hidden from the user.

At times, you may want to specify the actual E-mail address. For example, the recipient may not exist within the local mail system, so an actual address is necessary. IDSMail™ allows addresses to be specified using the following format:

**name <address>**

Either the name, address, or both may be specified for any recipient. The only requirement is that IF an address is specified, it must be enclosed within the angle braces.

For example:

Bob Williams <72392,7321>
The Boss <boss@acme.com>
<fax:415-353-5619>
<Jim Smith at WesternPO>
Mary Frank

are all valid recipient descriptors.

In some mail systems, certain characters in the recipient descriptor always signify an address. IDSMail™ will recognize these descriptors as addresses, even if the angle brackets are omitted. For information on these and other mail-system specific considerations, select the mail system below:

Internet Address Formats

MAPI Address Formats

VIM Address Formats

MHS Address Formats

VINES Address Formats

Compuserve Address Formats

# MAPI Address Formats

In MAPI, user names are normally used for messages sent to members of the same workgroup or mail system. To send to users OUTSIDE the local mail system, or to send the message via fax or some other gateway, an address is required.

IDSMail™ recognizes a recipient descriptor containing a colon as being an address, even if not contained in angle braces. For example, the following are equivalent recipient descriptors:

fax:776-1267
<fax:776-1267>

# VIM Address Formats

**cc:Mail**

In cc:Mail, user names are normally used for messages sent to members of the same post office. No address information is required. To send to users OUTSIDE the local post office, an address is required.

IDSMail™ recognizes the @ character or the word "at" (any case, separated by at least one space on each side of the word) as an address, even if not enclosed in angle braces. For example, the following are equivalent recipient descriptors:

> Therese Mayfield at HQPostOffice
> <Therese Mayfield At HQPostOffice>
> Therese Mayfield AT HQPostOffice
> Ms. Mayfield <Therese Mayfield At HQPostOffice>

IDSMail can also be used to post messages to cc:Mail bulletin boards. To do so, the bulletin board name is specified in place of the recipient name. To post to bulletin boards in an external post office, use the addressing format specified above.

**Lotus Notes**

Notes addresses are different from cc:Mail. In Notes, providing only the user name is not sufficient. The Notes address must be a fully qualified address such as:

> <Alan Buck/Captura Software/US>
> <Vince Lutheran/Wiltron/DE>

# MHS Address Formats

For MHS, IDSMail™ will use the entire recipient descriptor for the address UNLESS an address is specifically enclosed within angle brackets. In that case, only the portion between the angle brackets will be used and any name information preceding the angle brackets will be discarded.

# Banyan VINES IM Address Formats

In Banyan VINES Intelligent Messaging, the concept of a name is missing. Recipient descriptors are always addresses in the form of user@group@organization. The angle brackets may be omitted for these addresses. For example, the following are equivalent recipient descriptors:

Alicia@Accounting@Acme
<Alicia@Accounting@Acme>

# Internet Address Formats

Internet addresses are generally of the format **user@entity.xxx** where xxx is generally com, gov, edu, org, or other abbreviations for addresses outside of the United States. Internet addresses are usually case-sensitive.

A name can optionally be added ahead of the address for clarity. The name itself is not used for routing purposes, and can in fact be anything the sender desires. For example, the following are equivalent recipient descriptors:

> Kathy Williams <kathy@hercompany.com>
> kathy@hercompany.com
> Mom <kathy@hercompany.com>
> Mrs. Williams <kathy@hercompany.com>

Unlike some other mail systems, the name itself is not enough for proper routing. The address must be included.

# Compuserve Address Formats

If the Compuserve Driver for Microsoft Mail is in use, the recipient descriptor can be a name, address, or both. If only the name is specified, the name must appear in the Compuserve address book on the local machine. This is similar to MAPI: the address doesnt need to be specified if the system can resolve it from the directory information.

If only the Compuserve address is specified, it must be enclosed in angle brackets. A name can be used in conjunction with the address. The name will be ignored for addressing purposes, but will appear in the message when it reaches its destination.

The following are all valid recipient descriptors for Compuserve:

Bob Wills <74662,3921>
My Buddy Bob <74662,3921>
<74662,3921>

The following is valid IF Bob Wills is a member of the Compuserve address book on the local machine.

Bob Wills

For Compuserve access, the CompuServe MS Mail Driver must be present on the workstation. This driver can be downloaded free of charge from Compuserve (GO MAP-1)

At the time of writing, IDS has identified some problems with this driver (v1.02) and have informed Compuserve. We hope that these will be resolved in the near future. For more information on using the Compuserve driver with IDSMail™, please visit our web site. The URL is in the README file.

Until the driver problems are resolved, the IDSMail™/Compuserve link should be considered a Beta release and **is not supported by IDS**.

# Contacting IDS

Intuitive Data Solutions is available on the World Wide Web at:

**http://www.intuitive-data.com**

Please visit our site regularly to find out about the latest enhancements to the IDSMail™ OLE Server, and to learn about other new products and services.

# Copyright, Trademarks, and Credits

## Copyright Notice



## Credits

IDSMail™ TCP/IP support licensed from Dolphin Systems Inc.
IDSMail™ file compression technology licensed from Inner Media Inc.

## Trademarks

**NOTE: All Address Book properties and methods require IDSMail Professional Edition.**

## Properties

Address
Bof
Bookmark1
Bookmark2
Comments
DialogAvailable
DialogCaption
DialogErrorOnCancel
DialogResultDelimiter
DialogType
EnumerationAvailable
Eof
MailObject
Name
NoMatch
RecordCountApproximate
RecordNumberApproximate

## Methods

CloseAddressBook
GetAddressBookItems
GetDialogResultItems
GetDialogResultString
Move
MoveFirst
MoveLast
MoveNext
MovePrevious
OpenAddressBook
SeekName
ShowAddressBookDialog

# Address (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This string property returns the full address of the current AddressBook record.

## Usage

Read-only, Default = empty string

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses

```
idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop
```

## See Also:

Address Formats

# Bof (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This short integer property returns True (-1) if the Address Book record pointer is set to a position before the first record, or if the Address Book is empty. The property returns False (0) otherwise.

If this property is True, values retrieved via the Name, Address, or Comments properties are invalid.

## Usage

Read-only

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses in reverse order

```
idsAddr.MoveLast
Do While Not idsAddr.Bof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MovePrevious
Loop
```

# Bookmark1 (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This variant property is used in conjunction with the Bookmark2 property to get a bookmark for the current Address Book record, or to return to a specific record at a later time. Bookmark1 and Bookmark2 may be queried in any order, but MUST be set in the order of Bookmark1 then Bookmark2. The record pointer is repositioned after the Bookmark2 property is set.

## Usage

Read/Write, No Default

## Example

```
    Remember the current position

bm1 = idsAddr.Bookmark1
bm2 = idsAddr.Bookmark2

    Enumerate the rest of the records

Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop

    Return to the previous position

idsAddr.Bookmark1 = bm1
idsAddr.Bookmark2 = bm2
```

# Bookmark2 (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This variant property is used in conjunction with the Bookmark1 property to get a bookmark for the current Address Book record, or to return to a specific record at a later time. Bookmark1 and Bookmark2 may be queried in any order, but MUST be set in the order of Bookmark1 then Bookmark2. The record pointer is repositioned after the Bookmark2 property is set.

## Usage

Read/Write, No Default

## Example

```
    Remember the current position

bm1 = idsAddr.Bookmark1
bm2 = idsAddr.Bookmark2

    Enumerate the rest of the records

Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop

    Return to the previous position

idsAddr.Bookmark1 = bm1
idsAddr.Bookmark2 = bm2
```

# CloseAddressBook (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method explicitly closes the Address Book. It is executed automatically when the AddressBook object is set to nothing or goes out of scope.

## Parameters

NONE

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Enumerate every fifth name and address

idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    skipRecords& = 5
    idsAddr.Move skipRecords&
Loop

    Close the address book

idsAddr.CloseAddressBook
```

# Comments (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This string property returns the comments (if any) associated with the current AddressBook record.

## Usage

Read-only, Default = empty string

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names, addresses, and comments

```
idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    theComments$ = idsAddr.Comments
    MsgBox theName$ + ", " + theAddress$ + ", " + theComments$
    idsAddr.MoveNext
Loop
```

# DialogAvailable (property)

## Applies To

AddressBook Object (All Protocols)

## Description

This short integer property returns True (-1) if the ShowAddressBookDialog method is supported on the current mail system.

## Usage

Read-only

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Show a dialog if supported

If idsAddr.DialogAvailable = True Then
    idsAddr.ShowAddressBookDialog
End If
```

# DialogCaption (property)

## Applies To

AddressBook Object (MAPI & Banyan VINES)

## Description

This string property sets the window caption for the Address Book dialog. The default value results in the default caption being used, which is typically "Address Book". Note that Novell GroupWise ignores this setting and always uses the caption "Address Book".

## Usage

Read/Write, Default = empty string which results in a default dialog caption

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.DialogCaption = "Select Recipients"
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items as a string

```
theResults$ = idsAddr.GetDialogResultString (IDSM_ITM_AB_NAME_ADDRESS)
```

# DialogErrorOnCancel (property)

## Applies To

AddressBook Object (MAPI & Banyan VINES)

## Description

This short integer property determines what happens if the user selects Cancel from the Address Book Dialog. If the property is True (-1) a trappable error is triggered. If False (0) execution proceeds normally.

## Usage

Read/Write, Default = False

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.DialogCaption = "Select Recipients"
idsAddr.DialogErrorOnCancel = True
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items as a string

```
theResults$ = idsAddr.GetDialogResultString (IDSM_ITM_AB_NAME_ADDRESS)
```

# DialogResultDelimiter (property)

## Applies To

AddressBook Object (MAPI & Banyan VINES)

## Description

This string property specifies the delimiter that is used between items returned from the GetDialogResultString method.

## Usage

Read/Write, Default = comma

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.DialogCaption = "Select Recipients"
idsAddr.DialogErrorOnCancel = True
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items as a string

```
idsAddr.DialogResultDelimiter = ";"
theResults$ = idsAddr.GetDialogResultString (IDSM_ITM_AB_NAME_ADDRESS)
```

# DialogType (property)

## Applies To

AddressBook Object (Banyan VINES only)

## Description

This short integer property is used only for Banyan VINES. It specifies the type of Address Book dialog displayed by ShowAddressBookDialog. The values for the property are:

IDSM_AB_DLG_SINGLE (1) - Specifies a dialog in which only one name may be chosen.

IDSM_AB_DLG_MULTIPLE (2) - Specifies a multiple selection dialog.

## Usage

Read/Write, Default = IDSM_AB_DLG_MULTIPLE

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.DialogCaption = "Select Recipients"
idsAddr.DialogErrorOnCancel = True
idsAddr.DialogType = IDSM_AB_DLG_SINGLE
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items as a string

```
idsAddr.DialogResultDelimiter = ";"
theResults$ = idsAddr.GetDialogResultString (IDSM_ITM_AB_NAME_ADDRESS)
```

# EnumerationAvailable (property)

## Applies To

AddressBook Object (All Protocols)

## Description

This short integer property returns True (-1) if the Address Book enumeration methods such as MoveFirst, MoveNext, GetAddressBookItems, etc. are available. The property returns False (0) otherwise.

## Usage

Read-only

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Enumerate all the names and addresses

If idsAddr.EnumerationAvailable = True Then
    idsAddr.MoveFirst
    Do While Not idsAddr.Bof
        theAddress$ = idsAddr.Address
        theName$ = idsAddr.Name
        MsgBox theName$ + ", " + theAddress$
        idsAddr.MoveNext
    Loop
End If
```

# Eof (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This short integer property returns True (-1) if the Address Book record pointer is set to a position after the last record, or if the Address Book is empty. The property returns False (0) otherwise.

If this property is True, values retrieved via the Name, Address, or Comments properties are invalid.

## Usage

Read-only

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Enumerate all the names and addresses

idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop
```

# GetAddressBookItems (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method fills arrays with the specified type of items from the address book. The method is only available for protocols in which the EnumerationAvailable property returns True. The first item will be placed at array index=0.

The first item comes from the current AddressBook object record. To begin from the first record, execute the MoveFirst method prior to GetAddressBookItems. After completion, this method leaves the record pointer at the last item retrieved. To iterate through all records in the Address Book, the MoveNext method must be executed after each GetAddressBookItems call.

**NOTE: This method is not supported for Visual Basic v3.0, Access v2.0, and other environments which implemented an older version of OLE support. In these cases, use the Address and Name properties and step through the address book using MoveFirst, MoveNext, etc.**

## Parameters

*itemType* - Short integer specifying the type of item to retrieve from the Address Book. One of the following constants:

IDSM_ITM_AB_NAME
IDSM_ITM_AB_ADDRESS
IDSM_ITM_AB_NAME_ADDRESS
IDSM_ITM_AB_NAME_COMMENTS
IDSM_ITM_AB_NAMEADDRESS_COMMENTS

*array1()* - An array of strings which will be filled with the primary parts of the available items. The array must be dimensioned large enough to hold all of the items, or an error will occur.

*array2()* - An additional array of strings which will be filled with the secondary parts of the available items. The array must be dimensioned large enough to hold all of the items, or an error will occur.

*maxItems* - Long integer designating the maximum number of items retrieved.

*itemCount* - Long Integer designating the number of items actually retrieved.

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
```

```
idsAddr.OpenAddressBook

    Retrieve all names and addresses in groups of 5

maxItems& = 5
Redim array1$(maxItems&)
Redim array2$(maxItems&)
idsAddr.MoveFirst
Do While Not idsAddr.Eof
    idsAddr.GetAddressBookItems IDSM_ITM_AB_NAME_ADDRESS, array1$(), _
                                                          array2$(), maxItems&, numItems&
    For i% = 0 to numItems&-1
        MsgBox array1$(i%) + ", " + array2$(i%)
    Next i%
    idsAddr.MoveNext
Loop
```

**IDSM_ITM_AB_NAME (value = 100)**

**GetAddressBookItems/GetDialogResultItems:**

The name field (e.g. "John Smith") is returned in array1. Elements of array2 are all empty.

**GetDialogResultString**

The name fields (e.g. "John Smith") separated by the DialogResultDelimiter.

**IDSM_ITM_AB_ADDRESS (value = 101)**

**GetAddressBookItems/GetDialogResultItems:**

The address field, in a format specific to the underlying mail system, is returned in array1. See Address Formats for examples. Elements of array2 are all empty.

**GetDialogResultString**

The address fields, in a format specific to the underlying mail system, separated by the DialogResultDelimiter. See Address Formats for examples.

**IDSM_ITM_AB_NAME_ADDRESS (value = 102)**

**GetAddressBookItems/GetDialogResultItems:**

The name is returned in array1 and the address is returned in array2. The address is in a format specific to the underlying mail system. See Address Formats for examples.

**GetDialogResultString**

The name and address fields combined in this format: "name <address>". This string may be passed to an IDSMail AddRecipient method as a fully-qualified recipient. See Address Formats for address examples. Each result item is separated by the DialogResultDelimiter.

**IDSM_ITM_AB_NAME_COMMENTS (value = 103)**

**GetAddressBookItems/GetDialogResultItems:**

The name is returned in array1 and the associated comments (if any) are returned in array2.

**GetDialogResultString**

The name fields (e.g. "John Smith") separated by the DialogResultDelimiter. This is the same as IDSM_ITM_AB_NAME.

**IDSM_ITM_AB_NAMEADDRESS_COMMENTS(value = 104)**

**GetAddressBookItems/GetDialogResultItems:**

Array1 returns the name and address fields combined in this format: "name <address>". Array2 contains the associated comments (if any). The address is in a format specific to the underlying mail system. See Address Formats for examples.

**GetDialogResultString**

The name and address fields combined in this format: "name <address>". This string may be passed to an IDSMail AddRecipient method as a fully-qualified recipient. See Address Formats for address examples. Each result item is separated by the DialogResultDelimiter. This is the same as IDSM_ITM_AB_NAME_ADDRESS.

# GetDialogResultItems (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method fills arrays with the items that the user selected in the last call to ShowAddressBookDialog. The first item will be placed at array index=0.

**NOTE: This method is not supported for Visual Basic v3.0, Access v2.0, and other environments which implemented an older version of OLE support. In these cases, use the GetDialogResultString method instead.**

## Parameters

*itemType* - Short integer specifying the type of item to retrieve from the dialog results. One of the following constants:

IDSM_ITM_AB_NAME
IDSM_ITM_AB_ADDRESS
IDSM_ITM_AB_NAME_ADDRESS
IDSM_ITM_AB_NAME_COMMENTS
IDSM_ITM_AB_NAMEADDRESS_COMMENTS

*array1()* - An array of strings which will be filled with the primary parts of the available items. The array must be dimensioned large enough to hold all of the items, or an error will occur.

*array2()* - An additional array of strings which will be filled with the secondary parts of the available items. The array must be dimensioned large enough to hold all of the items, or an error will occur.

*itemCount* - Long Integer designating the number of items actually retrieved.

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items in arrays

```
Redim array1$(100)
```

```
Redim array2$(100)
idsAddr.GetDialogResultItems IDSM_ITM_AB_NAME_ADDRESS, array1$(), _
                                                       array2$(), numItems&
For I% = 0 to numItems& - 1
     theName$ = array1$(i%)
     theAddress$ = array2$(i%)
Next i%
```

# GetDialogResultString (method)

## Description

This method returns a string with the results of the last call to ShowAddressBookDialog.

## Parameters

*itemType* - Short integer specifying the type of item to retrieve from the dialog results. One of the following constants:

IDSM_ITM_AB_NAME
IDSM_ITM_AB_ADDRESS
IDSM_ITM_AB_NAME_ADDRESS
IDSM_ITM_AB_NAME_COMMENTS
IDSM_ITM_AB_NAMEADDRESS_COMMENTS

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.DialogCaption = "Select Recipients"
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items as a string

```
idsAddr.DialogResultDelimiter = ";"
theResults$ = idsAddr.GetDialogResultString (IDSM_ITM_AB_NAME_ADDRESS)
```

# MailObject (property)

## Applies To

AddressBook Object (All protocols)

## Description

The MailObject property is an object property which must be passed the IDSMail Server object variable prior to opening an Address Book session.

## Usage

Read/Write, Default = Nothing

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

If using VB3, use this line instead:

```
Set idsAddr.MailObject = idsMail
```

# Move (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method allows movement forward or backward in the Address Book by an arbitrary number of records.

## Parameters

*numRecords* - A long integer variable specifying how many records to move. Positive values move forward (toward the end) of the Address Book. Negative values move backward (toward the beginning) of the Address Book.

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Enumerate every fifth name and address

idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    skipRecords& = 5
    idsAddr.Move skipRecords&
Loop
```

# MoveFirst (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method moves the record pointer to the first Address Book record.

## Parameters

NONE

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses

```
idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop
```

# MoveLast (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method moves the record pointer to the last Address Book record.

## Parameters

NONE

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses in reverse order

```
idsAddr.MoveLast
Do While Not idsAddr.Bof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MovePrevious
Loop
```

# MoveNext (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method moves the record pointer to the next Address Book record.

## Parameters

NONE

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses

```
idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop
```

# MovePrevious (method)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This method moves the record pointer to the last Address Book record.

## Parameters

NONE

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses in reverse order

```
idsAddr.MoveLast
Do While Not idsAddr.Bof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MovePrevious
Loop
```

# Name (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This string property returns the name field from the current AddressBook record (e.g "John Smith").

## Usage

Read-only, Default = empty string

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Enumerate all the names and addresses

```
idsAddr.MoveFirst
Do While Not idsAddr.Eof
    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext
Loop
```

# NoMatch (property)

## Applies To

AddressBook Object (VIM)

## Description

This short integer property is used in conjunction with the SeekName method. It returns True (-1) if the last SeekName method did not locate a record. Returns False (0) if SeekName succeeded.

## Usage

Read-only

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Seek the name John Smith by using the first few letters of the last name

idsAddr.SeekName IDSM_AB_SEEK_PARTIAL, Smi
If idsAddr.NoMatch = True Then
    MsgBox Name not found.
Else
    MsgBox Found:  + idsAddr.Name
End If
```

# OpenAddressBook (method)

## Applies To

AddressBook Object (All protocols)

## Description

This method opens the underlying Address Book for the mail system. The MailObject property must be specified prior to executing this method. This method must be executed prior to other AddressBook object methods.

## Parameters

NONE

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

# RecordCountApproximate (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This long integer property returns the approximate total number of records in the address book. Because the total is only approximate, it should not be used as a loop limit. It may be used in conjunction with the RecordNumberApproximate property for positioning scroll bars, progress bars, etc.

## Usage

Read-only

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Enumerate all the names and addresses

idsAddr.MoveFirst
Do While Not idsAddr.Eof

    recNum& = idsAddr.RecordNumberApproximate
    recTot& = idsAddr.RecordCountApproximate
    percentage! = recNum& / recTot& * 100

    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext

Loop
```

# RecordNumberApproximate (property)

## Applies To

AddressBook Object (VIM & MHS)

## Description

This long integer property returns the approximate record number of the current record in the address book. It may be used in conjunction with the RecordCountApproximate property for positioning scroll bars, progress bars, etc.  Property returns 1 for the first record.

## Usage

Read-only

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Enumerate all the names and addresses

idsAddr.MoveFirst
Do While Not idsAddr.Eof

    recNum& = idsAddr.RecordNumberApproximate
    recTot& = idsAddr.RecordCountApproximate
    percentage! = recNum& / recTot& * 100

    theAddress$ = idsAddr.Address
    theName$ = idsAddr.Name
    MsgBox theName$ + ", " + theAddress$
    idsAddr.MoveNext

Loop
```

# SeekName (method)

## Applies To

AddressBook Object (VIM)

## Description

This method is applicable to VIM only. It allows an address book entry to be seeked by name, either a full name or a partial name. The search begins at the current record position allowing for iterative "type-down" searches. To search the entire Address Book, execute the MoveFirst method prior to SeekName. The NoMatch property indicates whether or not a successful match occurred.

Note that in cc:Mail, the name must be specified in the format of "LastName, FirstName". For example, to find the name "John Smith" pass the string "Smith,John".

## Parameters

*fullOrPartial* - Short integer specifying either an exact (full) match or a match based on partial name information. Either IDSM_AB_SEEK_FULL (1) or IDSM_AB_SEEK_PARTIAL (2).

*theName* - String specifying the name to seek. See comments above for format.

## Example

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook

    Seek the name John Smith by using the first few letters of the last name

idsAddr.SeekName IDSM_AB_SEEK_PARTIAL, "Smi"
If idsAddr.NoMatch = True Then
    MsgBox Name not found.
Else
    MsgBox Found:  + idsAddr.Name
End If
```

# ShowAddressBookDialog (method)

## Applies To

AddressBook Object (MAPI & Banyan VINES)

## Description

The method launches a native Address Book dialog allowing the user to select one or more names/addresses. This method is only available if the DialogAvailable property returns True (-1). The results of the dialog may be retrieved via the GetDialogResultItems or GetDialogResultString methods.

The dialog behavior may be modified with via the DialogCaption, DialogErrorOnCancel, and DialogType properties.

## Parameters

{NONE}

## Example

Initialize objects

```
Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

Show an address book dialog

```
idsAddr.DialogCaption = "Select Recipients"
idsAddr.DialogErrorOnCancel = False
idsAddr.ShowAddressBookDialog
```

Retrieve the chosen items as a string

```
idsAddr.DialogResultDelimiter = ";"
theResults$ = idsAddr.GetDialogResultString (IDSM_ITM_AB_NAME_ADDRESS)
```

# The AddressBook Object

**NOTE: All Address Book properties and methods require IDSMail Professional Edition.**

The AddressBook component enables your application to directly access the E-mail user list (Address Book or Directory). The E-mail systems differ greatly in the features that they provide in this area. So, unlike the IDSMail server object, AddressBook functionality is **not** universal across platforms.

There are two basic methodologies used in the mail systems and in the AddressBook object. The DialogAvailable and EnumerationAvailable properties may be read at runtime to tell your application which methodologies are available. Those methodologies are:

**Address Book Dialog (MAPI & Banyan VINES)** - MAPI and Banyan VINES each provide a built-in Address Book dialog. This dialog is accessible via the ShowAddressBookDialog method. The user selects entries which are then returned via the GetDialogResultItems or GetDialogResultString methods.

**Enumeration (VIM & MHS)** - VIM and MHS do not provide dialogs, but do provide some lower-level accessibility to the members of the E-mail directory. The AddressBook object models the E-mail directory as a database. Methods such as MoveFirst, MoveNext, etc. are used to step through the database, just as would be done with a table in MS Access, FoxPro, SQLServer, etc.

Note that there is no Address Book dialog for VIM & MHS. The ShowAddressBookDialog method triggers a trappable error for those systems. And, you cannot use methods such as MoveNext, MoveLast, or SeekName with MAPI or Banyan VINES. These two access methodologies should be viewed as being mutually exclusive.

## AddressBook Object properties and methods

VB3/Access 2.0 Syntax Note

# VB3/Access 2.0 Syntax Note

**The syntax for setting the MailObject property is slightly different for VB3/Access 2.0 as shown below:**

```
    Initialize objects

Dim idsMail as Object
Dim idsAddr as Object
Set idsMail = CreateObject("IDSMailInterface.Server")
Set idsAddr = CreateObject("IDSMailInterface.AddressBook")
idsMail.ObjectKey = "ABC123"
idsMail.Login
idsAddr.MailObject = idsMail
idsAddr.OpenAddressBook
```

If using VB3/Access 2.0, use this line instead:

```
Set idsAddr.MailObject = idsMail
```