NMsmtp unit

The NMSMTP unit contains the TNMSMTP component, and its related objects and types.

Components
TNMSMTP

Objects
TPostMessage

Types
TFileItem
THeaderInComplete
TMailListReturn
TRecipientNotFound
TSubType

TPostMessage object

Properties See Also

Unit

NMsmtp

Description

The TPostMessage object is used for storing an outgoing E-Mail message. TNMSMTP contains an instance of TPostMessage as the PostMessage property.

See Also

TNMSMTP.PostMessage

TPostMessage Properties

TPostMessage
Legend

- Attachments
- Body
- Date
- FromAddress
- FromName
- LocalProgram
- ReplyTo
- Subject
- ToAddress
- ToBlindCarbonCopy
- ToCarbonCopy

Attachments property

Applies to
TPostMessage object

Declaration
```
property Attachments: TstringList;
```

Description
The Attachments property specifies a list of files to attach to the outgoing E-Mail message.

Note:
Only one filename per line is permitted.

Body property

Applies to
TPostMessage object

Declaration
```
property Body: Tstringlist;
```

Description
The Body property contains the body of the E-Mail message to send.

Date property

Applies to
TPostMessage object

Declaration
```
property Date: string;
```

Description
The Date property specifies the date the E-Mail was sent. You can set the date to any date you wish. If it is left blank, it is filled in with the current date.

Note:
Having outrageous dates (June 5, 1701; August 30, 2096; etc.) can have unpredictable results on E-Mail delivery.

FromAddress property

Applies to
TPostMessage object

Declaration
```
property FromAddress: string;
```

Description
The FromAddress property specifies the E-Mail address of the sender of the message

FromName property

Applies to
TPostMessage object

Declaration
```
property FromName: string;
```

Description
The FromName property specifies the name of the sender of the E-Mail message.

LocalProgram property

Applies to
TPostMessage object

Declaration
```
property LocalProgram: string;
```

Description
The LocalProgram property specifies the name of the application sending the E-Mail. This is stored in the X-Mailer part of the header.

ReplyTo property

Applies to
TPostMessage object

Declaration
```
property ReplyTo: string;
```

Description
The ReplyTo property specifies the E-Mail address that recipients can use to reply to the message.

Subject property

Applies to
TPostMessage object

Declaration
```
property Subject: string;
```

Description
The Subject property contains the subject of the E-Mail message to be sent.

ToAddress property

Applies to
TPostMessage object

Declaration
```
property ToAddress: Tstringlist;
```

Description
The ToAddress property specifies the primary recipients of the E-Mail message to be sent.

Note:
Only one E-Mail address is permitted per line.

ToBlindCarbonCopy property

Applies to
TPostMessage object

Declaration
```
property ToBlindCarbonCopy: Tstringlist;
```

Description
The ToBlindCarbonCopy property specifies recipients of the message to be sent that will be unaware of the carbon copy status of the message.

Note:
Only one E-Mail address is permitted per line.

ToCarbonCopy property

Applies to
TPostMessage object

Declaration
```
property ToCarbonCopy: Tstringlist;
```

Description
The ToCarbonCopy property specifies the list of E-Mail address that will receive carbon copies of the current message.

Note:
Only one E-Mail address is permitted per line.

TNMSMTP component

Properties Methods Events Tasks

Unit

NMsmtp

Description

The TNMSMTP component enables the sending of E-Mail via an internet mail server and the implementation of other commands specified in RFC 821.

TNMSMTP Properties
TNMSMTP
Legend

In TNMSMTP

ClearParams
EncodeType
FinalHeader
PostMessage
SubType
UserID

Derived from TPowersock

About
BeenCanceled
BeenTimedOut
BytesRecvd
BytesSent
BytesTotal
Connected
Handle
Host
LastErrorNo
LocalIP
Port
Proxy
ProxyPort
RemoteIP
ReplyNumber
ReportLevel
Status
TimeOut
TransactionReply
WSAInfo

Derived from TComponent

ComObject
ComponentCount
ComponentIndex
Components

- ComponentState
- ComponentStyle
- DesignInfo
- Owner
- Tag
- VCLComObject

TNMSMTP Methods

TNMSMTP
Legend

In TNMSMTP

- ExpandList
- ExtractAddress
- Verify
- ClearParameters
- SendMail

Derived from TPowersock

- Abort
- Accept
  - Cancel
- CaptureFile
- CaptureStream
- CaptureString
  - CertifyConnect
- Connect
  - Create
  - Destroy
- Disconnect
  - FilterHeader
  - GetLocalAddress
  - GetPortstring
- Listen
- read
- ReadLn
  - RequestCloseSocket
  - SendBuffer
- SendFile
- SendStream
- Transaction
- write
- writeln

Derived from TComponent

- DestroyComponents
- Destroying
- FindComponent
- FreeNotification
- FreeOnRelease
- GetParentComponent
- HasParent
- InsertComponent
- RemoveComponent
- SafeCallException

Derived from TPersistent

- Assign
- GetNamePath

Derived from TObject

- ClassInfo
- ClassName

ClassNameIs
ClassParent
ClassType
CleanupInstance
DefaultHandler
Dispatch
FieldAddress
Free
FreeInstance
GetInterface
GetInterfaceEntry
GetInterfaceTable
InheritsFrom
InitInstance
InstanceSize
MethodAddress
MethodName
NewInstance

TNMSMTP Events
TNMSMTP
Legend

In TNMSMTP

- OnAttachmentNotFound
- OnAuthenticationFailed
- OnEncodeEnd
- OnEncodeStart
- OnFailure
- OnHeaderIncomplete
- OnMailListReturn
- OnRecipientNotFound
- OnSendStart
- OnSuccess

Derived from TPowersock

- OnAccept
- OnConnect
  - OnConnectionFailed
- OnConnectionRequired
- OnDisconnect
- OnError
- OnHostResolved
- OnInvalidHost
- OnPacketRecvd
- OnPacketSent
- OnRead
- OnStatus

About the TNMSMTP component

TNMSMTP reference

Purpose

The TNMSMTP component enables the sending of mail via an internet mail server. It needs a TCP/IP stack, WSOCK32.DLL, which is available from many vendors including Microsoft, and is included with Windows 95, 98, and NT.

Tasks

The mail server to connect to is defined by the properties **Host** and **Port**.

A connection is established by the **Connect** Method and terminated with the **Disconnect** method.

Sending Internet E-Mail:
The PostMessage property contains the data that the E-Mail message will be composed of, and the actual posting of mail is done with the SendMail method.

Verifying the Existance of a User:
The existance of a user on a connected host can be determined using the Verify method.

Expand a Mailing List:
The members of a mailing list can be determined by using the ExpandList method, and writing an event handler for the OnMailListReturn event.

ClearParams property

See also Example

Applies to
TNMSMTP component

Declaration
```
property ClearParams: boolean;
```

Description
The ClearParams property specifies whether the fields of the **PostMessage** property will be cleared after the message is sent. If ClearParams is TRUE, the fields of the PostMessage property will be cleared when a message is sent using the **SendMail** method. If ClearParams is FALSE, the fields of PostMessage will not be cleared.

Default: TRUE

Scope: Published
Accessability: Runtime, Designtime

See also

PostMessage property
SendMail method

Example

To recreate this example, you will need to create a new blank Delphi application.

Place 5 TMemos, 8 TEdits, a TListBox, 3 TButtons, a TCheckBox, a TNMSMTP, a TRadioGroup, and a TOpenDialog on the form.

If you wish to label the controls, they do the following:

Edit1: Host property
Edit2: User ID property
Edit3: PostMessage.Date property
Edit4: PostMessage.FromAddress property
Edit5: PostMessage.FromName property
Edit6: PostMessage.LocalProgram property
Edit7: PostMessage.ReplyTo property
Edit8: PostMessage.Subject property
Memo1: PostMessage.ToAddress property.
Memo2: PostMessage.ToBlindCarbonCopy property
Memo3: PostMessage.ToCarbonCopy property
Memo4: PostMessage.Body property
Memo5: Status window
ListBox1: PostMessage.Attachments property
Button1: Connect/Disconnect button
Button2: SendMail button
Button3: Clears edit fields and parameters
CheckBox1: sets value of ClearParams property
OpenDialog1: Adds files to attach to the E-Mail
RadioGroup1: Specifies the file encode method**

**Add 2 items to RadioGroup1's Items property: MIME and UUEncode
(In that order)

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if NMSMTP1.Connected then
    NMSMTP1.Disconnect
  else
  begin
    NMSMTP1.Host := Edit1.Text;
    NMSMTP1.UserID := Edit2.Text;
    NMSMTP1.Connect;
  end;
end;
```

When Button1 is clicked, if NMSMTP1 is connected as specified by the **Connected** property, the **Disconnect** method is called. If there is no connection present, the **Host** property is set to the value in Edit1, the **UserID** property is set to the value of Edit2, and the **Connect** method is called to connect to the SMTP host.

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
```

```
begin
  if NMSMTP1.Connected then
  begin
    NMSMTP1.ClearParams := CheckBox1.Checked;
    NMSMTP1.SubType := mtPlain;
    case RadioGroup1.ItemIndex of
      0: NMSMTP1.EncodeType := uuMime;
      1: NMSMTP1.EncodeType := uuCode;
    end;
    NMSMTP1.PostMessage.FromAddress := Edit4.Text;
    NMSMTP1.PostMessage.FromName := Edit5.Text;
    NMSMTP1.PostMessage.ToAddress.Text := Memo1.Text;
    NMSMTP1.PostMessage.ToCarbonCopy.Text := Memo3.Text;
    NMSMTP1.PostMessage.ToBlindCarbonCopy.Text := Memo2.Text;
    NMSMTP1.PostMessage.Body.Text := Memo4.Text;
    NMSMTP1.PostMessage.Attachments.Text := ListBox1.Items.Text;
    NMSMTP1.PostMessage.Subject := Edit8.Text;
    NMSMTP1.PostMessage.LocalProgram := Edit6.Text;
    NMSMTP1.PostMessage.Date := Edit3.Text;
    NMSMTP1.PostMessage.ReplyTo := Edit7.Text;
    NMSMTP1.SendMail;
  end
  else
    ShowMessage('You need to connect before you can send your message');
end;
```

When Button2 is clicked, if NMSMTP1 is connected, the **ClearParams** property is set to the value of CheckBox1.Checked, to determine whether the parameters of PostMessage will be cleared after a successful SendMail or not. The **SubType** property is set to mtPlain, signifying that the message being sent is plain ASCII text with no special formatting. The **EncodeType** property is set to either uuMime or uuCode, depending on which item in RadioGroup1 is selected. The **PostMessage** property contains sub-properties that define an E-Mail message. The **FromAddress** sub-property is set to the E-Mail address entered in Edit4. The **FromName** sub-property is set to the name entered in Edit5. The **ToAddress** sub-property is a TStringList, to allow multiple recipients of a message, so it's Text is set to the value of Memo1's Text property. The **ToCarbonCopy** and **ToBlindCarbonCopy** sub-properties are also TStringLists. The ToCarbonCopy property is set to the addresses entered in Memo3.Text, and the ToBlindCarbonCopy property is set to the addresses entered in Memo2.Text. The **Body** sub-property of PostMessage contains the body of the E-Mail, and is set to the value of Memo4.Text. The list of files that are in ListBox1 are set to the **Attachments** sub-property. The **Subject** sub-property is set to the value entered into Edit8. The **LocalProgram** sub-property is set to the value entered in Edit6. The **Date** sub-property is set to the value entered in Edit3. The date entered will be stored in the header, even if the date is not a date, simply text. The **ReplyTo** sub-property is set to the value of Edit7. Finally, the message is sent with the **SendMail** method.
If there is no connection present, a message box is displayed informing the user that a connection is required to send a message.

Insert the following code into Button3's OnClick event:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  NMSMTP1.ClearParameters;
  Edit3.Clear;
  Edit4.Clear;
  Edit5.Clear;
  Edit6.Clear;
```

```
  Edit7.Clear;
  Edit8.Clear;
  Memo1.Clear;
  Memo2.Clear;
  Memo3.Clear;
  Memo4.Clear;
  Memo5.Clear;
  ListBox1.Clear;
end;
```

When Button3 is clicked, the **ClearParameters** method clears the parameters of the **PostMessage** property, and also clears the contents of the input fields on the form to receive a new mail message.

Insert the following code into ListBox1's OnKeyDown event:

```
procedure TForm1.ListBox1KeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_INSERT then
    if OpenDialog1.Execute then
      ListBox1.Items.Add(OpenDialog1.FileName);
  if Key = VK_DELETE then
    ListBox1.Items.Delete(ListBox1.ItemIndex);
end;
```

If the Insert key is pressed while ListBox1 has focus, OpenDialog1 is displayed so the user can pick a file.
If the user clicks the OK button in the Open Dialog, the filename is added to ListBox1's Items.
If the Delete key is pressed while ListBox1 has focus, the currently selected filename is removed from the list.

Insert the following code into NMSMTP1's OnAttachmentNotFound event:

```
procedure TForm1.NMSMTP1AttachmentNotFound(Filename: String);
begin
  Memo5.Lines.Add('File attachment '+FileName+' not found');
end;
```

If one of the files specified in ListBox1 does not exist when the Button2 is clicked, the OnAttachmentNotFound event is called. In this case, the status window, Memo5, is updated to inform the user that a specified attachment was not found, and which file it was that was not found.

Insert the following code into NMSMTP1's OnAuthenticationFailed event:

```
procedure TForm1.NMSMTP1AuthenticationFailed(var Handled: Boolean);
var
  S: String;
begin
  S := NMSMTP1.UserID;
  if InputQuery('Authentication Failed', 'Invalid User ID. New User ID: ', S) then
  begin
    NMSMTP1.UserID := S;
    Handled := TRUE;
  end;
end;
```

If the User ID specified by the UserID property is invalid, or the UserID property is blank and a User ID is required, the OnAuthenticationFailed event is called. In this case, the InputQuery function is used to give the user the opportunity to recitfy the error. If the user enters a new User ID, the Handled property is set to true, and authentication is attempted again. If the user just clicks the cancel button, the UserID property is not reset, and an exception is raised.

Insert the following code into NMSMTP1's OnConnect event:

```
procedure TForm1.NMSMTP1Connect(Sender: TObject);
begin
  Memo5.Lines.Add('Connected');
end;
```

When a connection is established with the SMTP host, the OnConnect event notifies the user of the connect by adding a line that reads Connected to Memo5.

Insert the following code into NMSMTP1's OnSendStart event:

```
procedure TForm1.NMSMTP1SendStart(Sender: TObject);
begin
  Memo5.Lines.Add('Sending Message');
end;
```

When a message is about to be sent, the OnSendStart event is called. In this instance, Memo5 is updated to inform the user that the message is being sent.

Insert the following code into NMSMTP1's OnEncodeStart event:

```
procedure TForm1.NMSMTP1EncodeStart(Filename: String);
begin
  Memo5.Lines.Add('Encoding '+FileName);
end;
```

If a message has file attachments, when they begin encoding, the OnEncodeStart event is called. In this instance, Memo5 displays the name of the file being encoded.

Insert the following code into NMSMTP1's OnEncodeEnd event:

```
procedure TForm1.NMSMTP1EncodeEnd(Filename: String);
begin
  Memo5.Lines.Add(FileName+' encoded');
end;
```

If a message has file attachments, when they complete encoding, the OnEncodeEnd event is called. In this instance, Memo5 displays the name of the file that has finished being encoded.

Insert the following code into NMSMTP1's OnFailure event:

```
procedure TForm1.NMSMTP1Failure(Sender: TObject);
begin
```

```
  Memo5.Lines.Add('Message delivery failure');
end;
```

If an outgoing message fails to be sent, the OnFailure event is called. In this instance, Memo5 updates to inform the user of the failure.

Insert the following code into NMSMTP1's OnSuccess event:

```
procedure TForm1.NMSMTP1Success(Sender: TObject);
begin
  Memo5.Lines.Add('Message sent successfully');
end;
```

When an outgoing message has been sent successfully, the OnSuccess event is called. In this example, Memo5 is updated to inform the user that the message was sent successfully.

Insert the following code into NMSMTP1's OnHeaderIncomplete event:

```
procedure TForm1.NMSMTP1HeaderIncomplete(var handled: Boolean; hiType: Integer);
var
  S: String;
begin
  case hiType of
    hiFromAddress:
      if InputQuery('Missing From Address', 'Enter From Address: ', S) then
      begin
        NMSMTP1.PostMessage.FromAddress := S;
        Handled := TRUE;
      end;

    hiToAddress:
      if InputQuery('Missing To Address', 'Enter To Address: ', S) then
      begin
        NMSMTP1.PostMessage.ToAddress.Text := S;
        Handled := TRUE;
      end;
  end;
end;
```

If the PostMessage property is missing information that is critical to the outgoing message being sent successfully, the OnHeaderIncomplete event is called. In this example, the hiType parameter is checked, and the user is given the opportunity to fill in the missing information. If the user fills in the information that is missing, the Handled parameter is set to TRUE, and the message continues to be sent. If the user clicks the cancel button instead of entering the missing information, an exception gets raised.

Insert the following code into NMSMTP1's OnRecipientNotFound event:

```
procedure TForm1.NMSMTP1RecipientNotFound(Recipient: String);
begin
  Memo5.Lines.Add('Recipient '+Recipient+' not found');
end;
```

If one of the recipients of the outgoing message in either the ToAddress, ToBlindCarbonCopy, or

ToCarbonCopy fields are known by the SMTP host to not exist, the OnRecipientNotFound event is called. In this example, Memo5 is updated to inform the user which recipient could not be found. If only one recipient is specified, an exception is raised because no valid recipients were found for the outgoing message.

Example Description:
When this application is run, enter the requested information into the Edit boxes and Memos. For the ToAddress, ToCarbonCopy, and TBlindCarbonCopy fields, multiple addresses may be entered, but they must each be on a separate line (carriage return/line feeds between them). Click Button1 to connect and Button2 to send the message. Button3 is used to clear the input fields so a new message may be entered. Clicking Button1 a second time will disconnect from the SMTP host.

EncodeType property

See also Example

Applies to
TNMSMTP component

Declaration
```
property EncodeType: UUMethods;
```

Description
The EncodeType property specifies what type of encoding TNMSMTP will use to encode files attached to an E-Mail message. There are only two options, uuMime, which uses MIME base 64 encoding, and uuCode, which uses UUEncode encoding.

Default: uuMime

Scope: Published
Accessability: Runtime, Designtime

See also

<<< See also of EncodeType property >>>

FinalHeader property

See also Example

Applies to
TNMSMTP component

Declaration
```
property FinalHeader: TExStringList;
```

Description
The FinalHeader property represents the header that is actually sent with the body of the E-Mail message. This property can be viewed and/or modified in the OnSendStart event.

Scope: Public
Accessability: Runtime

See also

<<< See also of FinalHeader property >>>

Example

To recreate this example, you will need to create a new blank Delphi application.

Place a TButton and TNMSMTP on the form.

***Before you begin writing code, fill in the following properties for NMSMTP1 in the Object Inspector:

PostMessage
Host
UserID (if necessary)

It is recommended that you send test E-Mails to either yourself or a colleague.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NMSMTP1.Connect;
end;
```

When Button1 is clicked, the application establishes a connection to the remote SMTP host.

Insert the following code into NMSMTP1's OnConnect event:

```
procedure TForm1.NMSMTP1Connect(Sender: TObject);
begin
  NMSMTP1.SendMail;
end;
```

When NMSMTP1 connects to the remote host, the OnConnect event executes the **SendMail** method.

Insert the following code into NMSMTP1's OnSendStart event:

```
procedure TForm1.NMSMTP1SendStart(Sender: TObject);
begin
  NMSMTP1.FinalHeader.Values['X-Priority'] := '1 (High)';
end;
```

Immediately before NMSMTP1 sends the outgoing message defined in the **PostMessage** property, the OnSendStart event is called. In the above example, the **FinalHeader** property is modified to include the X-Priority item. Many E-Mail clients recognize and use this item to catagorize mail based on urgency. Using the OnSendStart method and FinalHeader properties, custom header fields can be added easily.

Insert the following code into NMSTMP1's OnSuccess event:

```
procedure TForm1.NMSMTP1Success(Sender: TObject);
begin
  NMSMTP1.Disconnect;
end;
```

When the outgoing E-Mail message is delivered successfully, the client disconnects from the SMTP host.

OnAttachmentNotFound event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnAttachmentNotFound: TFileItem;
```

Description

The OnAttachmentNotFound event is called when a file that is to be attached to the outgoing E-Mail is not found. The TFileItem event passes the filename as a parameter, so the file that is missing can be identified.

See also

PostMessage.Attachments property

OnEncodeEnd event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnEncodeEnd: TFileItem;
```

Description

The OnEncodeEnd event is called when a file attached to the outgoing E-Mail has been completely encoded for transmission. The TFileItem event type passes the name of the file that has just been encoded.

See also

OnEncodeStart event
PostMessage.Attachments property

OnEncodeStart event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnEncodeStart: TFileItem;
```

Description

The OnEncodeStart event is called when a file attachment is about to be encoded for transmission. The TFileItem event type passes the name of the file about to be encoded as a parameter.

See also

OnEncodeEnd event
PostMessage.Attachments property

OnHeaderIncomplete event

See also Example

Applies to
TNMSMTP component

Declaration
```
property OnHeaderIncomplete: THeaderInComplete;
```

Description
The OnHeaderIncomplete event is called when one of the following properties of PostMessage are left blank:

- The FromAddress field
- Either the ToAddress, ToCarbonCopy, or ToBlindCarbonCopy fields.

The THeaderInComplete event type is a modified form of the THandlerEvent event type. In addition to the Handled boolean parameter, the hiType parameter is passed as an integer. This specifies which part of the header is missing. The possible values are listed below:

hiFromAddress: The FromAddress field has been left blank
hiToAddress: Either the ToAddress, ToCarbonCopy, or ToBlindCarbonCopy field is blank.

If handled is set to TRUE, the message is attempted again. If one of the required fields is still left blank, an exception is raised and the message is not sent, otherwise the message continues being sent.
If handled is set to FALSE, the message is not sent, and an exception is raised.

See also

PostMessage property
PostMessage.FromAddress property
PostMessage.ToAddress property
PostMessage.ToBlindCarbonCopy property
PostMessage.ToCarbonCopy property

OnMailListReturn event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnMailListReturn: TMailListReturn;
```

Description

The OnMailList return event is called when the ExpandList method receives E-Mail addresses. The TMailListReturn event type passes the E-Mail addresses as a parameter.

See also

ExpandList method

OnMailListReturn property example

OnRecipientNotFound event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnRecipientNotFound: TRecipientNotFound;
```

Description

The OnRecipientNotFound event is called when one of the recipients specified in the PostMessage property (in either the ToAddress, ToCarbonCopy, or ToBlindCarbonCopy) are not found. The TRecipientNotFound event type passes the address that couldn't be found as a parameter.

See also

PostMessage property

PostMessage property

See also Example

Applies to
TNMSMTP component

Declaration
```
property PostMessage: TPostMessage;
```

Description
The PostMessage property contains the message that is going to be sent. See the **TPostMessage** reference for details on the sub-properties of this property.

Scope: Published
Accessability: Runtime, Designtime

Notes:
If the ClearParams property is set to TRUE, when the **SendMail** method completes, the contents of the PostMessage property are cleared.

See also

SendMail method
TPostMessage object

SubType property

See also Example

Applies to
TNMSMTP component

Declaration
```
property SubType: TSubType;
```

Description
The SubType property sets the type of E-Mail text that is being sent. For example, sending an E-Mail written in HTML is still ASCII text, even though it is HTML. By setting the SubType property to mtHTML, the receiving E-Mail client will recognize the message as an HTML message, and display it as such if possible.

Range:
Values defined in the TSubType type.

Default: mtPlain

Scope: Published
Accessability: Runtime, Designtime

See also

<<< See also of SubType property >>>

UserID property

See also Example

Applies to
TNMSMTP component

Declaration
```
property UserID: string;
```

Description
The UserID property specifies the user name to log into the SMTP host. A UserID is not always necessary to connect to an SMTP host, but many servers will not allow sending of mail without a valid User ID.
If a UserID is required but not supplied, the **OnAuthenticationFailed** event is called.

Scope: Published
Accessability: Runtime, Designtime

See also

OnAuthenticationFailed event

ExpandList method

See also Example

Applies to

TNMSMTP component

Declaration

```
function ExpandList(MailList: string): boolean;
```

Description

The ExpandList method is used to retrieve the members of a mailing list on an SMTP server. The **MailList** parameter specifies the list to get names and/or addresses for.

When addresses are returned from the SMTP host, the **OnMailListReturn** event is called.

See also

OnMailListReturn event

Example

To recreate this example, you will need to create a new blank Delphi application.

Place 3 TEdits, a TMemo, 2 TButtons, and a TNMSMTP on the form.

Component Descriptions:

Edit1: Host property
Edit2: UserID property (if needed)
Edit3: name of the mailing list to Expand
Button1: Connect/Disconnect
Button2: Invoke the **ExpandList** method

Set the value of the **Enabled** property for Button2 to **false** in the Object Inspector.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if NMSMTP1.Connected then
    NMSMTP1.Disconnect
  else
  begin
    NMSMTP1.Host := Edit1.Text;
    NMSMTP1.UserID := Edit2.TExt;
    NMSMTP1.Connect;
  end;
end;
```

When Button1 is clicked, if there is a connection present with the SMTP host, the **Disconnect** method is called. If there is no connection present, the **Host** property is set to the name or IP address in Edit1, and the **UserID** property is set to the user ID in Edit2, and the **Connect** method is called to establish a connection.

Insert the following code into NMSMTP1's OnConnect event:

```
procedure TForm1.NMSMTP1Connect(Sender: TObject);
begin
  Button2.Enabled := TRUE;
end;
```

When a connection is established with the SMTP host, Button2 is enabled to allow use of the **ExpandList** method.

Insert the following code into NMSMTP1's OnDisconnect event:

```
procedure TForm1.NMSMTP1Disconnect(Sender: TObject);
begin
  Button2.Enabled := FALSE;
end;
```

When a connection is closed with the SMTP host, the OnDisconnect method disables Button2 to prevent the **ExpandList** method from being called when there is no connection present.

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (not NMSMTP1.ExpandList(Edit3.Text)) then
    ShowMessage('ExpandList failed (unsupported or list not found)');
end;
```

When Button2 is clicked, the **ExpandList** method is called. If it returns FALSE, the call to ExpandList failed, with the reason either being that the command is unsupported, or the list was not found on the server. If the call to ExpandList was successful, the **OnMailListReturn** method returns the address(es) listed by the server.

ExtractAddress method

Example

Applies to

TNMSMTP component

Declaration

```
function ExtractAddress(TotalAddress: string): string;
```

Description

The ExtractAddress method extracts an E-Mail address from a string. It is used mainly for internal purposes, but is made public for general use.

Parameters:

The TotalAddress parameter specifies the string to parse the E-Mail address out of. This function was designed to take a string formatted in the following manners:
Persons Name <email@host.ext>
Persons Name:email@host.ext

Return Value:

The return value of this function is the E-Mail address extracted from the string passed.

Notes:

ExtractAddress does not require a connection to an SMTP host.

Example

To recreate this example, you will need to create a new blank Delphi application.

Place 2 TEdits, a TButton, and a TNMSMTP on the form.

Component Descriptions:
Edit1: Full Address input
Edit2: Result of the **ExtractAddress** method
Button1: Executes the **ExtractAddress** method

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit2.Text := NMSMTP1.ExtractAddress(Edit1.Text);
end;
```

Example Description:
After running the application, type in a full E-Mail address in one of the following formats (or cut and paste the below examples):

Edward T. Smith <edsmith@netmastersllc.com>
Edward T. Smith :edsmith@netmastersllc.com

Clicking Button1 should display edsmith@netmastersllc.com in Edit2 (if the above address(es) are used).

Verify method

See also Example

Applies to

TNMSMTP component

Declaration

```
function Verify(UserName: string): boolean;
```

Description

The Verify method is used to verify the existence of a user on an SMTP host.

Parameters:

The UserName parameter specifies the user to verify. Some hosts require only the name of the user (the name before the @ in the address), while some hosts require the entire address to verify.

Return Value:

If the user is verified, the return value is TRUE, otherwise the return value is FALSE.

See also

<<< See also of Verify method >>>

Example

To recreate this example, you will need to create a new blank Delphi application.

Place 3 TEdits, 2 TButtons, and a TNMSMTP on the form.

Component Descriptions:

Edit1: Host property for NMSMTP1
Edit2: UserID property for NMSMTP1
Edit3: User to pass to the **Verify** method
Button1: Connect/Disconnect from the SMTP host
Button2: Verify the user specified in Edit3

Set the **Enabled** property of Button2 to **FALSE** in the Object Inspector.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if NMSMTP1.Connected then
    NMSMTP1.Disconnect
  else
  begin
    NMSMTP1.Host := Edit1.Text;
    NMSMTP1.UserID := Edit2.Text;
    NMSMTP1.Connect;
  end;
end;
```

When Button1 is clicked, if NMSMTP1 is connected to the SMTP host, the **Disconnect** method is called to disconnect. If there is no connection present, the **Host** property is set to the value in Edit1, the **UserID** property is set to the value in Edit2, and the **Connect** method is called to connect to the SMTP host.

Insert the following code into NMSMTP1's OnConnect event:

```
procedure TForm1.NMSMTP1Connect(Sender: TObject);
begin
  Button2.Enabled := TRUE;
end;
```

When a connection is established with the SMTP host, Button2 is enabled. Button2 executes the **Verify** method, so if there is no connection present, the button can not be pressed.

Insert the following code into NMSMTP1's OnDisconnect event:

```
procedure TForm1.NMSMTP1Disconnect(Sender: TObject);
begin
  Button2.Enabled := FALSE;
end;
```

When the connection with the SMTP host is terminated, Button2 is disabled to prevent the **Verify** method from being invoked while no connected.

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if NMSMTP1.Verify(Edit3.Text) then
    ShowMessage(Edit3.Text+' verified')
  else
    ShowMessage(Edit3.Text+' not verified');
end;
```

When Button2 is clicked, the **Verify** method is called, passing the user name entered in Edit3 as the UserName parameter. If the Verify method returns true, the user exists on the remote host, and a message box is displayed stating that the user was verified. If the Verify method returns False, the user does not exist on the remote host, and a message box is displayed stating that the user was not verified.

Example Description:
After running the application, enter the host name or IP address of your SMTP host into Edit1. If you required a UserID to connect to your SMTP host, enter that User ID in Edit2. Click Button1 to connect to the remote host. When connected, Button2 will become enabled. Enter a username in Edit3, and click Button2. If a user with that name exists on the remote host, a message box will display that the user was verified. If the user does not exist, a message box displays that the user could not be verified. Click Button1 once more to disconnect from the remote host.

ClearParameters method

See also Example

Applies to

TNMSMTP component

Declaration

```
procedure ClearParameters;
```

Description

The ClearParameters method clears the contents of the PostMessage property. The ToAddress field, ToCarbonCopy field, ToBlindCarbonCopy field, and Attachments field are all cleared. To clear the Body field, make a call to PostMessage.Body.Clear

Note:

The ClearParameters method is called automatically after each call to the **SendMail** method if the **ClearParams** property is set to TRUE.

See also

ClearParams property
PostMessage property

SendMail method

See also Example

Applies to

TNMSMTP component

Declaration

```
procedure SendMail;
```

Description

The SendMail method sends the E-Mail message defined by the **PostMessage** property. If the **ClearParams** property is set to TRUE, the **ClearParameters** method is called when the message is sent.

Immediately before the message is sent, the **OnSendStart** event is called.
If the message is sent successfully, the **OnSuccess** event is called.
If there is an error during the sending of the message, the **OnFailureEvent** is called.

See also

ClearParameters method
ClearParams property
OnFailure event
OnSendStart event
OnSuccess event
PostMessage property

OnAuthenticationFailed event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnAuthenticationFailed: THandlerEvent;
```

Description

The OnAuthenticationFailed event is called when the client attempts to connect to the remote host, and a User ID is required, but one is not present, or an invalid User ID is provided. The purpose of this is to give the user the opportunity to rectify this problem.

If the Handled parameter is set to TRUE, the User ID is sent to the remote host again. If the User ID is still invalid, an exception is raised an the connect fails.

If the Handled parameter is set to FALSE (the default), an exception is raised and the connect fails.

See also

UserID property

OnFailure event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnFailure: TNotifyEvent;
```

Description

The OnFailure event is called when an outgoing E-Mail message is not sent successfully. If this event is executed, the E-Mail message that is being sent did not get delivered.

See also

OnSuccess event

OnSendStart event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnSendStart: TNotifyEvent;
```

Description

The OnSendStart event is called immediately before the outgoing E-Mail message is sent. This is the last time before the message is sent that it can be modified.

See also

OnFailure event
OnSuccess event

OnSuccess event

See also Example

Applies to

TNMSMTP component

Declaration

```
property OnSuccess: TNotifyEvent;
```

Description

The OnSuccess event is called when the outgoing E-Mail message has been delivered successfully.
If a message is not sent successfully the **OnFailure** event is called.
The **OnSendStart** event signifies the beginning of the message transmission.

See also

OnFailure event
OnSendStart event

TFileItem type

Unit
NMsmtp

Declaration
```
type
  TFileItem = procedure(Filename: string) of object;
```

Description
The TFileItem event type is used in cases where a filename needs to be passed as a parameter for the event.

THeaderInComplete type

Unit
NMsmtp

Declaration
```
type
  THeaderInComplete = procedure(var handled: boolean; hiType: integer) of
object;
```

Description
The THeaderInComplete event is called when a header item is missing. It is a modified THandler event, passing an integer in addition to the boolean Handled parameter.

TMailListReturn type

Unit
NMsmtp

Declaration
```
type
  TMailListReturn = procedure(MailAddress: string) of object;
```

Description
The TMailListReturn event type is used when an E-Mail address is needed as a parameter.

TRecipientNotFound type

Unit
NMsmtp

Declaration
```
type
  TRecipientNotFound = procedure(Recipient: string) of object;
```

Description
The TRecipientNotFound event type is used for passing an E-Mail address to the event as a parameter when the recipient can not be found.

TSubType type

Unit
NMsmtp

Declaration
```
type
  TSubType = (mtPlain, mtEnriched, mtSgml, mtTabSeperated, mtHtml);
```

Description
The TSubType type is used for specifying a sub-type for e-mail documents.

mtPlain: Plain Text
mtEnriched: Rich Text Format
mtSgml: An SGML (Standard Generalized Markup Language) Document
mtTabSeperated: Tab Separated Text
mtHTML: An HTML (Hyper Text Markup Language) Document

Legend

- Run-time only
- Read-Only
- Published
- Protected
- Key item

Heirarchy

TObject
|
TPersistent
|
TComponent
|
TPowersock