NMFtp unit

The TNMFTP unit contains the TNMFTP component, and it's supporting types and objects.

Components
TNMFTP

Objects
TFTPDirectoryList
TFTPUnixList
TFTPNETWAREList
TFTPDOSList
TFTPVMSList
TFTPMVSList
TFTPVMList
TFTPMACOSList
TFTPAS400List
TFTPOTHERList

Types
TCmdType
TFailureEvent
TNMListItem
TSuccessEvent
TUnsupportedEvent

TFTPDirectoryList object

Properties Methods

Unit

NMFtp

Description

The **TFTPDirectoryList** object is the base class for all of the other FTP Directory List parsing objects. If you encounter an FTP host that does not comply with any of the predefined list objects currently available, you can use this class to derive your own list parser.

Each property of this object is a TStringList. The indexes of each property correspond with the same index in each of the other properties. So **name**[1] and **Attribute**[1] contain the name and attributes for the second (TStringLists are 0-based) item in the listing.

TFTPDirectoryList Properties

TFTPDirectoryList
Legend

▶	Attribute
▶	ModifDate
▶	name
▶	Size

TFTPDirectoryList Methods

TFTPDirectoryList
Legend

Clear
🔑 ParseLine

Attribute property

Applies to
TFTPDirectoryList object

Declaration
```
property Attribute: TStringlist;
```

Description
The **Attribute** property specifies the attributes of the list items.

ModifDate property

Applies to
TFTPDirectoryList object

Declaration
```
property ModifDate: TStringlist;
```

Description
The **ModifDate** property specifies the dates of the last time the files were modified.

name property

Applies to
TFTPDirectoryList object

Declaration
```
property name: TStringlist;
```

Description
The **name** property specifies the names of the files listed.

Size property

Applies to
TFTPDirectoryList object

Declaration
```
property Size: TStringlist;
```

Description
The **Size** property specifies the sizes, in bytes, of the files in the listing.

Clear method

Applies to
TFTPDirectoryList object

Declaration
```
procedure Clear;
```

Description
The **Clear** method clears the properties of the directory listing object.

Notes:
After a call to the clear method, each of the TStringLists contain no data.

ParseLine method

Applies to

TFTPDirectoryList object

Declaration

```
procedure ParseLine(Line: string); virtual;
```

Description

The **ParseLine** method parses a line from a directory listing, and separates the file name, size, modified date, and attributes into their respective properties in the object.

TFTPUnixList component

Unit

NMFtp

Description

The **TFTPUnixList** component provides directory list parsing for FTP hosts that are being run from a Unix platform.

TFTPNETWAREList component

Unit

NMFtp

Description

The **TFTPNETWAREList** component provides directory list parsing for FTP hosts that are being run from a Novell Netware platform.

TFTPDOSList component

Unit
NMFtp

Description
The **TFTPDOSList** component provides directory list parsing for FTP hosts that are being run from a DOS or Windows platform.

TFTPVMSList component

Unit

NMFtp

Description

The **TFTPVMSList** component provides directory list parsing for FTP hosts that are being run from a VMS platform.

TFTPMVSList component

Unit

NMFtp

Description

The **TFTPMVSList** component provides directory list parsing for FTP hosts that are being run from an MVS platform.

TFTPVMList component

Unit

NMFtp

Description

The **TFTPVMList** component provides directory list parsing for FTP hosts that are being run from a VM platform.

TFTPMACOSList component

Unit

NMFtp

Description

The **TFTPMACOSList** component provides directory list parsing for FTP hosts that are being run from a Macintosh platform.

TFTPAS400List component

Unit
NMFtp

Description
The **TFTPAS400List** component provides directory list parsing for FTP hosts that are being run from an AS/400 platform.

TFTPOTHERList component

Unit

NMFtp

Description

The **TFTPOTHERList** component provides directory list parsing for FTP hosts that are being run on an unknown FTP host. This provides default directory list parsing that may or may not perform properly.

TNMFTP component

Heirarchy Properties Methods Events Tasks

Unit

NMFtp

Description

The purpose of the TNMFTP component is to transfer files to and from an internet/intranet FTP server via the FTP protocol. The use of this component requires a 32-bit TCP/IP stack, WSOCK32.DLL, which can be obtained from various vendors, including Microsoft, and is included with Windows 95, 98, and Windows NT.

TNMFTP Properties

TNMFTP
Legend

In TNMFTP

CurrentDir
FTPDirectoryList
OnListItem
ParseList
Password
UserID
Vendor

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

TNMFTP Methods

TNMFTP
Legend

In TNMFTP

- Allocate
- ChangeDir
- Delete
- DoCommand
- Download
- DownloadRestore
- List
- MakeDirectory
- Mode
- Nlist
- Reinitialize
- RemoveDir
- Rename
- Upload
- UploadAppend
- UploadRestore
- UploadUnique

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

TNMFTP Events

TNMFTP
Legend

In TNMFTP

- OnAuthenticationFailed
- OnAuthenticationNeeded
- OnFailure
- OnSuccess
- OnTransactionStart
- OnTransactionStop
- OnUnSupportedFunction

Derived from TPowersock

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

About the TNMFTP component

TNMFTP reference

Purpose

The purpose of the TNMFTP component is to transfer files to and from an internet/intranet FTP server via the FTP protocol.

RFC: RFC 959

Tasks

Before you can use the TNMFTP component for transferring files to and from a remote host, you must connect. This is accomplished by setting the **Host** and **Port** properties to those of a valid FTP server. Then, set the **UserID** and **Password** properties to a valid account on that server. Many servers will accept Anonymous as a user ID and your E-Mail address as a password for FTPing. After these properties are set, call the **Connect** method to connect to the server.

Getting a directory list on a remote host:
Once you are connected to the server, you can get a listing of the files and directories there by calling the **List** method, and writing an event handler for the **OnListItem** event.

Changing directory on a remote host:
You can change the directory you are currently in (called the 'working' directory) by calling the **ChangeDir** method, specifying a valid directory name to change to.

Uploading Files to a remote host:
To upload files to the directory you are currently in on the remote host, you call the **Upload** method. The Upload method takes the name of a file on your local computer, and a filename to store it as on the remote host as parameters. Please note, that you cannot upload to an FTP host unless you have sufficient rights to do so. This is most common allowed in a directory called 'incoming'. Also, if a file already exists on the server with the same name as the file name you specified in the Upload method, that file will be overwritten. See the **UploadUnique** method for a solution.

Downloading files from a remote host:
To download files from the remote host, you should do a List first, so that you know the file you are looking for will be available for download. Then, call the **Download** method, passing the name of the file you wish to download, and the path and name to save it to on your local drive as parameters. Please note, you will not be able to download files from everywhere on an FTP host. Most generally, you will be allowed to download many files from a directory called 'pub' (for public). Also, if a file with the same name exists on your local drive, it will be overwritten when you download the file from the remote host.

Create Directories on remote host:
To create a directory on the remote host, you must first have sufficient rights to do so. Most generally, you would be allowed to create directories in an 'incoming' directory on an FTP host. To create the directory, call the **MakeDirectory** method, and pass the name of the directory you wish to create as a parameter.

Remove directories on a remote host:
To remove a directory on the remote host, you must first have sufficient rights to do so. Once you do, you call the **RemoveDir** method, passing the name of the directory you wish to remove as a parameter.

CurrentDir property

See also Example

Applies to
TNMFTP component

Declaration
```
property CurrentDir: string;
```

Description
The **CurrentDir** property contains the name of the currently occupied directory on the remote system.

Scope: Public
Accessibility: RunTime, ReadOnly

Notes:
This property will be different after a successful call to the **ChangeDir** method.

See also

ChangeDir method
OnFailure event
OnSuccess event

Example

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

Place 3 TEdits, 10 TButtons, a TMemo, a TStringGrid, a TStatusBar, and a TNMFTP on the form.

Component Descriptions:

Edit1: FTP host
Edit2: FTP User ID
Edit3: FTP Password (set the PasswordChar property to *)
Button1: Connect/Disconnect
Button2: List
Button3: Change Directory
Button4: Make Directory
Button5: Remove Directory
Button6: Delete File
Button7: Rename File
Button8: Reinitialize
Button9: Authenticate
Button10: Get Current Directory
Memo1: Status Display
StringGrid1: Directory Listing Display
StatusBar1: Transfer progress *

* Set the **SimplePanel** property of StatusBar1 to TRUE in the Object Inspector
* Set the **ParseList** property of NMFTP1 to TRUE in the Object Inspector

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if NMFTP1.Connected then
    NMFTP1.Disconnect
  else
    begin
      NMFTP1.Vendor := NMOS_AUTO;
      NMFTP1.Host := Edit1.Text;
      NMFTP1.UserID := Edit2.Text;
      NMFTP1.Password := Edit3.Text;
      NMFTP1.Connect;
    end;
end;
```

When Button1 is clicked, if there is already a connection present, the **Disconnect** method is used to close the connection. If there is no connection present, the **Vendor** property is set to NMOS_AUTO, which will auto-detect the FTP host vendor, if possible. The **Host** property is set to the host name or IP address entered in Edit1. The **UserID** property is set to the username that has been entered in Edit2. The **Password** property is set to the value of Edit3, and a connection is established with the **Connect** method.

Insert the following code into Button2's OnClick event:

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

```
  NMFTP1.List;
end;
```

When Button2 is clicked, the **List** method is executed to get a listing of files and directories in the current directory. Since the **ParseList** property was previously set to TRUE, the directory listing will be parsed and it's fields placed in the **FTPDirectoryList** property.

Insert the following code into Button3's OnClick event:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  TheDir: String;
begin
  if InputQuery('Change Directory', 'Which directory?', TheDir) then
    NMFTP1.ChangeDir(TheDir);
end;
```

When Button3 is clicked, the **InputQuery** function is used to obtain the name of a directory. If the Ok button is clicked, the **ChangeDir** method attempts to change the current directory to the directory specified.

Insert the following code into Button4's OnClick event:

```
procedure TForm1.Button4Click(Sender: TObject);
var
  TheDir: String;
begin
  if InputQuery('Create Directory', 'Directory name?', TheDir) then
    NMFTP1.MakeDirectory(TheDir);
end;
```

When Button4 is clicked, the **InputQuery** function is used to obtain the name of a directory. If the Ok button is clicked, the **MakeDirectory** method attempts to create the directory specified.

Insert the following code into Button5's OnClick event:

```
procedure TForm1.Button5Click(Sender: TObject);
var
  TheDir: String;
begin
  if InputQuery('Remove Directory', 'Directory name?', TheDir) then
    NMFTP1.RemoveDir(TheDir);
end;
```

When Button5 is clicked, the **InputQuery** function is used to obtain the name of a directory. If the Ok button is clicked, the **RemoveDir** method attempts to remove the specified directory from the remote host.

Insert the following code into Button6's OnClick event:

```
procedure TForm1.Button6Click(Sender: TObject);
var
```

```
  TheFile: String;
begin
  if InputQuery('Delete File', 'File name?', TheFile) then
    NMFTP1.Delete(TheFile);
end;
```

When Button6 is clicked, the **InputQuery** function is used to obtain the name of a file. If the Ok button is clicked, the **Delete** method attempts to delete the file specified.

Insert the following code into Button7's OnClick event:

```
procedure TForm1.Button7Click(Sender: TObject);
var
  OldFile,
  NewFile: String;
begin
  if InputQuery('Rename file', 'File to rename?', OldFile) then
    if InputQuery('Rename file', 'New file name?', NewFile) then
      NMFTP1.Rename(OldFile, NewFile);
end;
```

When Button7 is clicked, the **InputQuery** function is called twice. The first time, the user is prompted for the file to rename. If the user enters a filename and clicks the Ok button, the second InputQuery asks for the new name for the file. If the OK button is also clicked here, the **Rename** method attempts to rename the file specified.

Insert the following code into Button8's OnClick event:

```
procedure TForm1.Button8Click(Sender: TObject);
begin
  ShowMessage('After reinitilizing, you must click the authenticate button');
  NMFTP1.Reinitialize;
end;
```

When Button8 is clicked, the **ShowMessage** procedure is used to display a warning that once the **Reinitialize** method is called, authentication is required. When the Ok button is clicked on the message, the Reinitialize method attempts to reinitialize the connection with the server back to the authentication state.

Insert the following code into Button9's OnClick event:

```
procedure TForm1.Button9Click(Sender: TObject);
var
  AnID,
  APass: String;
begin
  if InputQuery('Authentication needed', 'Enter User ID', AnID) then
    if InputQuery('Authentication needed', 'Enter Password', APass) then
      begin
        NMFTP1.DoCommand('USER '+AnID);
        NMFTP1.DoCommand('PASS '+APass);
      end;
end;
```

When Button9 is clicked the **InputQuery** function is used to obtain the user's User ID. If the Ok button is clicked, InputQuery is called again to obtain the user's password. If the Ok button is clicked a second time, the **DoCommand** method is used to send the USER and PASS commands to the remote host, effectively logging in to the FTP host.
***Please note that this is the **only** way to continue an FTP session once the **Reinitialize** method has been called.

Insert the following code into Button10's OnClick event:

```
procedure TForm1.Button10Click(Sender: TObject);
begin
  ShowMessage(NMFTP1.CurrentDir);
end;
```

When Button10 is clicked, the ShowMessage procedure is used to display the value of the **CurrentDir** property, which contains the current working directory on the remote host.

Insert the following code into NMFTP1's OnAuthenticationFailed event:

```
procedure TForm1.NMFTP1AuthenticationFailed(var Handled: Boolean);
var
  ThePass,
  TheID: String;
begin
  if MessageDlg('Authentication Failed. Retry?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
      ThePass := NMFTP1.Password;
      TheID := NMFTP1.UserID;
      InputQuery('Reauthenticate', 'Enter User ID', TheID);
      InputQuery('Reauthenticate', 'Enter Password', ThePass);
      NMFTP1.Password := ThePass;
      NMFTP1.UserID := TheID;
      Handled := TRUE;
    end;
end;
```

When authentication fails on the remote host (Password and UserID don't match, aren't correct), the **OnAuthenticationFailed** event is called. In this instance, a dialog box is displayed using the **MessageDlg** function. If the user clicks the Yes button to attempt authentication again, the **InputQuery** function is used to obtain a new UserID and Password. The **Password** and **UserID** properties are set to the new values accordingly, and the **Handled** parameter of the event is set to TRUE to allow the component to reauthenticate.

Insert the following code into NMFTP1's OnAuthenticationNeeded event:

```
procedure TForm1.NMFTP1AuthenticationNeeded(var Handled: Boolean);
var
  APass,
  AnID: String;
begin
  if NMFTP1.Password = '' then
    begin
```

```
      if InputQuery('Password needed', 'Enter password: ', APass) then
        begin
          NMFTP1.Password := APass;
          Handled := TRUE;
        end
      else
        Handled := FALSE;
    end;
  if NMFTP1.UserID = '' then
    begin
      if InputQuery('User ID needed', 'Enter User ID: ', AnID) then
        begin
          NMFTP1.UserID := AnID;
          Handled := TRUE;
        end
      else
        Handled := FALSE;
    end;
end;
```

If either the **UserID** or **Password** property are blank, the **OnAuthenticationNeeded** event is called. In this instance, if the Password property is blank, the **InputQuery** function is used to get a password. If the Ok button is clicked, the Password property is set to the new password, and the **Handled** parameter is set to FALSE. If the **UserID** property is blank, the InputQuery function is used to get a UserID. If the Ok button is clicked, the UserID property is set to the new value, and the **Handled** parameter is set to TRUE. If a password and/or user ID are not supplied, Handled is set to FALSE, which will cause an exception to be raised, and the connection to be cancelled.

Insert the following code into NMFTP1's OnConnect event:

```
procedure TForm1.NMFTP1Connect(Sender: TObject);
begin
  Memo1.Lines.Add('Connected');
  Button1.Caption := 'Disconnect';
end;
```

The **OnConnect** event is called once a connection has been established with the remote FTP host. In this case, Memo1 is updated to display the connection notice, and the caption of Button1 is set to Disconnect, since Button1 is used for connecting and disconnecting.

Insert the following code into NMFTP1's OnDisconnect event:

```
procedure TForm1.NMFTP1Disconnect(Sender: TObject);
begin
  Memo1.Lines.Add('Disconnected');
  Button1.Caption := 'Connect';
end;
```

When the **OnDisconnect** event is called when the connection to the remote FTP host has been terminated, the disconnection notice is added to Memo1 (status display memo), and Button1's caption is set to Connect, since Button1 is used both for connecting and disconnecting.

Insert the following code into NMFTP1's OnFailure event:

```
procedure TForm1.NMFTP1Failure(var Handled: Boolean; Trans_Type: TCmdType);
begin
  case Trans_Type of
    cmdChangeDir: Memo1.Lines.Add('ChangeDir failed');
    cmdMakeDir: Memo1.Lines.Add('MakeDir failed');
    cmdDelete: Memo1.Lines.Add('Delete failed');
    cmdRemoveDir: Memo1.Lines.Add('RemoveDir failed');
    cmdList: Memo1.Lines.Add('List failed');
    cmdRename: Memo1.Lines.Add('Rename failed');
    cmdUpRestore: Memo1.Lines.Add('UploadRestore failed');
    cmdDownRestore: Memo1.Lines.Add('DownloadRestore failed');
    cmdDownload: Memo1.Lines.Add('Download failed');
    cmdUpload: Memo1.Lines.Add('Upload failed');
    cmdAppend: Memo1.Lines.Add('UploadAppend failed');
    cmdReInit: Memo1.Lines.Add('Reinitialize failed');
    cmdAllocate: Memo1.Lines.Add('Allocate failed');
    cmdNList: Memo1.Lines.Add('NList failed');
    cmdDoCommand: Memo1.Lines.Add('DoCommand failed');
    cmdCurrentDir: Memo1.Lines.Add('CurrentDir failed');
  end;
end;
```

The **OnFailure** event is called when a command has failed to execute properly. In this instance, the **Trans_Type** parameter is checked, and Memo1 is updated to display which command failed to execute. The **Handled** parameter is left to FALSE (the default), so an exception will be raised due to the failure.

Insert the following code into NMFTP1's OnPacketRecvd event:

```
procedure TForm1.NMFTP1PacketRecvd(Sender: TObject);
begin
  StatusBar1.SimpleText := 'Received '+IntToStr(NMFTP1.BytesRecvd)+' bytes of
'+IntToStr(NMFTP1.BytesTotal)+' total';
end;
```

When data is received during a file transfer, the **OnPacketRecvd** event is called. This example updates the Statusbar StatusBar1 to display how many bytes of the total transfer have been received.

Insert the following code into NMFTP1's OnPacketSent event:

```
procedure TForm1.NMFTP1PacketSent(Sender: TObject);
begin
  StatusBar1.SimpleText := 'Sent '+IntToStr(NMFTP1.BytesSent)+' bytes of
'+IntToStr(NMFTP1.BytesTotal)+' total';
end;
```

When data is sent to the remote host during a file transfer, the **OnPacketSent** event is called. This example updates the Statusbar StatusBar1 to display how many bytes of the total transfer have been sent.

Insert the following code into NMFTP1's OnTransactionStart event:

```
procedure TForm1.NMFTP1TransactionStart(Sender: TObject);
```

```
begin
  Memo1.Lines.Add('Data transfer start');
end;
```

When a data transaction takes place in the TNMFTP component, the **OnTransactionStart** event is called to signify the data transaction's beginning. This example updates Memo1 to display that the data transfer has started.

Insert the following code into NMFTP1's OnTransactionStop event:

```
procedure TForm1.NMFTP1TransactionStop(Sender: TObject);
begin
  Memo1.Lines.Add('Data transfer end');
end;
```

When a data transaction completes in the TNMFTP component, the **OnTransactionStop** event is called to signify the data transaction's end. This example updates Memo1 to display that the data transfer has finished.

Insert the following code into NMFTP1's OnSuccess event:

```
procedure TForm1.NMFTP1Success(Trans_Type: TCmdType);
var
  I: Integer;
begin
  case Trans_Type of
    cmdList:
      begin
        for I := 0 to (StringGrid1.ColCount - 1) do
          StringGrid1.Cols[I].Clear;
        StringGrid1.RowCount := NMFTP1.FTPDirectoryList.name.Count;
        StringGrid1.ColCount := 4;
        StringGrid1.Cells[0, 0] := 'Filename';
        StringGrid1.Cells[1, 0] := 'File Size';
        StringGrid1.Cells[2, 0] := 'Modified Date';
        StringGrid1.Cells[3, 0] := 'Attributes';
        for I := 0 to (NMFTP1.FTPDirectoryList.name.Count - 1) do
          with NMFTP1.FTPDirectoryList do
            begin
              StringGrid1.Cells[0, I+1] := name[I];
              StringGrid1.Cells[1, I+1] := Size[I];
              StringGrid1.Cells[2, I+1] := ModifDate[I];
              StringGrid1.Cells[3, I+1] := Attribute[I];
            end;
      end;
    cmdChangeDir:
      begin
        Memo1.Lines.Add('ChangeDir successful');
        NMFTP1.List;
      end;
    cmdMakeDir: Memo1.Lines.Add('MakeDir successful');
    cmdRemoveDir: Memo1.Lines.Add('RemoveDir successful');
    cmdDelete: Memo1.Lines.Add('Delete successful');
    cmdRename: Memo1.Lines.Add('Rename successful');
    cmdReInit: Memo1.Lines.Add('Reinitialize successful');
    cmdCurrentDir: Memo1.Lines.Add('CurrentDir successful');
  end;
end;
```

When an FTP command has completed successfully, the **OnSuccess** event is called. In this case, the **Trans_Type** parameter is tested, and Memo1 is updated to display the success of the command.
If the command was a call to the **List** method, StringGrid1 is used to display the resulting directory list upon successful completion by using the **FTPDirectoryList** property to separate the different fields of the directory listing. If the command was a call to the **ChangeDir** method, the new directory listing is automatically initiated by the **List** method.

Insert the following code into NMFTP1's OnUnSupportedFunction event:

```
procedure TForm1.NMFTP1UnSupportedFunction(Trans_Type: TCmdType);
begin
  case Trans_Type of
    cmdChangeDir: Memo1.Lines.Add('ChangeDir not supported by this server');
    cmdMakeDir: Memo1.Lines.Add('MakeDir not supported by this server');
    cmdDelete: Memo1.Lines.Add('Delete not supported by this server');
    cmdRemoveDir: Memo1.Lines.Add('RemoveDir not supported by this server');
    cmdList: Memo1.Lines.Add('List not supported by this server');
    cmdRename: Memo1.Lines.Add('Rename not supported by this server');
    cmdUpRestore: Memo1.Lines.Add('UploadRestore not supported by this server');
    cmdDownRestore: Memo1.Lines.Add('DownloadRestore not supported by this server');
    cmdDownload: Memo1.Lines.Add('Download not supported by this server');
    cmdUpload: Memo1.Lines.Add('Upload not supported by this server');
    cmdAppend: Memo1.Lines.Add('UploadAppend not supported by this server');
    cmdReInit: Memo1.Lines.Add('Reinitialize not supported by this server');
    cmdAllocate: Memo1.Lines.Add('Allocate not supported by this server');
    cmdNList: Memo1.Lines.Add('NList not supported by this server');
    cmdDoCommand: Memo1.Lines.Add('DoCommand not supported by this server');
    cmdCurrentDir: Memo1.Lines.Add('CurrentDir not supported by this server');
  end;
end;
```

When an FTP command is not supported by the remote FTP host, the **OnUnSupportedFunction** event is called. In this case, the **Trans_Type** parameter is tested, and Memo1 is updated to display the command that was unsupported.

FTPDirectoryList property

See also Example

Applies to

TNMFTP component

Declaration

```
property FTPDirectoryList: TFTPDirectoryList;
```

Description

The **FTPDirectoryList** property is used only when the **ParseList** property is set to TRUE. FTPDirectoryList contains the directory listing obtained from the **List** method, with each of the elements of the listing separated into properties. See the **TFTPDirectoryList** object reference for details on the properties of this object.

See also

List method
OnListItem event

OnListItem event

See also Example

Applies to

TNMFTP component

Declaration

```
property OnListItem: TNMListItem;
```

Description

The **OnListItem** event is called when a directory listing is taking place. This event is called once for each item that gets listed.

Event Parameters:
The **Listing** parameter is the item being currently listed.

Notes:
This event gets called by the **List** and **NList** methods.

See also

FTPDirectoryList property
List method
NList method
ParseList property

Example

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

Place 3 TEdits, 9 TButtons, 2 TMemos, a TStatusBar, a TRadioGroup, and a TNMFTP on the form.

Component Descriptions:

Edit1: Host
Edit2: User ID
Edit3: Password
Button1: Connect/Disconnect
Button2: NList
Button3: ChangeDir
Button4: Download
Button5: Upload
Button6: UploadAppend
Button7: UploadUnique
Button8: Upload Restore
Button9: Abort
Memo1: Status Window
Memo2: Directory List display
RadioGroup1: Transfer Mode *
StatusBar1: File Transfer progress

* Change RadioGroup1.Items to MODE_ASCCI, MODE_IMAGE, and MODE_BYTE, in that order.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if NMFTP1.Connected then
    NMFTP1.Disconnect
  else
    begin
      NMFTP1.Host := Edit1.Text;
      NMFTP1.UserID := Edit2.Text;
      NMFTP1.Password := Edit3.Text;
      NMFTP1.Connect;
    end;
end;
```

When Button1 is clicked, if the **Connected** property is TRUE, a connection is present, and the **Disconnect** method is called to disconnect from the remote FTP host. If Connected is FALSE, there is no connection present, so the **Host** property is set to the value in Edit1 to specify the remote host to connect to. The **UserID** property is set to the value in Edit2, and the **Password** property is set to the value in Edit3. The UserID and Password are used to log in to the remote host when connected.

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Memo2.Clear;
  NMFTP1.NList;
```

```
end;
```

When Button2 is clicked, any text being displayed in Memo2 is cleared. The **NList** method is called, retrieving the names of all files and directories in the current working directory. The results of this method call can be seen in the **OnListItem** event.

Insert the following code into Button3's OnClick event:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  TheDir: String;
begin
  if InputQuery('Change directory', 'Directory name?', TheDir) then
    NMFTP1.ChangeDir(TheDir);
end;
```

When Button3 is clicked, the **InputQuery** function is used to obtain the name of the directory desired. If the Ok button is clicked, the **ChangeDir** method is used to change the current working directory to the directory specified.

Insert the following code into Button4's OnClick event:

```
procedure TForm1.Button4Click(Sender: TObject);
var
  RemoteFile,
  LocalFile: String;
  O: TOpenDialog;
begin
  if InputQuery('Download a file', 'File to download: ', RemoteFile) then
    begin
      O := TOpenDialog.Create(Self);
      try
        O.Title := 'Save file as';
        if O.Execute then
          begin
            LocalFile := O.FileName;
            case RadioGroup1.ItemIndex of
              0: NMFTP1.Mode(MODE_ASCII);
              1: NMFTP1.Mode(MODE_IMAGE);
              2: NMFTP1.Mode(MODE_BYTE);
            end;
            NMFTP1.Download(RemoteFile, LocalFile);
          end;
      finally
        O.Free;
      end;
    end;
end;
```

When Button4 is clicked, the InputQuery function is used to retrieve the name of a remote file to download to the local computer. if the Ok button is clicked, an OpenDialog is used to retrieve the name to store the file as on the local computer. If the Open button is clicked, the **Mode** method is called, changing the transfer mode depending on the selected item in RadioGroup1. Then the **Download** method is used to download the remote file to the local disk.

Insert the following code into NMFTP1's OnDisconnect event:

```
procedure TForm1.NMFTP1Disconnect(Sender: TObject);
begin
  Memo1.Lines.Add('Disconnected');
  Button1.Caption := 'Connect';
end;
```

When the local computer disconnects from the remote host, the **OnDisconnect** event is called. In this instance, the status display (Memo1) is updated to display the disconnect status, and the caption of Button1 is changed to Connect (Since Button1 is used for connecting and disconnecting).

Insert the following code into NMFTP1's OnPacketRecvd event:

```
procedure TForm1.NMFTP1PacketRecvd(Sender: TObject);
begin
  StatusBar1.SimpleText := IntToStr(NMFTP1.BytesRecvd)+' bytes received out of
'+IntToStr(NMFTP1.BytesTotal);
end;
```

When data is received from the remote host during a file transfer, the **OnPacketRecvd** event is called. In this instance, StatusBar1 displays the progress of the transfer by displaying the **BytesRecvd** property, which contains the number of bytes received, and the **BytesTotal** property, which contains the total number of bytes to transfer.

Insert the following code into NMFTP1's OnPacketSent event:

```
procedure TForm1.NMFTP1PacketSent(Sender: TObject);
begin
  StatusBar1.SimpleText := IntToStr(NMFTP1.BytesSent)+' bytes sent out of
'+IntToStr(NMFTP1.BytesTotal);
end;
```

When data is sent to the remote host during a file transfer, the **OnPacketSent** event is called. In this instance, StatusBar1 displays the progress of the transfer by displaying the **BytesSent** property, which contains the number of bytes sent, and the **BytesTotal** property, which contains the total number of bytes to transfer.

Insert the following code into NMFTP1's OnTransactionStart event:

```
procedure TForm1.NMFTP1TransactionStart(Sender: TObject);
begin
  Memo1.Lines.Add('Transaction Start');
end;
```

When a data transfer begins, the **OnTransactionStart** event is called. In this instance, the Status Memo, Memo1, displays that the transaction started.

Insert the following code into NMFTP1's OnTransactionStop event:

```
procedure TForm1.NMFTP1TransactionStop(Sender: TObject);
begin
  Memo1.Lines.Add('Transaction Stop');
end;
```

When a data transfer completes, the **OnTransactionStop** event is called. In this instance, the Status Memo, Memo1, displays that the transaction has finished.

Insert the following code into Button5's OnClick event:

```
procedure TForm1.Button5Click(Sender: TObject);
var
  LocalFile,
  RemoteFile: String;
  F: File of Byte;
  FSize: Integer;
  O: TOpenDialog;
begin
  O := TOpenDialog.Create(Self);
  try
    O.Title := 'Select file to upload';
    if O.Execute then
      if InputQuery('Choose Remote File Name', 'Filename?', RemoteFile) then
        begin
          LocalFile := O.FileName;
          case RadioGroup1.ItemIndex of
            0: NMFTP1.Mode(MODE_ASCII);
            1: NMFTP1.Mode(MODE_IMAGE);
            2: NMFTP1.Mode(MODE_BYTE);
          end;
          AssignFile(F, LocalFile);
          Reset(F);
          FSize := FileSize(F);
          CloseFile(F);
          NMFTP1.Allocate(FSize);
          NMFTP1.Upload(LocalFile, RemoteFile);
        end;
  finally
    O.Free;
  end;
end;
```

When Button5 is clicked, an Open Dialog is displayed to select a file on the local computer to upload to the remote host. If the Open button is clicked, the InputQuery function is used to retrieve the name the file will be stored as on the remote host. If the Ok button is clicked, the file is opened so it's size can be determined. The **Allocate** method is called to reserve space on the remote host for the file. This is not normally necessary, but is done here to illustrate the use of the method. The **Upload** method is used to send the file to the remote host.

Insert the following code into Button6's OnClick event:

```
procedure TForm1.Button6Click(Sender: TObject);
var
  LocalFile,
  RemoteFile: String;
  O: TOpenDialog;
begin
  O := TOpenDialog.Create(Self);
  try
    O.Title := 'Select file to upload';
    if O.Execute then
      if InputQuery('Choose Remote File Name', 'Filename?', RemoteFile) then
        begin
          LocalFile := O.FileName;
          case RadioGroup1.ItemIndex of
            0: NMFTP1.Mode(MODE_ASCII);
            1: NMFTP1.Mode(MODE_IMAGE);
            2: NMFTP1.Mode(MODE_BYTE);
          end;
          NMFTP1.UploadAppend(LocalFile, RemoteFile);
        end;
  finally
    O.Free;
  end;
end;
```

When Button6 is clicked, an Open Dialog is displayed to select a file on the local computer to upload to the remote host. If the Open button is clicked, the InputQuery function is used to retrieve the name the file will be stored as on the remote host. If the Ok button is clicked, the **UploadAppend** method is used to send the file to the remote host. If a file with the same name already exists on the remote host, the file being sent is appended to the end of the existing file.

Insert the following code into Button7's OnClick event:

```
procedure TForm1.Button7Click(Sender: TObject);
var
  LocalFile: String;
  O: TOpenDialog;
begin
  O := TOpenDialog.Create(Self);
  try
    O.Title := 'Upload file';
    if O.Execute then
      begin
        LocalFile := O.FileName;
        NMFTP1.UploadUnique(LocalFile);
      end;
  finally
    O.Free;
  end;
end;
```

When Button7 is clicked, an Open Dialog is displayed to select the file on the local computer to upload to the remote host. If the Open button is clicked, the **UploadUnique** method is used to store the file on the remote FTP host with a unique filename given by the FTP host.

Insert the following code into NMFTP1's OnFailure event:

```
procedure TForm1.NMFTP1Failure(var Handled: Boolean; Trans_Type: TCmdType);
begin
  case Trans_Type of
    cmdChangeDir: Memo1.Lines.Add('ChangeDir failure');
    cmdMakeDir: Memo1.Lines.Add('MakeDir failure');
    cmdDelete: Memo1.Lines.Add('Delete failure');
    cmdRemoveDir: Memo1.Lines.Add('RemoveDir failure');
    cmdList: Memo1.Lines.Add('List failure');
    cmdRename: Memo1.Lines.Add('Rename failure');
    cmdUpRestore: Memo1.Lines.Add('UploadRestore failure');
    cmdDownRestore: Memo1.Lines.Add('DownloadRestore failure');
    cmdDownload: Memo1.Lines.Add('Download failure');
    cmdUpload: Memo1.Lines.Add('Upload failure');
    cmdAppend: Memo1.Lines.Add('UploadAppend failure');
    cmdReInit: Memo1.Lines.Add('Reinitialize failure');
    cmdAllocate: Memo1.Lines.Add('Allocate failure');
    cmdNList: Memo1.Lines.Add('NList failure');
    cmdDoCommand: Memo1.Lines.Add('DoCommand failure');
    cmdCurrentDir: Memo1.Lines.Add('CurrentDir failure');
  end;
end;
```

When an FTP command fails, the **OnFailure** event is called. In this case, the **Handled** parameter is left the default (FALSE), so an exception will be raised, in addition to a failure notification being added to the status display (Memo1) noting the command that failed.

Insert the following code into NMFTP1's OnSuccess event:

```
procedure TForm1.NMFTP1Success(Trans_Type: TCmdType);
begin
  case Trans_Type of
    cmdChangeDir: Memo1.Lines.Add('ChangeDir success');
    cmdMakeDir: Memo1.Lines.Add('MakeDir success');
    cmdDelete: Memo1.Lines.Add('Delete success');
    cmdRemoveDir: Memo1.Lines.Add('RemoveDir success');
    cmdList: Memo1.Lines.Add('List success');
    cmdRename: Memo1.Lines.Add('Rename success');
    cmdUpRestore: Memo1.Lines.Add('UploadRestore success');
    cmdDownRestore: Memo1.Lines.Add('DownloadRestore success');
    cmdDownload: Memo1.Lines.Add('Download success');
    cmdUpload: Memo1.Lines.Add('Upload success');
    cmdAppend: Memo1.Lines.Add('UploadAppend success');
    cmdReInit: Memo1.Lines.Add('Reinitialize success');
    cmdAllocate: Memo1.Lines.Add('Allocate success');
    cmdNList: Memo1.Lines.Add('NList success');
    cmdDoCommand: Memo1.Lines.Add('DoCommand success');
    cmdCurrentDir: Memo1.Lines.Add('CurrentDir success');
  end;
end;
```

When an FTP command succeeds, the **OnSuccess** event is called. In this instance, the command that succeeded is displayed in the status display (Memo1).

Insert the following code into Button8's OnClick event:

```
procedure TForm1.Button8Click(Sender: TObject);
var
  LocalFile,
  RemoteFile: String;
  FPosition: Integer;
  FPos: String;
  O: TOpenDialog;
begin
  O := TOpenDialog.Create(Self);
  try
    O.Title := 'Select file to upload';
    if O.Execute then
      if InputQuery('Choose Remote File Name', 'Filename?', RemoteFile) then
        if InputQuery('Choose restoration point', 'Byte Count: ', FPos) then
          begin
            FPosition := StrToInt(Fpos);
            LocalFile := O.FileName;
            case RadioGroup1.ItemIndex of
              0: NMFTP1.Mode(MODE_ASCII);
              1: NMFTP1.Mode(MODE_IMAGE);
              2: NMFTP1.Mode(MODE_BYTE);
            end;
            NMFTP1.UploadRestore(LocalFile, RemoteFile, FPosition);
          end;
  finally
    O.Free;
  end;
end;
```

When Button8 is clicked, an OpenDialog is displayed to select a file on the local computer to upload to the remote host. If the Open button is clicked, the InputQuery function is used to retrieve the name the file will use on the remote host. The InputQuery function is called once more to obtain the position to restore the upload from. If the Ok button is clicked, the **Mode** method is used to set the mode as determined by RadioGroup1. The **UploadRestore** method continues a previously interrupted upload at the position specified.

Insert the following code into Button9's OnClick event:

```
procedure TForm1.Button9Click(Sender: TObject);
begin
  NMFTP1.Abort;
end;
```

When Button9 is clicked, the **Abort** method is called, aborting the current operation.

ParseList property

See also Example

Applies to
TNMFTP component

Declaration

```
property ParseList: boolean;
```

Description
The **ParseList** property determines whether to parse incoming directory listings into the **FTPDirectoryList** property or not.

Scope: Published
Accessability: Runtime, Designtime

Note:
If this property is set to FALSE, directory listings will not be parsed out, and will need to be handled in the **OnListItem** event.

See also

FTPDirectoryList property
List method
OnListItem event

Password property

See also Example

Applies to
TNMFTP component

Declaration
```
property Password: string;
```

Description
The **Password** property specifies the password used to log in to the remote FTP host.

Scope: Published
Accessability: Runtime, Designtime

Notes:
If the password supplied is invalid, the **OnAuthenticationFailed** event is called.
If there is no password supplied, the **OnAuthenticationNeeded** event is called.
The password supplied must correspond with the user ID specified by the **UserID** property.

See also

OnAuthenticationFailed event
OnAuthenticationNeeded event
UserID property

UserID property

See also Example

Applies to
TNMFTP component

Declaration

```
property UserID: string;
```

Description
The **UserID** property specifies the user ID to use when logging on to the remote FTP host.

Scope: Published
Accessability: Runtime, designtime

Notes:
If the User ID supplied is invalid, the **OnAuthenticationFailed** event is called.
If there is no User ID supplied, the **OnAuthenticationNeeded** event is called.
The User ID supplied must correspond with the password specified by the **Password** property.

See also

OnAuthenticationFailed event
OnAuthenticationNeeded event
Password property

Vendor property

See also Example

Applies to

TNMFTP component

Declaration

```
property Vendor: integer;
```

Description

The **Vendor** property specifies the vendor of the FTP host that is being connected to. This allows the component to parse directory listings sent from the host in the proper manner. If the Vendor is not known, the default value should auto detect the vendor for you.

Default: NMOS_AUTO

Range: Any of the following constants may be used for the **Vendor** property in your code:

NMOS_UNIX
NMOS_WINDOWS
NMOS_VM
NMOS_BULL
NMOS_MAC
NMOS_TOPS20
NMOS_VMS
NMOS_OS2
NMOS_MVS_IBM
NMOS_MVS_INTERLINK
NMOS_OTHER
NMOS_AUTO
NMOS_NT
NMOS_TANDEM
NMOS_AS400
NMOS_OS9
NMOS_NETWARE

Notes:

If the Vendor property is set to NMOS_AUTO, the component will try to auto-detect the type of host.

See also

FTPDirectoryList property

Allocate method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure Allocate(FileSize: Integer);
```

Description

The **Allocate** method is used for allocating space on the remote host for storing files.

Parameters:

The **FileSize** parameter specifies the number of bytes to allocate on the server for the incoming file.

Notes:

This method is not required under most circumstances. It has been included to comply with the RFC, and to provide support for servers that require disk space to be allocated before an upload can be performed.

See also

OnFailure event
OnSuccess event
OnUnSupportedFunction event
Upload method

ChangeDir method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure ChangeDir(DirName: string);
```

Description

The **ChangeDir** method changes the current (also called "working") directory.

Parameters:
The **DirName** parameter specifies the directory name to change to. This can be a directory that resides in the current working directory, or a full path.

Notes:
When the **ChangeDir** method is called, the **CurrentDir** property will change.
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called.
In either case, cmdChangeDir is passed as the **Trans_Type** parameter in the event.

See also

CurrentDir property
OnFailure event
OnSuccess event
OnUnSupportedFunction event

Delete method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure Delete(Filename: string);
```

Description

The **Delete** method deletes a file from the remote host.

Parameters:

The **FileName** parameter specifies the file on the remote host to delete. This can be the name of a file in the current working directory, or the path and filename of a file elsewhere on the remote system.

Notes:

You must have write priviledges in the current working directory in order to delete files.
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called.
In either case, cmdDelete is passed as the **Trans_Type** parameter in the event.

See also

OnFailure event
OnSuccess event
OnUnSupportedFunction event

DoCommand method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure DoCommand(CommandStr: string);
```

Description

The **DoCommand** method is used for sending a command to the remote FTP host. This method is especially useful for use with custom FTP hosts, that may have non-standard commands, or for support of unimplemented commands.

Parameters:

The **CommandStr** parameter is the command that is to be sent to the server. This can be as simple as 'PWD' (command for obtaining the name of the current directory), or as complex as a multi-file download.

Notes:

This method is used internally in the component as well for executing many of the commands available.

See also

OnFailure event
OnSuccess event

Download method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure Download(RemoteFile, LocalFile: string);
```

Description
The **Download** method is used for downloading files from the remote FTP host to the local computer.

Parameters:
The **RemoteFile** parameter specifies the name of a file in the current working directory on the remote host to be downloaded.
The **LocalFile** parameter specifies the name the file will have when retrieved and stored on the local machine. Please note, if a file with the same name already exists locally, **the existing file is overwritten.**
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdDownload is passed as the **Trans_Type** parameter in the event.

See also

Mode method
OnFailure event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
DownloadRestore method
Upload method

DownloadRestore method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure DownloadRestore(RemoteFile, LocalFile: string);
```

Description

The DownloadRestore method continues a file transfer that was previously interrupted. This can only be done if there is a part of the file downloaded onto the local drive.

Parameters:

The **RemoteFile** parameter specifies the name of the file to continue downloading.
The **LocalFile** parameter specifies the name of the file on the local drive that contains part of the remote file.

Notes:

Not all hosts support DownloadRestore.
If this command is successful, the **OnSuccess** event will be called, otherwise the **OnFailure** event will be called.

See also

Download method
Mode method
OnFailure event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
OnUnSupportedFunction event
UploadRestore method

Example

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

Place 7 TButtons, 3 TEdits, a TMemo, a TOpenDialog, a TStatusBar, and a TNMFTP on the form.

Component Descriptions:
Button1: Connect
Button2: Disconnect
Button3: Abort
Button4: Download
Button5: DownloadRestore
Button6: Change Directory
Button7: List
Edit1: FTP Host Name
Edit2: FTP User ID
Edit3: FTP Password
Memo1: Directory listing box

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NMFTP1.Host := Edit1.Text;
  NMFTP1.UserID := Edit2.Text;
  NMFTP1.Password := Edit3.Text;
  NMFTP1.Connect;
end;
```

Insert the following code into NMFTP1's OnConnect event:

```
procedure TForm1.NMFTP1Connect(Sender: TObject);
begin
  StatusBar1.SimpleText := 'Connected';
  NMFTP1.Mode(MODE_IMAGE);
end;
```

When the client connects to the remote FTP host, the **OnConnect** method is called. Here, the status bar is updated to inform the user of the connection, and the **Mode** method is called to set the data transfer mode to MODE_IMAGE.

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  NMFTP1.Disconnect;
end;
```

Insert the following code into NMFTP1's OnDisconnect event:

```
procedure TForm1.NMFTP1Disconnect(Sender: TObject);
begin
```

```
  StatusBar1.SimpleText := 'Disconnected';
end;
```

Insert the following code into NMFTP1's OnListItem event:

```
procedure TForm1.NMFTP1ListItem(Listing: String);
begin
  Memo1.Lines.Add(Listing);
end;
```

Insert the following code into Button3's OnClick event:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  NMFTP1.Abort;
end;
```

Insert the following code into Button4's OnClick event:

```
procedure TForm1.Button4Click(Sender: TObject);
var
  RemoteFileName: String;
begin
  if InputQuery('Download', 'Remote File: ', RemoteFileName) then
    if OpenDialog1.Execute then
      NMFTP1.Download(RemoteFileName, OpenDialog1.FileName);
end;
```

Insert the following code into NMFTP1's OnPacketRecvd event:

```
procedure TForm1.NMFTP1PacketRecvd(Sender: TObject);
begin
  StatusBar1.SimpleText := IntToStr(NMFTP1.BytesRecvd)+' bytes out of '+IntToStr(NMFTP1.BytesTotal)
+' received';
end;
```

Insert the following code into NMFTP1's OnTransactionStart event:

```
procedure TForm1.NMFTP1TransactionStart(Sender: TObject);
begin
  StatusBar1.SimpleText := 'Beginning Data Transfer';
end;
```

Insert the following code into NMFTP1's OnTransactionStop event:

```
procedure TForm1.NMFTP1TransactionStop(Sender: TObject);
begin
  StatusBar1.SimpleText := 'Data Transfer complete';
end;
```

Insert the following code into NMFTP1's OnUnSupportedFunction event:

```
procedure TForm1.NMFTP1UnSupportedFunction(Trans_Type: TCmdType);
begin
  if Trans_Type = cmdDownRestore then
    ShowMessage('DownloadRestore not supported');
end;
```

If the **DownloadRestore** method is not supported by the server currently connected to, the **OnUnSupportedFunction** event is called. Here, a message box is shown stating that the DownloadRestore method is unsupported.

Insert the following code into NMFTP1's OnSuccess event:

```
procedure TForm1.NMFTP1Success(Trans_Type: TCmdType);
begin
  if Trans_Type = cmdDownRestore then
    StatusBar1.SimpleText := 'DownloadRestore successful';
  if Trans_Type = cmdChangeDir then
    StatusBar1.SimpleText := 'Directory changed to '+NMFTP1.CurrentDir;
end;
```

If the **DownloadRestore** or **Download** method executes successfully, the **OnSuccess** event updates the status bar to display the success.

Insert the following code into Button6's OnClick event:

```
procedure TForm1.Button6Click(Sender: TObject);
var
  Dir: String;
begin
  if InputQuery('Change Directory', 'Directory Name: ', Dir) then
    NMFTP1.ChangeDir(Dir);
end;
```

Insert the following code into Button7's OnClick event:

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  Memo1.Clear;
  Memo1.Lines.Add('Listing of '+NMFTP1.CurrentDir);
  Memo1.Lines.Add('----------------------------');
  NMFTP1.List;
end;
```

Insert the following code into Button5's OnClick event:

```
procedure TForm1.Button5Click(Sender: TObject);
var
  RemoteFileName: String;
begin
```

```
  if InputQuery('Download Restore', 'Remote File: ', RemoteFileName) then
    if OpenDialog1.Execute then
      NMFTP1.DownloadRestore(RemoteFileName, OpenDialog1.FileName);
end;
```

When Button5 is clicked, the InputQuery function is used to retrieve the name of the remote file to download. If a filename is entered, and the ok button clicked, the OpenDialog is executed so the user can choose the file to continue downloading. If the user clicks the Open button, the **DownloadRestore** method is called to continue the interrupted download.

Running this example:
After compiling and running this example, connect to any give FTP server. Begin the download of a file using the download button, and once the download is underway, click the Abort button. Reconnect to the server, and the click the download restore button. Select the same remote file, and local file name and location as you did with the download, and the file will continue downloading.

List method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure List;
```

Description
The **List** method is used for getting a list of files and directories from the remote host. The listing retrieved is for the current working directory.

Notes:
For each item listed, the **OnListItem** event is called.
If the **ParseList** property is set to TRUE, the **FTPDirectoryList** property will contain the elements for each listing, including name, size, and attributes.
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdList is passed as the **Trans_Type** parameter in the event.

See also

FTPDirectoryList property
NList method
OnFailure event
OnListItem event
OnSuccess event
OnTransactionStart event
OnTransactionStop event

MakeDirectory method

See also   Example

Applies to
TNMFTP component

Declaration
```
procedure MakeDirectory(DirectoryName: string);
```

Description
The **MakeDirectory** method creates a directory in the current working directory on the remote FTP host.

Parameters:
The **DirectoryName** parameter specifies the name of the directory to create on the remote host. This can be a single directory name to be created in the current working directory, or a full directory path.

Notes:
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdMakeDir is passed as the **Trans_Type** parameter in the event.

See also

OnFailure event
OnSuccess event
OnUnSupportedFunction event
RemoveDir method

Mode method

See also Example

Applies to
TNMFTP component

Declaration

```
procedure Mode(TheMode: Integer);
```

Description
The **Mode** method changes the file transfer mode used when transfering files between the remote host and the local computer.

Parameters:
The **TheMode** parameter specifies the transfer mode to use. Any of the following values may be used:

MODE_ASCII - Sends data as ASCII text
MODE_IMAGE - raw binary data in 8-bit bytes
MODE_BYTE - raw binary data using variable-length bytes

See also

Download method
DownloadRestore method
Upload method
UploadAppend method
UploadRestore method
UploadUnique method

Nlist method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure Nlist;
```

Description

The **NList** method is used to retrieve just the names of files and directories in the current working directory.

Notes:

For each item that gets listed, the **OnListItem** event is called.
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called.
In either case, cmdNList is passed as the **Trans_Type** parameter in the event.

See also

List method
OnFailure event
OnListItem event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
OnUnSupportedFunction event

Reinitialize method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure Reinitialize;
```

Description
The **Reinitialize** method is used to reset the connection with the server. This leaves the connection with the remote host at the point right before the Authentication (UserID and Password) takes place.

Notes:
After a call to Reinitialize, the connection is left in an unusable state. Before proceeding with further FTP transactions, you must resend the user ID and password. Otherwise, any commands executed will fail. See the **Example** for more details.

See also

OnFailure event
OnSuccess event
OnUnSupportedFunction event
Password property
UserID property

RemoveDir method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure RemoveDir(DirectoryName: string);
```

Description
The **RemoveDir** method is used to remove a directory on the remote FTP host.

Parameters:
The **DirectoryName** parameter specifies the name of the directory to remove. This can be the name of a directory that resides within the current working directory, or a full directory path.

Notes:
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdRemoveDir is passed as the **Trans_Type** parameter in the event.

See also

MakeDirectory method
OnFailure event
OnSuccess event
OnUnSupportedFunction event

Rename method

See also Example

Applies to

TNMFTP component

Declaration

```
procedure Rename(Filename, FileName2: string);
```

Description

The **Rename** method is used to rename a file in the current working directory on the remote FTP host.

Parameters:

The **FileName** parameter specifies the name of the file that will be renamed.
The **FileName2** parameter specifies the new name for the file.
These parameters can be the name of a file in the current working directory, or the path and filename of a file elsewhere on the system.

Notes:

If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdRename is passed as the **Trans_Type** parameter in the event.

See also

OnFailure event
OnSuccess event
OnUnSupportedFunction event

Upload method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure Upload(LocalFile, RemoteFile: string);
```

Description
The **Upload** method is used for sending a file on the local computer to the remote host.

Parameters:
The **LocalFile** parameter specifies the name of the file on the local computer that will be sent to the remote host.
The **RemoteFile** parameter specifies the name the file will have on the remote host when it is sent.

Notes:
Please note, if a file with the same name already exists on the remote host, **the existing file is overwritten.**
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdUpload is passed as the **Trans_Type** parameter in the event.

See also

Download method
OnFailure event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
OnUnSupportedFunction event
UploadAppend method
UploadRestore method
UploadUnique method

UploadAppend method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure UploadAppend(LocalFile, RemoteFile: string);
```

Description
The **UploadAppend** method is used for storing a file on the local computer on the remote FTP host. If a file with the same name already exists, the new file is appended to the end of the existing file.

Parameters:
The **LocalFile** parameter specifies the name of the file on the local computer to send to the remote host.
The **RemoteFile** parameter specifies the name the file will have on the remote FTP host. If a file with this name already exists, the local file is appended to the end of the existing remote file.

Notes:
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdAppend is passed as the **Trans_Type** parameter in the event.

See also

Download method
OnFailure event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
OnUnSupportedFunction event
Upload method
UploadRestore method
UploadUnique method

UploadRestore method

See also Example

Applies to
TNMFTP component

Declaration
```
procedure UploadRestore(LocalFile, RemoteFile: string; Position: Integer);
```

Description
The **UploadRestore** method is used for continuing the upload of a file on the local computer to the remote FTP host.

Parameters:
The **LocalFile** parameter specifies the name of the file on the local computer to send the the remote host.
The **RemoteFile** parameter specifies the name the file will have when sent to the remote host. When using this method, this file should already partly exist on the remote FTP host.
The **Position** parameter specifies how far into the file to start the transfer. If 100 bytes were sent successfully before the transfer was interrupted previously, the position would be 101.

Notes:
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdUpRestore is passed as the **Trans_Type** parameter in the event.

See also

Download method
OnFailure event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
OnUnSupportedFunction event
Upload method
UploadAppend method
UploadUnique method

UploadUnique method

See also Example

Applies to
TNMFTP component

Declaration

```
procedure UploadUnique(LocalFile: string);
```

Description

The **UploadUnique** method is used for upload a file to the remote host. The name of the file on the local computer is used for the filename on the remote FTP host. If a file with the same name already exists, a unique name is used for the file.

Parameters:
The **LocalFile** parameter specifies the file on the local computer that will be sent to the remote host.

Notes:
If the file specifies already exists on the remote host, the file will be given a new name when it is uploaded.
If this command succeeds, the **OnSuccess** event will be called, otherwise the **OnFailure** event is called. In either case, cmdUpload is passed as the **Trans_Type** parameter in the event.

See also

Download method
OnFailure event
OnSuccess event
OnTransactionStart event
OnTransactionStop event
OnUnSupportedFunction event
Upload method
UploadAppend method
UploadRestore method

OnAuthenticationFailed event

See also Example

Applies to

TNMFTP component

Declaration

```
property OnAuthenticationFailed: THandlerEvent;
```

Description

The **OnAuthenticationFailed** event is called when either the **UserID** property or the **Password** property are invalid. This could be due to an invalid account name (UserID), or an invalid password.

Event Parameters:

If the **Handled** parameter is set to TRUE, authentication will be attempted again to complete the login with the remote host. If the **Handled** parameter is set to FALSE (the default), or if the second attempt at logging in fails, an exception will be raised.

See also

OnAuthenticationNeeded event
Password property
UserID property

OnAuthenticationNeeded event

See also Example

Applies to
TNMFTP component

Declaration

```
property OnAuthenticationNeeded: THandlerEvent;
```

Description
The **OnAuthenticationNeeded** event is called when either the **UserID** property or the **Password** property are blank.

Event Parameters:
If the **Handled** parameter is left FALSE, an exception will be raised, and the connection will be aborted.
If the **Handled** parameter is set to TRUE, and a UserID and/or Password are provided, the authentication will be attempted again. If it fails a second time, an exception will be raised and the connection is aborted.

Notes:
This event is typically called when the **Connect** method is called, and there is no user ID and/or password supplied.

See also

OnAuthenticationFailed event
Password property
UserID property

OnFailure event

See also Example

Applies to
TNMFTP component

Declaration
```
property OnFailure: TFailureEvent;
```

Description
The **OnFailure** event is called if an executed FTP command fails.

Event Parameters:
If the **Handled** parameter is set to TRUE, an exception is not raised.
If the **Handled** parameter is set to FALSE, an exception will be raised due to the failing command.
The **Trans_Type** parameter specifies the command that failed. The following are the possible values for this parameter:

cmdChangeDir
cmdMakeDir
cmdDelete
cmdRemoveDir
cmdList
cmdRename
cmdUpRestore
cmdDownRestore
cmdDownload
cmdUpload
cmdAppend
cmdReInit
cmdAllocate
cmdNList
cmdDoCommand
cmdCurrentDir

See also

Allocate method
ChangeDir method
CurrentDir property
Delete method
DoCommand method
Download method
DownloadRestore method
List method
MakeDirectory method
NList method
OnSuccess event
Reinitialize method
RemoveDir method
Rename method
Upload method
UploadAppend method
UploadRestore method

OnSuccess event

See also Example

Applies to

TNMFTP component

Declaration

```
property OnSuccess: TSuccessEvent;
```

Description

The **OnSuccess** event is called when a command completes successfully.

Event Parameters:

The **Trans_Type** parameter specifies the command that completed successfully. The possible values for this parameter are listed below:

cmdChangeDir
cmdMakeDir
cmdDelete
cmdRemoveDir
cmdList
cmdRename
cmdUpRestore
cmdDownRestore
cmdDownload
cmdUpload
cmdAppend
cmdReInit
cmdAllocate
cmdNList
cmdDoCommand
cmdCurrentDir

See also

Allocate method
ChangeDir method
CurrentDir property
Delete method
DoCommand method
Download method
DownloadRestore method
List method
MakeDirectory method
NList method
OnFailure event
Reinitialize method
RemoveDir method
Rename method
Upload method
UploadAppend method
UploadRestore method

OnTransactionStart event

See also Example

Applies to

TNMFTP component

Declaration

```
property OnTransactionStart: TNotifyEvent;
```

Description

The **OnTransactionStart** event is called each time data is sent from the remote host to the local computer using the Data socket.

Notes:

The following methods will execute the OnTransactionStart event:

- List
- Download
- UploadUnique
- Upload
- NList
- DownloadRestore
- UploadAppend
- UploadRestore

See also

Download method
DownloadRestore method
List method
NList method
OnTransactionStop event
UploadUnique method
Upload method
UploadAppend method
UploadRestore method

OnTransactionStop event

See also Example

Applies to

TNMFTP component

Declaration

```
property OnTransactionStop: TNotifyEvent;
```

Description

The **OnTransactionStop** event is called when a data transfer from the remote FTP host to the local computer has completed.

Notes:
The following methods will cause the OnTransactionStop event to execute:

List
Download
UploadUnique
Upload
NList
DownloadRestore
UploadAppend
UploadRestore

See also

Download method
DownloadRestore method
List method
NList method
OnTransactionStart event
UploadUnique method
Upload method
UploadAppend method
UploadRestore method

OnUnSupportedFunction event

See also Example

Applies to

TNMFTP component

Declaration

```
property OnUnSupportedFunction: TUnsupportedEvent;
```

Description

The **OnUnSupportedFunction** event is called when an FTP command fails to execute because it is not implemented on the remote FTP host.

Event Parameters:

The Trans_Type parameter specifies which command failed to execute. The possibles values for this parameter are listed below.

cmdChangeDir
cmdMakeDir
cmdDelete
cmdRemoveDir
cmdList
cmdRename
cmdUpRestore
cmdDownRestore
cmdDownload
cmdUpload
cmdAppend
cmdReInit
cmdAllocate
cmdNList
cmdDoCommand
cmdCurrentDir

See also

DoCommand method
Download method
List method
Rename method
UploadUnique method
Upload method

TCmdType type

Unit
NMFtp

Declaration
```
type
  TCmdType = (cmdChangeDir, cmdMakeDir, cmdDelete, cmdRemoveDir, cmdList,
cmdRename, cmdUpRestore, cmdDownRestore, cmdDownload, cmdUpload, cmdAppend,
cmdReInit, cmdAllocate, cmdNList, cmdDoCommand, cmdCurrentDir);
```

Description
The **TCmdType** type provides a simple way of referencing the commands available to the TNMFTP component.

Member Descriptions:

cmdChangeDir - ChangeDir method
cmdMakeDir - MakeDirectory method
cmdDelete - Delete method
cmdRemoveDir - RemoveDir method
cmdList - List method
cmdRename - Rename method
cmdUpRestore - UploadRestore method
cmdDownRestore - DownloadRestore method
cmdDownload - Download method
cmdUpload - Upload and UploadUnique methods
cmdAppend - UploadAppend method
cmdReInit - Reinitialize method
cmdAllocate - Allocate method
cmdNList - NList method
cmdDoCommand - DoCommand method
cmdCurrentDir - accessing the CurrentDir property

TFailureEvent type

Unit

NMFtp

Declaration

```
type
  TFailureEvent = procedure(var Handled: Boolean; Trans_Type: TCmdType) of
object;
```

Description

The **TFailureEvent** event type is used for the **OnFailure** event.
The **Handled** parameter determines whether the component's default actions will be carried out or not.
See event descriptions for a more detailed description of the role the Handled parameter takes.
The **Trans_Type** parameter specifies the command that failed.

TNMListItem type

Unit
NMFtp

Declaration
```
type
  TNMListItem = procedure(Listing: string) of object;
```

Description
The **TNMListItem** event type is used for listing strings items one by one in an event. This is used for the **OnListItem** event, which returns listings from the **List** method and the **NList** method.
The **Listing** parameter contains the current item that has been listed.

TSuccessEvent type

Unit
NMFtp

Declaration
```
type TSuccessEvent = procedure(Trans_Type: TCmdType) of object;
```

Description
The **TSuccessEvent** event type is used by the **OnSuccess** event when a command executes successfully.
The **Trans_Type** parameter specifies the command that executed successfully.

TUnsupportedEvent type

Unit
NMFtp

Declaration
```
type
  TUnsupportedEvent = procedure(Trans_Type: TCmdType) of object;
```

Description
The **TUnsupportedEvent** event type is used in the **OnUnSupportedFunction** event. This passes the command that is unsupported to the event for handling by the application developer.

Legend

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

Heirarchy

TObject
|
TPersistent
|
TComponent
|
TPowersock