Psock unit

The PSock unit contains TPowersock, the base class for the FastNet Tools for Delphi, and TNMGeneralServer, a generic internet server that can be used to derive your own custom servers. It also contains a few routines that may be useful for Internet Application Development.

Components

TPowersock
TNMGeneralServer

Routines

NthPos
NthWord
StreamLn

 TPowersock component

Heirarchy——————Properties Methods Events Tasks

Unit

Psock

Description

The TPowersock component is the base class for many of the other components in the FastNet Tools for Delphi toolset. It can be used as a base for creating controls that deal with other protocols, or for creating custom protocols.

TPowersock Properties
TPowersock
Legend

In 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

TPowersock Methods
TPowersock
Legend

In 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

TPowersock Events
TPowersock
Legend

In TPowersock

- OnAccept
- OnConnect
  - OnConnectionFailed
- OnConnectionRequired
- OnDisconnect
- OnError

OnHostResolved
OnInvalidHost
OnPacketRecvd
OnPacketSent
OnRead
OnStatus

About the TPowersock component

TPowersock reference

Purpose

The TPowersock component serves as the base class for TCP communications using various internet protocols. It can be used to create new controls that handle protocols not yet supported by the FastNet Tools for Delphi, or to create a new protocol to solve a custom problem.
NOTE It is not advised that TPowersock be used as a stand-alone component.

Tasks

It is not advised that the TPowersock component be used as a stand-alone component. Instead, another class should be derived from the TPowersock component. Once another component has been derived from TPowersock, the client can connect by calling the Connect method to connect to the remote host once the Host and Port properties have been set. Data can be received from the remote host by calling the read, ReadLn, CaptureFile, CaptureStream, and CaptureString methods. Data can be sent to the remote host by calling the write, writeln, SendFile, and SendStream methods. The Transaction method both sends data to the remote host, and reads the reply from the server.

If you are planning on creating a new component descending from the TPowersock component, see the source code for more details on the inner workings of TPowersock. If you do not have the source code for the FastNet Tools for Delphi, visit the NetMasters website for information on how to obtain it at **http://www.netmastersllc.com**

BeenCanceled property

See also Example

Declaration

```
property BeenCanceled: boolean;
```

Description

The BeenCanceled property is TRUE if the current operation has been canceled, and FALSE if the current operation has not been canceled.

Scope: Public
Accessibility: Runtime, Readonly

See also

Cancel method

BeenTimedOut property

See also Example

Declaration

```
property BeenTimedOut: boolean;
```

Description

The BeenTimedOut property is TRUE if the current operation has timed out, or FALSE if the current operation has not timed out.

Scope: Public
Accessibility: Runtime, Readonly

See also

TimeOut property

BytesRecvd property

See also Example

Declaration

```
property BytesRecvd: longint;
```

Description

The BytesRecvd property contains the number of bytes received from the current data transfer.

Scope: Public
Accessibility: Runtime, Readonly

See also

BytesSent property
BytesTotal property
CaptureStream method
CaptureString method
CaptureFile method
OnPacketRecvd event
OnPacketSent event

Example

The OnPacketRecvd event is defined as Public in TPowersock. It is exposed as Published in some of the protocol specific components that are derived from TPowersock.

To recreate this example, you will need to create a new blank Delphi application. Drop a TStatusBar, a TButton, and a TNMHTTP component onto the form, and set the TStatusBar's SimplePanel property to **true**.

Select the TNMHTTP component, and insert the following code into the OnPacketRecvd event on the events tab in the Object Inspector.

```
procedure TForm1.NMHTTP1PacketRecvd(Sender: TObject);
begin
  StatusBar1.SimpleText := IntToStr(NMHTTP1.BytesRecvd)+' bytes out of
'+IntToStr(NMHTTP1.BytesTotal)+' transferred';
end;
```

Select the TButton, and insert the following code into the button's OnClick event

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NMHTTP1.Get('http://www.netmastersllc.com');
end;
```

Running the application and clicking the button should make the statusbar update as it retrieves the document using the TNMHTTP component.

Example Description:
In this example, when Button1 is clicked, the OnPacketRecvd event is updating the status bar to inform the user when data comes in, by displaying how many bytes out of the total transfer have been received by the client.

BytesSent property

See also Example

Declaration

```
property BytesSent: longint;
```

Description

The BytesSent property contains the number of bytes sent during the current data transaction.

Scope: Public
Accessibility: Runtime, Readonly

See also

BytesRecvd property
BytesTotal property
OnPacketRecvd event
OnPacketSent event
SendStream method
SendFile method

Example

The OnPacketSent event is defined as Public in TPowersock. It is exposed as Published in some of the protocol specific components that are derived from TPowersock.

To recreate this example, you will need to create a new blank Delphi application. Drop a TStatusBar, a TButton, 3 TEdit controls, and a TNMSMTP component onto the form, and set the TStatusBar's SimplePanel property to **true**.

Select the TNMSMTP component, and insert the following code into the OnPacketSent event on the events tab in the Object Inspector.

```
procedure TForm1.NMSMTP1PacketRecvd(Sender: TObject);
begin
  StatusBar1.SimpleText := IntToStr(NMSMTP1.BytesSent)+' bytes out of
'+IntToStr(NMSMTP1.BytesTotal)+' transferred';
end;
```

Insert the following code into the TNMSMTP's OnConnect event:

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

Insert the following code into the TNMSMTP's OnSuccess event:

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

Select the TButton, and insert the following code into the button's OnClick event

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NMSMTP1.Host := Edit1.Text;
  NMSMTP1.UserID := Edit2.Text;
  NMSMTP1.PostMessage.FromAddress := Edit3.Text;
  NMSMTP1.PostMessage.ToAddress.Text := Edit3.Text;
  NMSMTP1.PostMessage.Subject := 'Testing BytesSent';
  for I := 1 to 10 do
    NMSMTP1.PostMessage.Body.Add('Test line '+IntToStr(I));
  NMSMTP1.Connect;
end;
```

Example Description:
Now run the application. In Edit1, type the name or IP address of your SMTP server. If you do not know what this is, ask your network administrator. Type your user ID for the SMTP server (if required) into Edit2. Type your E-Mail address into Edit3. When you click Button1, you will be sending an E-Mail to yourself, and when the packet is sent (in this example, there is only one), the status bar will update to display how many bytes were sent.

BytesTotal property

See also Example

Declaration

```
property BytesTotal: longint;
```

Description

The BytesTotal property contains the total number of bytes to send or receive in the current data transaction.

Scope: Public
Accessibility: Runtime, Readonly

See also

BytesSent property
BytesRecvd property
CaptureStream method
CaptureString method
CaptureFile method
OnPacketRecvd event
OnPacketSent event
SendStream method
SendFile method

Connected property

Example

Declaration

```
property Connected: boolean;
```

Description

The Connected property is TRUE if the client is currently connected to the remote host, and FALSE if the client is not connected.

Scope: Public
Accessibility: Runtime, Readonly

Handle property

Example

Declaration

```
property Handle: tSocket;
```

Description

The Handle property contains the value of the socket currently being used for WinSock communications. This property can be used if WinSock API calls need to be made directly.

Scope: Public
Accessibility: Runtime, Readonly

Example

To recreate this example, you will need to create a new blank Delphi application. Drop a TButton and a TNMHTTP on the form.

In the USES clause at the top of the unit, add Winsock to the list of units used.

Select the button, and in the OnClick event, insert the following code

```
procedure TForm1.Button1Click(Sender: TObject);
var
  gudtLinger:Tlinger;
begin
  gudtLinger.l_onoff := 1;
  gudtLinger.l_linger := 0;
  setsockopt(NMHTTP1.Handle, SOL_SOCKET, SO_LINGER, @gudtLinger, 4);
end;
```

In this example, handle is used to directly access Winsock function setsockopt, which is used for setting socket options. For more information on the Winsock API, go to http://www.sockets.com

Host property

See also Example

Declaration

```
property Host: String;
```

Description

The Host property contains the name or dotted IP address of the remote host to connect to.

Scope: Published
Accessibility: Runtime, Designtime

See also

Connect method
Port property

LastErrorNo property

Example

Declaration

```
property LastErrorNo: integer;
```

Description

The LastErrorNo property contains the last socket error reported.

Scope: Public
Accessibility: Runtime

LocalIP property

See also Example

Declaration

```
property LocalIP: string;
```

Description

The LocalIP property contains the dotted IP address of the local computer. If more than one IP address is present, only the first address is returned.

Scope: Public
Accessibility: Runtime, Readonly

See also

RemoteIP property

Example

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

Drop a TButton, a TMemo, and a TPowersock on the form.

In the OnClick event for the TButton, insert the following code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 13;
  Powersock1.Connect;
end;
```

In the OnConnect event for the TPowersock control, insert the following code:

```
procedure TForm1.Powersock1Connect(Sender: TObject);
begin
  Memo1.Liens.Add('Connected');
  Memo1.Lines.Add('Local Address: '+Powersock1.LocalIP);
  Memo1.Lines.Add('Remote Address: '+Powersock1.RemoteIP);
end;
```

In the OnHostResolved event for the TPowersock control, insert the following code:

```
procedure TForm1.Powersock1HostResolved(Sender: TObject);
begin
  Memo1.Lines.Add('Host Resolved');
end;
```

In the OnStatus event for the TPowersock control, insert the following code:

```
procedure TForm1.Powersock1Status(Sender: TObject; Status: String);
begin
  Memo1.Lines.Add(Powersock1.Status);
  Memo1.Lines.Add('Last WinSock error: '+IntToStr(Powersock1.LastErrorNo));
  If Powersock1.BeenCanceled then
    Memo1.Lines.Add('Input/ouput operation canceled');
  If Powersock1.BeenTimedOut then
    Memo1.Lines.Add('Operation timed out');
end;
```

Example Description:
Run the application, and then click Button1.
When Button1 is clicked, a connection is established with www.netmastersllc.com on port 13. Once www.netmastersllc.com has been resolved to a valid IP address, the OnHostResolved event will state that fact by adding a line to Memo1. When the connection is established, the OnConnect event updates the Memo1 to inform the user that a connection has been established. Also, the dotted IP address of the local machine, the LocalIP property, is added to Memo1, in addition to the IP address of the remote computer, the RemoteIP property. As status messages are available, they are added to Memo1's lines, as well as the last Winsock error encountered. If the current operation has been canceled, the BeenCanceled property will be TRUE, and the cancellation will be displayed in Memo1. If the current operation has timed out, the BeenTimedOut property will be True, and a timeout message will be added to Memo1.

OnHostResolved event

Example

Declaration
```
property OnHostResolved: TOnHostResolved;
```

Description
The OnHost resolved event is called when the remote host address has been resolved to an IP address.

Notes:
If the specified host name or IP Address is invalid, the OnHostResolved event will not be triggered, and the OnInvalidHost event will be used instead

OnStatus event

See also Example

Declaration

```
property OnStatus: TOnStatus;
```

Description

The OnStatus event is called when there is a status change in the component.

See also

Status property

Port property

See also Example

Declaration

```
property Port: Integer;
```

Description

The Port property specified the port number of the remote host to connect to, or in the case of a server, the port number that the server will listen on.

Scope: Published
Accessibility: Runtime*, Designtime

Notes:
* In the case of the TNMGeneralServer, the Port property must be set during designtime.

The Create method for each component sets the Port property to the default port for that protocol.

See also

Connect method
Listen method

RemoteIP property

See also Example

Declaration

```
property RemoteIP: string;
```

Description

The RemoteIP property contains the dotted IP address of the remote host. This property is not set until the client has connected to the remote host.

Scope: Public
Accessibility: Runtime, Readonly

See also

LocalIP property

ReplyNumber property

See also Example

Declaration

```
property ReplyNumber: Smallint;
```

Description

The ReplyNumber property contains the numerical result from a transaction, if any.

Scope: Public
Accessibility: Runtime, Readonly

See also

Transaction method

Example

```
procedure TForm1.Button1Click(Sender: TObject);
begin
      try
        Transaction('GET /');
        if ReplyNumber<299 then
          raise Exception.create('Get transaction failed');
        else
          ....
end;
```

In this example a Get command is issued and if the reply has a ReplyNumber larger that 299 an error exception is raised.

ReportLevel property

See also Example

Declaration

```
property ReportLevel: integer;
```

Description

ReportLevel controls the amount of detail that is reported by the OnStatus event and the Status property.

Default: Status_Informational

Scope: Published
Accessibility: DesignTime, RunTime

Range: Status_None = 0
Status_Informational = 1
Status_Basic = 2
Status_Routines = 4
Status_Debug = 8
Status_Trace = 16

See also

OnStatus event
Status property

Status property

See also Example

Declaration
```
property Status: String;
```

Description
The Status property contains the last status message that was passed as the status parameter in the OnStatus event.

Scope: Public
Accessibility: RunTime, ReadOnly

See also

OnStatus event
ReportLevel property

TimeOut property

See also Example

Declaration

```
property TimeOut: Integer;
```

Description

The TimeOut property specifies the amount of time (in milliseconds) to wait for a response from the socket before an exception is raised and the current operation is aborted. If TimeOut is 0, an exception is never raised, and operations never time out.

Default: 0

Scope: Published
Accessibility: Designtime, Runtime

See also

BeenTimedOut property
CaptureStream method
CaptureString method
CaptureFile method
read method
readln method
SendFile method
SendStream method
write method
writeln method

TransactionReply property

See also Example

Declaration

```
property TransactionReply: String;
```

Description

The TransactionReply property contains the result from the last command sent to the server.

Scope: Public
Accessibility: Runtime, ReadOnly

See also

Transaction method

Example

```
procedure TForm1.Button1Click(Sender: TObject);
begin
      try
        Transaction('GET /');
        if ( TransactionReply='-ERR' ) then
          raise Exception.create('Get transaction failed');
        else
          ....
end;
```

In this example a Get command is issued and if the Reply is '-ERR' an error exception is raised.

WSAInfo property

Example

Declaration

```
property WSAInfo: TStringList;
```

Description

The WSAInfo property contains information about the current version of Winsock including version number and vendor.

Scope: Public
Accessibility: Runtime, ReadOnly

Example

```
procedure TForm1.Button4Click(Sender: TObject);
begin
   Memo3.Lines.Add(Powersock1.WSAInfo);
end;
```

In this example, when Button4 is clicked, the information on the current version of Winsock is displayed in Memo3.

Accept method

See also Example

Declaration

```
function Accept: Word; virtual;
```

Description

The Accept method accepts a connection from a remote computer that is requesting a connection. The socket of the accepted connection is the return value of the function.

Return Value:

The result of this function is the socket descriptor for the connecting client.

See also

Listen method

Example

Before you will be able to reproduce this example, you will first need to subclass the TPowersock component. The reason for this is to expose the OnAccept event as published to control acceptance of connections. Select Component | New Component from Delphi's menu. Use TPowersock as the ancestor, and name your new class TSPowersock. Modify the class definition so it appears as follows:

```
type
  TSPowersock = class(TPowersock)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
    property OnAccept;
  end;
```

Save this file, and install it into the component library by selecting Component | Install Component.

This example is not functioning properly, and will be rewritten.

read method

See also Example

Declaration

```
function read(value: word): string;
```

Description

The read method reads a specific number of bytes from the currently connected socket, and returns them as a string.

Parameters:

value = The value parameter specifies the number of bytes to read from the socket. If value is 0, then all of the data currently waiting to be read from the socket is returned.

See also

readln method
CaptureStream method
CaptureString method
CaptureFile method

Example

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

Place a TButton, TEdit, and a TPowersock on the form.

Insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 19;
  Powersock1.Connect;
  Edit1.Text := Powersock1.Read(100);
  Powersock1.Disconnect;
end;
```

Example Description:
When the application is run, and Button1 is clicked, Powersock1 connects to www.netmastersllc.com on port 19 (port 19 is the standard port for CharGen, which simply sends an endless stream of ASCII characters to the client). Edit1 is then filled with the first 100 characters sent from the remote host, which is read by the Read method, and Powersock1 disconnects.

ReadLn method

See also Example

Declaration

```
function ReadLn: string;
```

Description

The ReadLn method reads data from the socket until a Line Feed is encountered, and returns the data read as a string.

Warning:
If there is no carriage return/linefeed pair in the incoming data, the readln method will go on until it encounters them. . .which in some cases could be never.

See also

CaptureStream method
CaptureString method
read method

Example

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

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

Insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 7;
  Powersock1.Connect;
  Powersock1.Write(Edit1.Text+#13#10);
  Edit2.Text := Powersock1.Readln;
  Powersock1.Disconnect;
end;
```

Example Description:
When Button1 is clicked, Powersock1 connects to www.netmastersllc.com on port 7 (port 7 is the standard port for Echo, which just sends back any text sent to it). The write method is used to send whatever text is in Edit1, plus a carriage return (#13) and linefeed (#11), to the remote host. (Writeln could have been used, but this illustrates what the write method does) Edit2 is then filled with the carriage return/line feed terminated string the server sends back by using the readln method. Powersock1 then disconnects from the remote host.

Transaction method

See also Example

Declaration
```
function Transaction(const CommandString: String): String; virtual;
```

Description
The Transaction method sends a command to the remote host, and returns the reply from the host as a string. The ReplyNumber property will also be set to the numerical value of the reply, if any.

Parameters:
CommandString = The CommandString parameters specifies the command to send to the server.

See also

ReplyNumber property
TransactionReply property

Abort method

Declaration
```
procedure Abort; virtual;
```

Description
The Abort method aborts the current operation. This method is overridden in the descendant classes, because each protocol has different requirements to abort an operation.

Cancel method

See also Example

Declaration

```
procedure Cancel;
```

Description

The Cancel method cancels the current input or output operation, and disconnects from the remote host.

Note:
When the Cancel method is called, an exception is raised.

See also

BeenCanceled property
CaptureStream method
CaptureString method
CaptureFile method
read method
readln method
SendFile method
SendStream method
write method
writeln method

Example

To recreate this example, you will need to create a new blank Delphi application.
Drop 2 TButtons, a TNMSMTP, 3 TEdits, and a TMemo on a form.

Insert this code into the Button1's OnClick event:

```
procedure TFrm1.Button1Click(Sender: TObject);
begin
  NMSMTP1.Host := Edit1.Text
  NMSMTP1.UserID := Edit2.Text;
  NMSMTP1.PostMessage.ToAddress.Text := Edit3.Text;
  NMSMTP1.PostMessage.FromAddress := Edit3.Text;
  NMSMTP1.PostMessage.Subject := 'Test message');
  NMSMTP1.PostMessage.Body.Add('This is a test message');
  NMSMTP1.PostMessage.Body.Add('If this message is delivered, the Cancel button (Button2) wasn''t
pressed fast enough');
  NMSMTP1.Connect;
end;
```

Insert this code into NMSMTP1's OnConnect event:

```
procedure TForm1.NMSMTP1Connect(Sender: TObject);
begin
  Memo1.Lines.Add('Connected, sending message');
  NMSMTP1.SendMail;
end;
```

Insert this code into NMSMTP1's OnSuccess event:

```
procedure TForm1.NMSMTP1Success(Sender: TObject);
begin
  Memo1.Lines.Add('Success, you didn''t press Button2 fast enough');
  if NMSMTP1.Connected then
    NMSMTP1.Disconnect;
end;
```

Insert this code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Powersock1.Cancel;
end;
```

Insert this code into NMSMTP1's OnDisconnect event:

```
procedure TForm1.NMSMTP1Disconnect(Sender: TObject);
begin
  Memo1.Lines.Add('Disconnected');
end;
```

Example Description:
Run the application.
Type your E-Mail address into Edit3. Type the Host name or IP Address of your SMTP server into Edit1. If you do not know what this is, ask your network administrator. If you require a user ID when you connect to

your SMTP server, type that into Edit2, or else leave it blank.
When you have done this, click Button1. When the words Connected, sending message appear in Memo1, click Button2. When you click button 2, an exception is raised, and the OnDisconnect event is called, which notifies the user of a disconnect. If Button2 is not pressed before the message is successfully sent, the OnSuccess event adds a line to Memo1 notifying the user that the button was not pressed enough, and calls the Disconnect method to disconnect from the remote host if the client is still connected, as reported by the Connected property.

CaptureFile method

See also Example

Declaration

```
procedure CaptureFile(FileName: String);
```

Description

The CaptureFile method captures all data from a socket, and stores it in the file specified by the FileName parameter. Data will continue to be captured from the socket until the socket closes.

Parameters:

FileName = The FileName parameter specifies the name of the file to store the data in. This must be a valid filename and path (path is optional).

See also

CaptureStream method
CaptureString method

Example

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

Drop 2 TButtons, a TEdit, and a TPowersock onto the form.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 19;
  Powersock1.Connect;
end;
```

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Powersock1.Cancel;
  if Powersock1.BeenCanceled then
    ShowMessage('I/O Operation Canceled');
end;
```

Insert the following code into Powersock1's OnConnect event:

```
procedure TForm1.Powersock1Connect(Sender: TObject);
begin
  Powersock1.CaptureFile(Edit1.Text);
end;
```

Example Description:
When you run this application, type a valid file path and name into Edit1. When you click Button1, Powersock1 connects the www.netmastersllc.com on port 19, which is the CharGen port (CharGen just sends ASCII characters until the client disconnects). When the connection is established, the CaptureFile method begins capturing data from the socket and storing it in the file that was specified by Edit1. When Button2 is clicked, the Cancel method is called to Cancel the I/O operation and disconnect the socket. A message is displayed if the Cancel method is successful. Once Button2 has been clicked, there should be a file in the location specified containing just ASCII text.

CaptureStream method

See also Example

Declaration

```
procedure CaptureStream(MainStream: TStream; Size: longint);
```

Description

The CaptureStream method captures data from the socket until a specified number of bytes have been captured.

Parameters:

MainStream = The MainStream parameter specifies the stream to stored the data in
Size = The Size parameter specifies the number of bytes to read into the stream. If Size is -1, then data continues to be read from the socket into the stream until the socket closes.

See also

CaptureFile method
CaptureString method

Example

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

Place a TButton, a TPowersock, and a TMemo on the form.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 19;
  Powersock1.Connect;
end;
```

Insert the following code into Powersock1's OnConnect event:

```
procedure TForm1.Powersock1Connect(Sender: TObject);
var
  MyStream: TMemoryStream;
  S: String;
begin
  MyStream := TMemoryStream.Create;
  try
    Powersock1.CaptureStream(MyStream, 256);
    SetLength(S, MyStream.Size);
    MyStream.Write(S, MyStream.Size);
    Memo1.Text := S;
  finally
    MyStream.Free;
  end;
end;
```

Example Description:
When this example is run and Button1 is clicked, the application connects to www.netmastersllc.com on port 139 (port 139 is the standard port for CharGen, which just sends ASCII characters to the client). When the connection has been established, Powersock1's OnConnect event creates a memory stream, and uses the CaptureStream method to capture data from the remote host. This data is then placed into a string (since CharGen hosts just send ASCII characters), and the string is displayed in Memo1. Note that all code involving the TMemoryStream is placed within a **try**...**finally** block, so that the stream will be freed even if an error occurs.

CaptureString method

See also Example

Declaration

```
procedure CaptureString(var AString: String; Size: longint);
```

Description

The CaptureString method captures data of a specified size from the socket and stores it into a string.

Parameters:

AString = The AString parameter is the string variable that the captured data is stored in.
Size = the Size parameter specifies the number of bytes to read from the socket into the string. If Size is -1, then data continues to be read from the socket into the stream until the socket closes.

See also

CaptureFile method
CaptureStream method

Example

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

Place 2 TButtons, a TPowersock, and a TEdit on the form.

Insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 19;
  Powersock1.Connect;
end;
```

Insert this code into Powersock1's OnConnect event:

```
procedure TForm1.Powersock1Connect(Sender: TObject);
var
  S: String;
begin
  Powersock1.CaptureString(S, 25);
  Memo1.Text := S;
end;
```

Insert this code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Powersock1.Disconnect;
end;
```

Example Description:
When this application is run and Button1 is clicked, the client connects to the remote host www.netmastersllc.com on port 19 (port 19 is the standard port for CharGen, which simply sends an endless stream of ASCII characters to the client until disconnect). When the client has connected, the OnConnect event of Powersock1 captures a string of 25 characters from the remote host and stores it in S using the CaptureString method, passing S and 25 as the paramters. The text in S is then displayed in Memo1. Button2 then should be clicked to disconnect from the remote host.

CertifyConnect method

Example

Declaration

```
procedure CertifyConnect;
```

Description

The CertifyConnect method checks to see if the client is connected to the remote host. If there is no connection present, an OnConnectionRequired event is fired.

Notes:

The OnConnectionRequired event is protected in TPowersock, but is exposed in many of the protocol-specific components as a published event.

Example

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

Drop 3 TButtons, a TMemo, and a TPowersock on the form.

Insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 80;
  Powersock1.Connect;
end;
```

Insert this code into Powersock1's OnConnect event:

```
procedure TForm1.Powersock1Connect(Sender: TObject);
begin
  Memo1.Lines.Add('Connected');
end;
```

Insert this code into Powersock1's OnDisconnect event:

```
procedure TForm1.Powersock1Disconnect(Sender: TObject);
begin
  Memo1.Lines.Add('Disconnected');
end;
```

Insert this code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Powersock1.Disconnect;
end;
```

Insert this code into Button3's OnClick event:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  try
    Powersock1.CertifyConnect;
    Memo1.Lines.Add('Connection present');
  except
    Memo1.Lines.Add('Connection not present');
  end;
end;
```

Example Description:
In this example, Button1's purpose is to connect to the remote host, which in this example is www.netmastersllc.com When a connection is established, the OnConnect event will notify the user of the application by displaying the message Connected in Memo1. When Button2 is clicked, the client will disconnect from the remote host, and the OnDisconnect event will display the message Disconnected in Memo2. When Button3 is pressed the CertifyConnect method is called from within a try...except loop. If there is no connection present, the exception is handled by adding the message Connection not present

to Memo1. If there is a connection present, no exception is raised, and the message Connection present is added to Memo1, and the except part of the statement is bypassed.

Connect method

See also Example

Declaration

```
procedure Connect; virtual;
```

Description

The Connect method connects the client to the remote host.

Prerequisites:
Host property and Port property must first be set before this method can be called. In the protocol-specific components that descend from TPowersock, the Port property is already set to the protocol's standard port.

Notes:
If the host name specified is invalid an exception is raised if the Handled variable in the OnInvalidHost event is not set to true, or if the OnInvalidHost event does not set the invalid host name to a valid host name.

If the connection fails, the OnConnectionFailed event is called, and an exception is raised.

See also

Disconnect method
Host property
Port property

Disconnect method

See also Example

Declaration
```
procedure Disconnect; virtual;
```

Description
The Disconnect method disconnects the client from the remote host.

Notes:
When the client has disconnected successfully, the OnDisconnect event is called.

See also

Connect method

FilterHeader method

Example

Declaration

```
procedure FilterHeader(HeaderStream: TFileStream);
```

Description

The FilterHeader method filters out the MIME header from the data arriving at the connected socket, and stores it in HeaderStream.

Parameters:

HeaderStream = The HeaderStream parameter specifies the file stream to store the MIME header into.

Example

```
procedure TForm1.Button4Click(Sender: TObject);
var
  FStream: TFileStream;
begin
  FStream := TFileStream.Create('HEAD.HDR', fmCreate);
  try
    Powersock1.FilterHeader(FStream);
    Powersock1.CaptureFile('C:\DATA.FIL');
  finally
    FStream.Free;
  end;
end;
```

In this example, when Button4 is clicked, a filestream is created, and the MIME header from the incoming data is stored in the file HEAD.HDR. Then, the rest of the data coming in from the socket is stored in the file C:\DATA.FIL. Finally, the filestream is freed.

Listen method

See also Example

Declaration

```
procedure Listen(sync: boolean);
```

Description

The Listen method listens for TCP connections on the port specified by the Port property.

Parameters:

sync = if sync is FALSE, when a connection comes in, the OnAccept event is called, where the connection can either be accepted using the Accept method, or it can be ignored.
if sync is TRUE, the call to Accept must be made directly after the call to Listen to accept the connection.

See also

Accept method
OnAccept

RequestCloseSocket method

Example

Declaration

```
procedure RequestCloseSocket;
```

Description

The RequestCloseSocket method closes the currently active socket, but does not destroy it.

Example

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

Place 2 TButtons, a TMemo, and a TPowersock on the form.

Insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 19;
  Powersock1.Connect;
end;
```

Insert this code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Powersock1.RequestCloseSocket;
end;
```

Insert this code into Powersock1's OnConnect event:

```
procedure TForm1.Powersock1Connect(Sender: TObject);
begin
  Memo1.Lines.Add('Connected');
end;
```

Insert this code into Powersock1's OnDisconnect event:

```
procedure TForm1.Powersock1Disconnect(Sender: TObject);
begin
  Memo1.Lines.Add('Disconnected');
end;
```

Example Description:
When this application is run, clicking Button1 will connect the client to www.netmastersllc.com on port 19 (port 19 is the standard port for CharGen, which sends ASCII characters until the client disconnects). When the client has connected, Powersock1's OnConnect event adds a line reading Connected to Memo1. When Button2 is clicked, the RequestCloseSocket method attempts to close the socket, disconnecting the client. This causes Powersock1's OnDisconnect event to add a line that says Disconnected to Memo1.

SendBuffer method

See also Example

Declaration

```
procedure SendBuffer(value: PChar; buflen: word);
```

Description

The SendBuffer method sends data from a buffer to the remote host.

Parameters:

value = The value parameter specifies the buffer to send data from.
buflen = The buflen parameter specifies the number of bytes to send from the buffer

See also

write method
writeln method

SendFile method

See also Example

Declaration

```
procedure SendFile(FileName: String);
```

Description

The SendFile method sends the contents of a file to the remote host.

Parameters:

FileName = The FileName parameter specifies the filename (and optionally path) of the file to send to the remote host. FileName must contain a valid path and filename in order to succeed.

See also

SendStream method
write method
writeln method

SendStream method

See also Example

Declaration

```
procedure SendStream(MainStream: TStream);
```

Description

The SendStream method sends the contents of a stream to the remote host.

Parameters:

MainStream = The MainStream parameter specifies the stream that contains the data that is to be sent to the remote host. MainStream must contain a valid stream.

See also

SendFile method
write method
writeln method

write method

See also Example

Declaration

```
procedure write(value: String);
```

Description

The write method sends a string to the remote host.

Parameters:

value = The value parameter contains the string to be sent to the remote host
NOTE A Carriage Return/Line Feed is NOT appended to the end of the string passed.

See also

SendFile method
SendStream method
writeln method

writeln method

See also Example

Declaration

```
procedure writeln(value: string);
```

Description

The write method sends a string to the remote host. A Carriage Return/Line Feed is appended to the end of the string.

Parameters:

value = The value parameter contains the string to be sent to the remote host

See also

SendFile method
SendStream method
write method

OnConnect event

See also Example

Declaration
```
property OnConnect: TNotifyEvent;
```

Description
The OnConnect event is called when a connection is established with the remote host. This corresponds to WinSock's FD_CONNECT message.

See also

Connect method

OnConnectionFailed event

See also Example

Declaration
```
property OnConnectionFailed: TNotifyEvent;
```

Description
The OnConnectionFailed event is called when a connection fails to be established with the remote host.

See also

Connect method

Example

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

Place a TButton and a TPowersock on the form.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := '127.0.0.1';
  Powersock1.Port := 80;
  Powersock1.Connect;
end;
```

Insert this code into Powersock1's OnConnectionFailed event:

```
procedure TForm1.Powersock1ConnectionFailed(Sender: TObject);
begin
  ShowMessage('Connection Failed');
end;
```

Example Description:
When the application is run and Button1 is clicked, Powersock1's host is set to 127.0.0.1 (this is the IP address for local host, so it will try to connect to the computer you are using), and the port is set to 80, which is the standard port for the HTTP protocol. If there is a web server running on the computer you are doing development on, please disable it while you are doing this example, or run the example on another computer. Then the client tries to connect. If the connection fails, the OnConnectionFailed event will show a message saying Connection Failed, and will also raise an exception stating that the Connection Failed.

OnDisconnect event

See also Example

Declaration
```
property OnDisconnect: TNotifyEvent;
```

Description
The OnDisconnect event is called when the client disconnects from the server. This corresponds to WinSock's FD_CLOSE message.

See also

Disconnect method

OnInvalidHost event

See also Example

Declaration

```
property OnInvalidHost: THandlerEvent;
```

Description

The OnInvalid host event is called when the host specified by the Host property is invalid. If the handled parameter is set to TRUE, then the connection is attempted again. If handled is set to FALSE, an exception is raised. If handled is set to TRUE, and the specified host is still invalid, an exception is raised regardless of the value of the handled parameter.

See also

Connect method
Host property

Example

To recreate this example, you will need to create a new blank Delphi application.
Place a TButton, TMemo, and TNMHTTP on the form.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NMHTTP1.Get('www.safgbweg.com');
end;
```

Insert the following code into NMHTTP1's OnSuccess event

```
procedure TForm1.NMHTTP1Success(Cmd: CmdType);
begin
  if Cmd = CmdGet then
    Memo1.Text := NMHTTP1.Body;
end;
```

Insert the following code into NMHTTP1's OnInvalidHost event

```
procedure TForm1.Powersock1InvalidHost(var handled: Boolean);
var
  TmpStr: String;
begin
  If InputQuery('Invalid Host!', 'Specify a new host:', TmpStr) then
  Begin
    NMHTTP1.Host := TmpStr;
    Handled := TRUE;
  End;
end;
```

Example Description:
In this example, when Button1 is clicked, the host name specified in the URL given to the Get method is invalid, so the OnInvalidHost event displays a dialog to the user using the InputQuery function, allowing them to specify a new host name. If a new host name is specified, then the connect is attempted again using the new host name. If the Cancel button within the dialog box is clicked, an exception is raised. If an HTTP server is successfully connected to, the retrieved document will be shown in Memo1.

OnPacketRecvd event

See also Example

Declaration

```
property OnPacketRecvd: TNotifyEvent;
```

Description

The OnPacketRecvd event is called when data is received from the remote host. Used in conjunction with the BytesRecvd and BytesTotal properties, the progress of a data transaction can be monitored.

See also

BytesRecvd property
BytesTotal property

OnPacketSent event

See also Example

Declaration
```
property OnPacketSent: TNotifyEvent;
```

Description
The OnPacketSent event is called when data is sent to the remote host. Used in conjunction with the BytesSent property and BytesTotal property, the progress of a data transfer can be monitored.

See also

BytesSent property
BytesTotal property

 TNMGeneralServer component

Heirarchy Properties Methods Events Tasks

Unit
Psock

Description
The TNMGeneralServer component is provided for use as a base class for developing multi-threaded internet servers, be it a custom server or a server that supports RFC standards. The TNMGeneralServer component is not meant to be used as a stand-alone component, although it can be used as such.

TNMGeneralServer Properties

TNMGeneralServer

Legend

Derived from TPowersock

About

BeenCanceled

BeenTimedOut

BytesRecvd

BytesSent

BytesTotal

Connected

Handle

Host

LastErrorNo

LocalIP

Port

RemoteIP

ReplyNumber

ReportLevel

Status

TimeOut

TransactionReply

WSAInfo

Derived from TComponent

ComObject

ComponentCount

ComponentIndex

Components

ComponentState

ComponentStyle

DesignInfo

Owner

Tag

VCLComObject

TNMGeneralServer Methods
TNMGeneralServer
Legend

In TNMGeneralServer
- Serve

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

TNMGeneralServer Events
TNMGeneralServer
Legend

In TNMGeneralServer

OnClientContact

Derived from TPowersock
OnConnect
OnConnectionFailed
OnDisconnect
OnHostResolved
OnInvalidHost
OnPacketRecvd
OnPacketSent
OnRead
OnStatus

About the TNMGeneralServer component

TNMGeneralServer reference

Purpose

The TNMGeneralServer component is provided for use as a base class for developing multi-threaded internet servers, be it a custom server or a server that supports RFC standards. The TNMGeneralServer component is not meant to be used as a stand-alone component, although it can be used as such.

Tasks

By creating a descendant class of the TNMGeneralServer, and overriding the Serve method, you can customize how the server will respond to a client connection. Interaction with the client is accomplished using the read/write methods from TPowersock including ReadLn and writeln.

NOTE The Port property must be set during designtime.

Serve method

See also Example

Declaration

```
procedure Serve; virtual;
```

Description

The Serve method is a virtual method meant to be overridden by descendant classes. The Serve method is the method that gets called each time a client connects to the server. Be advised that the Serve method is executed in another thread, so VCL objects and methods cannot be used within this method.

NOTES:

The Serve method does nothing if called directly from the TNMGeneralServer class. It must be used (overridden) in a class descending from TNMGeneralServer.

When using the Serve method in a descendant class, you do not make a call to inherited Serve;

When the Serve method concludes, the client is disconnected from the server.

See also

ReadLn method
writeln method

Example

This example will demonstrate how to create a simple server that will send the day and time as a string to a client when a connection arrives.
To do this, you must first create a descendant of TNMGeneralServer by selecting Component | New Component from the Delphi 4 menu. Name this new component TDateServ, and choose TNMGeneralServer as the ancestor.

Next, you must override the Serve method. Modify the **public** section of the TDateServ class definition so that it mirrors the declaration below:

```
type
  TDateServ = class(TNMGeneralServer)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
    procedure Serve; override;
  published
    { Published declarations }
  end;
```

Now, in the **implementation** section of the unit, you must write the code for the Serve method, which contains only one line:

```
procedure TDateServ.Serve;
var
  DT: String;
  St: TMemoryStream;
  Sl: TStringList;
begin
  DT := DateTimeToStr(now);
  writeln(DT);
  SendBuffer(@DT[1], Length(DT));
  St := TMemoryStream.Create;
  try
    St.WriteBuffer(DT[1], Length(DT));
    SendStream(St);
  finally
    St.Free;
  end;
  Sl := TStringList.Create;
  try
    Sl.Text := DT;
    Sl.SaveToFile('Date.fil');
  finally
    Sl.Free;
  end;
  SendFile('Date.fil');
  DeleteFile('Date.fil');
end;
```

The Port property should be set the listen on the default port for the DayTime protocol (since this server simply returns the date and time to the client), so the Create method should be overridden so the port is

set correctly, as shown below:

```
constructor TDateServ.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Port := 13;
end;
```

Once doing that, you must also update the class definition in the **interface** section of the unit so that it mirrors the code below:

```
type
  TDateServ = class(TNMGeneralServer)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    procedure Serve; override;
  published
    { Published declarations }
  end;
```

Here is the complete source for the TDateServ component:

BEGIN UNIT DATESERV.PAS

```
unit DateServ;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Psock;

type
  TDateServ = class(TNMGeneralServer)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    procedure Serve; override;
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
```

```
  RegisterComponents('Samples', [TDateServ]);
end;

{ TDateServ }

procedure TDateServ.Serve;
var
  DT: String;
  St: TMemoryStream;
  Sl: TStringList;
begin
  DT := DateTimeToStr(now);
  writeln(DT);
  SendBuffer(@DT[1], Length(DT));
  St := TMemoryStream.Create;
  try
    St.WriteBuffer(DT[1], Length(DT));
    SendStream(St);
  finally
    St.Free;
  end;
  Sl := TStringList.Create;
  try
    Sl.Text := DT;
    Sl.SaveToFile('Date.fil');
  finally
    Sl.Free;
  end;
  SendFile('Date.fil');
  DeleteFile('Date.fil');
end;

constructor TDateServ.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Port := 13;
end;

end.
```

END UNIT DATESERV.PAS

Next, this component should be installed into Delphi by selecting the Component | Install Component from the menu. For more information on this, see Delphi's online help.

Now, create a new blank Delphi application.

Add a newly created TDateServ to your form, and a TMemo component to the form.

Insert the following code into DateServ1's OnClientContact event:

```
procedure TForm1.DateServ1ClientContact(Sender: TObject);
begin
  Memo1.Lines.Add('Client Connected');
end;
```

Now run the application. Don't be alarmed if it looks like it's not doing anything. It's waiting for a client to connect.
To test it, you can use either the TNMDayTime component (write a new application, or use the demo), or use the windows Telnet application, connecting to the Daytime port, using 127.0.0.1 (local host) as the host address.

Example Description:
When a client connects to the server, the Serve method is invoked. The date is sent to the client 4 times, which isn't a very useful thing in itself, but it illustrates how the different methods of sending data work. The writeln method simply takes a string, writes it to the socket, and appends a carriage return/line feed to it.
The SendBuffer method takes a PChar as a paramter, and an integer specifying the length of the data. It then sends out that number of bytes from the passed PChar. The SendStream method takes a stream as a parameter, and a size. It then writes the specified number of bytes from the stream to the socket. The SendFile method takes only a filename as a parameter. It then opens the file and sends it's contents to the socket.When the end of the Server method is reached, the host disconnects the client.

OnClientContact event

Example

Declaration
```
property OnClientContact: TNotifyEvent;
```

Description
The OnClientContact event is called each time a client connects to the server.

NthPos function

See also Example

Unit
Psock

Declaration
```
function NthPos(InputString: String; Delimiter: Char; Number: integer):
integer;
```

Description
The NthPos function returns the position of the specified occurrence of the delimiter specified.

Parameters:
InputString = The InputString parameter contains string to parse
Delimiter = The Delimiter parameter specifies the character to use as a separator
Number = The Number parameter specifies the occurrence of the delimiter to return the position of

Note:
The use of this function does not require any of the FastNet components to be used in an application. However, the Psock unit must be in the uses clause for the unit the function is used in.

See also

NthWord function

Example

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

Add a TButton and 4 TEdits to the form.

Be sure to add Psock to the **uses** clause. Your uses clause should look something like this:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, PSock;
```

Next, insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit4.Text := IntToStr(NthPos(Edit1.Text, Edit2.Text[1], StrToIntDef(Edit3.Text,2)));
end;
```

Example Description:
When you run the application, type in a sentence or just a bunch of words into Edit1. If you like to cut and paste, use this one: This*is*very*informative
Type in the character used to separate words, or just any letter in the sentence, in Edit2. I used *
In Edit3, type the occurence of the character you typed in Edit2. If you want to know the position of the 3rd occurence of the * character, type a 3.
Next, click Button1. A number should appear in Edit4. If you used the values specified above, the number should be 13. If you count the characters in Edit1, the 13th character will be the 3rd occurence of the * character.

NthWord function

See also Example

Unit
Psock

Declaration
```
function NthWord(InputString: String; Delimiter: Char; Number: integer):
String;
```

Description
The NthWord function returns a word from a string

Parameters:
InputString = The InputString parameter specifies the string to parse
Delimiter = The Delimiter parameter specifies the character that separates words within the string
Number = The Number parameter specifies the index of the word to return from the function

Note:
The use of this function does not require any of the FastNet components to be used in an application. However, the Psock unit must be in the uses clause for the unit the function is used in.

See also

NthPos function

Example

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

Add a TButton and 4 TEdits to the form.

Be sure to add Psock to the **uses** clause if it is not there already. Your **uses** clause should look something like this:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, PSock;
```

Next, insert this code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit4.Text := NthWord(Edit1.Text, Edit2.Text[1], StrToIntDef(Edit3.Text,2));
end;
```

Example Description:
When you run the application, type in a sentence or just a bunch of words into Edit1. If you like to cut and paste, use this one: This*is*very*informative
Type in the character used to separate words, or just any letter in the sentence, in Edit2. I used *
In Edit3, type the index of the word you wish to find. For the above example, I used 3.
Next, click Button1. A word should appear in Edit4. If you used the above example, the word should be very.

StreamLn procedure

Example

Unit

Psock

Declaration

```
procedure StreamLn(AStream: Tstream; Astring: String);
```

Description

The StreamLn method appends a string into a stream.

Parameters:

AStream = The AStream parameter specifies the stream to append the string to
Astring = The Astring parameter specifies the string to append to the stream

Note:

The use of this function does not require any of the FastNet components to be used in an application. However, the Psock unit must be in the uses clause for the unit the function is used in.

Example

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

Place a TEdit, TMemo, and 2 TButtons on the form.

Be sure that Psock is in the **uses** clause in your unit. Your uses clause should look something like this:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PSock, StdCtrls;
```

In the **interface** section of the unit, right before the **implementation** section, where there is a **var** statement, modify it so that it looks like the code below:

```
var
  Form1: TForm1;
  Strm: TMemoryStream;
```

Insert the following code into Form1's OnCreate event:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Strm := TMemoryStream.Create;
end;
```

Insert the following code into Form1's OnDestroy event:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Strm.Free;
end;
```

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  StreamLn(Strm, Edit1.Text);
end;
```

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  S: String;
begin
  Strm.Position := 0;
  SetLength(S, Strm.Size);
  Strm.Read(S[1], Strm.Size);
  Memo1.Text := S;
end;
```

Example Description:
When the application is run, Button1 will add whatever text is in Edit1 to the stream Strm using the Streamln procedure. When Button2 is clicked, the entire contents of the stream Strm are displayed in Memo1.

Legend

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

Heirarchy

TObject
|
TPersistent
|
TComponent

GetLocalAddress method

See also

Declaration

```
function GetLocalAddress: String;
```

Description

The GetLocalAddress method will return the local IP address formatted for sending to an FTP server when initiating a data transfer.

Notes:

The string returned by this function is not a standard dotted IP address in xxx.xxx.xxx.xxx format.
The purpose of GetLocalAddress is solely for communication with an FTP server

See Also

GetPortString method

GetPortString method

See also

Declaration
```
function GetPortString: String;
```

Description
The GetPortString method will return the port string required for initiating a data transfer with an FTP server.

Notes:
The string returned by this function is not a standard port number, rather it is a string recognized by FTP hosts when the user initiates a data transfer.
The purpose of GetPortString is solely for communication with an FTP server

See Also

GetLocalAddress method

ResolveRemoteHost method

Declaration
```
procedure ResolveRemoteHost;
```

Description
The ResolveRemoteHost method resolves the remote host specified by the Host property into a valid IP address as well as setting up the TPowersock component for connecting to the host specified

Warnings:
It is not reccomended that the ResolveRemoteHost method be called directly. Rather, allow the TPowersock component to make use of it during the Connect method.

Create method

See also Example

Declaration

```
constructor Create(AOwner: TComponent); override;
```

Description

The create method allocates memory and constructs a safely initialized instance of a component. See TComponent.Create for details on inherited actions. TPowersock also initializes WinSock when it is created.

Notes:

The Create method for the protocol-specific components that descend from TPowersock sets the Port property to the standard port for that protocol.

See Also

Destroy method
Free method

Example

```
procedure TForm1.Button1Click(Sender: TObject);
var
  PSock: TPowersock;
begin
  PSock := TPowersock.Create(Self);
  try
    Psock.Host := 'www.netmastersllc.com';
    Psock.Port := 7;
    PSock.TimeOut := 5000;
    PSock.Connect;
    PSock.Writeln('Hi');
    ShowMessage(PSock.ReadLn);
    Psock.Disconnect;
  finally
    Psock.Free;
  end;
end;
```

In this example, when Button 1 is clicked, an instance of TPowersock is created with the Create method. Enclosed in a Try...Finally block the instance of TPowersock is used to send data to a server, and receive a reply, then disconnect. In the Finally section, the instance of TPowersock is Freed.

Destroy method

See also

Declaration
```
destructor Destroy; override;
```

Description
You should not call the Destroy method in your application. Instead, call the Free method.
The Destroy method Disposes of the component and its owned components.See TComponent.Destroy for details on inherited actions. TPowersock also cleans up WinSock when it is destroyed.

See Also

Create method
Free method

OnRead event

See also Example

Declaration
```
property OnRead: TNotifyEvent;
```

Description
The OnRead event is called when there is incoming data being sent from the remote host. This corresponds to WinSock's FD_READ message.

Notes:
The OnRead event is where you would use the read, readln, and other data-retrieving methods to get data from the remote host as it comes in.

See Also

OnConnect event
OnDisconnect event

Example

Before you can make use of the OnRead event, you either have to:

A) Create a component that descends from TPowersock and expose the OnRead event as published, or
B) Write a method and assign it to the OnRead event.

Using the "A" method is the easiest way. Using the "B" method can be overwhelming if you don't know what you're doing.

Here is an example of the "B" method:

Create a new blank Delphi application. Drop a TButton and a TPowersock onto the form.
In the OnClick event for the TButton component, add the following code

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Powersock1.Host := 'www.netmastersllc.com';
  Powersock1.Port := 13;
  Powersock1.ReportLevel := 1;
  Powersock1.Timeout := 5000;
  Powersock1.Connect;
end;
```

Write the following code into Form1's unit somewhere in the **implementation** section.

```
// This is the procedure that will be assigned to the OnRead event
procedure TForm1.ReadEvent(Sender: TObject);
// Must have Sender: TObject as a parameter for a TNotifyEvent
begin
  ShowMessage(Powersock1.Readln);
end;

procedure TForm1.FormCreate(Sender: TObject);
// This is the form's OnCreate event
begin
  Powersock1.OnRead := Readevent;
end;
```

Be sure to add the ReadEvent method to your class definition in the Interface part of the unit, like this:

```
class
  TForm1 = class(TForm)
    Powersock1: TPowersock;
    ..
  public
    procedure ReadEvent(Sender: TObject);
   . . .
```

Example Description:
When Button1 is clicked, the TimeOut property is set to 5000 (5 seconds) so that in the event of an invalid host or other time-delaying reaction the connection attempt will stop. The ReportLevel property is set to 1 so that only basic informational status messages will be sent through. Then a connection is established with the host specified, in this case, www.netmastersllc.com, at port 13. Port 13 is the standard port for the DayTime protocol. When the remote host sends the date and time, the OnRead event will trigger, displaying the date and time using the ShowMessage procedure.

About property

Declaration
```
property About: TNMReg;
```

Description
The About property displays a dialog box displaying information about registering the components, obtaining upgrades, and getting technical support

Scope: Published
Accessibility: Designtime, Readonly

Example

OnAccept event

See also Example

Declaration
```
property OnAccept: TNotifyEvent;
```

Description
The OnAccept event is called when there is an incoming socket connection when the Listen method has been invoked with the sync parameter as FALSE.

Notes:
Use the Accept method to accept the incoming connection. If this is not done, the connection is refused.

See Also

Accept method
Listen method

Example

OnError event

See also

Declaration
```
property OnError: TOnErrorEvent;
```

Description
The OnError event is called when a Winsock error occurs. The ErrNo parameter specifies the numerical representation of the error that occurred, and the ErrMsg parameter is the error message associated with the error number.

See Also

LastErrorNo property

OnConnectionRequired event

See also Example

Declaration
```
property OnConnectionRequired: THandlerEvent;
```

Description
The OnConnectionRequired event is called whenever a methods is called that requires the client to be connected to a remote host. If Handled is set to True, the method is attempted again, if not, the method is aborted, and an exception is raised. If Handled is set to true, and the client is still not connected, an exception is raised.

See Also

Connect method
OnConnectionFailed event

Example

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

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

Set the **Host** property of NMEcho1 to www.netmastersllc.com in the Object Inspector.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NMEcho1.Connect;
end;
```

Insert the following code into Button2's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  NMEcho1.Disconnect;
end;
```

Insert the following code into Button3's OnClick event:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit2.Text := NMEcho1.Echo(Edit1.Text);
  Label1.Caption := FloatToStr(NMEcho1.ElapsedTime);
end;
```

Insert the following code into NMEcho1's OnConnectionRequired event:

```
procedure TForm1.NMEcho1ConnectionRequired(var Handled: Boolean);
begin
  if MessageDlg('Connection Required', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  begin
    NMEcho1.Connect;
    Handled := TRUE;
  end;
end;
```

Example Description:
When this application is run, try pressing Button3 (performing an echo) before pressing Button1 (Connect). This will cause the OnConnectionRequired event to be called, since calling the Echo method requires a connection. The MessageDlg function will ask if the user wishes to connect. If the user clicks the Yes button, the client connects, and Handled is set to TRUE, so the echo is attempted again. If the user clicks no, an exception is raised, and the echo attempt is aborted. If you press Button1 before you press Button3, the echo is performed without a hitch. Button2 serves only to disconnect the client from the host at the conclusion of the demo.

Proxy property

See also Example

Declaration
```
property Proxy: String;
```

Description
The Proxy property determines the name or IP Address of a Proxy server, if one is used. If a proxy server isn't being used, this must be left blank.

Scope: Protected (in TPowersock)

Notes:
Unless a Proxy server is being used, leave this property blank.
In certain protocol-specific components derived from TPowersock, the Proxy property is Published and may be access during either runtime or designtime.

Proxy property - See Also

Example

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

NOTE
If you do not use a proxy server at your current location, you will not be able to use this example. If you are not sure if you are using a proxy server or not, ask your network administrator.

Place a TButton, 2 TEdits, a TMemo, and a TNMHTTP on the form.

Insert the following code into Button1's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Edit1.Text <> '') and (Edit2.Text <> '') then
  begin
    NMHTTP1.Proxy := Edit1.Text;
    NMHTTP1.ProxyPort := StrToInt(Edit2.Text);
  end;
  NMHTTP1.Get('http://www.netmastersllc.com');
  Memo1.Text := NMHTTP1.Body;
end;
```

Example Description:
When this application is run, enter the dotted IP address or host name of the proxy server you use in Edit1. Type the port number you connect to the proxy server on in Edit2. If you are unsure of either of these things, ask your network administrator. Once these values have been set, click Button1. The host name/ip address in Edit1 is assigned to the Proxy property, and the number in Edit2 is assigned to the ProxyPort property after it has been converted into an integer. Then, the Get method retrieves the document located at http://www.netmastersllc.com using the proxy server specified, and displayes it in Memo1.

ProxyPort property

See also Example

Declaration

```
property ProxyPort: integer;
```

Description

The ProxyPort property specifies the port of a Proxy server to connect to, if one is used.

Scope: Protected (in TPowersock)

Notes:
This port must be set to the proper port of the proxy server if a proxy is to be used. If a proxy server is not being used, this property should be left at 0.
In certain protocol-specific components derived from TPowersock, the ProxyPort property is Published and may be accessed during either designtime or runtime.

See Also

Proxy property

THandlerEvent type

Declaration
type
THandlerEvent = **procedure**(var Handled: Boolean) **of** **Object**;

Description
The THandler event type is used to override default actions taken by various controls that descend from TPowersock. See each event description for details on the effects of changing the handled parameter.

Parameters
The **Handled** parameter specifies if you wish to execute the default actions taken by the component in question for the current operation. Typically, setting this paramter to TRUE will override the default component action, and allow you to change the behavior of the component in the event.

TOnErrorEvent type

Declaration:

type
  TOnErrorEvent = **procedure**(Sender: TComponent; Errno: Word; Errmsg: **string**) **of object**;

Description:
The TOnErrorEvent type is used for events that handle errors that have both a numerical representation and a string message associated with the number.

Parameters:
Errno = The Errno parameter specifies the numerical value of the error.
Errmsg = The Errmsg parameter is the text message that describes, at least partially, the Error encountered.

Heirarchy

TObject
|
TPersistent
|
TComponent
|
TPowersock