TIPPort DELPHI component

properties methods events

This Component is for all those people wishing to make DELPHI based network programs. Two levels of interface are provided, one for simple access when the component is a single connection or listening port,the other for advanced projects where multi-connections are required or for the programmer who wishes to do more complex interactions with the sockets sub-system. All sockets use Asynchronous notifications, via events, to allow the programmer to make non-blocking programs. This component is dynamically linked to WINSOCK.DLL and therefore does not hold this DLL open whilst designing projects within DELPHI.

Requires a 16-bit WINSOCK.DLL which is provided by 3rd Parties e.g. Trumpet 2.0b) or Microsoft. This has been tested with Trumpet 2.0b.

Sample program of Finger and GetNetTime are included to to demonstrate the use of this component.

ConnectTimeout

```
Word
```

The Connect Timeout Property is the length of time in seconds that a connecting port will attempt to connect to its remote before generating an OnTimeOut event. Default = 60 seconds if set to zero then no timeout event is generated.

```
IPPort1.ConnectTimeOut := 120; {two minute timeout on connection}
```

Properties

* = readonly at design time

BufferSize	ConnectTimeout	Description*
LocalPort	LocalAddress*	LocalHostName*
MaxSockets*	RemotePort	RemoteAddress
SystemStatus*	WinsockVersion*	WholeLines

Events are generated for

OnDataAvailable	When Data is ready to be received
OnSessionAccept	When an inbound connection needs to be created
OnTimeOut	When a connection exceeds the ConnectTimeout Value
OnSocketClose	When a Socket is closed by the remote
OnSocketConnect	When a Socket has successfully connected
OnDataReady	When the remote is Ready to accept data

Simple Methods

The Simple Methods are for simple single connection or listening ports these just allow the
simple programming of socket connections without having to keep track of the socket that is
currently in use. Listening sockets are limited to a single incoming connection

ConnectToRemote
ListenForRemote
StopListening
AcceptRemote
SendToRemote
GetFromRemote
DisconnectFromRemote

Byte Ordering Methods

Windows Sockets may return data that is in network byte order or require data in network byte order. This allows for multi platform support (e.g... IBM PC , Apple Mac etc.)

These methods allow you to convert between the two formats and are just wrappers to the Winsock functions

NetworkLong NetworkInt
HostLong HostInt

Advanced Methods

These Methods are use for handling multiple connections with the component or for doing specific
data transfers

OpenSocket
ConnectSocket
ListenOnSocket
AcceptConnection
ShutSocket
SendData
GetData
SendDataRaw
GetDataRaw

BufferSize

```
Word
```

This is the size of the internal data buffer use to receive data from the connected port if WholeLines is set to true then if the return buffer is completely filled then the port will buffer the next set of packets until a packet which is less than BufferSize is returned then all data is return into the passed stringlist.

The default value = 1024.

```
IPPort1.BufferSize := 4096;
```

Description

`string`

This is the description string provided by your WINSOCK.DLL provider and is read-only at design time

LocalPort

`string`

The Local Port Property is for listening ports to listen on a specified port such as 'smtp' or '25' . For outbound connections setting this value to 0 will use any port that is currently available.

```
IPPort1.LocalPort := 'smtp';
IPPort1.ListenForRemote;
```

LocalAddress

`string`

The Local Address property is your local host address in IP format and is readonly
e.g.. '123.45.100.99'

LocalHostName

`string`

The Local Host Name property is a readonly property which displays the name of your local host
e.g.. 'ipport.co.uk'

MaxSockets

`word`

This property is the current number of available sockets supported by your Sockets provider
and is read-only

RemotePort

`string`

The Remote Port Property allows the destination socket port to be specified and can either be the port name description or number.

e.g. 'smtp' or '25'

It must be specified with RemoteAddress property to make successful connections.

```
IPPort1.RemotePort := 'nntp';
```

RemoteAddress

```
string
```

The Remote Address is used to specify the remote socket to connect to with the ConnectToRemote and ConnectSocket methods. This and RemotePort must be set to make a successful connection.

The RemoteAddress can be in either IP address or HostName format
e.g. '123.01.99.67' or 'ipport.co.uk'.

```
IPPort1.RemoteAddress := 'borland.com';
```

SystemStatus

`string`

The SystemStatus Property is a Read-Only property and provides the system status from the Winsock provider.

WinsockVersion

`string`

The WinsockVersion is a readonly property which returns the current implementation of Win Sockets as provided by your Sockets provider.

WholeLines

```
Boolean
```

The Whole Lines property works in conjunction with the BufferSize property in determining what to do with receive buffers when they are returned full. If Wholelines is true and the received data equals the current buffersize the internal buffer is expanded and another receive issued until the data returned is less than buffersize at which point the data is returned. This is to try to concatenate buffers so that returned data lines are not split.

```
IPPort1.WholeLines := false;
```

ConnectToRemote

```
function ConnectToRemote: integer;
```

The ConnectToRemote method connects your application to a remote socket specified by the RemoteAddress and RemotePort Properties

If ConnectTimeOut is none zero then the OnTimeOut Event will be called if the connection is not
established by the connectTimeOut value

The Return Value is <0 on error

```
IPPort1.RemoteAddress := 'borland.com'
IPPort1.RemotePort := '80';
IPPort1.ConnectToRemote;
```

ListenForRemote

```
function ListenforRemote: Integer;
```

The Listen for Remote Method sets up a sockets listener port on the specified by the LocalPort property

The return value is < 0 on error

```
IPPort1.LocalPort := 'smtp';
error := IPPort1.ListenForRemote;
```

StopListening

```
procedure StopListening;
```

This Method closes the previously created listening port with ListenforRemote method

```
Form1.OnClose(Sender: TComponent);
begin
      IPPort1.StopListening;
end;
```

AcceptRemote

```
procedure AcceptRemote(var RemoteName: string; var RemoteAddr: string; var
RemotePort: string)
```

The AcceptRemote procedure allows the user to accept the incoming remote connection
the procedure creates a new outbound connection for the incoming connection so that SendToRemote
and GetFromRemote methods can be called

This procedure is usually called in the OnSessionAvaliable event

```
IPPort1.OnSessionAvailable(Socket: TSocket);
var
      RemName, RemAddr; RemPort: string;
begin
      IPPort1.AcceptRemote(RemName,RemAddr,RemPort);
      MessageDlg( ' Connection from ' + RemAddr + ' Received', mtInformation,
[mbOK],0);
end;
```

SendToRemote

```
function SendToRemote(Data: TStringList); LongInt
```

The Send to remote Method sends data onto the connected session. The return Value is the number of bytes sent.

```
IPPort1.OnDataReady(Socket: TSocket);
var
      Data: TStringList;
      DataLen : LongInt;
begin
      Data := TStringList.Create;
      Data.Add('Hello World');
      DataLen := IPPort1.SendToRemote(Data);
end;
```

GetFromRemote

```
function GetFromRemote(Var Data: TStringList): Longint;
```

The GetFromRemote method gets data from the socket connection and is usually called in the OnDataAvailable event. The Passed TStringList is filled with the returned data specified by the Buffersize and wholelines properties.

```
Ipport1.OnDataAvailable(Socket: TSocket);
var
      Data: TStringList;
      DataLen : LongInt;
begin
      Data := TStringList.Create;
      DataLen := IPPort1.GetFromRemote(Data);
      Memo1.Lines := Data;
      Data.Free;
end;
```

DisconnectFromRemote

```
Procedure DisconnectFromRemote;
```

The DisconnectFromRemote Method Closes the current outbound socket connection

```
IPPort1.OnSocketClose(Socket: TSocket);
begin
      IPPort1.DisconnectFromRemote;
      StatusBar.Caption := 'Disconnected';
end;
```

NetworkLong

```
function NetworkLong(LongInt); LongInt
```

The NetworkLong Method converts Host ordered byte LongInt data to Network ordered byte data

```
NetValue := Ipport1.NetworkLong(HostValue);
```

NetworkInt

```
Function NetworkInt(Integer):Integer;
```

The NetworkInt Method converts a local byte order Integer to Network Byte order integer

```
NetValue := IPPort1.NetworkInt(HostValue);
```

HostLong

```
function HostLong(LongInt): LongInt;
```

The HostLong Method converts a LongInt network ordered byte to Host ordered Bytes

```
HostValue := IPPort1.HostLong(NetValue);
```

HostInt

```
function HostInt(Integer): Integer;
```

The HostInt method converts Network byte ordered integer to a Host byte ordered integer

```
HostValue := IPPort1.HostInt(NetValue);
```

OpenSocket

```
function OpenSocket: TSocket;
```

The Open socket create a new socket for outbound connection or listening the return socket handle is used in subsequent method calls.

if the returned TSocket < 0 then a socket was unable to be allocated

```
MailserverSocket := IPPort1.Opensocket;
IPPort1.LocalPort := 'smtp';
IPPort1.ListenOnSocket(MailServerSocket);
```

ConnectSocket

```
function ConnectSocket(TSocket); Integer;
```

The ConnectSocket method connects the passed socket, created using OpenSocket to the Remote address and port specified in the RemoteAddress and RemotePort properties. The Error Status is returned.

```
WebSocket := IPPort1.OpenSocket;
IPPort1.RemoteAddress := 'borland.com';
IPPort1.RemotePort := '80'; {WWW};
IPPort1.ConnectSocket(WebSocket);
```

ListenOnSocket

```
function ListenOnSocket(Socket:TSocket; Queue: Integer): Integer;
```

The ListenonSocket method starts a listening port on the local host, using the port specified in the LocalPort property for the passed socket, created by the OpenSocket Method.

```
MailServer := IPPort1.OpenSocket;
IPPort1.LocalPort := 'smtp';
IPPort1.ListenOnSocket(MailServer, 5)
```

AcceptConnection Method

```
function AcceptConnection(Socket: TSocket, var RemoteName: string; var
RemoteAddr: string; var RemotePort string): TSocket;
```

The AcceptConnection method accepts a remote connection to the local host and returns a new socket on which the data transfer mechanism should use. This is usually called in the OnSessionAvailable Event

The Socket parameter is the socket which is doing the listening and should have been previously called with ListenOnSocket. The RemoteName,Address and Port variable string parameters are filled with the connecting ports details

```
IPPort1.OnSessionAvailable(Socket: TSocket)
var
      RNam,RAddr,RPort: string;
begin
      if Socket = MailServer then
      begin
            MClient := AcceptConnection(Socket, RNam,RAddr,RPort);
      end
end;
```

ShutSocket

```
procedure ShutSocket(Socket: TSocket);
```

The ShutSocket Methods Closes all actions on the passed socket handle. The socket handle should no longer be used.

```
IPPort1.OnSocketClose(socket: TSocket)
begin
      IPPort1.ShutSocket(Socket);
end;
```

SendData Method

```
function SendData(Socket: TSocket; Data : TStringList): LongInt;
```

The SendData Method sends the passed tstringlist to the socket specified in the socket parameter. it returns the number of bytes sent.

```
IPPort1.OnDataReady(Socket: TSocket)
var
      Data : TStringList;
      DataLen : LongInt;
begin
      Data := TSringList.Create;
      Data.LoadFromFile('Memo.msg');
      DataLen := IPPort1.SendData(Socket, Data);
      Data.Free;
end;
```

GetData Method

```
function GetData(Socket: TSocket; var Data TStringList): Longint;
```

The GetData Method returns data from the socket into the TStringList returning the number of bytes it works inconjunction with the BufferSize and WholeLines Properties and is described in the
BufferSize Property.

```
IPPort1.OnDataAvailable(Socket: TSocket);
var
      Data: TStringList;
      DataLen : LongInt;
begin
      IPPort.WholeLines := False;
      IPPort1.BufferSize := 32768;
      Data := TStringList.Create;
      DataLen := IPPort1.GetData(Socket, Data);
      Data.SavetoFile('Memo1.txt');
      Data.Free;
end;
```

GetDataRaw Method

```
function GetDataRaw(Socket: TSocket; Buffer: Pointer; BuffLen Integer);
Integer;
```

The GetDataRaw Method allows returned data to be returned into a users buffer where a TStringList would be unsuitable.

```
IPPort1.OnDataAvailable(Socket: TSocket);
var
      DataLen: LongInt;
      NetTime: LongInt;
      HostTime:TDateTime;
begin
      DataLen := IPPort1.GetDataRaw(Socket, @NetTime, sizeof(NetTime));
      HostTime := ConvertNetTimeToHostTimeFormat(NetTime);
      Memo1.Text :=  DateTimeToStr(HostNetTime);
end;
```

SendDataRaw Method

```
function SendRawData(Socket:TSocket; Buffer: Pointer; BuffLen Integer);
Integer;
```

The SendRawDataMethod allows the users own buffer to be sent over the connected socket. This is where a TStringlist is unsuitable.

```
IPPort1.OnDataReady(Socket: TSocket)
var
      DataLen : Integer;
      NetTime: LongInt;
begin
      NetTime := ConvertTimeToNetFormat(Now);
      DataLen := IPPort1.SendDataRaw(Socket, @NetTime, 4);
end;
```