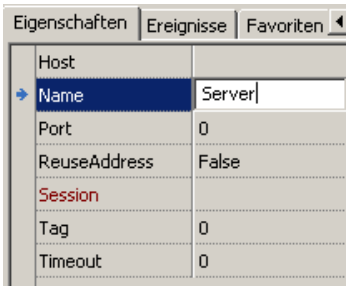

1.) Client-Server-Kommunikation

Zunächst wird an einem einfachen Beispiel gezeigt, wie mit der TCP-Komponente aus dem LNet-Paket eine Verbindung aufgebaut wird, bei der Strings übertragen werden.

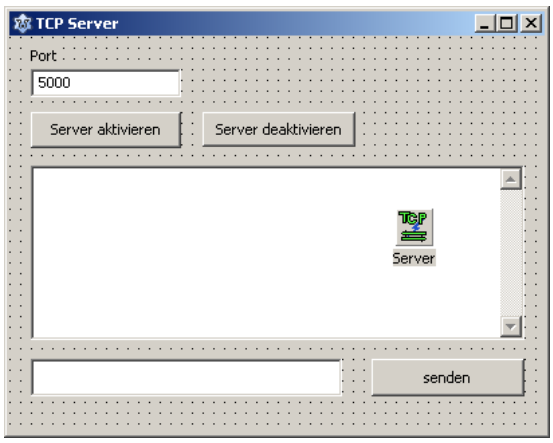
Sowohl der Server als auch der Client benutzen dieselbe Komponente: ***TLTCPComponent***

Zur besseren Unterscheidung benenne ich die Komponenten, nachdem sie auf das Formular für den *Server* bzw. den *Client* gezogen wurde in „Server“ bzw. „Client“ um.



2.) Der Server (Server I)

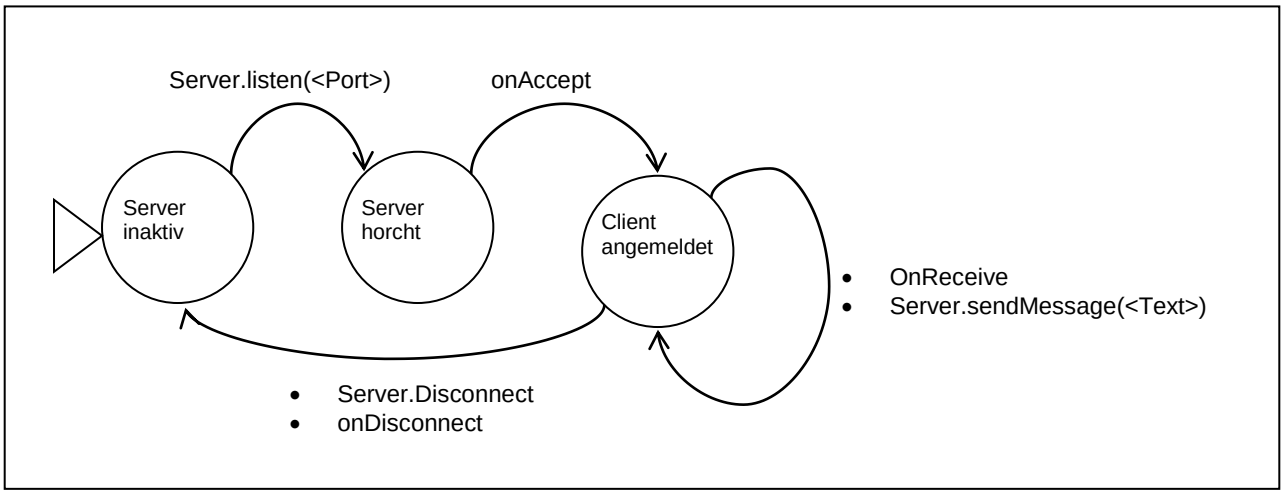
Beim Server wird der *Port* , auf dem der Server horcht, eingegeben. Beim Klick auf den Button *Server aktivieren* fängt die Server-Komponente an, auf dem angegebenen Port auf eingehende Nachrichten zu horchen. *Server deaktivieren* schaltet die Komponente wieder aus. Statusmeldungen und empfangene Strings werden im Memo-Feld angezeigt.



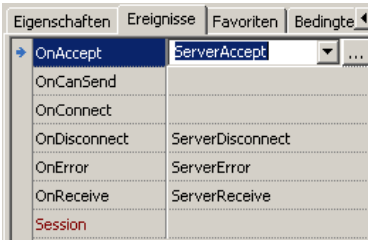
Die Datenübertragung erfolgt über das TCP-Protokol (das im Gegensatz zu UDP verbindungsorientiert ist, d.h. innerhalb des Protokolls erfolgt eine automatische Kontrolle über den Erfolg).
Der Server wird über den Befehl “listen” aktiv geschaltet. Er horcht nun am angegebenen Port, der aus dem Edit-Feld ausgelesen und übergeben wird.

```
procedure TForm1.Button1Click(Sender: TObject);
var Port : integer;
      ServerIP : String;
      Verbindungsaufbau : boolean;
begin
     Port := StrToInt(LabeledEdit1.Text);
     Verbindungsaufbau := Server.Listen(Port);
end;
```

Das Zustandsdiagramm macht die Arbeitsweise des Servers deutlich:



Beim Aufbau einer Verbindung, wenn sich ein Client anmeldet, wird das Ereignis „onAccept“ ausgelöst:



In der Eingangsmeldung kann über das Attribut *LocalAddress* die IP-Adresse des Client ausgelesen werden.

```
procedure TForm1.ServerAccept(aSocket: TLSocket);
begin
  Memo1.append ('Verbindung zum Client mit der IP' + aSocket.LocalAddress + '
wurde hergestellt');
end;
```

Um eine Sendung zu empfangen wird das Ereignis „OnReceive“ genutzt, das bei eingehenden Nachrichten ausgelöst wird. Die hereinkommende Nachricht liegt an der Socketverbindung „aSocket“ an und kann dort über die getter-Methode abgegriffen werden.

```
procedure TForm1.ServerReceive(aSocket: TLSocket);
var Eingangstext : String;
```

```
begin
  aSocket.GetMessage(Eingangstext);
  Memo1.append ('Eingehende Nachricht: ' + Eingangstext);
end;
```

Da lediglich Zeichenketten versendet werden sollen nutzen wir die Funktion „sendMessage“ der Komponente:

```
procedure TForm1.Button3Click(Sender: TObject);
var sendetext : String;
begin
  sendetext := Edit1.text;
  if (sendetext <> ' ') then
     Server.SendMessage(sendetext);
end;
```

Schließlich werden noch die Methoden zur Fehlermeldung, zum Trennen der Verbindung und die Mitteilung über das Abmelden des Clients implementiert:

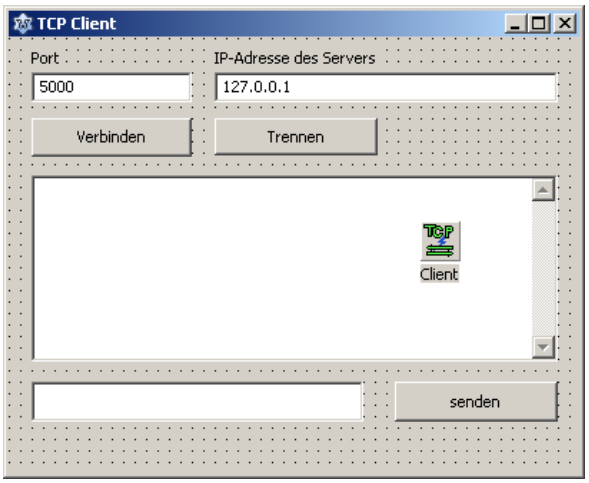
```
procedure TForm1.ServerError(const msg: string; aSocket: TLSocket);
begin
  Memo1.Append('Fehlermeldung: ' + msg);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Server.Disconnect(true);
  Memo1.append ('Server wurde deaktiviert');
end;

procedure TForm1.ServerDisconnect(aSocket: TLSocket);
begin
   Memo1.append ('Verbindung zum Client wurde getrennt');
end;
```

3.) Der Client

Der Client ist fast genauso wie der Server aufgebaut. Die Komponente wird zur besseren Identifikation zu „Client“ umbenannt. Zudem muss dem Client nicht nur der Port bekannt sein, auf dem der Server horcht, sondern auch die IP-Adresse. Werden die Programme lokal auf demselben Rechner getestet reichen bei der IP-Adresse die Angaben „127.0.0.1“ bzw. „localhost“ aus.



Zunächst stellt der Client eine Verbindung zum aktiven Server her:

```
procedure TForm1.Button1Click(Sender: TObject);
var Port : integer;
    ServerIP : String;
    Verbindungsaufbau : boolean;
begin
    Port := StrToInt(LabeledEdit1.Text);
    ServerIP := LabeledEdit2.Text;
    Verbindungsaufbau := Client.Connect(ServerIP,Port);
    if Verbindungsaufbau then
      Memo1.Append('Die Verbindung wird aufgebaut . . .');
end;
```

Alternativ funktioniert auch:

```
Client.Host:= ServerIP;
Client.Port:= Port;
Client.Connect;
```

oder auch nur:

```
Client.Connect(ServerIP,Port);
```

Die übrigen Prozeduren können quasi eins zu eins vom Server übernommen werden.

Text senden:

```
procedure TForm1.Button3Click(Sender: TObject);
var sendetext : String;
begin
  sendetext := Edit1.text;
  if (sendetext <> '') then
     client.SendMessage(sendetext);
end;
```

Beim Akzeptieren der Verbindung seitens des Servers wird die Meldung nicht über “onAccept” sondern über „onConnect“ mitgeteilt.

```
procedure TForm1.ClientConnect(aSocket: TLSocket);
begin
  Memo1.append ('Verbindung zum Server mit der IP ' + aSocket.LocalAddress +'
wurde hergestellt');
end;
```

Empfang einer Nachricht:

```
procedure TForm1.ClientReceive(aSocket: TLSocket);
var Eingangstext : String;
begin
  aSocket.GetMessage(Eingangstext);
  Memo1.append ('Eingehende Nachricht: ' + Eingangstext);
end;
```

Aktives Trennen, Trennungs- und Fehlermeldung:

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

procedure TForm1.ClientDisconnect(aSocket: TLSocket);
begin
```

```
   Memo1.append ('Verbindung zum Server wurde getrennt');
end;

procedure TForm1.ClientError(const msg: string; aSocket: TLSocket);
begin
  Memo1.Append('Fehlermeldung: ' + msg);
end;
```

4.) Erweiterung – mehrere Clients an einem Server (Server II)

Bislang konnte nur ein Client mit dem Server kommunizieren. Für z.B. einen einfachen Chat in einem lokalen Netz (Computerraum) ist es notwendig, dass der Server Nachrichten an alle Clients schickt. Im Beispiel soll die Nachricht im Textfeld an alle angeschlossenen Clients gesendet werden. Die Erweiterung hierzu ist relativ einfach. Der Client bleibt so, wie er ist. Nur der Server wird erweitert.

Die Prozedur „senden“ unter dem Button3 wird um einen Iterator erweitert.

```
procedure TForm1.Button3Click(Sender: TObject);
var sendetext : String;
begin
  sendetext := Edit1.text;
  if (sendetext <> '') then
    begin
    Server.IterReset;
      while (Server.IterNext = true) do
         Server.SendMessage(sendetext,server.Iterator);
    end;
end;
```

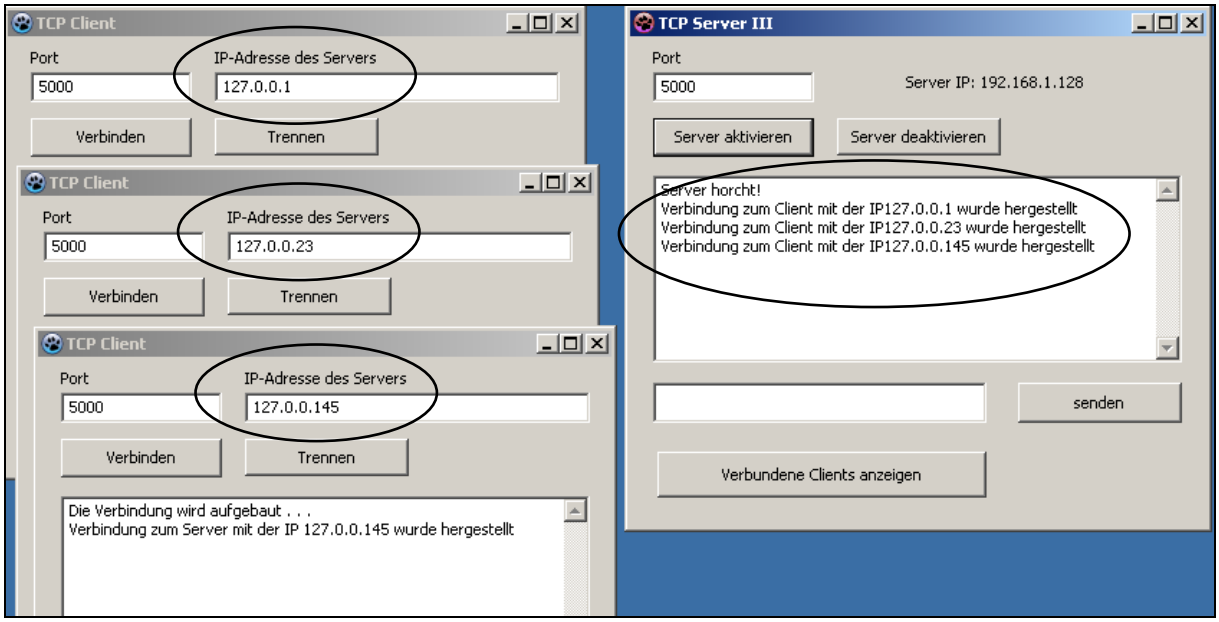
Die Socket-Verbindungen werden von der TCP-Komponente in einer verketteten Liste verwaltet. Die erste geöffnete Socket-Verbindung ist dabei die Server-Verbindung. Auf diese zeigt der Iterator bei ***IterReset***. ***IterNext*** liefert die nächste offene Socket-Verbindung. Solange diese existiert, wird die Nachricht gesendet. Der zweite Übergabeparameter in der sendMessage-Methode ist die geöffnete Socket-Verbindung.

Zudem macht es nun auch Sinn, bei der Akzeptanz eines Clients dessen IP-Adresse anzeigen zu lassen. Eine kleine Ergänzung in der Receive-Methode des Servers ist notwendig:

```
procedure TForm1.ServerReceive(aSocket: TLSocket);
var Eingangstext : String;
begin
  aSocket.GetMessage(Eingangstext);
  Memo1.append ('Eingehende Nachricht von IP '+ aSocket.PeerAddress
  +': '+  Eingangstext);
end;
```

5.) Erweiterung – Clientübersicht und Nachricht vom Server an nur einen Client (Server III)

Zunächst soll eine Liste der angemeldeten Clients angezeigt werden. Zum Test am lokalen Rechner –also mehrere Clients und ein Server auf demselben PC- kann die IP-Adresse des Ziel-Servers bei jedem einzelnen Client mehr oder weniger beliebig variiert werden:

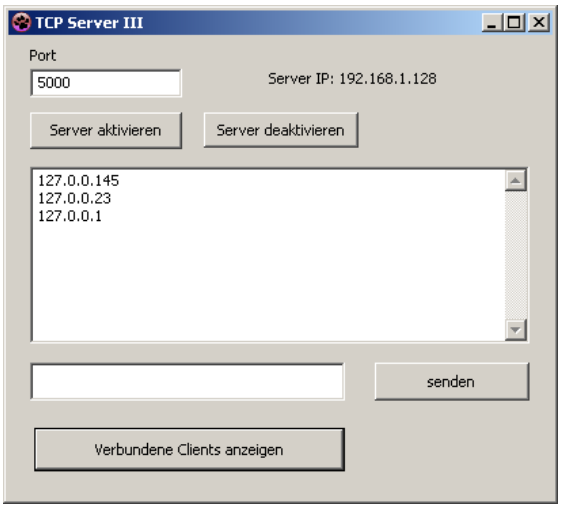


Nun kann der Server beim Anmelden eines Clients dessen IP auslesen und anzeigen. Dabei wird die „onAccept“-Methode erweitert:

```
procedure TForm1.ServerAccept(aSocket: TLSocket);
begin
  Memo1.append ('Verbindung zum Client mit der IP' + aSocket.LocalAddress
  + ' wurde hergestellt');
end;
```

Ebenso kann die Liste der Clients kompakt auf Knopfdruck angezeigt werden. Eine Alternative zu diesem Code steht in den Kommentarstrichen.

```
procedure TForm1.Button4Click(Sender: TObject);
// var n : integer;    --->  alternativ
begin
  Memo1.Clear;
  server.IterReset;
  // n := 0;      ---> alternativ
  while server.IterNext do
    begin
    Memo1.Append(Server.Iterator.PeerAddress);
    //n := n+1;    --> alternativ
    // Memo1.Append(Server.Socks[n].PeerAddress);  ---> alternativ
    end;
end;
```



Dieses Resultat (die „nackte“ IP-Liste) hat den Vorteil, dass per Mausklick die IP-Adresse übernommen und eine Testsendung an den Client geschickt werden kann.
Die Methode „OnDblClick“ der Memo-Felds liest zunächst die angeklickte IP-Adresse als String aus und schickt dann eine Testnachricht (die natürlich nach Belieben auch aus einem Edit-Feld entnommen werden kann) an die angeklickt IP-Adresse.
Dazu durchläuft der Iterator wieder die Socket-Liste und schickt dich Nachricht an die entsprechende Adresse.
Hinweis: Sollten mehrere Clients unter derselben IP vorhanden sein (kann beim lokalen Test auf einem einzigen Rechner passieren), wird die Nachricht selbstverständlich an alle Clients mit dieser IP verschickt.

```
procedure TForm1.Memo1DblClick(Sender: TObject);
var ClientIP : String;
begin
  ClientIP := Memo1.seltext;
  server.IterReset;
  while server.IterNext do
    if (ClientIP = server.Iterator.PeerAddress) then
         server.SendMessage('Testnachricht',Server.Iterator);
end;
```