Wie baue ich mir einen Chat

Wie der Titel schon sagt, geht es in diesem Tutorial darum, wie man einen Chat mit den Sockets realisiert.
Ihr solltet dazu mit der IDE und der Sprache von Delphi zumindest ein wenig vertraut sein.
Das Beispiel wurde unter Windows XP Home und Delphi 7 Enterprise geschrieben.

Teil 1: Die Vorbereitungen

Wir starten Delphi *g*. Danach klicken wir auf Datei->Neu->Anwendung, soweit nicht schon passiert. Nun können wir schon mal das Formular benennen, etwa "Mein erster Chat" oder was immer euch beliebt. Wir brauchen auch nicht die ganze Fenstergröße, nur ungefähr 385x285. Wir stellen nun die Eigenschaft **Autosize** auf **false**. In den **BorderIcons** nehmen wir **biMaximize** raus und stellen den **BorderStyle** auf **bsSingle**, damit keiner auf die blöde Idee kommt, die Anwendung zu vergrößern und so unser (mehr oder weniger) tolles Design kaputt zu machen. Jetzt wird das ganze erst einmal abgespeichert, am besten in einem eigenen Ordner. Das war erst einmal alles.

Teil 2: Die Komponenten

Gleich zu Anfang: Ich benutze die Indy-Komponenten, wer sie nicht hat sollte sie downloaden (http://www.nevrona.com/indy) und installieren. Auf das Formular kommen fürs erste ein Memo, drei Edit-Felder, drei Buttons, eine ComboBox und sechs Labels. Das wären die visuellen Komponenten (das sind die, die man später auch im fertigen Programm sieht). Zur Veranschaulichung hier ein Bild:

Zusätzlich brauchen wir einen ClientSocket, ein ApplicationEvents, ein IdIPWatch und einen Timer. Das ganze könnt ihr erstmal übersichtlich anordnen, z.B. oben das Memo, darunter die drei Edits und neben jedes davon einen Button.

Wir müssen auch noch die Property **Port** des ClientSockets auf irgendeine Zahl stellen, vielleicht 7530, oder was auch immer euch beliebt. Diese Zahl sollte jedoch nicht unter 250 und über 65000 liegen. Schreibt euch die Zahl am Besten auf, wir brauchen sie später noch. Da Viele die Sockets nicht standardmäßig installiert haben, erkläre ich hier mal, wie sie installiert werden: Im Menü **Komponente** auf **Packages installieren** klicken. Danach auf Hinzufügen, ins Verzeichnis **\Borland\Delphi7\Bin** wechseln und die Datei **dclsockets70.bpl** öffnen. Schon sind sie in der Registerkarte **Internet** zu finden.

Teil 3: Der Code

Wir deklarieren zunächst eine **globale** Variable namens "Nickname" vom Typ **string**.
Ins FormCreate kommt folgendes:

```
Label6.Caption:=IdIPWatch1.LocalIP; //Damit oben im Fenster schon beim
FormCreate die aktuelle IP-Adresse angezeigt wird
Button1.Enabled:=false;//Damit man nicht schon vorher draufdrückt
Button2.Enabled:=false;//              -''-
```

Wenn man auf den Knopf "Nick setzen" drückt sollte folgendes passieren:

```
Nickname:=Edit3.Text;
Button3.Enabled:=false;
Edit3.Enabled:=false;
Button2.Enabled:=true;
```

Damit bekommt die Variable "Nickname" den Wert des Edit-Feldes zugewiesen.
Nun können wir Edit3 & Button3 disablen, da der Nickname ja nun festgesetzt ist. Enablen tun wir jedoch Button2, damit man die IP des Servers angeben kann.

A Apropos Button2, auf dessen Druck sollte dies geschehen:

```
ClientSocket1.Host:=Edit2.Text;
ClientSocket1.Active:=true;
Button2.Enabled:=false;
Edit2.Enabled:=false;
Button1.Enabled:=true;
Nachricht:=Edit3.Text+' hat den Raum betreten';
delay(500);
ClientSocket1.Socket.SendText(Nachricht);
```

Wir tun mit diesem Code folgendes: Die IP-Adresse des Servers wird festgesetzt, der ClientSocket1 wird aktiviert (um überhaupt etwas senden zu können) und Button2 & Edit2 werden ge-disablet. Danach senden wir einen Text, der den anderen Teilnehmern signalisiert, dass jemand den Chat betreten hat. Aber da ist noch etwas: Die Prozedur **delay**.
Sie erlaubt uns, eine Pause im Programm einzulegen, es aber nicht einzufrieren.
Dazu schreiben wir in den **type**-Abschnitt diesen Code (irgendwo zwischen oder unter die ganzen anderen Procedures):

```
procedure Delay(time: word);
```

Und gleich danach im implementation-Teil fügen wir das hier ein:

```
procedure TForm1.delay(time:word);
var Start: Integer;
begin
  Start:=GetTickCount;
  while (GetTickCount)-(Start) <= time do
    Application.ProcessMessages;
end;
```

Womit wir auch diese Prozedur gemeistert hätten.

Nun kommt das, was passiert, wenn wir auf den "Senden"-Knopf drücken:

```
if (ClientSocket1.Active=true) and (Edit1.Text<>'') then
ClientSocket1.Socket.SendText(NickName+': '+Edit1.Text);
```

```
Edit1.Text:='';
```
So, nun die Erklärung: Um nicht zu senden wenn der Socket inaktiv ist fragen wir den Status ab. Außerdem wäre es dumm, wenn man einen Leerstring senden könnte (wozu auch, wäre ja sinnlos). Jetzt wird eine Kombination aus Nickname, ': ' und dem Text verschickt. Das sähe dann im Fenster der anderen ungefähr so aus:
```
Gerhard: So sieht das dann aus
```

OK, das könnten wir vorerst auch mal kompilieren. Es würde auch funktionieren, bloß dass man noch nix verschicken kann, weil kein Server da ist. Aber den schreiben wir später schon auch noch.

Aber zuerst müssen wir ja etwas unternehmen, damit der Text, der von anderen (oder auch von einem selbst) gesendet wird, auch im Memo erscheint. Das machen wir, indem wir folgendes ins **OnClientRead** schreiben:
```
Form1.Show;
Memo1.Lines.Add(Socket.ReceiveText);
```

Jetzt stellt sich die Frage: Wozu das **Form1.Show**?
Ganz einfach: Wenn man das Chat-Fenster in den Hintergrund verbannt, würde man ja normalerweise nicht mitkriegen, wenn jemand einen Text schickt. So wird nun aber das Fenster in den Vordergrund geholt. So bleibt man immer auf dem Laufenden.

So, jetzt kümmern wir uns mal um den Timer. Der soll dafür sorgen, dass die IP-Adresse alle paar Sekunden aktualisiert wird.
In die Prozedur OnTimer kommt also das hier:
```
Label6.Caption:=IdIPWatch1.LocalIP;
```

Das wäre eigentlich schon fast alles für den Client, bloß das fehlt noch:
```
ClientSocket1.Socket.SendText(Edit3.Text +' hat den Chat verlassen.');
delay(500);
ClientSocket1.Close;
end;
```

Wir schicken nun einen Text, der die anderen Teilnehmer darüber informiert, dass man den Chat verlassen hat. Dann warten wir eine halbe Sekunde, damit der Text auch wirklich durch die Leitung ist. Danach schließen wir den Socket.

Das war’s jetzt aber endgültig für den Client. Jetzt kommt aber schon der nächste Teil: Der Server. Machen wir’s wieder wie im ersten Teil.

Teil 4: Die Vorbereitungen für den Server

OK, wir starten wieder ein neues Projekt und speichern es im gleichen Ordner wie den Client ab; eventuell könnten wir eine Ordnerstruktur anlegen, so dass wir einen übergeordneten Ordner namens **Chat-Projek**t anlegen und in diesem Ordner zwei weitere mit den Namen **Client** und **Server**. Ihr könnt sie natürlich auch anders nennen oder tun was immer euch beliebt, aber so ist es (finde ich) am übersichtlichsten. Zurück in Delphi nennen wir das Formular **Server** oder so ähnlich. Aus den **BorderIcons** fliegt **biMaximize** raus und den **BorderStyle** stellen wir auch hier auf **bsSingle**. Genau wie beim Client wird die Eigenschaft **Autosize** auf **false** gestellt.

Teil 5: Die Komponenten für den Server

An visuellen Komponenten brauchen wir eigentlich nur ein Memo. Ansonsten werden ein TServerSocket und ein TIdIPWatch benötigt. Wenn es euch gefällt, könnt ihr ein TXPManifest hinmachen, um dem ganzen einen Touch von XP zu verleihen, das ist aber nicht zwingend notwendig. Wenn wir das alles gemacht haben, können wir das Memo so ausrichten, dass der letzte Rasterpunkt des Formulars an allen Rändern gerade noch sichtbar ist; die **Anchors** setzen wir alle, also **akLeft**, **akTop, akRight** und **akBottom**. So wächst das Memo immer mit. Danach stellen wir die **Scrollbars** noch auf **ssVertical**, damit man auch allen Text lesen kann. Zu guter Letzt brauchen wir auch noch ein Label. Wir müssen den Port des ServerSockets auf die Zahl stellen, die wir uns vorhin notiert haben, also in unserem Fall die Zahl 7530.

Teil 6: Der Code des Servers

Im Code-Fenster kommt nun Folgendes ins FormCreate:

```
label2.caption:=IdIPWatch1.LocalIP;
```

Damit bezwecken wir wieder mal, dass die IP gleich angezeigt wird, genau wie schon beim Client.

Damit der Server uns auch alles anzeigt, was von den einzelnen Clients geschickt wird, müssen wir etwas ins **OnClientRead** schreiben, und zwar folgendes:

```
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var Text:string;
    i: Integer;
begin
Text:=Socket.ReceiveText;
Memo1.Lines.add(Timetostr(now)+': '+Text);
  For I := 0 to ServerSocket1.Socket.ActiveConnections - 1 do
   begin
   with ServerSocket1.Socket.Connections [I] do
  if (Connected) then SendText(Text);
end;
end;
```

Nun erkläre ich erstmal, was hier passiert: Vom Server wird der gesendete Text der Clients empfangen und in eine Variable geschrieben. Dann wird dieser Text im Memo angezeigt, und zwar mit Uhrzeit, damit man zurückverfolgen kann, wann was war. Nun werden in einer Schleife alle Clients durchlaufen und jedem wird der Text zugeschickt, damit alle den Text erhalten und anzeigen können. Wir fragen dabei auch ab, ob die Clients verbunden sind, denn sonst gäbe es einen Fehler.

Das war auch schon fast alles für den Server, es fehlt nur noch etwas im **FormClose**, und zwar lassen wir uns alles, was in der Zeit, in der der Server gelaufen ist, gesendet wurde, in eine Textdatei schreiben, um eventuellen späteren Streitigkeiten vorzubeugen. Das ist eigentlich ganz einfach und geht so:

```
ServerSocket1.Close;
Memo1.Lines.SaveToFile('Chat-Log.txt');
```

Wir schließen den Socket, damit wir das Programm beenden können und dann speichern wir wie gesagt den Text des Memos in eine Textdatei. Das war jetzt aber auch wirklich alles.

Achtung: Wenn euer Virenscanner Alarm schlägt wenn ihr den Server öffnen wollt, dann wundert euch bitte nicht. Manche Scanner erkennen angeblich einen Trojaner namens BDS/Mantice.10.Cli! Das ist normal bei diesem Serverprogramm.