MyStringEchoServer		Prof. Dr.-Ing. habil. L. Winkler	HOCHSCHULE MITTWEIDA UNIVERSITY OF APPLIED SCIENCES
CPP-Project mit TServerSocket	2015-01	Fakultät Informations- und Elektrotechnik	

1 Ziel des Projektes

- Umgang mit einem Server-Socket der Klasse TServerSocket aus der RAD-Studio-XE2-IDE.
- Es soll eine Server-Anwendung programmiert werden, die kommende Verbindungen an TCP-Port 6666 annimmt.
- Der Echo-Server erwartet einen AnsiString, wendet auf diesen die Funktion UpperCase() an und sendet den String zurück.
- Ein String-EchoClt steht zum Test des Servers zum Download bereit.

2 Die Klasse TServerSocket[1]

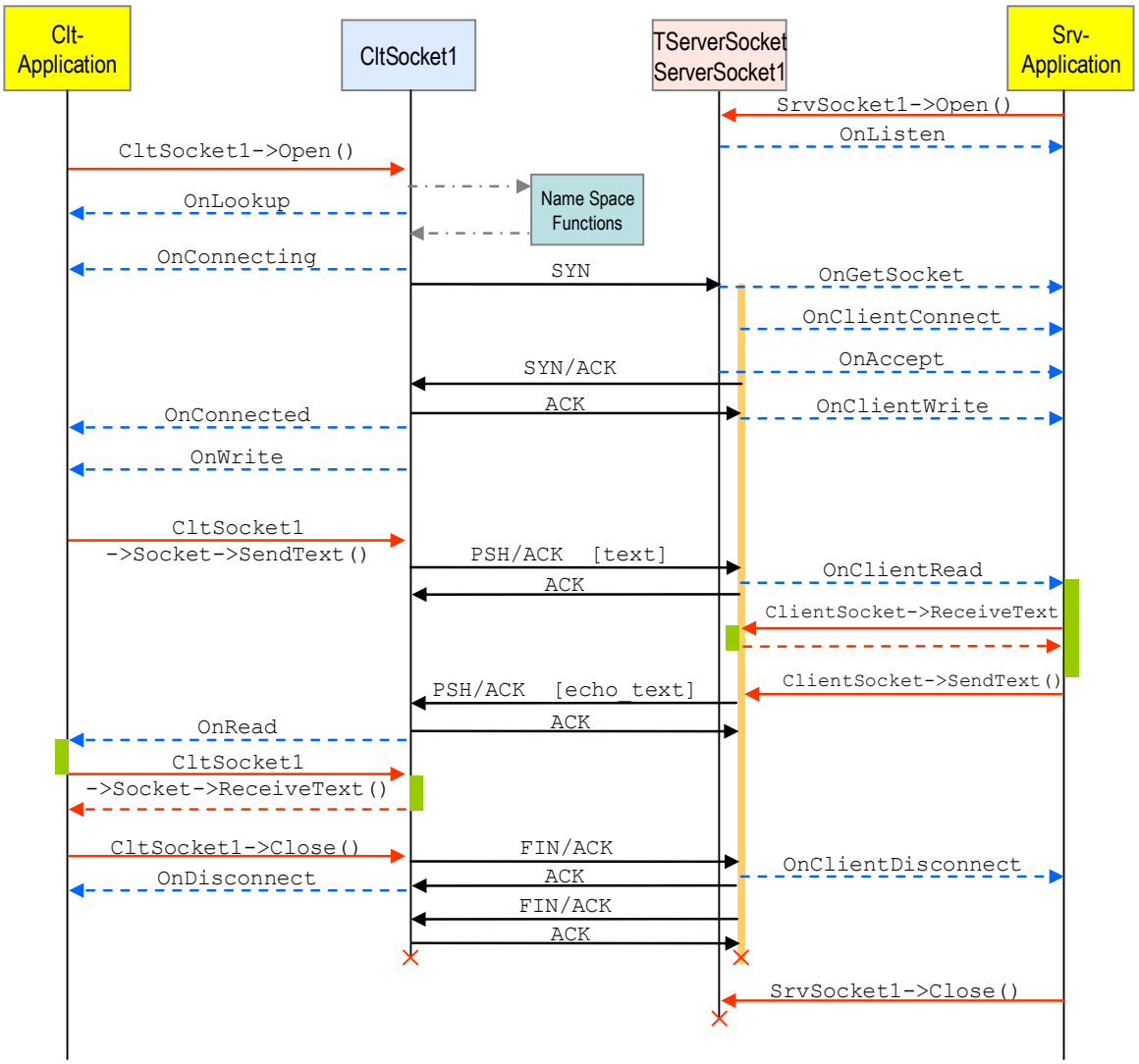
Wesentliche Eigenschaften, Methoden und Ereignisse dieser Klasse, werden nachfolgend vorgestellt.

Properties der Klasse TServerSocket (Palette Internet)		
Active	False|True	Standardwert ist False. Setzt man Active auf True, wird der ServerSocket sofort beim Start der Anwendung aktiviert und wartet auf ankommende Verbindungswünsche.
Name	z.B. ServerSocket1	Name des Serversockets
ServerType	stNonBlocking| stThreadBlocking	Bestimmt, ob alle vom Serversocket akzeptierten Verbindungen nichtblockierend sind oder ob automatisch ein separater Thread angegeben wird.
Port oder	80 | 21 | 25	Hier muss man die Portnummer des Dienstes angeben, für den der ServerSocket errichtet wurde. Man gibt entweder die Portnummer oder den Service an.
Service	http|ftp |smtp	Anstelle der Portnummer kann man auch das Anwendungsprotokoll bzw. den Dienst angeben. Daraus wird dann mittels interner Socketfunktionen der Port ermittelt.
Socket		Bezeichnet das TServerWinSocket-Objekt, das den Endpunkt der Datenverbindung zu einem Client beschreibt. Diese Eigenschaft existiert nur zur Laufzeit.
Methoden der Klasse TServerSocket (Palette Internet), *wenn Name= ServerSocket1 ist*		
ServerSocket1->**Open()**		Der Socket wird aktiviert und befindet sich im Listen-Zustand. Property ServertSocket->Active hat anschließend den Wert True. Hat Property ServerSocket1->ServerType den Wert stNonBlocking, erzeugt der Socket in Reihenfolge folgende Ereignisse: OnGetSocket, OnClientConnect, OnAccept, OnClientWrite. Hat Property ServertSocket1->ServerType den Wert stThreadBlocking wird für jede ankommende ein neuer Thread erzeugt und die Ereignisse OnGetThread, OnThreadStart, OnThreadEnd erzeugt.
ServerSocket1->**Close()**		Der ServerSocket wird beendet, alle allokierten Ressourcen werden frei gegeben.
Socket->**SendText(***AnsiString***)**		Zum ClientSocket einen Text vom Typ AnsiString senden
AnsiString=Socket->**ReceiveText()**		Vom ClientSocket einen Text vom Typ AnsiString empfangen
int=Socket->**SendBuf(***&buf,anzahl***)**		Zum ClientSocket eine bestimmte *anzahl* Byte senden, die in *buf* stehen. Im Rückgabewert wird angezeigt, wie viele Bytes tatsächlich gesendet wurden.
int=Socket->**ReceiveBuf(***&buf, anzahl***)**		Vom ClientSocket Bytes empfangen, die dann in *buf* stehen. *anzahl* gibt an, wie groß der *buf* ist. Im Rückgabewert steht die Anzahl Bytes, die in *buf* übergeben werden.
Weitere Schreib- und Lesemethoden →siehe Hilfe		
Ereignisse der Klasse TServerSocket (Palette Internet), *wenn Name= ServerSocket1 ist*		
Nur wenn ServerSocket1->ServerType=stNonBlocking		
ServerSocket1**Listen**		ServerSocket zeigt damit an, dass er bereit ist, ankommende Verbindungswünsche zu akzeptieren.
ServerSocket1**GetSocket**		ServerSocket zeigt damit an, das ein Verbindungswunsch vorliegt und ein neues Socket-Objekt als Verbindungsendpunkt zu einem Client erzeugt wurde.
ServerSocket1**Accept**		ServerSocket zeigt damit an, dass eine kommender Verbindungswunsch akzeptiert wurde. Es existiert ein Socket-Objekt über das die Kommunikation mit dem Client realisiert wird.
ServerSocket1Client**OnWrite**		Mit diesem Ereignis zeigt das erzeugte Socket-Objekt an, dass die Verbindung zum Client aufgebaut ist und jetzt Daten gesendet werden könnten.
ServerSocket1Client**OnRead**		Dieses Ereignis wird ausgelöst, wenn Daten zum Lesen aus dem Socket-Objekt bereitstehen. Der Anwendung wird damit eine einfache Möglichkeit gegeben aus den Socket-Objekt zu lesen.
ServerSocket1Client**OnDisconnect**		Dieses Ereignis wird ausgelöst, wenn die Verbindung vom Client, z.B. mit Close(), beendet wurde.
ServerSocket1Client**OnError**		Dieses Ereignis tritt ein, wenn im Socket-Objekt ein Fehler auftritt. Setzen Sie den Parameter ErrorCode auf 0, um das Entstehen einer ESocketError-Exception zu verhindern.
Nur wenn ServerSocket1->ServerType=stThreadBlocking		
ServerSocket1**OnGetThread**		Tritt ein, wenn der ServerSocket für eine neue ankommende Verbindung einen neuen Thread erzeugt.
ServerSocket1**OnThreadStart**		Tritt ein, wenn die Verbindung von einem Client akzeptiert und hergestellt wurde und benutzbar ist.
ServerSocket1**OnThreadEnd**		Tritt ein, wenn das Socket-Objekt zu einem Client beendet wird und damit auch der Thread.

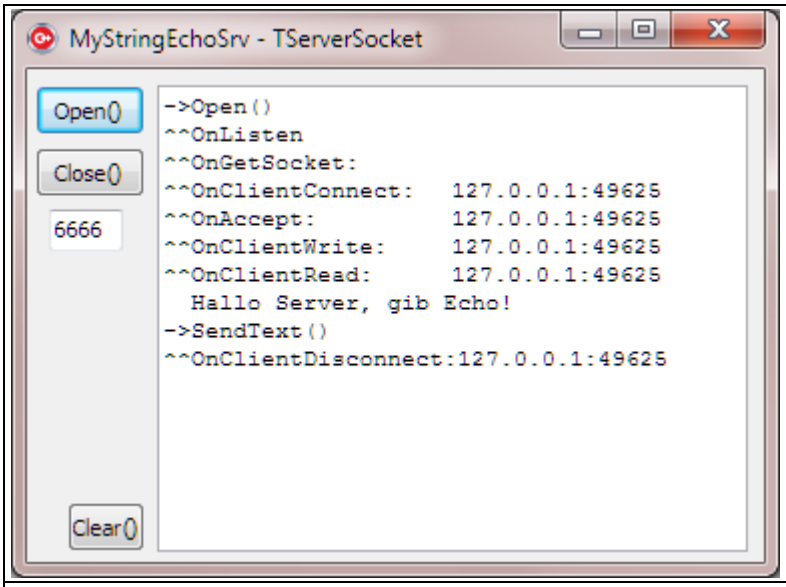
[1] Die Komponente ist nicht standardmäßig in die IDE eingebunden.
Sollten sie TServerSocket nicht in der Palette „Internet“ finden, gehen sie zum Einbinden der Komponente wie folgt vor:
(1) Menü: Komponenten→Packages installieren (2)→Hinzufügen
(3)C|D|X:\Programme\Embarcadero\RAD Studio\9.0\bin\dclsockets160.bpl wählen →Öffnen
(4)Die Komponentenpalette „Internet“ hat jetzt zwei neue Komponenten: TClientSocket, TServerSocket

3 Ablauf einer Client-Server-Verbindung

In der nachfolgenden Abbildung sieht man alle Methoden, Ereignisse zwischen den Anwendungsteilen und den zugehörigen Sockets der Client- und Serverseite.
Man beachte, dass der Serversocket eine reine Verwaltungsfunktion hat, nämlich ankommende Verbindungen entgegenzunehmen und ein Socket-Objekt zu erzeugen. Der Serversocket ist anschließend wieder frei und kann weitere Verbindungswünsche annehmen. Alle während dieser Zeit ankommenden Verbindungswünsche werden in eine Warteschlange aufgenommen, deren Größe einstellbar ist.
Die Datenkommunikation zwischen Serveranwendung und Clientanwendung wird dann über das neu errichtete Socket-Objekt abgewickelt. In dieser Abbildung ist nur ein Socket-Objekt dargestellt. Hätten sich z.B. fünf Clients mit dem Server erfolgreich "verbunden", existiert neben dem Serversocket weitere fünf Socket-Objekte zur Abwicklung der fünf Connections.



4 Realisierung des Projektes MyStringEchoSrv



- Erzeugen Sie ein neues Projekt "MyStringEchoSrv" im gleichnamigen Order.
- Installieren Sie die Komponente dclsockets160.bpl entsprechen der Anleitung Fußnote 1, falls diese nicht vorhanden ist.
- Editieren Sie die Oberfläche entsprechend der Abbildung.
- Erzeugen Sie Schritt für Schritt den Programmcode, entsprechend dem BeispielCode.

- Verwenden Sie bitte folgende Klassen und Objektnamen:
  - TButton: Open_, Close_,
  - TBitBtn: BitBtn1
  - TEdit: Edit1
  - TRichEdit: RichEdit1
  - TServerSocket: ServerSocket1

```
//==notwendiger Programmtext==
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Open_Click(TObject *Sender)
{
          RichEdit1->Lines->Add("->Open() ");
          ServerSocket1->Port=Edit1->Text.ToInt();
          ServerSocket1->Open();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1ClientRead(TObject *Sender,
      TCustomWinSocket *Socket)
{
         RichEdit1->Lines->Add("^^OnClientRead:\t"+Socket->RemoteAddress+":"+
                                              IntToStr(Socket->RemotePort));
         AnsiString text= Socket->ReceiveText();
         RichEdit1->Lines->Add("  "+text);
         RichEdit1->Lines->Add("->SendText()");
         Socket->SendText(text.UpperCase());
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Close_Click(TObject *Sender)
{
        RichEdit1->Lines->Add("->Close() ");
        ServerSocket1->Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
        RichEdit1->Clear();
}
//== nicht notwendiger Programmtext, dient nur zur Ereignisanzeige-----------
void __fastcall TForm1::ServerSocket1Listen(TObject *Sender,
      TCustomWinSocket *Socket)
{
        RichEdit1->Lines->Add("^^OnListen");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1Accept(TObject *Sender,
      TCustomWinSocket *Socket)
{                                       // \t entspricht Tabulator[2]
         RichEdit1->Lines->Add("^^OnAccept:\t\t"+Socket->RemoteAddress+":"+
                                             IntToStr(Socket->RemotePort));
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1GetSocket(TObject *Sender, int Socket,
      TServerClientWinSocket *&ClientSocket)
{                             // \n entspricht Zeilenschaltung
         RichEdit1->Lines->Add("\n^^OnGetSocket");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1ClientConnect(TObject *Sender,
      TCustomWinSocket *Socket)
{
         RichEdit1->Lines->Add("^^OnClientConnect:\t"+Socket->RemoteAddress+":"+
                                             IntToStr(Socket->RemotePort));
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1ClientDisconnect(TObject *Sender,
      TCustomWinSocket *Socket)
{
         RichEdit1->Lines->Add("^^OnClientDisconnect:"+Socket->RemoteAddress+":"+
                                             IntToStr(Socket->RemotePort));
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1ClientError(TObject *Sender,
      TCustomWinSocket *Socket, TErrorEvent ErrorEvent, int &ErrorCode)
{
        RichEdit1->Lines->Add("^^OnError: "+Socket->RemoteAddress+" "+
                                             IntToStr(ErrorCode));
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ServerSocket1ClientWrite(TObject *Sender,
      TCustomWinSocket *Socket)
{
        RichEdit1->Lines->Add("^^OnClientWrite:\t"+Socket->RemoteAddress+":"+
                                             IntToStr(Socket->RemotePort));
}
//---------------------------------------------------------------------------
```

2
\t Tabulator
\n Line Feed, LF
\r Carriage Return, CR
\b Back Space, BS
\f Form Feed, FF