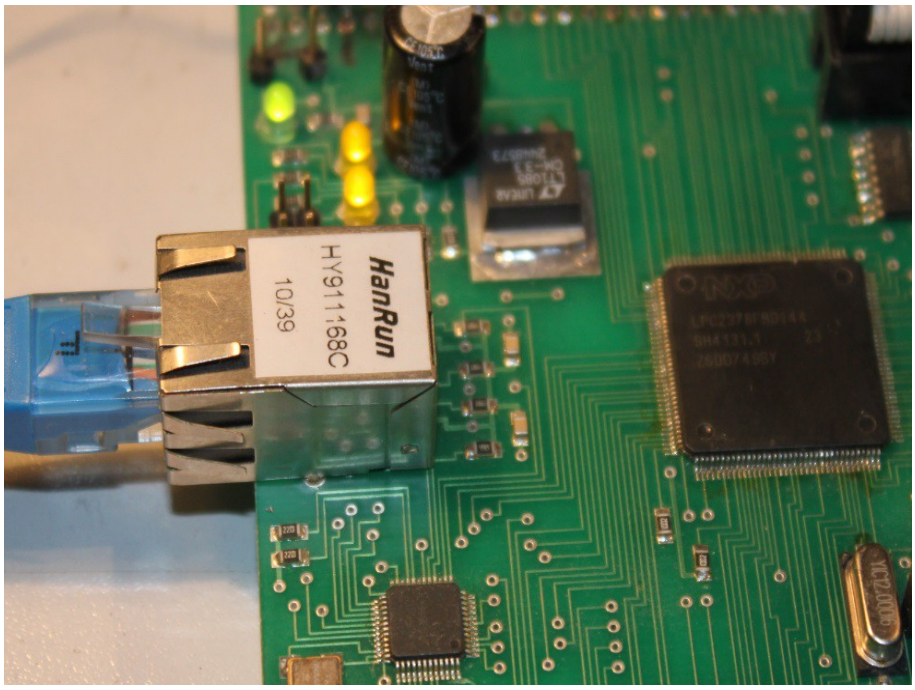
Andrzej Pawluczuk

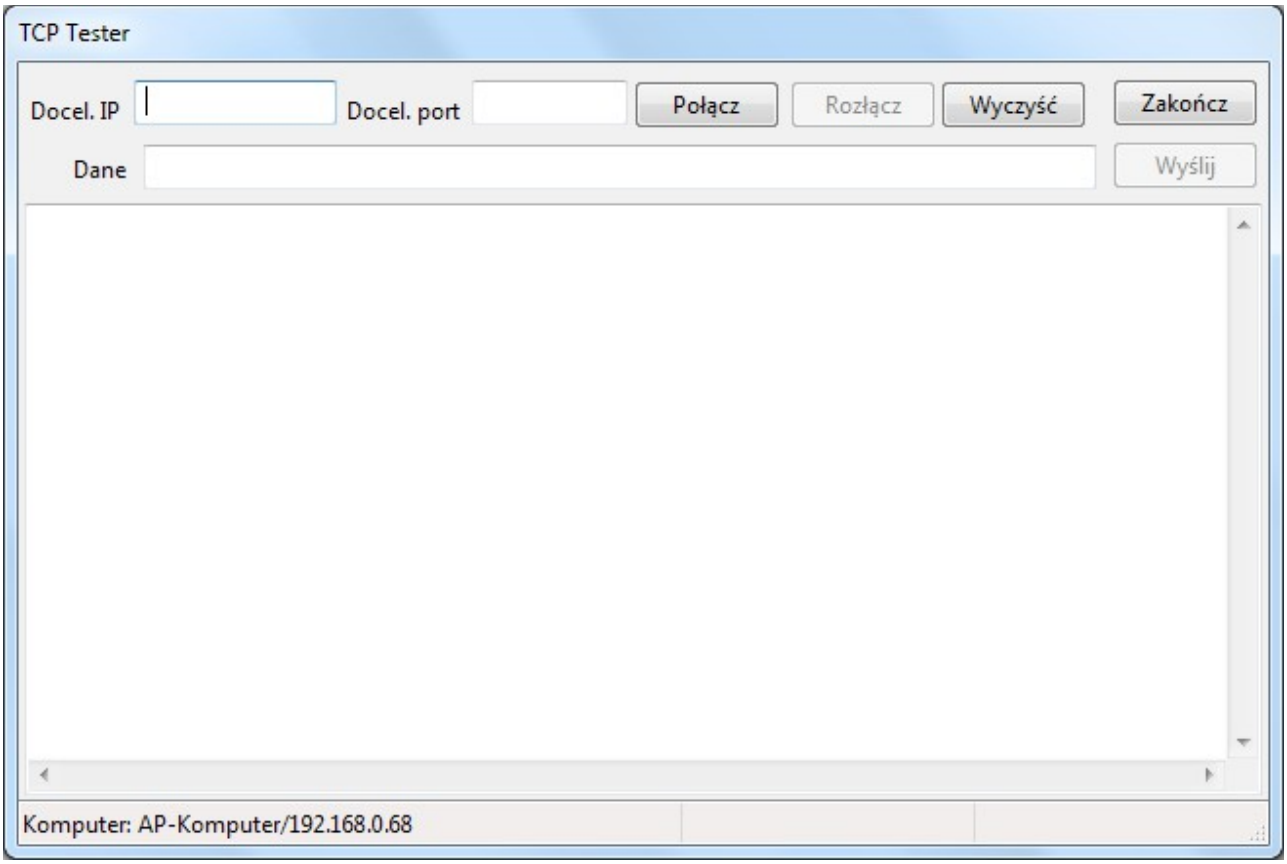


Prosty program klienta TCP w LAZARUS

Białystok, maj 2021

TCPTESTER to prosty program klienta posługujący się protokołem sieci TCP. Pozwala on na nawiązanie połączenia do określonego serwera (identyfikowanego przez adres IP i numer portu) i dalej na wysyłanie oraz odbieranie z sieci strumienia TCP. Program ten jest utworzony w środowisku LAZARUS, wolnym (nie wymagającym jakiejkolwiek licencji) narzędziu pozwalającym na tworzenie programów w języka PASCAL. Środowisko to jest wzorowane na DELPHI, więc każdy, kto próbował swych sił w DEPLHI bez problemów będzie się poruszał w LAZARUS.

By program łatwo dawał się uruchomić, najprościej jest utworzyć odpowiedni skrót na pulpicie. Po uruchomieniu programu ukazuje się okienko jak na ilustracji 4.

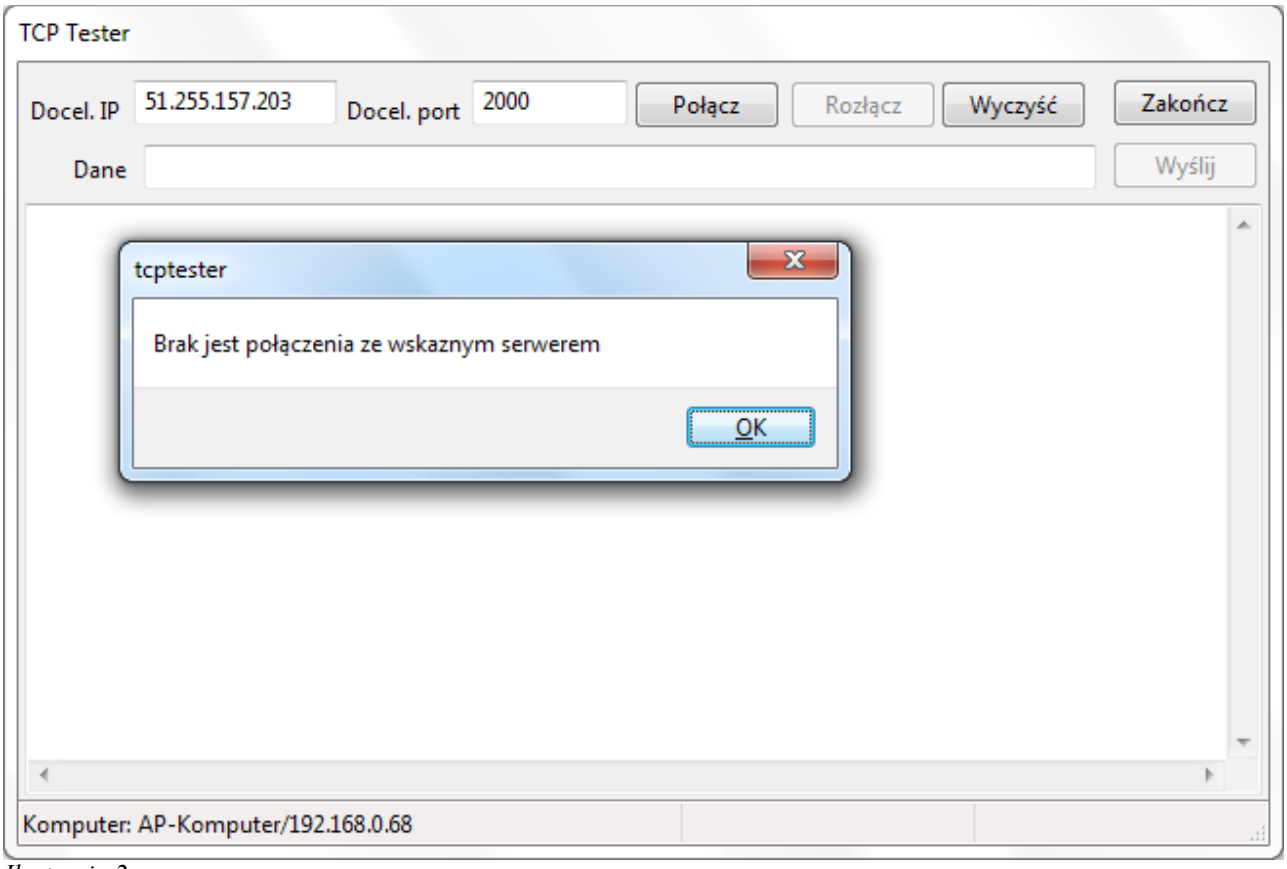


Ilustracja 1

Pole „*Docel. IP*” jest przewidziane na wprowadzenie adresu IP miejsca docelowego. Podobną funkcję spełnia „*Docel. Port*” – określa numer portu TCP docelowego miejsca. Te pola muszą być wypełnione przed operacją nawiązania połączenia. Połączenie następuje po kliknięciu na przycisk „*Połącz*”. Przy operacji połączenia tworzone jest gniazdko TCP, którego numer portu jest wylosowany przez windows (adres IP jest adresem komputera → te dane są pokazane na dole okienka). Wysyłaną treść należy wpisać w polu „*Dane*”. Kliknięcie na przycisk „*Wyślij*” wysyła wpisaną treść jako dane w nawiązanym połączeniu TCP.

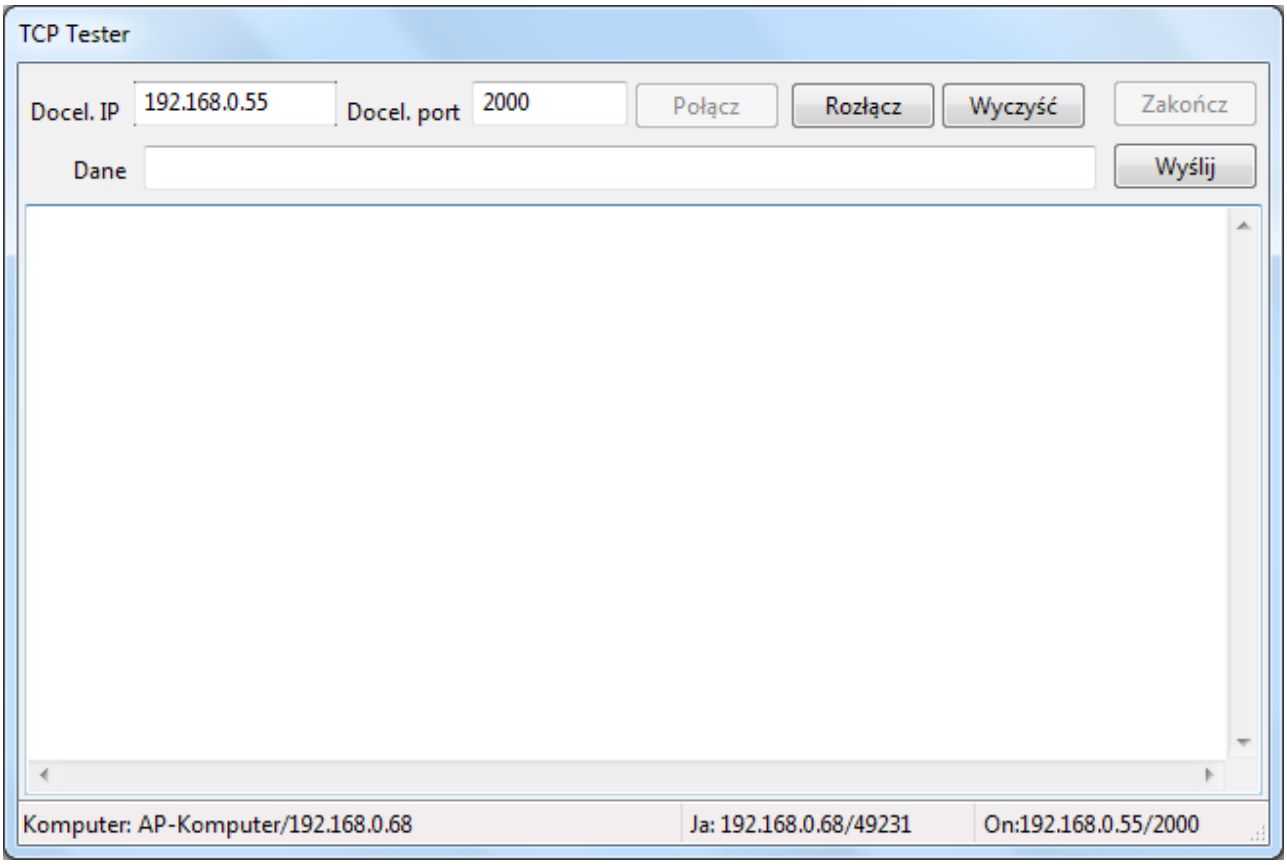
W okienku są wyświetlane dane, które są nadawane oraz odbierane. „Charakter” informacji jest oznaczony znaczkiem „<” lub „>”. Symbol „<” oznacza, że dane zostały wysłane, symbol „>” oznacza, że dane zostały odebrane (ilustracja 4).

Próba połączenia do stacji, która nie jest osiągalna jest sygnalizowana następująco: ilustracja 2 (serwer elportal.pl z pewnością nie nasłuchuje na porcie o numerze 2000).



Ilustracja 2

W sytuacji nawiązania połączenia, jego szczegóły są wyświetlone na dole okienka programu (ilustracja 3).



Ilustracja 3

Zakończenie „sesji komunikacyjnej” wymaga zamknięcia połączenia, do tego służy przycisk „*Rozłącz*”. Po zamknięciu połączenia możliwe jest nawiązanie kolejnego połączenia.

Budowa programu

Program, jak wspomniano wyżej jest utworzony w środowisku LAZARUS. Całość akcji rozgrywa się w „unicie” zawartym w pliku *tcptesterunit.pas*. Jest tam zdefiniowany komponent programu.

```
unit TCPTesterUnit ;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs,
  StdCtrls, ComCtrls, ExtCtrls, blcksock;
```

Użyty w programie komponent obsługi sieci zawarty jest w module *blockscok* i musi być on wymieniony na liście *uses*.

```
type

  { TTCPTesterForm }

  TTCPTesterForm = class(TForm)
    DestIPLabel                 : TLabel ;
    DestPortLabel               : TLabel ;
    DataLabel                   : TLabel ;
    ConnectButton               : TButton ;
    DisconnectButton            : TButton ;
    ExitButton                  : TButton ;
    ClearButton                 : TButton ;
    SendDataButton              : TButton ;
    DestIPEdit                  : TEdit ;
    DestPortEdit                : TEdit ;
    DataEdit                    : TEdit ;
    StatusBar                   : TStatusBar ;
    GeneralTimer                : TTimer ;
    TraceMemo                   : TMemo ;
    procedure ClearButtonClick ( Sender : TObject ) ;
    procedure ConnectButtonClick ( Sender : TObject ) ;
    procedure DisconnectButtonClick ( Sender : TObject ) ;
    procedure ExitButtonClick ( Sender : TObject ) ;
    procedure FormCreate ( Sender : TObject ) ;
    procedure GeneralTimerTimer ( Sender : TObject ) ;
    procedure SendDataButtonClick ( Sender : TObject ) ;
  private
    TCPSocket                 : TTCPBlockSocket ;
```

Element komponentu obsługi programu:
TCPSocket : TTCPBlockSocket.

```
    IPAddress                 : String ;
    { private declarations }
  public
    function AcceptOperation : boolean ;
    { public declarations }
  end;

var
```

```
  TCPTesterForm: TTCPTesterForm;

implementation

{$R *.lfm}

const
  MachinePanelIndex              = 0 ;
  SrcPanelIndex                  = 1 ;
  DestPanelIndex                 = 2 ;


function ValidPortNo ( Port : String ) : boolean ;
```

Funkcja, której zadaniem jest weryfikacja, czy wprowadzony numer portu jest poprawny (ma być liczbą 16-bitową różną od zera). Jest funkcją pomocniczą w operacji nawiązania połączenia.

```
var
  PortNumer             : integer ;

begin (* ValidPortNo *)
  if not TryStrToInt ( Port , PortNumer ) then
  begin (* 1 *)
    ShowMessage ( 'Niepoprawny numer portu.' ) ;
    ValidPortNo := false ;
    exit ;
  end (* 1 *) ;
  if ( PortNumer <= 0 ) or ( PortNumer > 65535 ) then
  begin (* 1 *)
    ShowMessage ( 'Numer portu spaza zakresu' ) ;
    ValidPortNo := false ;
    exit ;
  end (* 1 *) ;
  ValidPortNo := true ;
end (* ValidPortNo *) ;


function TTCPTesterForm . AcceptOperation : boolean ;
begin (* TTCPTesterForm . AcceptOperation *)
  if MessageDlg ( 'Potwierdzenie' , 'Czy potwierdzasz operacjÄ™?' ,
mtConfirmation , [ mbYes , mbNo ] , 0 ) = mrYes then
    AcceptOperation := true
  else
    AcceptOperation := false ;
end (* TTCPTesterForm . AcceptOperation *) ;


procedure TTCPTesterForm.FormCreate ( Sender : TObject ) ;
```

Dodatkowa funkcja tworząca komponent programu. Operacja tworzenia może zostać „przechwycona” (to realizuje procedura *TTCPTesterForm.FormCreate*) i można do tego wtrącić swoje trzy grosze. Jest ona wywołana jednorazowo w chwili tworzenia komponentu programu, czyli uruchomienia programu.

```
var
  MachineName          : String ;

begin (* TTCPTesterForm.FormCreate *)
```

Tworzy nowy komponent związany z obsługą sieci.

```
  TCPSocket := TTCPBlockSocket . Create ;
```

```
  TCPSocket . Family := SF_IP4 ;
```

Określa, że jest to sieć w wariancie IPV4 (standardowa sieć, do której jesteśmy przyzwyczajeni i adresach pisanych w notacji czterech liczb rozdzielonych znakiem kropki). Ponieważ (w sieciach rozległych) zaczyna brakować „wolnych" adresów, powoli do użytku wchodzi sieć w wariancie IPV6 (na razie w sieciach domowych nie szybko się zadomowi).

```
  MachineName := TCPSocket . LocalName ;
```

Wywołanie TCP*Socket.LocalName* dostarcza od windozy nazwę naszego komputera. Jest to niezbędny element, by poznać własny adres IP.

```
  IPAddress := TCPSocket . ResolveName ( MachineName ) ;
```

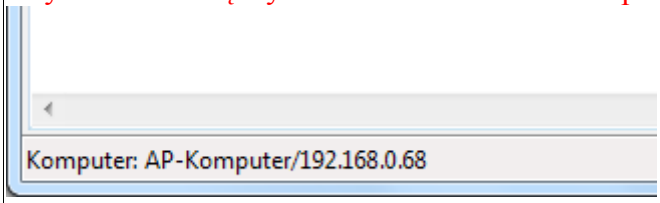
Wywołanie funkcji TCP*Socket.ResolveName*, gdzie w parametrach jest podana nazwa komputera, daje jego adres IP (czyli adres IP naszego komputera).

```
  TCPSocket . Destroy ;
```

Po tej operacji komponent związany z obsługą sieci jest usuwany. Jego istnienie ma sens jedynie w trakcie istnienia sesji połączeniowej. Tutaj był utworzony jedynie w celu „zdobycia" adresu własnego komputera.

```
  StatusBar . Panels . Items [ MachinePanelIndex ] . Text := 'Komputer: ' +
MachineName + '/' + IPAddress ;
```

Uzyskane dane są wyświetlone na dole okienka programu.




```
  DisconnectButton . Enabled := false ;
  SendDataButton . Enabled := false ;
  GeneralTimer . Enabled := false ;
end (* TTCPTesterForm.FormCreate *) ;


procedure TTCPTesterForm.ExitButtonClick ( Sender : TObject ) ;
begin (* TTCPTesterForm.ExitButtonClick *)
  if AcceptOperation then
    Application . Terminate ;
end (* TTCPTesterForm.ExitButtonClick *) ;


procedure TTCPTesterForm.SendDataButtonClick ( Sender : TObject ) ;
```

Procedura do wysyłania danych jako strumień TCP.

```
var
  Message                : string ;

begin (* TTCPTesterForm.SendDataButtonClick *)
  Message := DataEdit . Text ;
  TCPSocket . SendString ( DataEdit . Text ) ;
```

Samo fizyczne wysłanie to wywołanie *TCPSocket.SendString*.

```
  TraceMemo . Lines . Add ( '<' + Message ) ;
end (* TTCPTesterForm.SendDataButtonClick *) ;
```

Wyświetlenie informacji w okienku wraz ze znacznikiem '<' symbolizującym dane wychodzące.

```
procedure TTCPTesterForm.ConnectButtonClick ( Sender : TObject ) ;
```

Obsługa kliknięcia na przycisk „Połącz”. Wymagane jest, by w odpowiednich okienkach był wpisany adres IP miejsca docelowego oraz numer portu TCP, na który należy nawiązać połączenie.

```
var
  DestinationAddress     : string ;
  DestinationPort        : string ;
  LocalPortNo            : integer ;

  function ValidAddress ( Address : String ) : boolean ;
```

Wewnątrz procedury wykorzystywana jest pomocnicza funkcja weryfikująca poprawność wprowadzonego adresu IP (ma być w powszechnie stosowanej notacji czterech liczb rozdzielonych kropkami).

```
  var
    Index                : cardinal ;
    StrSize              : cardinal ;

    function ExtractNumber : boolean ;
    var
      SubStr             : string ;
      Numb               : integer ;
      Ch                 : char ;
      LoopExit           : boolean ;

    begin (* ExtractNumber *)
      SubStr := '' ;
      LoopExit := false ;
      repeat
        if Index > StrSize then
        begin (* 1 *)
          LoopExit := true ;
        end (* 1 *)
        else
        begin (* 1 *)
          Ch := Address [ Index ] ;
          Index := Index + 1 ;
          if Ch = '.' then
            LoopExit := true
          else
            SubStr := SubStr + Ch ;
        end (* 1 *)
      until LoopExit ;
      if Length ( SubStr ) <= 0 then
      begin (* 1 *)
        ExtractNumber := false ;
        exit ;
      end (* 1 *) ;
      if not TryStrToInt ( SubStr , Numb ) then
      begin (* 1 *)
        ExtractNumber := false ;
        exit ;
      end (* 1 *) ;
```

```
      if Numb > 255 then
        ExtractNumber := false
      else
        ExtractNumber := true ;
    end (* ExtractNumber *) ;

  begin (* ValidAddress *)
    StrSize := Length ( Address ) ;
    Index := 1 ;
    if not ExtractNumber then
    begin (* 1 *)
      ValidAddress := false ;
      exit ;
    end (* 1 *) ;
    if not ExtractNumber then
    begin (* 1 *)
      ValidAddress := false ;
      exit ;
    end (* 1 *) ;
    if not ExtractNumber then
    begin (* 1 *)
      ValidAddress := false ;
      exit ;
    end (* 1 *) ;
    if not ExtractNumber then
    begin (* 1 *)
      ValidAddress := false ;
      exit ;
    end (* 1 *) ;
    ValidAddress := true ;
  end (* ValidAddress *) ;

begin (* TTCPTesterForm.ConnectButtonClick *)
```

Weryfikacja poprawności wprowadzonych danych.

```
  DestinationAddress := Trim ( DestIPEdit . Text ) ;
  if Length ( DestinationAddress ) <= 0 then
  begin (* 1 *)
    DestinationAddress := '127.0.0.1' ;
    DestIPEdit . Text := DestinationAddress ;
  end (* 1 *) ;
  if not ValidAddress ( DestinationAddress ) then
  begin (* 1 *)
    ShowMessage ( 'Niepoprawny adres IP.' ) ;
    exit ;
  end (* 1 *) ;
  DestinationPort := Trim ( DestPortEdit . Text ) ;
  if not ValidPortNo ( DestinationPort ) then
    exit ;
```

Nawiązanie połączenia wymaga utworzenia gniazda TCP (windows nada mu jakiś numer portu w dostępnej w windozie puli). W sumie numer portu klienta nie ma znaczenia.

```
  TCPSocket := TTCPBlockSocket . Create ;
  TCPSocket . Family := SF_IP4 ;
```

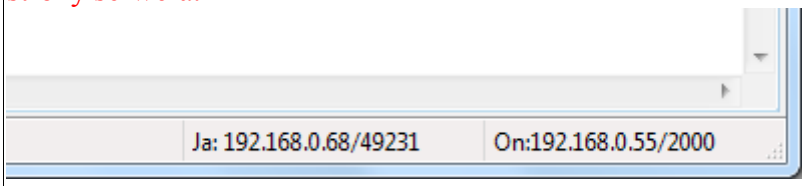
Realizacja nawiązania połączenia. Funkcja *TCPSocket.Connect* dostaje w parametrach adres IP i numer portu miejsca docelowego.

```
  TCPSocket . connect ( DestinationAddress , DestinationPort ) ;
```

funkcja *TCPSocket.GetLocalSinPort* zwraca numer portu, jeżeli ten numer portu wynosi 0, to połączenie nie zostało nawiązane (serwer nie jest dostępny).

```
  LocalPortNo := TCPSocket . GetLocalSinPort ;
  if LocalPortNo = 0 then
  begin (* 1 *)
    ShowMessage ( 'Brak jest połączenia ze wskaznym serwerem' ) ;
    TCPSocket . Destroy ;
    exit ;
  end (* 1 *) ;
  TraceMemo . Clear ;
```

Na dole okienka wyświetlone zostają parametry połączenia: „Ja” to adres IP i numer portu strony klienta, „On” adres IP i numer portu strony serwera.




```
  StatusBar . Panels . Items [ SrcPanelIndex ] . Text := 'Ja: ' + IPAddress +
'/' + IntToStr ( LocalPortNo ) ;
  StatusBar . Panels . Items [ DestPanelIndex ] . Text := 'On:' +
DestinationAddress + '/' + DestinationPort ;
```

Po połączeniu uruchamiany zostaje zegar (*GeneralTimer.Enabled:=true;*), który bierze udział w odbiorze danych ze strumienia TCP.

```
  GeneralTimer . Enabled := true ;
  DisconnectButton . Enabled := true ;
  SendDataButton . Enabled := true ;
  ConnectButton . Enabled := false ;
  ExitButton . Enabled := false ;
end (* TTCPTesterForm.ConnectButtonClick *) ;


procedure TTCPTesterForm.DisconnectButtonClick ( Sender : TObject ) ;
```

Obsługa kliknięcia na przycisk „Rozłącz”. Zatrzymany jest zegar (*GeneralTimer.Enabled:=false;*) i zwolniony komponent obsługi sieci (*TCPSocket.Destroy*). Usunięcie komponentu zawiera w sobie operacją zamknięcia połączenia (wysłanie sygnału FIN).

```
begin (* TTCPTesterForm.DisconnectButtonClick *)
  if AcceptOperation then
  begin (* 1 *)
    TCPSocket . Destroy ;
    StatusBar . Panels . Items [ SrcPanelIndex ] . Text := '' ;
    StatusBar . Panels . Items [ DestPanelIndex ] . Text := '' ;
    GeneralTimer . Enabled := false ;
    DisconnectButton . Enabled := false ;
    SendDataButton . Enabled := false ;
    ConnectButton . Enabled := true ;
    ExitButton . Enabled := true ;
  end (* 1 *) ;
end (* TTCPTesterForm.DisconnectButtonClick *) ;


procedure TTCPTesterForm.ClearButtonClick ( Sender : TObject ) ;
begin (* TTCPTesterForm.ClearButtonClick *)
```

```
  if AcceptOperation then
    TraceMemo . Clear ;
end (* TTCPTesterForm.ClearButtonClick *) ;


procedure TTCPTesterForm.GeneralTimerTimer ( Sender : TObject ) ;
```

W programie co określony (konfigurowalny) interwał czasu wywoływana jest funkcja *TTCPTesterForm.GeneralTimerTimer*. Jej zadaniem jest sprawdzenie, czy windoza nie ma odebranych danych przeznaczonych dla nas (przysłanych do nas przez stronę serwera).

```
  var
    Data                  : String ;

begin (* TTCPTesterForm.GeneralTimerTimer *)
  GeneralTimer . Enabled := false ;
  Data := TCPSocket . RecvPacket ( 0 ) ;
```

Próba odebrania danych...

```
  if length ( Data ) <> 0 then
```

… jeżeli zostały odebrane jakieś dane (długość łańcucha znaków jest niezerowa), to...

```
    TraceMemo . Lines . Add ( '>' + Data ) ;
```

… są wyświetlone w okienku wraz ze znacznikiem '>' symbolizującym dane odebrane.

```
  GeneralTimer . Enabled := true ;
end (* TTCPTesterForm.GeneralTimerTimer *) ;


end.
```