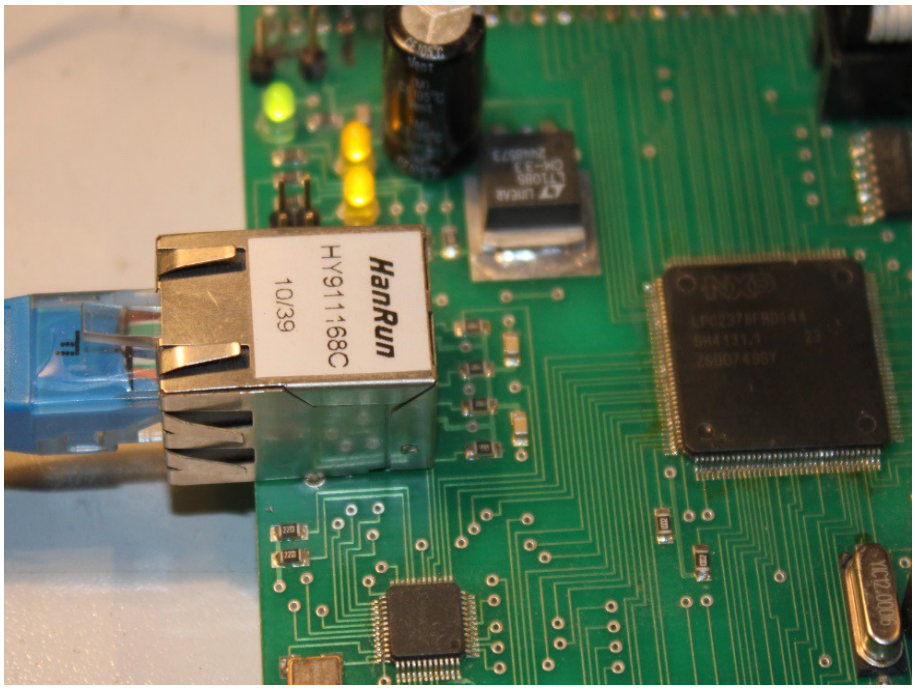
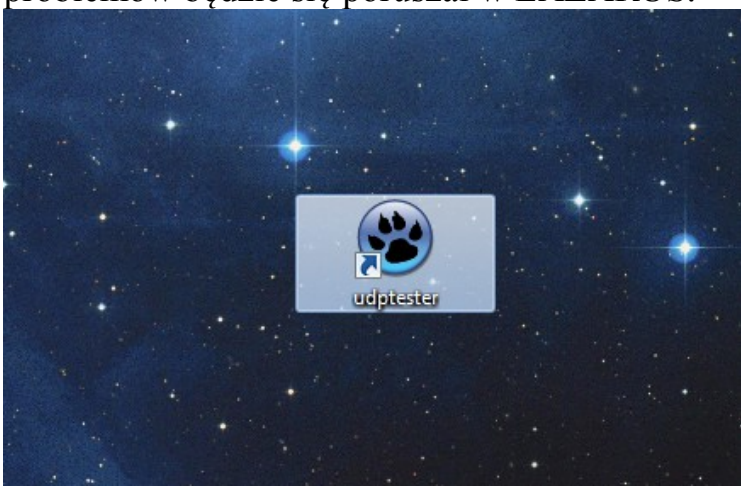
Andrzej Pawluczuk



Prosty klient/serwer UDP w LAZARUS

Białystok, grudzień 2020

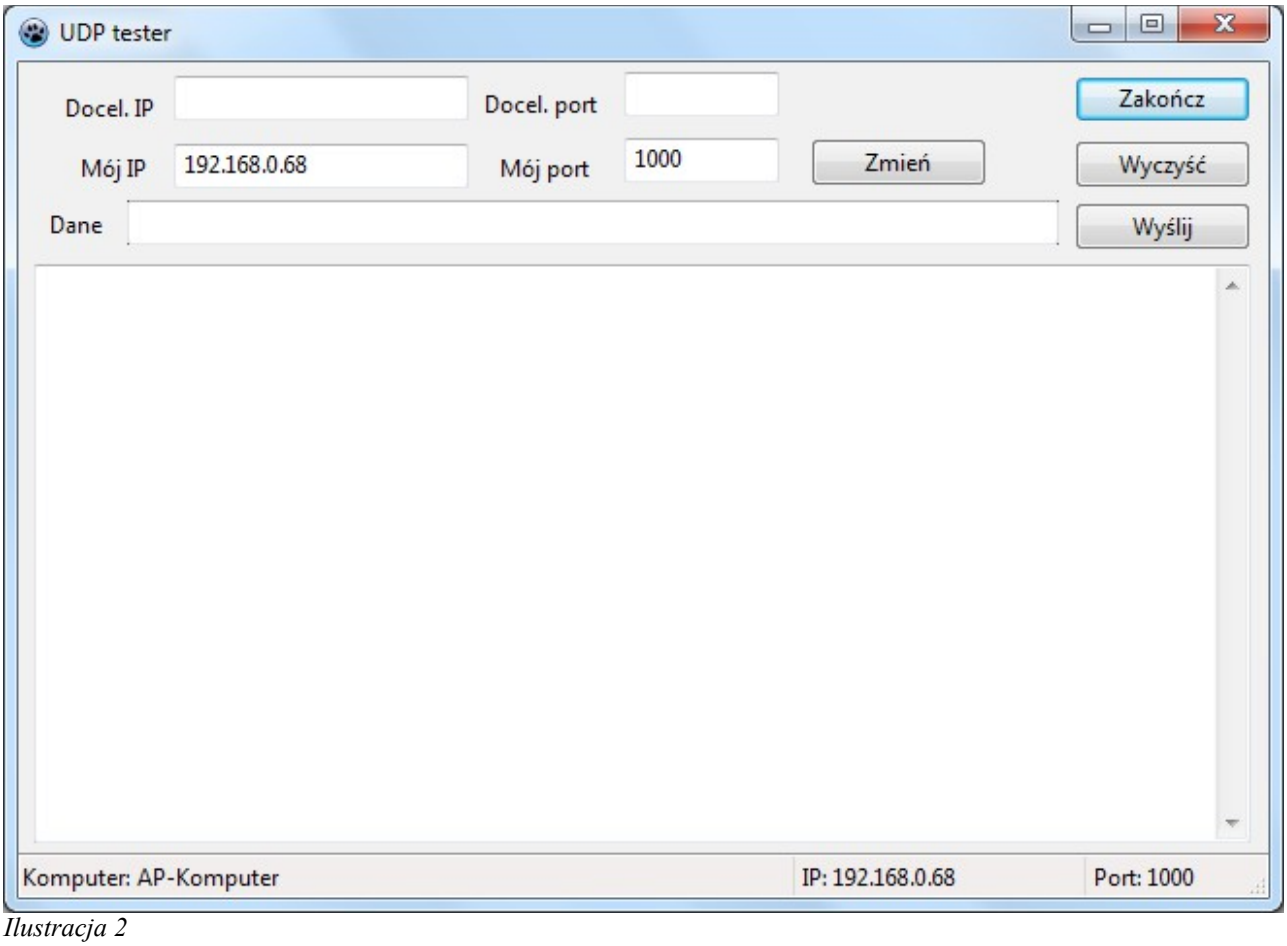
UDPTESTER to prosty program obsługujący protokół sieciowy UDP. Pozwala on na wysyłanie oraz odbieranie z sieci datagramów (pakietów) UDP. Jak wiadomo z własności sieci, protokół UDP jest najprostszą możliwością przesyłania danych via sieć ethernetowa. Program ten jest utworzony w środowisku LAZARUS, 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.



Ilustracja 1

By program łatwo dawał się uruchomić, najprościej jest utworzyć odpowiedni skrót na pulpicie. Uruchomienie sprowadza się do dwukliku na ikonkę programu.

Po uruchomieniu programu ukazuje się okienko jak na ilustracji 2.

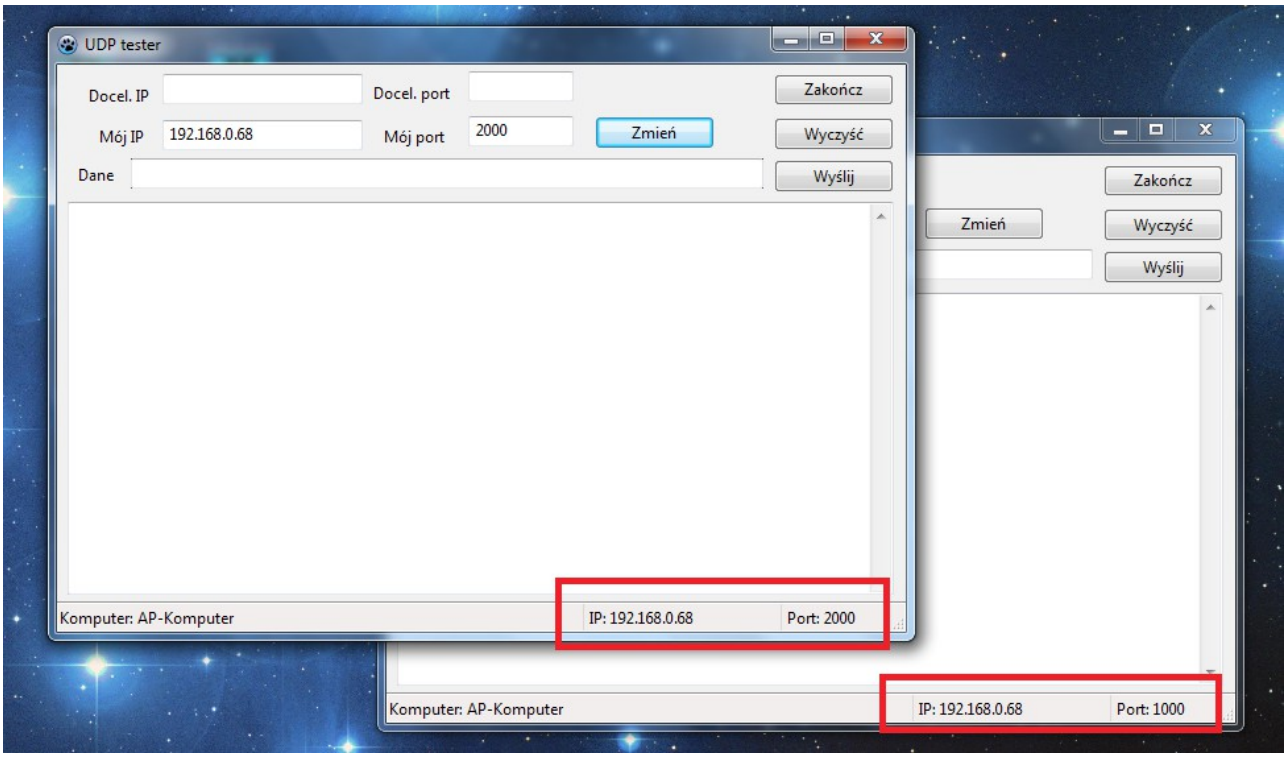


Ilustracja 2

W polu „Mój IP” jest wyświetlony adres IP komputera, w którym został uruchomiony program (tego pola nie daje się edytować). Obok jest pole na wprowadzenie numer portu UDP, poprzez który program będzie odbierał pakiety UDP. Wstępnie jest zaproponowany numer portu *1000*, jednak można go zmienić. W tym celu należy wpisać nową wartość i kliknąć na przycisk „Zmień”.

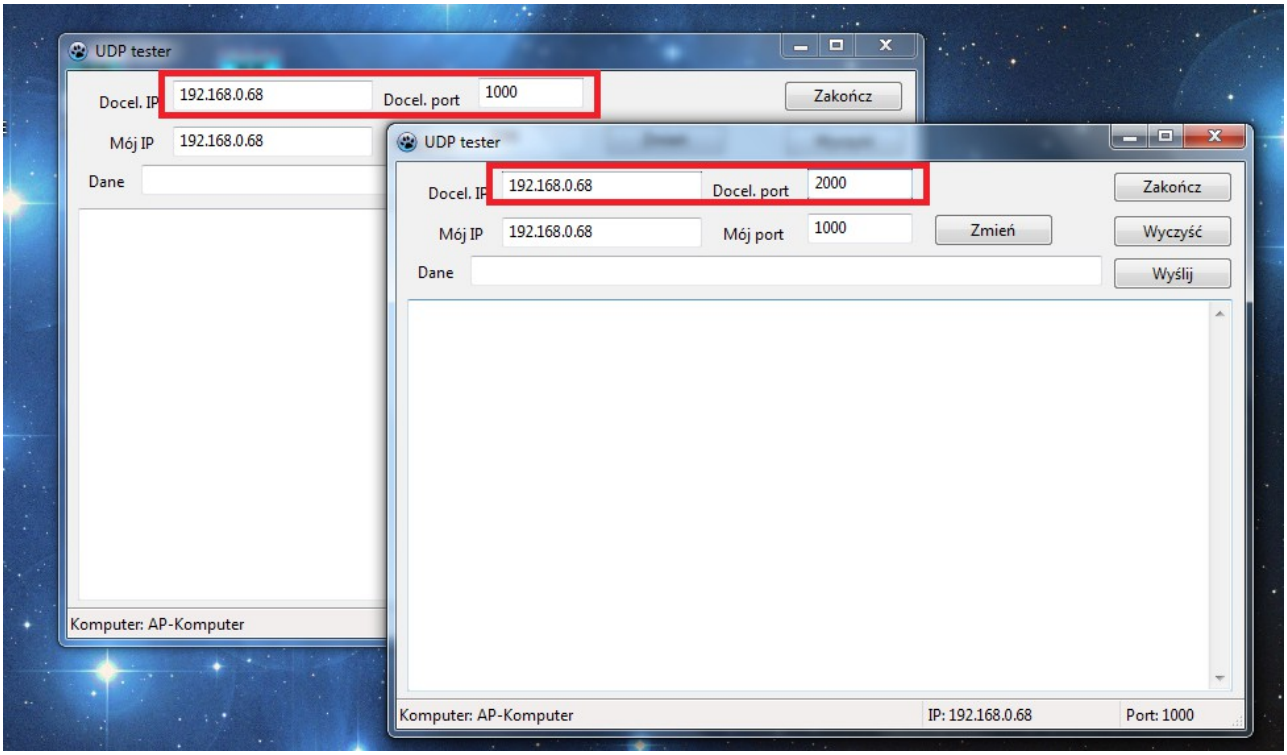
Pole „*Docel. IP*” jest przewidziane na wprowadzenie adresu IP miejsca docelowego. Podobną funkcję spełnia „*Docel. Port*” – określa numer portu UDP docelowego miejsca. Te pola muszą być wypełnione przed wysłaniem jakichkolwiek danych. Wysyłaną treść należy wpisać w polu „*Dane*”. Kliknięcie na przycisk „*Wyślij*” wysyła wpisaną treść jako pakiet UDP. W pakiecie tym miejscem źródłowym (nadawcą) jest to co znajduje się w okienkach „*Mój IP*” oraz „*Mój port*”. Miejscem docelowym (adresatem) jest to co jest wpisane w „*Docel. IP*” i „*Docel. Port*”.

Docelowy adres IP może również dotyczyć własnego komputera. Można wykonać eksperyment: uruchomić program dwukrotnie. W drugiej „kopii” programu zmienić numer własnego portu przykładowo na *2000* (koniecznie kliknąć na „Zmień”). Powstają dwa różne „końce” do przesyłania danych. Obsługa sieci nie „dyskutuje” tylko przesyła dane z jednego miejsca do drugiego (to, że oba „końce” są w obrębie jednego komputera nie stanowi problemu).



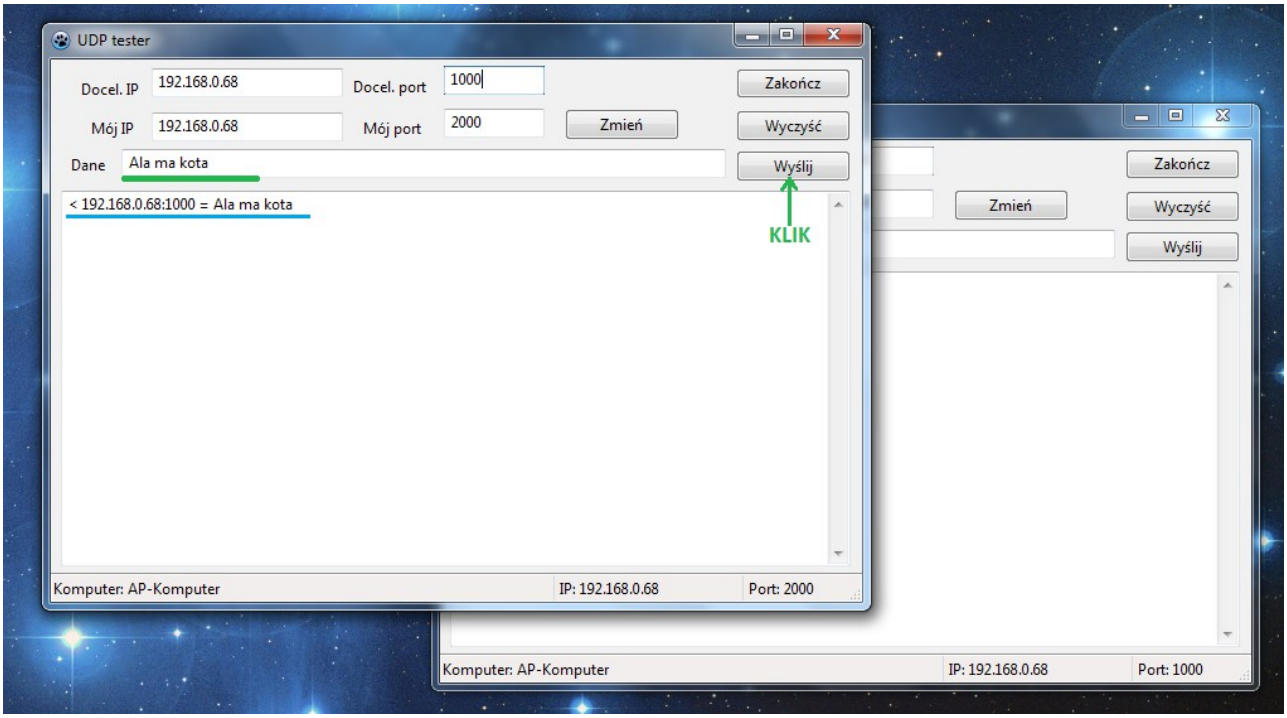
Ilustracja 3

Zostało podać jeszcze adresatów: dla każdego programu jako adres IP wpisany jest własny adres IP oraz jako numery portów są podane numery „na krzyż” (ilustracja 4).



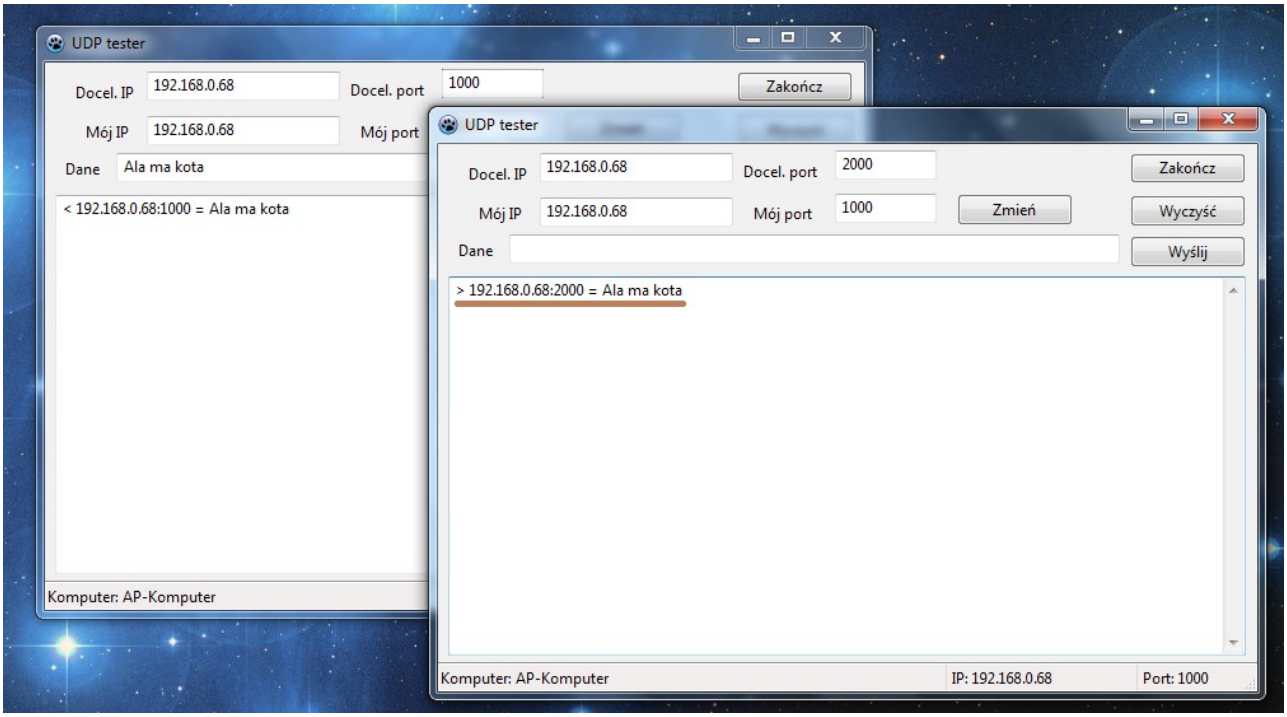
Ilustracja 4

Powstał „wirtualny kabelek” łączący dwa miejsca i jest on gotowy do transmitowania danych (ilustracja 5). Wpisane dane po kliknięciu są wysyłane do drugiej stacji.



Ilustracja 5

W drugiej stacji (ilustracja 6), dane są odebrane.



Ilustracja 6

W okienku są wyświetlane dane, które są nadawane lub odbierane. Zawierają one adres IP i numer portu miejsca docelowego (przy nadawaniu) lub adres IP i numer portu nadawcy (przy odbieraniu). „Charakter” informacji jest oznaczony znaczkiem „<” lub „>”. Symbol „<” oznacza, że dane zostały wysłane, symbol „>” oznacza, że dane zostały odebrane.

Budowa programu

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

```
unit udptesterunit ;

{$mode objfpc}{$H+}

interface

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

Do istniejącej listy unitów należy dodać ten, który zawiera obsługę sieci: *blcksock*.

```
type

  { TUDPTesterForm }

  TUDPTesterForm = class ( TForm )
    MyIPAddressLabel          : TLabel ;
    DestIPLabel               : TLabel ;
    DestPortLabel             : TLabel ;
    MyPortNoLabel             : TLabel ;
    DataLabel                 : TLabel ;
    ClearButton               : TButton ;
    ExitButton                : TButton ;
    ChangeMyPortButton        : TButton ;
    SendDataButton            : TButton ;
    DestIPEdit                : TEdit ;
    DestPortEdit              : TEdit ;
    MyPortEdit                : TEdit ;
    MyIPEdit                  : TEdit ;
    DataEdit                  : TEdit ;
    StatusBar                 : TStatusBar ;
    TextMemo                  : TMemo ;
    GeneralTimer              : TTimer ;
    procedure ClearButtonClick ( Sender : TObject ) ;
    procedure ExitButtonClick ( Sender : TObject ) ;
    procedure FormCreate ( Sender : TObject ) ;
    procedure FormDestroy ( Sender : TObject ) ;
    procedure GeneralTimerTimer ( Sender : TObject ) ;
    procedure ChangeMyPortButtonClick ( Sender : TObject ) ;
    procedure SendDataButtonClick ( Sender : TObject ) ;
  private
    UDPSocket                 : TUDPBlockSocket ;
```

Jednym z elementów komponentu programu jest *UDPSocket* typu *TUDPBlockSocket*. Definicja ta pochodzi z dołączonego unitu *blcksock*. Ten element zostanie utworzony w procedurze, która tworzy komponent programu (patrz: *TUDPTesterForm.FormCreate)*.

```
    function AcceptOperation : boolean ;
  public
    { public declarations }
```

```
  end ;

const
  DefaultUDPListenPortNo          = 1000 ;
```

Domyślny numer w polu „*mój numer portu*”.

```
  MachinePanelIndex               = 0 ;
  IPAddressPanelIndex             = 1 ;
  PortNoPanelIndex                = 2 ;
  InputDataSymbol                 = '> ' ;
  OutputDataSymbol                = '< ' ;
  NetParamSeparator               = ':' ;
  DataSeparator                   = ' = ' ;

var
  UDPTesterForm                   : TUDPTesterForm ;
```

Komponent programu.

```
implementation
{$R *.lfm}

{ TUDPTesterForm }


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


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

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

```
  var
    MachineName          : String ;
    IPAddress            : String ;
begin (* TUDPTesterForm . FormCreate *)
  UDPSocket := TUDPBlockSocket . Create ;
```

Tworzy nowy komponent związany z obsługą sieci.

```
  UDPSocket . 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).

```
  TextMemo . Clear ;
  MyPortEdit . Text := IntToStr ( DefaultUDPListenPortNo ) ;
  MachineName := UDPSocket . LocalName ;
```

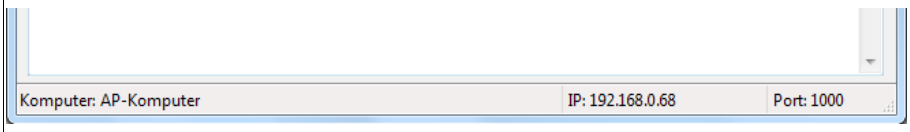
Wywołanie *UDPSocket.LocalName* dostarcza od windozy nazwę naszego komputera. Jest to niezbędny element, by poznać własny adres IP.

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

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

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

Uzyskana nazwa komputera oraz adres IP są wyświetlone w linii statusowej, numer portu to już jest znany i nie trzeba go zdobywać.




```
  MyIPEdit . Text := IPAddress ;
```

Ten sam adres IP jest umieszczony w polu edycyjnym „*Mój IP*”, jednak to pole ma ustawioną własność, że jest nieedytowalne.

```
  ChangeMyPortButtonClick ( Sender ) ;
```

Wywołanie procedury *ChangeMyPortButtonClick* symuluje kliknięcie na przycisk „*Zmień*” mój numer portu.

```
  GeneralTimer . Enabled := true ;
end (* TUDPTesterForm . FormCreate *) ;


procedure TUDPTesterForm . FormDestroy ( Sender : TObject ) ;
```

Kończenie działania programu między innymi zajmuje się usunięciem z pamięci utworzonych komponentów (przede wszystkim komponentu programu). Podobnie jak w *FormCreate*, można „przechwycić” część operacji związanych z usuwaniem komponentów. Taką funkcję ma *TUDPTesterForm.FormDestroy* i w niej jest usuwany utworzony komponent do obsługi sieci.

```
begin (* TUDPTesterForm . FormDestroy *)
  UDPSocket . Destroy ;
end (* TUDPTesterForm . FormDestroy *) ;


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

W programie co określony (konfigurowalny) interwał czasu wywoływana jest funkcja *TUDPTesterForm.GeneralTimerTimer*. Jej zadaniem jest sprawdzenie, czy windoza nie ma odebranego pakietu UDP przeznaczonego dla nas (adresowanego na port o numerze „*Mój numer portu*” → gniazdko do nasłuchu).

```
  var
    MemoMessage          : String ;
    Data                 : String ;
    RemIPAdr             : String ;
    RemPort              : integer ;
begin (* TUDPTesterForm . GeneralTimerTimer *)
  GeneralTimer . Enabled := false ;
  Data := UDPSocket . RecvPacket ( 0 ) ;
```

Próba odebrania pakietu...

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

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

```
  begin (* 1 *)
    RemIPAdr := UDPSocket . GetRemoteSinIP ;
```

… odczytany jest adres IP nadawcy...

```
    RemPort := UDPSocket . GetRemoteSinPort ;
```

… i numer portu nadawcy.

```
    MemoMessage := InputDataSymbol + RemIPAdr + NetParamSeparator +
                   IntToStr ( RemPort ) + DataSeparator + Data ;
```

Powstaje łańcuch znaków jako połączenie:

- znaczka „>”
- odczytany adres IP
- znak „:” jako separator pomiędzy adresem IP i numerem portu,
- numer portu (który należy przekonwertować z postaci binarnej na znakową),
- kolejny separator „=”
- odebrane dane.

```
    TextMemo . Lines . Add ( MemoMessage ) ;
```

Utworzony łańcuch znaków zostaje wyświetlony w okienku programu.

```
  end (* 1 *) ;
  GeneralTimer . Enabled := true ;
end (* TUDPTesterForm . GeneralTimerTimer *) ;


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 operacjach wprowadzenia numeru portu.

```
  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 *) ;


procedure TUDPTesterForm.ChangeMyPortButtonClick ( Sender : TObject ) ;
```

Reakcja programu na kliknięcie na przycisk „Zmień" mój numer portu.

```
  var
    PortNo                : String ;

begin (* TUDPTesterForm.ChangeMyPortButtonClick *)
  PortNo := Trim ( MyPortEdit . Text ) ;
  if not ValidPortNo ( PortNo ) then
    exit ;
```

Jest wyekstrahowany tekst zawierający numer portu (jako pole będące łańcuchem znaków). Zapis numeru portu podlega sprawdzeniu, czy jest to liczba z właściwego zakresu.

```
  UDPSocket . Destroy ;
```

Istniejący komponent obsługi sieci ulega rozwiązaniu *UDPSocket . Destroy* (jest on wstępnie utworzony w procedurze *FormCreate* → tworzącej komponent programu i przy okazji komponent obsługi sieci).

```
  UDPSocket := TUDPBlockSocket . Create ;
```

Ponownie utworzony...

```
  UDPSocket . Bind ( '0.0.0.0' , MyPortEdit . Text ) ;
```

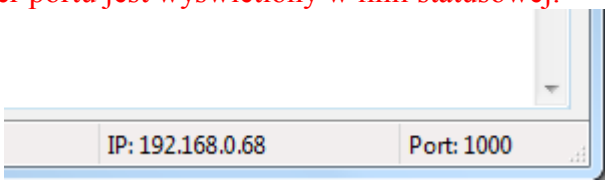
nadany mu wymagany numer portu (wyżej występujące utworzenie nadaje mu numer z wolnej puli systemu i nie musi to być numer, którego oczekujemy).

```
  UDPSocket . Listen ;
```

Utworzone gniazdko UDP jest przełączone do nasłuchu.

```
  StatusBar . Panels . Items [ PortNoPanelIndex ] . Text := 'Port: ' +
                                                    MyPortEdit . Text ;
```

Nowa numer portu jest wyświetlony w linii statusowej.




```
end (* TUDPTesterForm.ChangeMyPortButtonClick *) ;


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

Procedura do wysyłania danych jako pakiety UDP.

```
  var
    MemoMessage           : String ;
    DestinationAddress    : String ;
    DestinationPort       : String ;
    DataText              : String ;

  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 (* TUDPTesterForm . SendDataButtonClick *)
  DestinationAddress := Trim ( DestIPEdit . Text ) ;
  if Length ( DestinationAddress ) <= 0 then
```

Jeżeli pole na docelowy adres IP jest <puste>, to...

```
  begin (* 1 *)
    DestinationAddress := '127.0.0.1' ;
```

...program postawi tam adres IP=127.0.0.1 (adres zawsze oznaczający: do samego siebie").

```
    DestIPEdit . Text := DestinationAddress ;
  end (* 1 *) ;
  if not ValidAddress ( DestinationAddress ) then
```

Przed wysłaniem pakietu UDP weryfikowana jest poprawność zapisu docelowego adresu IP...

```
  begin (* 1 *)
    ShowMessage ( 'Niepoprawny adres IP.' ) ;
    exit ;
  end (* 1 *) ;
  DestinationPort := Trim ( DestPortEdit . Text ) ;
```

Weryfikowany jest numer portu.

```
  if not ValidPortNo ( DestinationPort ) then
    exit ;
  DataText := DataEdit . Text ;
  if Length ( DataText ) <= 0 then
    DataText := '<puste>' ;
```

Jeżeli wysyłany łańcuch znaków jest pusty, to jest podmieniany na odpowiedni napis (ot, taka fantazja programisty).

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

Wywołanie procedury *UDPSocket.connect* określa docelowe miejsce na transmitowane dane. Używając protokołu UDP nie zachodzi żadne nawiązanie połączenia (nazwa funkcji *connect* jest trochę „na wyrost" i może być myląca).

```
  UDPSocket . SendString ( DataText ) ;
```

Samo fizyczne wysłanie.

```
  MemoMessage := OutputDataSymbol + DestinationAddress + NetParamSeparator +
                 DestinationPort + DataSeparator + DataText ;
```

Powstaje łańcuch znaków jako połączenie:

- znaczka „<"
- docelowego adresu IP
- znak „:" jako separator pomiędzy adresem IP i numerem portu,
- docelowego numeru portu
- kolejny separator „="
- wysyłane dane.

```
  TextMemo . Lines . Add ( MemoMessage ) ;
```

Utworzony łańcuch znaków zostaje wyświetlony w okienku programu.

```
end (* TUDPTesterForm . SendDataButtonClick *) ;
```

```
procedure TUDPTesterForm . ExitButtonClick ( Sender : TObject ) ;
```

Reakcja programu na kliknięcie na przycisk „Zakończ” → po akceptacji kończy się działanie programu.

```
begin (* TUDPTesterForm . ExitButtonClick *)
  if AcceptOperation then
    Application . Terminate ;
end (* TUDPTesterForm . ExitButtonClick *) ;


procedure TUDPTesterForm.ClearButtonClick ( Sender : TObject ) ;
begin (* TUDPTesterForm.ClearButtonClick *)
  if AcceptOperation then
    TextMemo . Clear ;
end (* TUDPTesterForm.ClearButtonClick *) ;



end.
```