

# Capítulo 9

## Desenvolvendo utilitários para Internet

Neste capítulo, iremos desenvolver uma série de utilitários para Internet.


### Portas Abertas: Seja Bem-Vindo

O primeiro utilitário visa alertar aos amigos leitores sobre um grave problema: *Portas Abertas*.


Quanto de vocês trancam a porta de suas casas ao anoitecer, ou até mesmo à luz do dia? Acredito que a maioria. Com o forte avanço da Internet em todo o planeta, nos deparamos com o mesmo problema em nossos computadores: *Portas Abertas*. Neste capítulo iremos desenvolver um aplicativo para *scanear* e apresentar as portas que estão abertas num determinado servidor.


Para facilitar a compreensão de todos, estou apresentando um exemplo muito simples, onde o usuário informa o *Nome do Servidor (ou endereço IP)* e o *intervalo de portas a serem scanneadas*. Os leitores com conhecimentos mais avançados ou *know-how em Threads* irão me crucificar por falta das benditas. Acontece que o principal objetivo deste exemplo é demonstrar o uso do componente *TCPClient*, e não do uso de *Threads*. Isso iria complicar um pouco a compreensão dos nossos amigos leitores. Aos amigos que se encaixam neste perfil, minhas sinceras desculpas. Bem, continuando, agora vem a parte boa: *mão-na-massa*.


Vamos iniciar um novo projeto no Delphi (grave a *unit* como *un\_scan.pas* e o projeto como *pscan.drp*) e inserir os objetos que seguem:

OBJETO		
	pnTopo – TPanel	
Objeto	Propriedade	Valor
TPanel	Align	alTop
	Caption	
	Name	pnTopo


Com o foco no objeto *pnTopo* insira os seguintes objetos:


OBJETO		
	Label1 – TLabel	
Objeto	Propriedade	Valor
TLabel	Caption	HOST
	Left	16
	Top	14


OBJETO		
	Label2 – TLabel	
Objeto	Propriedade	Valor
TLabel	Caption	Porta Inicial
	Left	16
	Top	43

OBJETO		
	Label3 – TLabel	
Objeto	Propriedade	Valor
TLabel	Caption	Porta Final
	Left	232
	Top	43


Agora vamos inserir os objetos de controle e interatividade com o usuário. Mantenha o foco no objeto *pnTopo* e insira os seguintes objetos:

OBJETO		
	NomeServidor – TEdit	
Objeto	Propriedade	Valor
TEdit	Name	NomeServidor
	Left	104
	Top	10
	Width	290

OBJETO		
	Inicio– TSpinEdit [Samples]	
Objeto	Propriedade	Valor
TSpinEdit	Name	Inicio
	Left	104
	Top	40


OBJETO		
	<b>Fim– TSpinEdit [Samples]</b>	
Objeto	Propriedade	Valor
TSpinEdit	Name	Fim
	Left	320
	Top	40

Neste ponto vamos inserir o botão para iniciar o processo de *scanner de portas*.

OBJETO		
	<b>btScan– TButton</b>	
Objeto	Propriedade	Valor
TButton	Name	btScan
	Caption	Scan
	Left	416
	Top	8

Com isso concluímos a primeira etapa do projeto.


Prosseguindo, vamos criar o módulo de saída das informações. Agora com o foco no formulário e não mais no objeto *pnTopo*, insira um objeto do tipo *Tpanel* alterando as seguintes propriedades:

OBJETO		
	<b>pnDados– TPanel</b>	
Objeto	Propriedade	Valor
TPanel	Align	alClient
	Name	pnDados


Vamos inserir os objetos deste painel, como segue:


OBJETO		
	<b>ListadePortas– TMemo</b>	
Objeto	Propriedade	Valor
TMemo	Name	ListadePortas
	Height	180
	Left	16
	Top	76
	Width	500


Dentro deste mesmo painel *pnDados*, insira outro objeto do tipo *TPanel* alterando as propriedades que seguem:


OBJETO		
	<b>PnProgresso– TPanel</b>	
Objeto	Propriedade	Valor
TPanel	Name	pnProgresso
	Caption	
	Height	65
	Left	16
	Top	8
	Width	500

Amigos, sei que é um pouco cansativo, mas estamos quase no final. Com o foco no objeto *pnProgresso* insira os seguintes objetos:


OBJETO		
	<b>btParar– TButton</b>	
Objeto	Propriedade	Valor
TButton	Name	btParar
	Caption	Parar
	Left	400
	Top	32

OBJETO		
	<b>ProgressBar1– TProgressBar</b>	
Objeto	Propriedade	Valor
TProgressBar	Name	ProgressBar1
	Left	16
	Top	8
	Width	470

OBJETO		
	<b>Label4 – TLabel</b>	
Objeto	Propriedade	Valor
TLabel	Caption	Scaneando Porta...
	Left	16
	Top	36

OBJETO		
	porta – TLabel	
Objeto	Propriedade	Valor
TLabel	Caption	0
	Left	112
	Top	36

Ufa!!! E para concluir a “enorme” lista de objetos, insira um do tipo *TTCPClient* que se encontra na seção *Internet*.

OBJETO		
	TCPClient1 – TTCPClient [Internet]	
Objeto	Propriedade	Valor
TTCPClient	Name	TCPClient1

Finalmente vamos codificar o nosso projeto. Crie uma variável global pertencente à nossa *classe Form1*., com o nome *Parar*, do tipo *Integer*; veja:

```
var
  Form1: TForm1;
  parar:integer; // variavel auxiliar

implementation
```

A variável *Parar* será utilizada para finalizar o processo de *Scanner das Portas*. Neste ponto é que poderíamos criar uma *Thread*, mas não vamos complicar.

Só para aliviar um pouco a forte tensão, vamos dar uma olhadinha na interface do nosso projeto (*figura 9.1*):



*Figura 9.1 Projeto Port Scanner*

Vamos codificar o botão *btScan*, responsável pelo núcleo do nosso projeto. No evento *onClick* do objeto *btScan*, insira o código que segue (para facilitar, numerei as linhas de programação, de forma que possamos analisar melhor o código).

```

001 procedure TForm1.btScanClick(Sender: TObject);
002 var i:integer;
003 begin
004 try
005     ListadePortas.Clear; // Limpa a Lista de Portas
006
007     parar:=0; // 0 = continua, 1 = conclui
008
009     // Definições da barra de progresso
010
011     ProgressBar1.Max:=Fim.Value;
012     ProgressBar1.Min:=Inicio.Value;
013
014     PainelProgresso.Visible:=True;
015     TcpClient1.RemoteHost := NomeServidor.Text;
016
017     for i := Inicio.Value to Fim.Value do
018     begin
019         if parar=1 then break; // finaliza o laço
020
021         ProgressBar1.Position:=i;
022         porta.Caption:=inttostr(i);
023
024         Application.ProcessMessages;
025         TcpClient1.RemotePort := inttostr(i);
026         TcpClient1.Active:=true;
027
028         if TcpClient1.Connect then
029             ListadePortas.Lines.Add('Porta ['+ inttostr(i) + ']'
030             aberta');
031
032         TcpClient1.Disconnect; // desconeta porta
033
034     end; //for loop
035 except
036     on E:Exception do begin
037         ListadePortas.Lines.Add('Erro: ' + E.Message);
038     end; // end on do begin
039 end; //try block
040
041 ListadePortas.Lines.Add('Scaneamento das portas finalizado !');
042
043 PainelProgresso.Visible:=False;
044 end;

```

Vamos ao detalhamento do código:

A linha 002 declara uma variável “*i*” que irá auxiliar no laço de contagem das portas.

```
var i:integer;
```

Na linha 004 iniciamos uma proteção de erros da aplicação.

```
try
```

A linha 005 limpa o conteúdo do objeto *ListaPortas*.

```
ListadePortas.Clear
```

Na linha 007 inicializamos a variável *parar* com o valor 0, de forma que o sistema *continue scanneando* as portas até o limite solicitado pelo usuário, ou através do pressionamento da tecla *Parar*, fazendo com que a variável receba o valor 1.

```
parar:=0;
```

As linhas 011 e 012 configuram o objeto *ProgressBar1* de maneira que o mesmo fique compatível com as informações porta inicial e final. Com isso temos um progresso adequado.

```
ProgressBar1.Max:=Fim.Value;  
ProgressBar1.Min:=Inicio.Value;
```

A linha 014 torna visível o objeto *PainelProgresso*.

```
PainelProgresso.Visible:=True;
```

A linha 015 configura o servidor remoto do objeto *TcpClient1*.

```
TcpClient1.RemoteHost := NomeServidor.Text;
```

A linha 017 inicia um *loop* baseado nas informações *Porta Inicial e Final*.

```
for i := Inicio.Value to Fim.Value do
```

Já na linha 019 nossa aplicação verifica se existe a obrigação de paralisar o *loop*. Esta informação vem do botão *btParar*.

```
if parar=1 then break;
```

A linha 021 posiciona a barra de progresso em relação ao andamento do *loop*.

```
ProgressBar1.Position:=i;
```

Na linha 022 apenas mostramos ao usuário através do objeto *Porta*, qual *porta* está sendo *scanneada* no momento.

```
porta.Caption:=inttostr(i);
```

A linha 024 solicita ao Windows que processe as informações da aplicação, de maneira que a mesma não tenha o efeito *congelamento*.

```
Application.ProcessMessages;
```

A linha 025 configura a porta que deve ser *scanneada*, a 026 tenta ativar, e a 028 verifica se houve sucesso na ativação, e em caso afirmativo, a linha 029 adiciona no objeto *ListadePortas* a informação de que a *Porta* está aberta.

```
TcpClient1.RemotePort := inttostr(i);  
TcpClient1.Active:=true;  
  
if TcpClient1.Connect then  
ListadePortas.Lines.Add('Porta [' + inttostr(i) + '] aberta');
```

A linha 031 desconecta a porta independente do seu estado (aberta ou fechada).

```
TcpClient1.Disconnect;
```

A linha 032 finaliza o *loop*. As linhas 033, 034 e 035 tratam qualquer exceção ocorrida no bloco protegido (try...except...end).

```
end; //for loop
except
  on E:Exception do begin
    ListadePortas.Lines.Add('Erro: ' + E.Message);
  end; // end on do begin
end; //try block
```

Concluindo esta rotina, a linha 039 apresenta uma mensagem indicando o fim do *scaneamento das portas* e a 041 torna o objeto *PainelProgresso* invisível.

```
ListadePortas.Lines.Add('Scaneamento das portas finalizado !');

PainelProgresso.Visible:=False;
```

Para concluir o nosso projeto, devemos codificar o botão *btParar* com o seguinte código (no evento *OnClick*):

```
Parar:=1;
```

Amigos, agora é só executar o aplicativo, informar o nome do servidor (caso seja seu próprio equipamento, digite *localhost*, no campo *Nome do Servidor*), e o intervalo de portas a serem *scanneadas*. A *figura 9.2* ilustra nosso utilitário *scanneando* um servidor.



*Figura 9.2 Scanner em ação*

#### IMPORTANTE



Embora seja possível, não recomendo o uso deste aplicativo para *scannear* portas de servidores não-autorizados.

Normalmente utilizamos este tipo de aplicativo para vigiar nosso “quintal” e não o do vizinho.

*Artigo originalmente publicado na Revista The Club, nº 94*



**Listagem 9.1**

```

unit un_scan;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, IdBaseComponent, IdComponent, IdTCPConnection,
  IdTCPClient, Sockets, ComCtrls, ExtCtrls, Spin;

type
  TForm1 = class(TForm)
    TcpClient1: TTcpClient;
    pnTopo: TPanel;
    BtScan: TButton;
    NomeServidor: TEdit;
    Label2: TLabel;
    Panel2: TPanel;
    ListadePortas: TMemo;
    PainelProgresso: TPanel;
    ProgressBar1: TProgressBar;
    Label1: TLabel;
    porta: TLabel;
    Inicio: TSpinEdit;
    Label3: TLabel;
    Label4: TLabel;
    Fim: TSpinEdit;
    btParar: TButton;
    procedure BtScanClick(Sender: TObject);
    procedure btPararClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  parar:integer; // variavel auxiliar

implementation

{$R *.dfm}

procedure TForm1.BtScanClick(Sender: TObject);
var i:integer;
begin
  try
    ListadePortas.Clear; // Limpa a Lista de Portas

    parar:=0; // 0 = continua, 1 = conclui

    // Definições da barra de progresso

    ProgressBar1.Max:=Fim.Value;
    ProgressBar1.Min:=Inicio.Value;
  
```

```

PainelProgresso.Visible:=True; // Visualiza painel progresso

TcpClient1.RemoteHost := NomeServidor.Text; // define o HOST

for i := Inicio.Value to Fim.Value do
begin
    if parar=1 then break; // finaliza o laço

    ProgressBar1.Position:=i;    // atualiza ProgressBar
    porta.Caption:=inttostr(i);  // atualiza Label Porta

    Application.ProcessMessages; // solicita ao Windows prioridade no
processamento da aplicação

    TcpClient1.RemotePort := inttostr(i); // define porta a ser scaneada

    TcpClient1.Active:=true; // tenta ativar a porta

    if TcpClient1.Connect then
        ListadePortas.Lines.Add('Porta ['+ inttostr(i) + '] aberta');

    TcpClient1.Disconnect; // desconecta porta

    end; //for loop
except
    on E:Exception do begin
        ListadePortas.Lines.Add('Erro: ' + E.Message);
    end; // end on do begin
end; //try block

ListadePortas.Lines.Add('Scaneamento das portas finalizado!');

PainelProgresso.Visible:=False;
end;

procedure TForm1.btPararClick(Sender: TObject);
begin
    Parar:=1;
end;

end.

```

## Ping

Neste tópico vamos desenvolver uma nova versão para o *ping*. Mas Facunte, o que é *ping*? O utilitário *ping*, encontrado na maioria dos sistemas operacionais, como Windows e Linux, é utilizado para “*ouvir*” uma resposta de um determinado servidor, e medir o tempo da mesma. Esta resposta pode ser nula, quando não se consegue estabelecer uma comunicação com o servidor, ou terminal remoto. Normalmente os testes de *ping* são feitos em ***servidores de Internet, firewall, roteadores e servidores de banco de dados***. Vejamos um exemplo deste utilitário rodando no sistema operacional Windows:

```

C:\>ver
Microsoft Windows XP [versão 5.1.2600]
C:\>ping Servidor

Disparando contra servidor [200.173.173.142] com 32 bytes de dados:

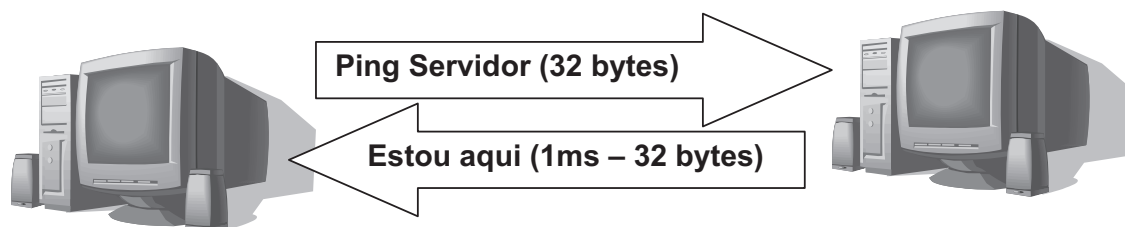
Resposta de 200.173.173.142: bytes=32 tempo<1ms TTL=128
Resposta de 200.173.173.142: bytes=32 tempo<1ms TTL=128
Resposta de 200.173.173.142: bytes=32 tempo<1ms TTL=128
Resposta de 200.173.173.142: bytes=32 tempo<1ms TTL=128

Estatísticas do Ping para 200.173.173.142:
    Pacotes: Enviados = 4, Recebidos = 4, Perdidos = 0 (0% de perda),
    Aproximar um número redondo de vezes em milissegundos:
        Mínimo = 0ms, Máximo = 0ms, Média = 0ms
C:\>

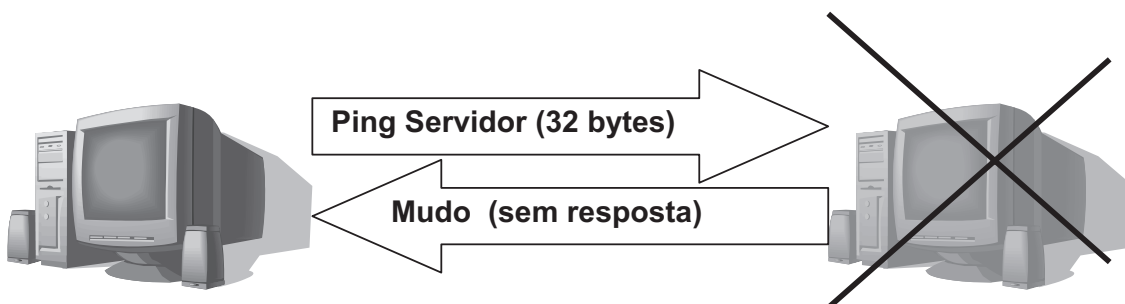
```

*Figura 9.3 – Ping sendo executado no Windows XP*

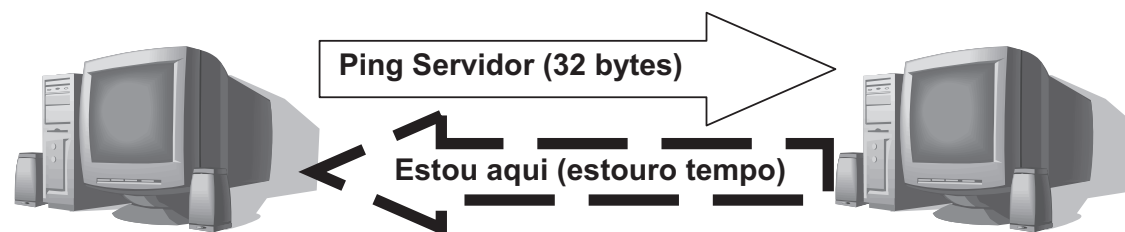
Neste exemplo, executamos o utilitário *ping* no sistema operacional Windows XP, solicitando uma resposta do computador **Servidor**. Repare que o *ping* dispara um pacote de **32 bytes** no endereço IP (**200.173.173.142**), relacionado ao **Servidor**, e “**ouve**” quatro respostas num tempo menor que 1ms (um milissegundo). Com isso poderemos medir a eficiência no tráfego de dados entre cliente/servidor, seja lá qual for a sua origem (servidor de banco de dados, aparelho celular, servidor Internet, PDA, entre outros). Veja alguns exemplos ilustrativos do utilitário *Ping* (figuras 9.4, 9.5 e 9.6).



*Figura 9.4 – Ping disparado contra o Servidor com resultado satisfatório*



*Figura 9.5 – Ping disparado contra o Servidor sem resposta*



*Figura 9.6 – Ping disparado contra o Servidor com resultado nada agradável (lento)*

Como vimos, existem três possibilidades de resultado: satisfatório, sem resposta e lento. Com isso podemos fazer testes de performance em diversos tipos de servidores, e descobrir, em alguns casos, onde está o problema de lentidão.

Bem, amigos, agora vem a parte boa: *mão-na-massa*.

Inicie um projeto do tipo CLX (Delphi 6 ou superior), pois poderemos executar nossa aplicação no Linux, compilando o projeto no Kylix. Grave a *unit* com o nome *un\_ping.pas* e o projeto como *ping.dpr*. Insira o seguinte objeto, alterando suas propriedades.

Objeto TPanel [Standard]	
Align	alTop
Name	pnEndereco

Com o foco no objeto *pnEndereco*, insira os objetos que seguem, alterando suas respectivas propriedades:

Objeto TLabel[Standard]	
Caption	Endereço
Left	12
Top	20

Objeto TEdit [Standard]	
Left	72
Name	edEndereco
Top	16
Width	210

Objeto TBitBtn [additional]	
Caption	Ping
Left	296
Name	btPing
Top	16
Width	75

Agora com o foco no formulário, insira outro objeto do tipo *TPanel* e altere as propriedades que seguem:

Objeto TPanel [Standard]	
Align	AlClient
Name	PnResultado

Com o foco no objeto *pnResultado* insira um objeto do tipo *TMemo* e altere as seguintes propriedades:

Objeto TMemo [standard]	
Height	160
Left	16
Name	mmResultado
Lines	(deixe em branco)
Top	16
Width	500

Com isso teremos um formulário parecido com a *figura 9.7*.



**Figura 9.7** Aplicação Ping em tempo de projeto

Neste ponto iremos inserir o nosso objeto chave: *TidICMPClient*. Muita “hora” nesta “calma”, ou melhor, muita “calma” nesta “hora”, amigos, não é mais um imposto do governo, é apenas o nosso componente chave.

Facunte, que nome complicado de componente! O que quer dizer ICMP? ICMP quer dizer **I**nternet **C**ontrol **M**essage **P**rotocol, ou Protocolo Internet de Controle de Mensagens. Este protocolo normalmente é utilizado pelos utilitários que iremos desenvolver, e basicamente serve para enviar mensagens a servidores remotos.

Ok, então vamos inserir o nosso objeto *TidICMPClient*, e alterar as seguintes propriedades:

Objeto TPanel [Standard]	
Name	NossoICMP
ReceiveTimeout	10000

Perceba que alteramos a propriedade *ReceiveTimeout*. Esta propriedade indica o tempo máximo em milissegundos para uma resposta do servidor, gerando um *timeout* quando ultrapassado este tempo.

Vamos codificar o botão *btPing*, inserindo o código que segue no evento *OnClick*:

```
var
  i: integer;
begin
  btPing.Enabled:=False;
  try
    NossoICMP.Host := edEndereco.Text;
    for i := 1 to 4 do
      begin
        NossoICMP.Ping;
        Application.ProcessMessages;
      end;
    finally
      btPing.Enabled := True;
    end;
  end;
```

Vamos analisar o código. Primeiro, estamos declarando uma variável do tipo *integer* para nos auxiliar no laço do *ping*.

```
var
  i: integer;
```

Em seguida desabilitamos o botão *btPing*, impedindo o usuário de clicar várias vezes e causar um efeito redundante.

```
btPing.Enabled:=False;
```

No próximo ponto, protegemos o código com *try* para que eventuais erros sejam ignorados em nossa aplicação.

```
try
```

Em seguida configuramos a propriedade *Host* do nosso objeto *NossoICMP* de acordo com a informação do usuário extraída do objeto *edEndereco*.

```
NossoICMP.Host := edEndereco.Text;
```

Finalmente iniciamos o laço, executando o comando *Ping* 4 vezes (padrão), e solicitando ao S.O. uma atualização da aplicação.

```
for i := 1 to 4 do
  begin
    NossoICMP.Ping;
    Application.ProcessMessages;
  end;
```

E para concluir esta parte do código, finalizamos o bloco protegido, habilitando novamente o botão *btPing*.

```
finally
  btPing.Enabled := True;
end;
```

Agora, iremos codificar o evento *OnReply* do objeto *NossoICMP*. Este evento é responsável pelas informações de resposta.

Coloque o código a seguir no evento *OnReply*:

```
var
  sTime:string;
begin
  if (AReplyStatus.MsRoundTripTime = 0) then
    sTime := '<1'
  else
    sTime := '=';

  mmResultado.Lines.Add(Format('%d bytes de %s: sequencia#=%d tempo-de-vida(ttl)=%d tempo=%s%d ms',
    [AReplyStatus.BytesReceived,
      AReplyStatus.FromIpAddress,
      AReplyStatus.SequenceId,
      AReplyStatus.TimeToLive,
      sTime,
      AReplyStatus.MsRoundTripTime]));
end;
```

Nossa Facunte, agora complicou! Que código maluco é esse? Amigos, à primeira vista, realmente é meio maluco, mas iremos desvendar rapidamente o mistério. Primeiro declaramos uma variável para nos auxiliar na composição da resposta:

```
var
  sTime:string;
```

Em seguida estamos verificando a propriedade *MsRoundTripTime* da constante *AReplyStatus*. Tudo bem, e de onde vem esta constante, e o que quer dizer *MsRoundTripTime*? Bem, a constante foi declarada no cabeçalho do evento *OnReply*, vejam:

```
procedure TForm1.NossoICMPReply(ASender: TComponent;
  const AReplyStatus: TReplyStatus);
```

Esta é uma declaração padrão para o evento *OnReply* do objeto *NossoICMP*. Com relação à propriedade *MsRoundTripTime*, ela representa o tempo em milissegundos do resultado de retorno do pacote. Caso o resultado seja 0, então atribuímos o valor “< 1” à nossa variável *sTime*, senão, atribuímos o valor “= “.

Prosseguindo com a nossa análise, agora vem o ponto forte da nossa aplicação: a linha de resultado.

```
mmResultado.Lines.Add(Format('%d bytes de %s: sequencia#=%d tempo-de-vida(ttl)=%d
tempo=%s%d ms',
  [AReplyStatus.BytesReceived,
    AReplyStatus.FromIpAddress,
    AReplyStatus.SequenceId,
    AReplyStatus.TimeToLive,
    sTime,
    AReplyStatus.MsRoundTripTime]));
```

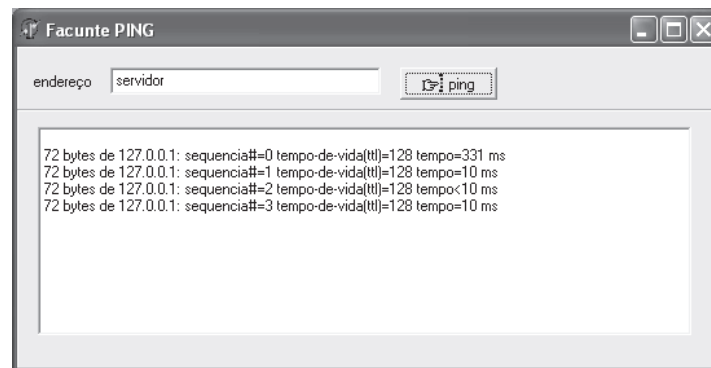
Percebam que estamos adicionando conteúdo ao nosso objeto *mmResultado*:

```
mmResultado.Lines.Add
```

Estamos nos aproveitando da função *Format* para facilitar a composição da linha de resultado. Para quem não conhece a função *Format*, sugiro uma breve visita no *Help* do *Delphi*, onde sempre encontramos ricas informações. Vejamos os significados das propriedades da constante *AReplyStatus*.

```
BytesReceived   = bytes recebidos
FromIpAddress   = endereço do Host
SequenceId      = número da sequência
TimeToLive      = Tempo de vida do pacote
```

Agora vamos executar o nosso projeto e ver o resultado. A *figura 9.8* ilustra o nosso projeto em execução.



**Figura 9.8 Projeto Ping em execução**

**Missão Cumprida**

Ao final deste tópico, espero ter esclarecido a todos a utilidade do *Ping*, bem como ter transmitido com clareza as funcionalidades do objeto *TidICMPClient*.

*Artigo originalmente publicado na Revista Clube Delph, nº 33*

**Listagem 9.2 unit un\_ping**

```
unit un_ping;

interface

uses
  SysUtils, Types, Classes, QGraphics, QControls, QForms, QDialogs,
  QStdCtrls, QExtCtrls, QButtons, IdBaseComponent, IdComponent, IdRawBase,
  IdRawClient, IdIcmpClient;

type
  TForm1 = class(TForm)
    pnEndereco: TPanel;
    PnResultado: TPanel;
    edEndereco: TEdit;
    btPing: TBitBtn;
    Label1: TLabel;
    mmResultado: TMemo;
    NossoICMP: TIidIcmpClient;
    procedure btPingClick(Sender: TObject);
    procedure NossoICMPReply(ASender: TComponent;
      const AReplyStatus: TReplyStatus);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.xfm}

procedure TForm1.btPingClick(Sender: TObject);
var
  i: integer;
begin
  btPing.Enabled:=False;
  try
    NossoICMP.Host := edEndereco.Text;
    for i := 1 to 4 do
      begin
        NossoICMP.Ping;
```



```

        Application.ProcessMessages;
    end;
finally
    btPing.Enabled := True;
end;

end;

procedure TForm1.NossoICMPReply(ASender: TComponent;
    const AReplyStatus: TReplyStatus);
var
    sTime:string;
begin
    if (AReplyStatus.MsRoundTripTime = 0) then
        sTime := '<1'
    else
        sTime := '=';

    mmResultado.Lines.Add(Format('%d bytes de %s: sequencia#=%d tempo-de-vida(ttl)=%d
tempo%s%d ms',
[AReplyStatus.BytesReceived,
AReplyStatus.FromIpAddress,
AReplyStatus.SequenceId,
AReplyStatus.TimeToLive,
sTime,
AReplyStatus.MsRoundTripTime]));
end;

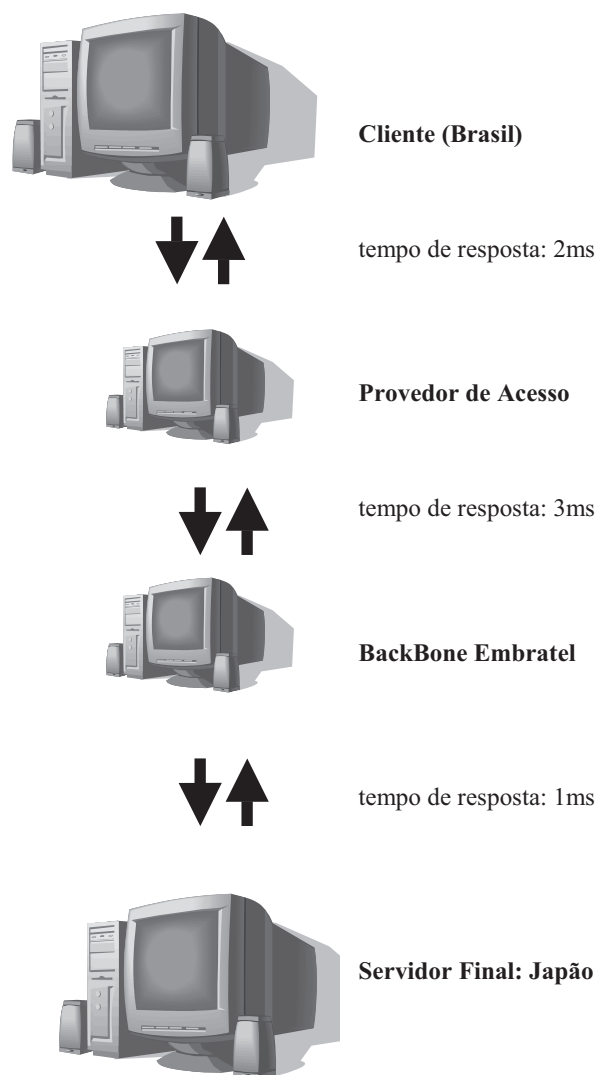
end.

```

## TraceRoute (Tracert)

Neste tópico vamos desenvolver uma nova versão para o *trace route*.

Muito semelhante ao nosso utilitário *ping*, mas com funcionalidades mais completas, apresentando todos os servidores por onde trafegam os pacotes de informações, entre cliente e servidor final. A *figura 9.9* ilustra uma simulação do *trace route*.



**Figura 9.9 Simulação do Trace Route**

Com o *trace route* poderemos descobrir falhas ou lentidão de resposta em todos os servidores envolvidos no processo de envio e recebimento de pacotes. Normalmente falamos que o site X está extremamente lento. Nem sempre é o próprio site, pode ser um dos servidores envolvidos (conhecido como *hope*) no processo.

Nosso utilitário é uma adaptação de um programa de demonstração que vem no pacote da INDY Components. Vamos iniciar o desenvolvimento do nosso utilitário.

Através das opções *File/New...CLX Application*, crie uma nova aplicação. Grave a *unit* com o nome *un\_tracert.pas* e o projeto como *tracert.dpr*. Insira os objetos que seguem alterando suas respectivas propriedades.

Objeto TPanel [Standard]	
Align	alTop
Name	pnTopo
Height	155

Com o foco no objeto *pnTopo*, insira os objetos que seguem, alterando suas respectivas propriedades:

Objeto TPanel [Standard]	
Align	alTop
Name	pnDados
Height	60

Ainda com o foco no *PnTopo*, insira um objeto do tipo *TMemo*, e altere as seguintes propriedades.

Objeto TListBox [Standard]	
Align	alClient
Name	lbLog

Agora com o foco no objeto *PnDados*, insira os objetos que seguem, alterando suas respectivas propriedades:

Objeto TLabel[Standard]	
Caption	Host Destino
Left	12
Top	8

Objeto TLabel[Standard]	
Caption	Máximo Hopes
Left	12
Top	40

Objeto TEdit [Standard]	
Left	110
Name	edTarget
Text	(deixar em branco)
Top	5
Width	280

Objeto TSpindEdit [Common Controls]	
Left	110
Name	seMaxHops
Top	32
Value	50
Width	65

Objeto TButton [Standard]	
Caption	Executa
Left	184
Name	btExecuta
Top	32
Width	75

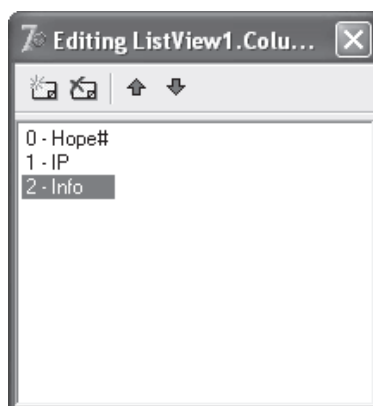
Objeto TButton [Standard]	
Caption	Cancela
Left	270
Name	btCancela
Top	32
Width	75

Agora com o foco no Formulário, insira outro componente do tipo *TPanel* alterando as seguintes propriedades.

Objeto TPanel [Standard]	
Align	AlClient
Name	PnInfo

Com o foco no objeto *pnInfo* insira um componente do tipo *TListView*, alterando as seguintes propriedades.

Objeto TListView [Common Controls]	
Align	alClient
Columns	insira três colunas como ilustra a <i>figura 9.10</i>
Name	LvTrace
ViewStyle	vsReport



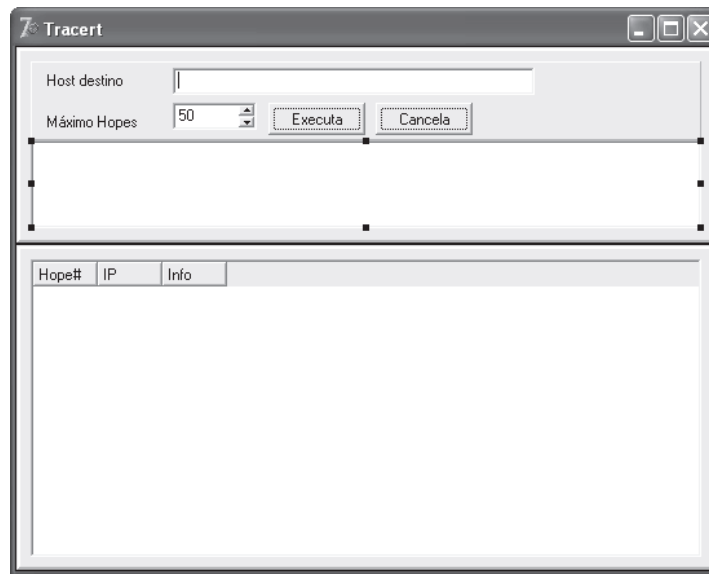
**Figura 9.10** Columns.items do listview1

Insira os objetos que seguem, alterando as respectivas propriedades.

Objeto idICMPClient [Indy Clients]	
Name	idICMPClient

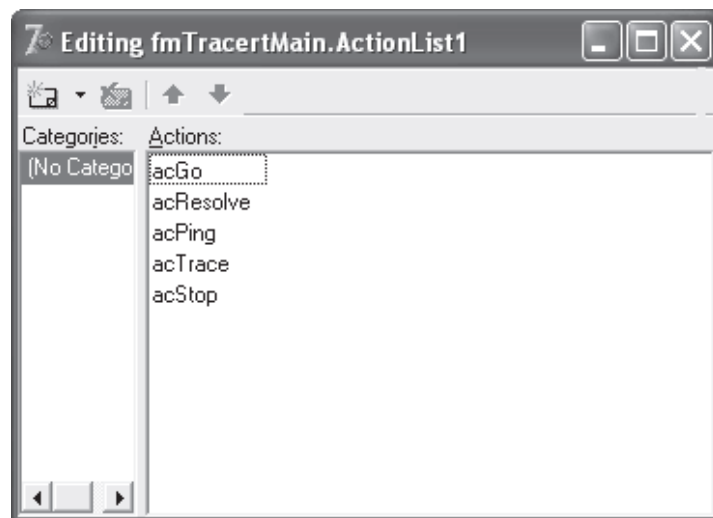
Objeto idAntiFreeze [Indy Misc]	
Name	idAntiFreeze

A *figura 9.11* ilustra nosso formulário.



**Figura 9.11** Formulário do projeto Tracert

Para facilitar a organização das nossas *procedures e ações*, vamos utilizar o componente *ActionList*. Insira um objeto do tipo *TActionList* e através do duplo-clique insira as seguintes ações (*figura 9.12*).



**Figura 9.12** Ações do projeto

Agora vamos codificar nosso projeto. Na seção **private** insira o código que segue.

```
bResolved: Boolean;
ResolvedHost: String;
Stopped: Boolean;
function PingHost(Host: string; TTL: Integer): boolean;
function FindItem(TTL: Integer; Add: boolean): TListItem;
```

Nesta seção, definiremos algumas variáveis e funções com o objetivo de auxiliar em alguns processos.

```
bResolved: Boolean;
ResolvedHost: String;
Stopped: Boolean;
```

bResolved                para atribuir *true* ou *false* na operação de *ping*  
 ResolvedHost           para atribuir o *host (hope)* em operação  
 Stopped                atribui o valor *true* para informar a paralisação das operações.

```
function PingHost(Host: string; TTL: Integer): boolean;
function FindItem(TTL: Integer; Add: boolean): TListItem;
```

Na cláusula **uses** insira as seguintes *Units*.

```
uses IdStack, IdException;
```

Agora vamos codificar as ações: No evento *OnExecute* da *Action acGO*, insira o código que segue:

```
try
  Stopped := false;
  acGo.Enabled := false;
  acStop.enabled := true;
  acResolve.execute;
  if bResolved and not stopped then
  begin
    acPing.execute;
    if not stopped then
      acTrace.Execute;
  end;
  acGo.Enabled := true;
  acStop.enabled := false;
finally
  { ok }
end; { try/finnaly }
```

Nesta rotina estamos atribuindo as definições iniciais do nosso projeto, e executando as rotinas de *ping* e *trace*. No evento *OnExecute* da *Action acResolve*, insira o código que segue:

```
bResolved := false;
lbLog.Items.Append(Format('resolvendo %s', [edTarget.text]));
try
  Application.ProcessMessages;
  ResolvedHost := gStack.WSGetHostByName(edTarget.text);
  bResolved := true;
  lbLog.Items.Append(format('%s resolvido %s', [edTarget.text, ResolvedHost]));
except
  on e: EIdSocketError do
    lbLog.Items.text := lbLog.Items.text + e.message;
  end;
```

Neste código estamos *resolvendo* um *host (hope)*, ou melhor, verificando informações e possibilidade de contato com o *host*. Repare que estamos utilizando a função *WSGetHostByName*, da *unit IdStack*, para mapear e *resolver* o *host*.

```
ResolvedHost := gStack.WSGetHostByName(edTarget.text);
```

Se a operação for bem sucedida, adicionamos no *lbLog* o resultado; caso contrário, a mensagem de erro.

#### **Operação bem sucedida**

```
lbLog.Items.Append(format('%s resolvido %s', [edTarget.text, ResolvedHost]));
```

**Erro.**

```
except
  on e: EIdSocketError do
    lbLog.Items.text := lbLog.Items.text + e.message;
```

No evento *OnExecute* da *Action acPing*, insira o seguinte código.

```
PingHost(ResolvedHost, seMaxHops.value);
Application.ProcessMessages;
```

Aqui estamos executando a chamada da função *PingHost* (será criada posteriormente), com o *host* atual, e o máximo de *hops*. No evento *OnExecute* da *Action acTrace*, insira o código que segue.

```
var
  TTL: Integer;
  Reached: boolean;
  aItem: TListItem;
begin
  TTL := 0;
  reached := false;
  lvTrace.Items.Clear;
  repeat
    inc(TTL);
    IdIcmpClient.Host := ResolvedHost;
    IdIcmpClient.TTL := TTL;
    IdIcmpClient.ReceiveTimeout := 5000;
    IdIcmpClient.Ping;
    aItem := FindItem(TTL, True);
    aItem.SubItems.Clear;
    case IdIcmpClient.ReplyStatus.ReplyStatusType of
      rsEcho:
        begin
          aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
          aItem.SubItems.Append(format('Reached in : %d ms',
[IdIcmpClient.ReplyStatus.MsRoundTripTime]));
          reached := true;
        end;
      rsError:
        begin
          aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
          aItem.SubItems.Append('Unknown error.');
```

```

        aItem.SubItems.Append(format('TTL=%d',
[IdIcmpClient.ReplyStatus.TimeToLive]));
        end;
    end; // case
    Application.ProcessMessages;
until reached or (TTL > seMaxHops.value) or Stopped;

```

Este código é um pouco mais complexo, mas vamos analisar passo a passo. Inicialmente, estamos definindo algumas variáveis auxiliares.

TTL                define o *hope* atual.  
 Reached          define se o *TTL* atual foi resolvido  
 aItem            objeto do tipo *TListItem* para ser adicionado em nosso *LvTrace*.

```

var
    TTL: Integer;
    Reached: boolean;
    aItem: TListItem;

```

Em seguida temos a atribuição inicial das variáveis.

```

TTL := 0;
reached := false;
lvTrace.Items.Clear;

```

O bloco que segue inicia um laço, até que uma das três situações (resolvido – reached = true, TTL > hopes, ou solicitação de cancelamento) seja verdadeira. Incrementamos o *TTL*, e atribuímos ao objeto *idICMPClient* algumas configurações, como *Host* e *TTL*. Executamos o método *ping* do *idICMPClient*, e em seguida preparamos o objeto *aItem* para receber a informação do resultado da operação.

```

repeat
    inc(TTL);
    IdIcmpClient.Host := ResolvedHost;
    IdIcmpClient.TTL := TTL;
    IdIcmpClient.ReceiveTimeout := 5000;
    IdIcmpClient.Ping;
    aItem := FindItem(TTL, True);
    aItem.SubItems.Clear;

```

Os próximos blocos tratam os diversos tipos de resultado do método *ping*. O bloco que segue trata o tipo de resposta (**rsEcho**) e adiciona um subitem ao objeto *aItem*.

```

case IdIcmpClient.ReplyStatus.ReplyStatusType of
    rsEcho:
    begin
        aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
        aItem.SubItems.Append(format('Reached in : %d ms',
[IdIcmpClient.ReplyStatus.MsRoundTripTime]));
        reached := true;
    end;

```

O bloco seguinte recebe um tipo de resposta de erro (**rsError**) e também adiciona um subitem ao objeto *aItem*.

```

rsError:
begin
    aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
    aItem.SubItems.Append('Unknown error. ');
end;

```



O próximo bloco trata o *TimeOut* excedido (**rsTimeOut**), definido como 5000 milsegundos.

```
rsTimeOut:
begin
  aItem.SubItems.Append('?.?.?.?');
  aItem.SubItems.Append('Timed out. ');
end;
```

Em seguida, temos o tratamento de uma operação *não resolvida* (**rsErrorUnreachable**).

```
rsErrorUnreachable:
begin
  aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
  aItem.SubItems.Append(format('Destination network unreachable',
[IdIcmpClient.ReplyStatus.MsRoundTripTime]));
  break;
end;
```

E por fim, temos o tratamento de erro (**rsErrorTTLExceeded**), onde é excedido o número máximo de tentativas.

```
rsErrorTTLExceeded:
begin
  aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
  aItem.SubItems.Append(format('TTL=%d',
[IdIcmpClient.ReplyStatus.TimeToLive]));
end;

end; // case
Application.ProcessMessages;
until reached or (TTL > seMaxHops.value) or Stopped;
```

É um código realmente complexo, mas estudando com calma e atenção, entenderemos cada método implementado.

Agora no evento *OnExecute* da *Action acStop*, insira o código que segue.

```
Stopped := true;
acStop.enabled := false;
```

Com isso estamos solicitando uma parada nas operações.

Agora vamos atribuir as *Actions acGO* e *acStop*, ao objeto *btExecuta* e *btCancela*, respectivamente. Para tanto, basta selecionar a *Action* desejada do objeto em questão. Para finalizar nosso projeto, devemos criar as rotinas que seguem.

```
function PingHost(Host: string; TTL: Integer): boolean;
function FindItem(TTL: Integer; Add: boolean): TListItem;
```

Primeiro vamos criar a rotina **PingHost**. Insira o código que segue em nossa *unit*.

```
function TForm1.PingHost(Host: string; TTL: Integer): Boolean;
begin
  result := false;
  IdIcmpClient.Host := Host;
  IdIcmpClient.TTL := TTL;
  IdIcmpClient.ReceiveTimeout := 5000;
  IdIcmpClient.Ping;
  case IdIcmpClient.ReplyStatus.ReplyStatusType of
    rsEcho:
      begin
```

```

        lbLog.Items.Append(format('resposta do host %s in %d milesegundos', [
IdIcmpClient.ReplyStatus.FromIpAddress,
IdIcmpClient.ReplyStatus.MsRoundTripTime]));
        result := true;
    end;
    rsError:
        lbLog.Items.Append('Erro desconhecido');
    rsTimeout:
        lbLog.Items.Append('Timed out. ');
    rsErrorUnreachable:
        lbLog.Items.Append(format('Host %s não resolvido',
[IdIcmpClient.ReplyStatus.FromIpAddress]));
    rsErrorTTLExceeded:
        lbLog.Items.Append(format('Hope %d %s: TTL expirou',
[IdIcmpClient.TTL,
IdIcmpClient.ReplyStatus.FromIpAddress]));
    end; // case
end;

```

Repare que é muito parecida com a *Action acTrace*, só que neste caso, estamos resumindo o resultado das operações em nosso objeto *lbLog*. Agora vamos criar a rotina **FindItem**. Insira o código que segue em nossa *unit*.

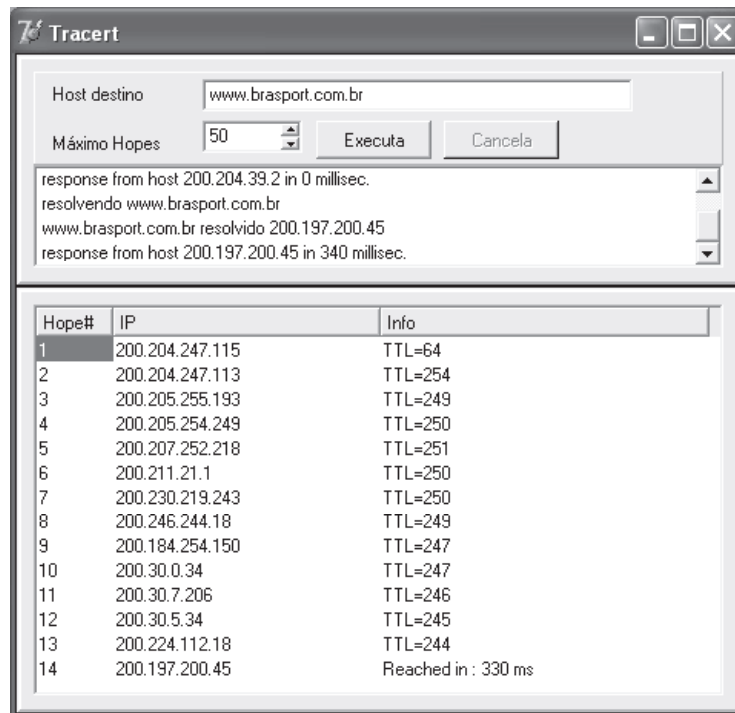
```

function Tform1.FindItem(TTL: Integer; Add: boolean): TListItem;
var
    i: Integer;
begin
    result := nil;
    // Find the TTL item
    if lvTrace.Items.Count < TTL Then
    begin
        for i := 0 to lvTrace.Items.Count - 1 do
        begin
            if StrToIntDef(lvTrace.Items[i].Caption, -1) = TTL then
            begin
                result := lvTrace.Items[i];
                Break;
            end;
        end;
    end;
    if not assigned( result ) then
    begin
        // Not found, add it
        result := lvTrace.Items.Add;
        result.Caption := IntToStr(TTL);
    end;
end;

```

O objetivo desta função é retornar informações adicionadas no objeto *lvTrace*, baseada na *TTL* atual, e utilizada pela *Action acTrace*.

Agora vamos gravar e compilar nossa aplicação. A *figura 9.13* ilustra nossa aplicação sendo executada.



**Figura 9.13 Tracert em ação**

### Listagem 9.3 un\_tracert.pas

```

unit un_tracert;

interface

uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls, QForms,
  QDialogs, QStdCtrls, QExtCtrls, QComCtrls, QActnList, IdComponent,
  IdRawBase, IdRawClient, IdTcmpClient, IdBaseComponent, IdAntiFreezeBase,
  IdAntiFreeze;

type
  TForm1 = class(TForm)
    pnTopo: TPanel;
    pnInfo: TPanel;
    pnDados: TPanel;
    Label1: TLabel;
    edTarget: TEdit;
    Label2: TLabel;
    seMaxHops: TSpinEdit;
    btExecuta: TButton;
    btCancela: TButton;
    lvTrace: TListView;
    ActionList1: TActionList;
    acGo: TAction;
    acResolve: TAction;
  end;

```

```

    acPing: TAction;
    acTrace: TAction;
    acStop: TAction;
    IdAntiFreeze1: TIdAntiFreeze;
    IdIcmpClient: TIdIcmpClient;
    lbLog: TListBox;
    procedure acGoExecute(Sender: TObject);
    procedure acResolveExecute(Sender: TObject);
    procedure acPingExecute(Sender: TObject);
    procedure acTraceExecute(Sender: TObject);
    procedure acStopExecute(Sender: TObject);
private
    { Private declarations }
    bResolved: Boolean;
    ResolvedHost: String;
    Stopped: Boolean;
    function PingHost(Host: string; TTL: Integer): boolean;
    function FindItem(TTL: Integer; Add: boolean): TListItem;

public
    { Public declarations }
end;

var
    Form1: TForm1;

implementation

uses idStack, IdException;
{$R *.xfm}

procedure TForm1.acGoExecute(Sender: TObject);
begin
    try
        Stopped := false;
        acGo.Enabled := false;
        acStop.enabled := true;
        acResolve.execute;
        if bResolved and not stopped then
            begin
                acPing.execute;
                if not stopped then
                    acTrace.Execute;
            end;
        acGo.Enabled := true;
        acStop.enabled := false;
    finally
        { ok }
    end; { try/finally }
end;

end;

procedure TForm1.acResolveExecute(Sender: TObject);
begin
    bResolved := false;
    lbLog.Items.Append(Format('resolvendo %s', [edTarget.text]));

```

```

try
    Application.ProcessMessages;
    ResolvedHost := gStack.WSGetHostByName(edTarget.text);
    bResolved := true;

lbLog.Items.Append(format('%s resolvido %s',[edTarget.text, ResolvedHost]));
except
    on e: EIdSocketError do
        lbLog.Items.text := lbLog.Items.text + e.message;
    end;

end;

procedure TForm1.acPingExecute(Sender: TObject);
begin
    PingHost(ResolvedHost, seMaxHops.value);
    Application.ProcessMessages;
end;

procedure TForm1.acTraceExecute(Sender: TObject);
var
    TTL: Integer;
    Reached: boolean;
    aItem: TListItem;
begin
    TTL := 0;
    reached := false;
    lvTrace.Items.Clear;
    repeat
        inc(TTL);
        IdIcmpClient.Host := ResolvedHost;
        IdIcmpClient.TTL := TTL;
        IdIcmpClient.ReceiveTimeout := 5000;
        IdIcmpClient.Ping;
        aItem := FindItem(TTL, True);
        aItem.SubItems.Clear;
        case IdIcmpClient.ReplyStatus.ReplyStatusType of
            rsEcho:
                begin
                    aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
                    aItem.SubItems.Append(format('Reached in : %d ms',
[IdIcmpClient.ReplyStatus.MsRoundTripTime]));
                    reached := true;
                end;
            rsError:
                begin
                    aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
                    aItem.SubItems.Append('Unknown error. ');
                end;
            rsTimeOut:
                begin
                    aItem.SubItems.Append('????.?');
                    aItem.SubItems.Append('Timed out. ');
                end;
            rsErrorUnreachable:
                begin
                    aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
                    aItem.SubItems.Append(format('Destination network unreachable',

```

```

[IdIcmpClient.ReplyStatus.MsRoundTripTime]));
    break;
end;
rsErrorTTLExceeded:
begin
    aItem.SubItems.Append(IdIcmpClient.ReplyStatus.FromIpAddress);
    aItem.SubItems.Append(format('TTL=%d',
[IdIcmpClient.ReplyStatus.TimeToLive]));
end;
end; // case
Application.ProcessMessages;
until reached or (TTL > seMaxHops.value) or Stopped;

end;

procedure TForm1.acStopExecute(Sender: TObject);
begin
    Stopped := true;
    acStop.enabled := false;
end;

function TForm1.PingHost(Host: string; TTL: Integer): Boolean;
begin
    result := false;
    IdIcmpClient.Host := Host;
    IdIcmpClient.TTL := TTL;
    IdIcmpClient.ReceiveTimeout := 5000;
    IdIcmpClient.Ping;
    case IdIcmpClient.ReplyStatus.ReplyStatusType of
        rsEcho:
        begin
            lbLog.Items.Append(format('response from host %s in %d millisec.',
            [
                IdIcmpClient.ReplyStatus.FromIpAddress,
                IdIcmpClient.ReplyStatus.MsRoundTripTime
            ]));

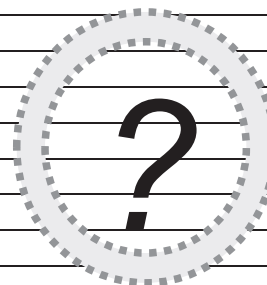
            result := true;
        end;
        rsError:
            lbLog.Items.Append('Unknown error. ');
        rsTimeOut:
            lbLog.Items.Append('Timed out. ');
        rsErrorUnreachable:
            lbLog.Items.Append(format('Host %s reports destination network unreachable.',
            [
                IdIcmpClient.ReplyStatus.FromIpAddress
            ]));
        rsErrorTTLExceeded:
            lbLog.Items.Append(format('Hope %d %s: TTL expired.',
            [
                IdIcmpClient.TTL,
                IdIcmpClient.ReplyStatus.FromIpAddress
            ]));
    end; // case
end;

```

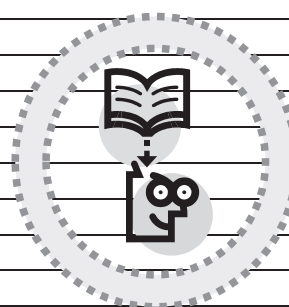
```
function TForm1.FindItem(TTL: Integer; Add: boolean): TListItem;
var
  i: Integer;
begin
  result := nil;
  // Find the TTL item
  if lvTrace.Items.Count < TTL Then
  begin
    for i := 0 to lvTrace.Items.Count - 1 do
    begin
      if StrToIntDef(lvTrace.Items[i].Caption, -1) = TTL then
      begin
        result := lvTrace.Items[i];
        Break;
      end;
    end;
  end;
  if not assigned( result ) then
  begin
    // Not found, add it
    result := lvTrace.Items.Add;
    result.Caption := IntToStr(TTL);
  end;
end;

end.
```

**Anotações de Dúvidas**



**Preciso Revisar**



**Anotações Gerais**

