Under Construction: Usenet News Reading

by Bob Swart

My first use of the internet was in the early 1980s, when it was still called Usenet, for sending email or reading newsgroup articles. Today, while the web may have become very important, the internet's email and newsgroup facilities are still vital to me. We've covered reading (SMTP) and sending (POP3) email messages in previous columns, so this time I'd like to focus on reading Usenet news articles using the NNTP protocol.

News Articles

Email messages and mailing lists contain messages from a sender to a *specified* receiver. A newsgroup is different, it's like a mailing list, in that you don't reach just one person but a potentially very large group. The messages (called *articles*) are posted and stored for a certain time. Any newsgroup reader can retrieve an article. Usually articles are either removed after a certain time or after a certain number of articles are in the newsgroup, the oldest articles being removed to make space for new ones.

When you subscribe to a mailing list you will receive all the *new* messages posted to that mailing list from that moment on. With a newsgroup, however, the default behaviour is that you get *all the articles currently available*, each time, which can be hundreds!

Newsgroups are maintained by news servers. Reading and Posting to the newsgroups is done using NNTP (Network News Transport Protocol) and in this article we'll use this protocol to communicate with a news server, obtain a list of newsgroups and read articles.

NNTP Server

I will assume that you have an NNTP newsgroup server available to connect to, somewhere. If not, or if you can't connect to the internet from your local machine, then you can always use the 30 days trial version of the DNEWS NNTP Server (to find it, search on DNEWS at www.altavista.com). You can use DNEWS to set up newsgroups on your intranet.

So, assuming an NNTP server is available, we'll now focus on writing an NNTP client in Delphi.

NNTP Client

NNTP comprises several simple commands that an NNTP client can send to an NNTP server. With `LIST` we get a list of available newsgroups, with `GROUP` followed by the name of a newsgroup we can 'join' a newsgroup, and with `ARTICLE` followed by an article number we can retrieve that particular article from the current newsgroup. We can also use `HEAD` and `STAT` to get the header and contents of an article (as opposed to using the `ARTICLE` command which gets the header and contents combined). Finally, we can perform a `QUIT` command to terminate the connection with the NNTP server.

We *can* use the available `TNMNNTP` component provided with Delphi 4, but the source code for these components is not included and it's always instructive to find out how things 'work' inside. For our custom `TBNNTP` VCL component, we can implement the same basic architecture as for the `TBPOP3` component (see Issue 36, August 1998). We create a `TClientSocket` component to connect to our news server (like news.shoresoft.com or an IP address such as 194.229.217.69) at port 119. Listing 1 shows a bare bones `TBNNTP` component that can connect to a news host and extract the list of available newsgroups.

The basic architecture is based on a `TClientSocket` component, which we create to connect to port 119 for the NNTP protocol. We assign an event handler for the `OnRead` event (when the NNTP server sends us data) and for the `OnWrite` event (when we need to send data to the NNTP server).

As soon as the connection is made, we wait for the welcome message from the NNTP server and then call the `OnWrite` event handler to send our first command (the value of the `Command` variable): the `LIST` of newsgroups. After we receive the list (in the `OnRead` event handler), we terminate the connection by sending a `QUIT` command in the `OnWrite` event handler.

Available Newsgroups

Listing 2 shows a console application to test the `TBNNTP` component by retrieving a list of newsgroups from my own newsgroup server at news.shoresoft.com. This server carries about 100 newsgroups, so retrieving the list doesn't take too long. However, most ISPs provide nearly all the newsgroups: many thousands! Downloading this list is not something to do *too* often... The output (Listing 3) shows a list of newsgroups.

The next step involves selecting a newsgroup and reading the articles in it. However, let's first take a look at a potential problem with our first attempt.

Socket.ReceiveText

The data sent to us (received in the `OnRead` event handler) can be obtained by looking at the `Socket.ReceiveText` property. This property holds the welcome message and list of newsgroups, for example, and will contain entire articles as well (as we'll see later). A potential problem is that `ReceiveText` may not be big enough to contain all the information in one transaction. The data is split into multiple parts and we need to

```
unit DrBobNEW;
{$DEFINE DEBUG}
interface
uses Classes, ScktComp;
type
  TBNNTP = class(TComponent)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  public
    procedure Connect;
  protected
    _Socket: TClientSocket;
    procedure SocketRead(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure SocketWrite(Sender: TObject;
      Socket: TCustomWinSocket);
  private
    fNewsServer: String;
  published
    property NewsServer: String
      read fNewsServer write fNewsServer;
  private
    Status: String;
  end;
implementation
uses SysUtils, Forms;
const
  CmdStart =  0;
  CmdList  =  1;
  CmdQuit  = 42;
  NNTP = 119;
  CRLF = #13#10;
var Command: Integer = CmdStart;
constructor TBNNTP.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  _Socket := TClientSocket.Create(Self);
  _Socket.Port := NNTP;
  _Socket.OnRead := SocketRead;
  _Socket.OnWrite := SocketWrite;
end {Create};
destructor TBNNTP.Destroy;
begin
  _Socket.OnRead := nil;
  _Socket.Free;
  _Socket := nil;
  inherited Destroy
end {Destroy};
procedure TBNNTP.Connect;
begin
  Command := CmdStart;
  _Socket.Active := False;
  _Socket.Host := fNewsServer;
  _Socket.Open;
  repeat
    Application.ProcessMessages
  until
    Command >= CmdQuit
end {Connect};
procedure TBNNTP.SocketRead(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Status := Socket.ReceiveText;
  while (Length(Status) > 0) and
    (Status[Length(Status)] in [#10,#13]) do
    Delete(Status,Length(Status),1);
 {$IFDEF DEBUG}
  writeln(Status);
 {$ENDIF}
  case Command of
    CmdStart : Command := CmdList;
    CmdList  : Command := CmdQuit
  end;
  SocketWrite(Sender, Socket)
end {SocketRead};
procedure TBNNTP.SocketWrite(Sender: TObject;
  Socket: TCustomWinSocket);
var Send: String;
begin
  Send := '';
  case Command of
    CmdList : Send := 'LIST';
    CmdQuit : Send := 'QUIT'
  end;
 {$IFDEF DEBUG}
  if Send <> '' then writeln(Command,'> ',Send);
 {$ENDIF}
  Socket.SendText(Send + CRLF)
end {SocketWrite};
end.
```

➤ *Listing 1: First attempt at unit DrBobNEW.*

listen to multiple `OnRead` events until we get all the data. Fortunately, a terminator has been defined as a single dot on an otherwise empty line.

If you used the example program in Listing 2 to get a list of newsgroups from your ISP you may find that the program doesn't run correctly: it only displays the first batch of newsgroups and then performs a `QUIT` without waiting for the other newsgroups to arrive.

Sometimes, `Socket.ReceiveText` doesn't contain the complete text we want to receive, and we should not stop receiving text until the combined `Socket.ReceiveTexts` do signal the end, with the single dot on an empty line: we'll call it the End-Of-Data (`EOD`) marker.

So, if `Socket.ReceiveText` does not contain the `EOD` as the last part we should continue with the current `Command`, but without sending the `Command` to the NNTP server. Instead, we should be sending an empty string to the NNTP server, to

```
program News;
{$APPTYPE CONSOLE}
uses DrBobNEW;
begin
  with TBNNTP.Create(nil) do
  try
    NewsServer := 'news.shoresoft.com'; { hosting my DrBob.* newsgroups }
    Connect { and retrieve newsgroup list }
  finally
    Free
  end
end.
```

➤ *Above: Listing 2*

➤ *Below: Listing 3*

```
200 194.229.217.69 DNEWS Version 4.6r, SO, posting OK
1> LIST
215 list of newsgroups follows
drbob 2 2 y
drbob.website 2 2 y
drbob.internet 2 2 y
drbob.internet.book 2 2 y
drbob.internet.tools 2 2 y
drbob.wizards 2 2 y
drbob.wizards.headconv 2 2 y
drbob.startrek 2 2 y
.
42> QUIT
```

indicate that we received the text and are now ready to receive more.

The trick is to determine whether or not the `EOD` does occur as the last part of the received test. The following was my first attempt (I use `-4` because I'm not sure if the final CR/LF has been stripped):

```
EOD := Pos(CRLF+'.',Socket.ReceiveText)
  >= (Length(Socket.ReceiveText)-4);
```

This works in almost all cases, but will fail when a dot occurs on an otherwise empty line somewhere within the article itself. `Pos` will return the first position and will hence fail to set `EOD` to `True`.

A better solution is the following snippet of code, where we copy the last part of `Socket.ReceiveText` and see if the `EOD` occurs in that last part:

```
EOD := Pos(CRLF+'.',
  Copy(Socket.ReceiveText,
  Length(Socket.ReceiveText)-4,
  5)) > 0;
```

Finally, to keep the current `Command` variable unchanged but make sure no `Command` is sent from the `OnWrite` event handler, we can use a semaphore. Since the problem will usually occur when retrieving articles from newsgroups, I decided to use the `ArtNr` variable (the number of the article to be retrieved) as the semaphore. If the `ArtNr` is positive, then we need to perform the `Command` (in this case, `ARTICLE` followed by the `ArtNr`, but if `ArtNr` is negative we're still busy retrieving the current article, so we should send an empty command instead, and just wait until one of the following `Socket.ReceiveTexts` contains the `EOD`. The complete implementation is in Listing 4.

BobNews

BobNews is an application to test our new component: see Figure 1. Listing 5 shows the event handling code needed.

Using BobNews to connect to news.shoresoft.com (and note that I've filtered all non-DrBob specific newsgroups from the list of available newsgroups here), we can double click on a newsgroup to retrieve all the available articles, then click in the grid with the headers to view the individual articles themselves.

Of course, this is just a quick and dirty newsgroup reader with quite a number of limitations. First of all, it retrieves every article from a selected newsgroup (and like I said before, this can potentially mean hundreds of articles). It would be better to keep track of the articles read already (in previous sessions), and only download the new articles. This would also mean storing already retrieved articles on your local machine (where you can delete them by hand if you want to).

The same is true for the list of available newsgroups to choose from, which you don't want to collect every time you connect with this newsreader to your NNTP server (unless you use the newsreader in a fast intranet environment, or you have a local ISP who makes very few newsgroups available).

And of course at this time we have no capability to reply to articles or post new articles in the newsgroups (having spent two years with read-only access to newsgroups, I know how frustrating this can be).

➤ *Below and facing page: Listing 4, final DrBobNEW.*

```
unit DrBobNEW;
{$DEFINE DEBUG}
interface
uses Classes, {$IFDEF DEBUG}StdCtrls,{$ENDIF} ScktComp;
const MaxGroups = 256;
type
  TBNNTP = class(TComponent)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  public
   {$IFDEF DEBUG}
    StatusMemo: TMemo; { pointer to Form's Memo }
   {$ENDIF}
    procedure Connect;
    procedure JoinNewsGroup(const NewsGroup: String);
    procedure ReadArticle(ArticleNr: Integer);
    procedure Disconnect;
  protected
    _Socket: TClientSocket;
    procedure SocketRead(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure SocketWrite(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure Wait;
  private
    fNewsServer: String;
  published
    property NewsServer: String
      read fNewsServer write fNewsServer;
  private // newgroups
    fNumGroups: Integer;
    fNewsGroups: Array[0..MaxGroups-1] of String;
    function GetNewsGroup(Index: Integer): String;
  public
    property NewsGroups: Integer read fNumGroups;
    property NewsGroup[Index: Integer]: String
      read GetNewsGroup;
  private // articles
    fFirstArticle,fLastArticle: Integer;
    fArticles: Array of String;
    function GetArticle(Index: Integer): String;
  public
    property FirstArticle: Integer read fFirstArticle;
    property LastArticle: Integer read fLastArticle;
    property Article[Index: Integer]: String
      read GetArticle;
  private // internal
    WinSocket: TCustomWinSocket;
    Command: Integer;
    ArtNr: Integer;
    Status: String; { also NewsgroupName }
   {$IFDEF DEBUG}
    Indent: Integer;
   {$ENDIF}
  end;
procedure Register;
```

```
implementation
uses SysUtils, Forms;
const
  CmdStart =   0;
  CmdList  =   1; { list newsgroups }
  CmdJoin  =   2; { join newsgroup }
  CmdMess  =   3; { read article # }
  CmdDone  =  42; { signals ready }
  CmdQuit  = 666;
  NNTP = 119;
  CRLF = #13#10;
 {$IFDEF DEBUG}
  function Space(X: Integer): String;
  begin
    Result := '';
    while X > 0 do begin
      Result := Result + ' ';
      Dec(X)
    end
  end {Space};
 {$ENDIF}
constructor TBNNTP.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  _Socket := TClientSocket.Create(Self);
  _Socket.Port := NNTP;
  _Socket.OnRead := SocketRead;
  _Socket.OnWrite := SocketWrite;
 {$IFDEF DEBUG}
  Indent := 0;
  StatusMemo := nil;
 {$ENDIF}
  WinSocket := nil
end {Create};
destructor TBNNTP.Destroy;
begin
  _Socket.OnRead := nil;
  _Socket.OnWrite := nil;
  //if Assigned(WinSocket) and (Command <> CmdQuit) then
  //  WinSocket.SendText('QUIT'+ CRLF);
  WinSocket := nil;
  _Socket.Free;
  _Socket := nil;
 {$IFDEF DEBUG}
  StatusMemo := nil;
 {$ENDIF}
  inherited Destroy
end {Destroy};
function TBNNTP.GetNewsGroup(Index: Integer): String;
begin
  if Index < MaxGroups then
    Result := fNewsGroups[Index]
  else
    Result := ''
end {GetNewsGroup};
function TBNNTP.GetArticle(Index: Integer): String;
```

```
begin
  if (Index >= fFirstArticle) and
    ((Index-fFirstArticle) < Length(fArticles)) then
    Result := fArticles[Index-fFirstArticle]
  else
    Result := ''
end {GetArticle};
procedure TBNNTP.SocketRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
  i,j: Integer;
  EOD: Boolean; { end-of-data }
begin
 {$IFDEF DEBUG}
  if Assigned(StatusMemo) then
    StatusMemo.Lines.Add(Space(Indent)+'SocketRead');
 {$ENDIF}
  WinSocket := Socket; { save copy to talk back }
  Status := Socket.ReceiveText;
  while (Length(Status) > 0) and
    (Status[Length(Status)] in [#10,#13]) do
    Delete(Status,Length(Status),1);
  EOD := Pos(CRLF+'.',Copy(Status,Length(Status)-4,5)) > 0;
 {$IFDEF DEBUG}
  if Assigned(StatusMemo) then begin
    if Command <> CmdMess then
      StatusMemo.Lines.Add(Space(Indent)+Status)
    else
      StatusMemo.Lines.Add(Space(Indent)+Copy(
        Status,1,Pos(#13,Status)-1));
    StatusMemo.Update; { force repaint }
  end else if IsConsole then
    writeln(Status);
 {$ENDIF}
  case Command of
    CmdStart :
      begin
        Command := CmdList; { next: get newsgroup list }
        ArtNr := 0
      end;
    CmdList :
      begin
        fNumGroups := -1;
        while Length(Status) > 1 do begin
          Inc(fNumGroups);
          i := Pos(#13,Status);
          j := Pos(#10,Status);
          if (i = 0) and (j = 0) then
            i := Length(Status)
          else if j > i then
            i := j;
          j := 1;
          while (j < i) and (Status[j] > #32) do
            Inc(j);
          if fNumGroups > 0 then begin
            fNewsGroups[fNumGroups-1] :=Copy(Status,1,j-1);
            if fNewsGroups[fNumGroups-1] = '' then
              Dec(fNumGroups)
          end;
          Delete(Status,1,i);
          while (Length(Status) > 0) and
            (Status[1] in [#10,#13]) do
            Delete(Status,1,1)
        end;
        if (Status = '.') or EOD then
          Command := CmdDone
        else
          ArtNr := -1 { continue: need more data... }
      end;
    CmdJoin :
      begin
        i := Pos(' ',Status);
        Delete(Status,1,i); { status code }
        i := Pos(' ',Status);
        Delete(Status,1,i); { number of articles }
        i := Pos(' ',Status);
        try // first article
          fFirstArticle := StrToInt(Copy(Status,1,i-1))
        except
          fFirstArticle := 1
        end;
        Delete(Status,1,i);
        i := Pos(' ',Status);
        try // last article
          fLastArticle := StrToInt(Copy(Status,1,i-1))
        except
          fLastArticle := 1
        end;
        fArticles := nil;
        if fLastArticle >= fFirstArticle then
          SetLength(fArticles, fLastArticle-
            fFirstArticle+1); // allocate memory
       {$IFDEF DEBUG}
        if Assigned(StatusMemo) then
          StatusMemo.Lines.Add(Space(Indent) +
            IntToStr(fFirstArticle)+' to '+
            IntToStr(fLastArticle))
        else
          if IsConsole then
            writeln(fFirstArticle,' to ',fLastArticle);
       {$ENDIF}
        Command := CmdDone
      end;
    CmdMess:
      begin
        if ArtNr < 0 then  { remaining part of article }
          fArticles[-ArtNr-fFirstArticle] :=
            fArticles[-ArtNr-fFirstArticle] + Status
        else begin
          i := Pos(#13,Status);
          if i > 0 then begin
            Delete(Status,1,i);
            while (Length(Status) > 0) and
              (Status[1] in [#10,#13]) do
              Delete(Status,1,1)
          end;
          fArticles[ArtNr-fFirstArticle] := Status
        end;
        if EOD then
          Command := CmdDone
        else
          ArtNr := -abs(ArtNr) { get next part of article }
      end;
    CmdQuit: Command := CmdDone
  end;
  if Command <> CmdDone then
    SocketWrite(Sender, Socket)
end {SocketRead};
procedure TBNNTP.SocketWrite(Sender: TObject;
  Socket: TCustomWinSocket);
var Send: String;
begin
  Send := '';
  case Command of
    CmdList : if ArtNr >= 0 then Send := 'LIST';
    CmdJoin : Send := 'GROUP ' + Status;
    CmdMess : if ArtNr > 0 then
                Send := 'ARTICLE ' + IntToStr(ArtNr);
    CmdQuit : Send := 'QUIT'
  end;
 {$IFDEF DEBUG}
  if Assigned(StatusMemo) then
    StatusMemo.Lines.Add(Space(Indent)+'> '+Send)
  else
    if IsConsole then writeln('> '+Send);
 {$ENDIF}
  Socket.SendText(Send + CRLF)
end {SocketWrite};

procedure TBNNTP.Wait;
begin
{$IFDEF DEBUG}
  Inc(Indent);
  if Assigned(StatusMemo) then
    StatusMemo.Lines.Add(Space(Indent)+'Waiting...')
  else if IsConsole then
    writeln('Waiting...');
  Inc(Indent);
 {$ENDIF}
  repeat
    Application.ProcessMessages
  until Command = CmdDone;
{$IFDEF DEBUG}
  Dec(Indent);
  if Assigned(StatusMemo) then
    StatusMemo.Lines.Add(Space(Indent)+'Done.')
  else
    if IsConsole then writeln('Done.');
  Dec(Indent);
 {$ENDIF}
end {Wait};
procedure TBNNTP.Connect;
begin
  Command := CmdStart;
  _Socket.Active := False;
  _Socket.Host := fNewsServer;
  _Socket.Open;
  Wait
end {Connect};
procedure TBNNTP.Disconnect;
begin
  Command := CmdQuit;
  SocketWrite(Self,WinSocket);
  Wait
end {Connect};
procedure TBNNTP.JoinNewsGroup(const NewsGroup: String);
begin
  Status := NewsGroup;
  Command := CmdJoin;
  SocketWrite(Self,WinSocket);
  Wait
end {JoinNewsGroup};
procedure TBNNTP.ReadArticle(ArticleNr: Integer);
begin
  ArtNr := ArticleNr;
  Command := CmdMess;
  SocketWrite(Self,WinSocket);
  Wait
end {ReadArticle};
procedure Register;
begin
  RegisterComponents('Dr.Bob',[TBNNTP])
end;
end.
```

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, ComCtrls, Buttons,
  Grids, DrBobNEW;
type
  TFormNews = class(TForm)
    ToolBar: TPanel;
    StatusBar: TStatusBar;
    MemoStatus: TMemo;
    ListBoxNewsGroups: TListBox;
    Splitter: TSplitter;
    MemoArticle: TMemo;
    BtnConnect: TSpeedButton;
    BtnDisconnect: TSpeedButton;
    BtnJoin: TSpeedButton;
    BNNTP: TBNNTP;
    StringGridArticles: TStringGrid;
    procedure BtnConnectClick(Sender: TObject);
    procedure BtnDisconnectClick(Sender: TObject);
    procedure BtnJoinClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure StringGridArticlesClick(Sender: TObject);
  end;
var FormNews: TFormNews;
implementation
{$R *.DFM}
procedure TFormNews.FormCreate(Sender: TObject);
begin
  BNNTP.NewsServer := 'news.shoresoft.com';
  with StringGridArticles do begin
    ColWidths[0] := 42;
    ColWidths[1] := 156;
    ColWidths[2] := 200;
    ColWidths[3] := 200;
    Cells[0,0] := 'Nr.';
    Cells[1,0] := 'From:';
    Cells[2,0] := 'Subject:';
    Cells[3,0] := 'Date:'
  end
end;
procedure TFormNews.BtnConnectClick(Sender: TObject);
var i: Integer;
begin
  BtnConnect.Enabled := False;
  BNNTP.Connect;
  ListBoxNewsGroups.Items.Clear;
  for i:=0 to Pred(BNNTP.NewsGroups) do
    ListBoxNewsGroups.Items.Add(BNNTP.NewsGroup[i]);
  BtnDisconnect.Enabled := True;
  BtnJoin.Enabled := ListBoxNewsGroups.Items.Count > 0;
end;
procedure TFormNews.BtnDisconnectClick(Sender: TObject);
var i: Integer;
begin
  ListBoxNewsGroups.Items.Clear;
  for i:=1 to Pred(StringGridArticles.RowCount) do begin
    StringGridArticles.Cells[0,i] := '';
    StringGridArticles.Cells[1,i] := '';
    StringGridArticles.Cells[2,i] := '';
    StringGridArticles.Cells[3,i] := '';
  end;
  MemoArticle.Clear;
  BNNTP.Disconnect;
  BtnConnect.Enabled := True;
  BtnDisconnect.Enabled := False;
  BtnJoin.Enabled := False;
end;
procedure TFormNews.BtnJoinClick(Sender: TObject);
var
  i,j: Integer;
  Article: String;
begin
  if ListBoxNewsGroups.ItemIndex >= 0 then
    BNNTP.JoinNewsGroup(ListBoxNewsGroups.Items[
      ListBoxNewsGroups.ItemIndex]);
  for i:=1 to Pred(StringGridArticles.RowCount) do begin
    StringGridArticles.Cells[0,i] := '';
    StringGridArticles.Cells[1,i] := '';
    StringGridArticles.Cells[2,i] := '';
    StringGridArticles.Cells[3,i] := '';
  end;
  for i:=BNNTP.FirstArticle to BNNTP.LastArticle do begin
    BNNTP.ReadArticle(i);
    StringGridArticles.Cells[0,i-BNNTP.FirstArticle+1] :=
      IntToStr(i);
    Article := BNNTP.Article[i];
    j := Pos('From: ',Article);
    Delete(Article,1,j+5);
    j := Pos(#13,Article);
    StringGridArticles.Cells[1,i-BNNTP.FirstArticle+1] :=
      Copy(Article,1,j-1);
    Article := BNNTP.Article[i];
    j := Pos('Subject: ',Article);
    Delete(Article,1,j+8);
    j := Pos(#13,Article);
    StringGridArticles.Cells[2,i-BNNTP.FirstArticle+1] :=
      Copy(Article,1,j-1);
    Article := BNNTP.Article[i];
    j := Pos('Date: ',Article);
    Delete(Article,1,j+5);
    j := Pos(#13,Article);
    StringGridArticles.Cells[3,i-BNNTP.FirstArticle+1] :=
      Copy(Article,1,j-1);
  end;
  StringGridArticles.Row := 1;
  StringGridArticlesClick(Sender)
end;
procedure TFormNews.StringGridArticlesClick(Sender:
  TObject);
begin
  MemoArticle.Text := BNNTP.Article[StringGridArticles.Row+
    BNNTP.FirstArticle-1]
end;
end.
```

➤ *Listing 5: The key parts of the BobNews newsgroup reader.*

Next Time

Next month we'll focus on how to post articles to newsgroups. Who knows, we may end up with our very own newsgroup reader and writer, all written in Delphi (and from the ground up too!). If you want to send me suggestions or feedback, you're very welcome to do so, in the DrBob.internet.tools newsgroups at news.shoresoft.com/DrBob.internet.tools (of course, the BobNews program won't allow you to post articles just yet, but after next month it will).

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is a technical consultant and webmaster using Delphi, JBuilder and C++Builder for Bolesian (www.bolesian.com), and freelance technical author.

➤ *Figure 1: BobNews in action.*