Under Construction: Usenet News Reading, 2

by Bob Swart

Last month we covered the basics of NNTP (Network News Transfer Protocol), including a TBNNTP component that could connect to a news server, retrieve a list of newsgroups, subscribe to a newsgroup and retrieve articles from that newsgroup.

This time, we'll add the abilities to post messages to newsgroups, interpret the response status code and manage information about both the newsgroups we're subscribing to and the messages we've read already (so we don't have to retrieve the whole list of newsgroups and articles every time we connect to our news server).

NNTP Commands

Last time, we started our TBNNTP component and implemented the LIST, GROUP, ARTICLE (combined HEAD and STAT) and QUIT commands.

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 article from the current newsgroup. We can also use HEAD and STAT to get the header and contents of an article (rather than 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.

This time, we need to add some more commands, namely HELP and POST (for both replying to and posting new articles), and we'll implement response status code checking as well.

Help And Status

The HELP command can be quite useful, as I found that not all NNTP servers support all the commands that we implement in the TBNNTP component. The NNTP server will answer the HELP command with a summary of commands that are supported by this implementation of the server. The help itself is returned as text, terminated by the EOD character (a single period on an otherwise empty line).

To implement this, we need to send the HELP command in the SocketWrite event handler, and receive the summary of commands in the SocketRead event handler (see the full listing of TBNNTP in Listing 3 for details).

Some news servers don't support the POST command that we need as well. Of course, we can parse the output from the HELP command to see if POST is actually supported by the server, but there's an easier way to determine this using *response status codes*. With each command that we give to the news server, it responds with an acknowledgement message, starting with a response or status code. This is also the case for the welcome message (the first message we receive when we connect to the news server). In fact, the welcome message indicates whether or not the POST command is available to us; all we need to do is look at the welcome status code that accompanies this welcome message. Welcome code 200 means posting is allowed, while welcome code 201 means no posting is allowed (ie read-only access to the news server).

It's a small change to add a Boolean property ReadOnly to the TBNNTP component, that gets its value from the status code in the welcome message. This property will indicate, at runtime, whether we have read-only access to the server, or whether we can actually post new articles (see Listing 1).

The welcome message (and in fact each response status code) will be received by the SocketRead event and can be parsed in the first few lines of that routine, as shown in Listing 2.

For now, we just use the ResponseCode to determine the value of the fReadOnly property after we received the welcome message. We'll shortly start to use the response status code at another time as well, namely when posting articles.

Posting Articles

The POST command can be used to send a new article to the current newsgroup. After the POST command, the NNTP server can return code 340 (ready) or code 440 (permission denied). Response code 340 indicates that the article to be posted should now be sent, while code 440 indicates that posting is prohibited (in which case we also should have received a welcome code 201).

If posting is permitted, the article should be presented in the format specified by RFC850, which means it should start with a header containing fields for NEWSGROUPS, FROM (TO is not needed), SUBJECT and DATE, followed by an empty line, followed by the body of the article, terminated by a single period on an otherwise empty line. The tricky part is not listing the

➤ *Listing 1*

```
type
  TBNNTP = class(TComponent)
  ...
  private
    fReadOnly: Boolean;
  published
    property ReadOnly: Boolean read fReadOnly;
  end {TBNNTP};
```

current newsgroup name (that should be easy), but to include a valid date that isn't considered 'too long in the past' by the news server. The format that DNews accepts can be returned by a call to FormdatDateTime:

```
EditDate.Text := FormatDateTime(
  'dd mmm yyyy  hh:mm:ss', Now);
```

After the article's header and body have been completely sent by the client to the server, a final return code of 240 (meaning all is OK) or 441 (meaning the posting failed) will be sent, so we know the message is posted correctly and will be available to all news servers and readers all over the world shortly. Note that we get a code 441 (fail) if the news server considers the article too old, for example (which is why we need to include a correct date and time string), as shown in Listing 3.

Full source code of the final TBNNTP component can be found on disk with this issue. Don't hesitate to contact me by email at bob@bolesian.nl if you have any questions, comments or feedback on this article!

```
procedure TBNNTP.SocketRead(Sender: TObject; Socket: TCustomWinSocket);
var ResponseCode: Integer;
begin
  Status := Socket.ReceiveText;
  ResponseCode := StrToInt(Copy(Status,1,Pos('#32',Status)-1));
  ...
  case Command of
    CmdStart :
      begin
        fReadOnly := not (ResponseCode = 200);
        ...
      end;
    ...
  end;
  ...
end {SocketRead};
```

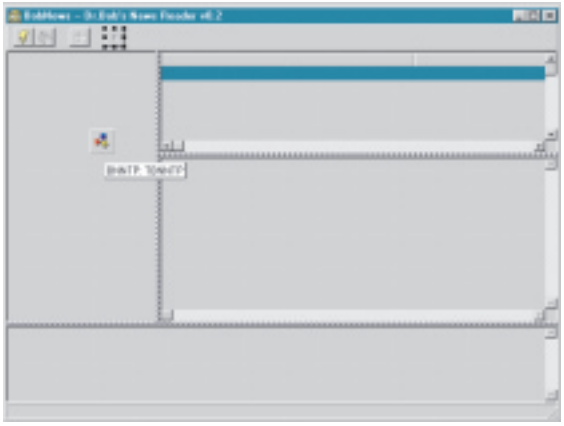
➤ *Listing 2*

BobNews v0.2

Version 0.2 of BobNews (we started version 0.1 last month) now also has the ability to post new messages or reply to existing messages in a newsgroup. Note that the Post (reply) button is disabled if the ReadOnly property is of the TBNNTP component is True and if we've joined a newsgroup: see Figure 1. If we click on the Post (reply) button, the code shown in Listing 4 will be executed.

This will pop up a new modal form that we can use to enter the subject and content of the article. Note that the Sender and Date are read-only fields (the value of the Date editbox is determined when this form is created): see Figure 2.

Further updates and enhancements to this application should include the addition of my TBSMTP



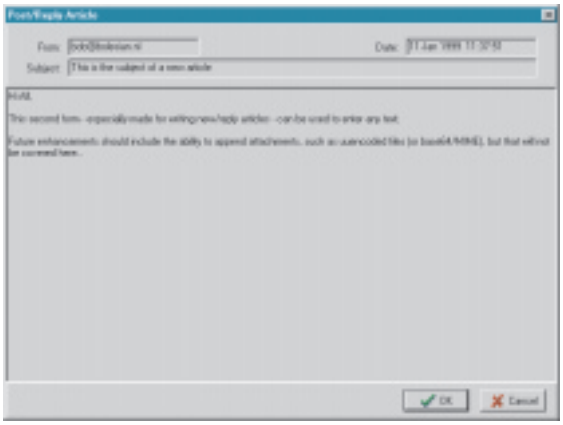
➤ *Figure 1*

➤ *Listing 3*

```
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 }
   {$ENDIF}
    procedure Connect;
    procedure JoinNewsGroup(const NewsGroup: String);
    procedure ReadArticle(ArticleNr: Integer);
    procedure PostArticle(const NewArticle: String);
    procedure Disconnect;
  protected
    _Socket: TClientSocket;
    procedure SocketRead(Sender: TObject; Socket:
      TCustomWinSocket);
    procedure SocketWrite(Sender: TObject; Socket:
      TCustomWinSocket);
    procedure Wait;
  private
    fNewsServer: String;
  published
    property NewsServer: String read fNewsServer
      write fNewsServer;
  private // readonly or posting articles?
    fReadOnly: Boolean;
    fNewArticle: String;
  published
    property ReadOnly: Boolean read fReadOnly;
  private // newgroups
    fNumGroups: Integer;
    fNewsGroups: Array[0..MaxGroups-1] of String;
    function GetNewsGroup(Index: Integer): String;
  public
    property NewsGroups: Integer read fNumGroups;
    property NewsGroup[Index: Integer]: String
      read GetNewsGroup;
  private // articles
    fFirstArticle,fLastArticle: Integer;
    fArticles: Array of String;
    function GetArticle(Index: Integer): String;
  public
    property FirstArticle: Integer read fFirstArticle;
    property LastArticle: Integer read fLastArticle;
    property Article[Index: Integer]: String
      read GetArticle;
  private // internal
    WinSocket: TCustomWinSocket;
    Command: Integer;
    ArtNr: Integer;
    Status: String; { also NewsgroupName }
   {$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 # }
  CmdPost  =   4; { post article }
  CmdPost2 =   5; { post content }
  CmdHelp  =   7; { summary commands }
  CmdDone  =  42; { signals ready }
  CmdQuit  = 666;
  NNTP = 119;
  CRLF = #13#10;
constructor TBNNTP.Create(AOwner: TComponent);
begin
                { ** LISTING CONTINUES ON FACING PAGE... **}
```

```
  inherited Create(AOwner);
  _Socket := TClientSocket.Create(Self);
  _Socket.Port := NNTP;
  _Socket.OnRead := SocketRead;
  _Socket.OnWrite := SocketWrite;
  fReadOnly := True;
 {$IFDEF DEBUG}
  Indent := 0;
  StatusMemo := nil;
 {$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};
{ … NOT ALL CODE SHOWN: SEE DISK FOR FULL LISTING }
procedure TBNNTP.SocketRead(Sender: TObject;
   Socket: TCustomWinSocket);
var
  i,j: Integer;
  EOD: Boolean; { end-of-data }
  ResponseCode: Integer;
begin
 {$IFDEF DEBUG}
  if Assigned(StatusMemo) then
    StatusMemo.Lines.Add(Space(Indent)+'SocketRead');
 {$ENDIF}
  WinSocket := Socket; { talk back? }
  Status := Socket.ReceiveText;
  ResponseCode := StrToInt(Copy(Status,1,
    Pos(' ',Status)-1));
  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;
  // Pos(CRLF+'.',Status) > (Length(Status)-4);
 {$IFDEF DEBUG}
  if Assigned(StatusMemo) then begin
    if Command <> CmdMess then
      StatusMemo.Lines.Add(Space(Indent)+Status)
    else StatusMemo.Lines.Add(Space(Indent)+Copy(
      Status,1,Pos(  #13,Status)-1));
    StatusMemo.Update; { force repaint }
  end else if IsConsole then
    writeln(Status);
 {$ENDIF}
  case Command of
    CmdStart:
      begin
        fReadOnly := not (ResponseCode = 200);
        Command := CmdList; { get newsgroup list }
        ArtNr := 0
      end;
    CmdPost:
      begin
        if ResponseCode = 340 then
          Command := CmdPost2
        else
          Command := CmdDone
      end;
    CmdPost2:
      begin
        Command := CmdDone
      end;
    CmdHelp:
      begin
        { receive summary of commands }
        Command := CmdDone;
      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 }
        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
            fFirstArticle := StrToInt(Copy(Status,1,i-1))
          except
            fFirstArticle := 1
          end;
          Delete(Status,1,i); { last article }
          i := Pos(' ',Status);
          try
            fLastArticle := StrToInt(Copy(Status,1,i-1))
          except
            fLastArticle := 1
          end;
          fArticles := nil;
          if fLastArticle >= fFirstArticle then
            // allocate
            SetLength(fArticles,fLastArticle-fFirstArticle+1);
         {$IFDEF DEBUG}
          if Assigned(StatusMemo) then
            StatusMemo.Lines.Add(Space(Indent)+IntToStr(
              fFirstArticle)+ ' to '+IntToStr(fLastArticle))
          else if IsConsole then
            writeln(fFirstArticle,' to ',fLastArticle);
         {$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) { negative }
        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);
    CmdPost  : Send := 'POST';
    CmdPost2 : Send := fNewArticle;
    CmdHelp  : Send := 'HELP';
    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.PostArticle(const NewArticle: String);
begin
  if not fReadOnly then begin
    fNewArticle := NewArticle;
    Command := CmdPost;
    SocketWrite(Self,WinSocket);
    Wait
  end
end {PostArticle};
procedure Register;
begin
  RegisterComponents('Dr.Bob',[TBNNTP])
end;
end.
```

and TBPOP3 components, to allow us to send and read email messages as well: often you need to reply by email to an article in a newsgroup. This, as well as including uuencoded attachments, is left as an excercise for the reader. Note that the full source code for TBSMTP and TBPOP3 can be found in Issues 35 and 36, so it shouldn't be too hard to extend BobNews *[The companion disk contents are also on our website at www.itecuk.com for download. Ed]*.



➤ *Figure 2*

➤ *Listing 4*

```
procedure TFormNews.BtnPostReplyClick(Sender: TObject);
begin
  with TFormPostReply.Create(Self) do
  try
    if ShowModal = mrOk then begin
      MemoArticle.Lines.Insert(0,'');
      MemoArticle.Lines.Insert(0,'Newsgroups: '+StatusBar.SimpleText);
      MemoArticle.Lines.Insert(0,'Subject: '+EditSubject.Text);
      MemoArticle.Lines.Insert(0,'Date: '+EditDate.Text);
      MemoArticle.Lines.Insert(0,'From: '+EditFrom.Text);
      MemoArticle.Lines.Add('.');
      BNNTP.PostArticle(MemoArticle.Text)
    end
  finally
    Free
  end
end;
```

Next Time

Distributed applications is one of the new areas where Delphi offers us support with MIDAS and CORBA, among others. Next time, I'd like to 'step down' and explore CORBA from the ground up. We'll see how we can make our own CORBA objects and communicate with them (even from other machines, as long as they are connected in some kind of network that supports CORBA). Should be quite interesting, and very useful as well, so *stay tuned*...

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is a technical consultant and webmaster using Delphi, JBuilder and C++Builder for Bolesian and freelance technical author. When he's not coding or writing, Bob likes to watch video tapes of Star Trek Voyager and Deep Space Nine with his 4.5-year-old son Erik Mark Pascal and his 2-year-old daughter Natasha Louise Delphine.