Under Construction: Automated Email

by Bob Swart

Next month, we'll be exploring the world of Artificial Intelligence again, specifically the concept of Intelligent Agents (or Robots) for the internet. But before we start on that path, this month our subject is the Simple Mail Transfer Protocol (SMTP).

So why bother about next month already? Well, because in this column we'll develop a useful technique that can be deployed by the upcoming web Robot for next month, and I'm sure it's something every webmaster would like to use at times as well. I'm talking about the ability to automatically send email messages (for example from the web server), without any human intervention whatsoever.

Protocols

For someone who doesn't know (or care) much about the underlying email message sending protocols, as long as the message gets there, I discovered a number of new acronyms when looking into this. The first one, SMTP, stands for Simple Mail Transfer Protocol, and is a very old protocol, originally designed for UNIX machines, but operational on other servers as well (such as the web/mail servers I connect to).

Apart from SMTP, I also learned about MAPI (Microsoft's Mail API) or the later IMAP (anyone know what that stands for?) and found some folk using Lotus Notes. But all I was looking for was a protocol for sending automatic email messages from the web/mail servers I connect to and it seemed that SMTP was implemented on all of them.

NetManage ActiveX TSMTP

When I started with this column, I first checked out the SIMPMAIL.DPR project, which is provided with Delphi 2.01, 3 and 4 in the directory DEMOS/INTERNET/SIMPMAIL. This small sample project illustrates the use of the NetManage SMTP and POP3 ActiveX controls. My friend John Kaster (who now works for Inprise in the US) already warned me about the POP3 control, which he ended up rewriting in ObjectPascal from scratch, and after a few tests with the SMTP control, I had to agree with him. Using an ActiveX control to manage a relatively low-level protocol as SMTP or POP3 does not seem to be the best solution to me.

Native SMTP

Right after I moved the `TSMTP` ActiveX control aside, I downloaded and read the latest RFC (Request For Comment) on the SMTP protocol standard, being RFC 821 by Jonathan B Postel, dated August 1982. SMTP uses a TCP/IP connection, that we can implement using Delphi's `TClientSocket` component. SMTP uses port 25, a value you must set in the `Port` property after you've created the `ClientSocket`.

But before we start to write our own native SMTP component, based on the `TClientSocket` control, let's first decide how to name this new component and how exactly to include the socket.

Naming Conventions

Apart from the fact that the NetManage ActiveX control is a bit bulky, it has another even worse drawback: it ships with Delphi and is called `TSMTP`. This means that any third-party vendor that also sells a `TSMTP` control will probably get into name collisions (meaning that the component user must remove the internet solutions pack package in order to make the name `TSMTP` available, or modify the source code for the third-party component changing the name to `TWhatEverSMTP` instead).

In order to prevent these problems (which are even worse if you consider the fact that not only component names but also unit names need to be unique among packages), a new website called the Delphi Prefix Registry has been set up by Steven Healey at

```
http://developers.href.com/
  registry/dpr.htm
```

➤ *Listing 1*

```
unit TBOBSMTP;
interface
uses Classes, ScktComp;
type
  TBSMTP = class(TComponent)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  public
    procedure SendMail;
  protected
    _Socket: TClientSocket;
    Step: Integer;
    procedure SocketRead(Sender: TObject; Socket: TCustomWinSocket);
    procedure SocketWrite(Sender: TObject; Socket: TCustomWinSocket);
  private
    fMailServer: String;
    fMessageTo: String; { only one "To" at a time }
    fMessageFrom: String;
    fMessageSubject: String;
    fMessageText: TStringList; { assign to TMemo.Lines, for example }
    procedure SetMessageText(NewText: TStringList);
  published
    property MailServer: String read fMailServer write fMailServer;
    property MessageTo: String read fMessageTo write fMessageTo;
    property MessageFrom: String read fMessageFrom write fMessageFrom;
    property MessageSubject: String read fMessageSubject write fMessageSubject;
    property MessageText: TStringList read fMessageText write SetMessageText;
  end;
```

which gives an overview of this problem and lists a set of vendor prefixes that others should hopefully take into consideration. My personal prefix for the components I develop for *The Delphi Magazine* and my website is B, so this month we'll be developing a TBSMTP component!

Since name clashes can occur at the unit name level as well, I decided to use the unit name TBOBSMTP for this component.

Inheritance And Delegation

There are basically two ways we can implement our native Delphi TBSMTP component. We can use inheritance and take the TClientSocket component and derive a TBSMTP component from (since we need the socket in the first place), or we can use delegation and derive TBSMTP from a basic non-visual TComponent, hiding the socket as property in the implementation details. I decided to use the latter approach, mainly because we can now create and initialise the ClientSocket inside the TBSMTP constructor, and hide the implementation details (like port number 25) from the user of the component who only wants to send an email message.

TBSMTP

The new component, named TBSMTP, starts with a constructor and destructor, and a public procedure SendMail to send the email message. We also need a few published properties to specify the SMTP mail server (like smtp.gatebolsan.nl), the recipient, the name of the sender (us, usually), the subject line, and finally a set of strings to hold the actual contents of the email message. If you ignore the protected parts, then this is exactly what the class definition in Listing 1 offers.

Note that I have only a single MessageTo property, which provides us with one email address to send the message to. There's also no MessageCc property (for carbon copy functionality), but if you read on you'll find that these features won't be hard to implement afterwards.

```
constructor TBSMTP.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  _Socket := TClientSocket.Create(Self);
  _Socket.Port := 25; // according to RFC 821
  _Socket.OnRead := SocketRead;
  _Socket.OnWrite := SocketWrite;
  Step := 0;
  fMessageText := TStringList.Create
end {Create};
destructor TBSMTP.Destroy;
begin
  fMessageText.Free;
  _Socket.Free;
  _Socket := nil;
  inherited Destroy
end {Destroy};
```

➤ *Listing 2*

```
procedure TBSMTP.SocketRead(Sender: TObject; Socket: TCustomWinSocket);
var Status: String;
begin
  Status := Socket.ReceiveText;
{$IFDEF DEBUG}
  writeln(log,Status);
{$ENDIF}
  SocketWrite(Sender, Socket)
end {SocketRead};
```

➤ *Listing 3*

Now that we've seen the design-time interface, let's look at the implementation details, and specifically the protected section where the internal _Socket field of type TClientSocket is defined. This is the placeholder for the socket control that we need to send the email message behind the scenes. Note that I use an underscore prefix in the name of this internal field, to remind me that this field is only used to implement the SMTP protocol, and is not visible to the outside world. We also need two event handler (callback) methods assigned to this _Socket control, for the OnRead and OnWrite events.

The TBSMTP constructor is the place where we need to create the TClientSocket field, assign the OnRead and OnWrite event handlers and set the Step field to 0 (Step is used to keep track of the 'steps' that have been done while actually sending the email message, as we'll see in just a minute). The last statement in the constructor allocates a TString class for the message text. This means we can add individual strings to this list, or simply assign a TMemo.Lines property to it, for example.

Of course, what gets constructed must be destructed and Listing 2 shows the constructor and the accompanying destructor.

Socket Event Handlers

The _Socket field, of type TClientSocket, contains two event handler properties that we must fill with event handler methods. The OnRead event is fired whenever the _Socket control gets data from the SMTP server. This usually is a simple confirmation, but can also be an error code, so this is a good place to add error detection and handling code.

After we actually read the information the server sent us, using ReceiveText, we can call the SocketWrite method to send something back to the SMTP server, as in Listing 3. Actually this makes it sound as if we're responding to the SMTP server, while in fact the server is responding to our data: we just wait until the server responds before sending more data.

The SocketWrite method is called in response to a SocketRead event, this is the place where we actually set up the communication with the SMTP server and, using six steps, send the message across. The member field Step is used to keep track of the current step. Based on the information from the RFC 821, the six steps that I identified are shown in Figure 1 (each step must be followed by an answer from the SMTP server before the next step can be sent).

1. Send HELO to initiate a conversation with the SMTP server *(note that this is not a typo, it is indeed the four letter 'HELO' not 'HELLO')*.
2. Send MAIL FROM: plus sender email address.
3. Send RCPT TO: plus recipient email address.
4. Send DATA and prepare to send the entire message.
5. Send the message, followed by a . (period) on a single line.
6. Send QUIT to terminate the conversation.

➤ *Figure 1: Stages in sending email using SMTP.*

In Listing 5, I added another step between steps 3 and 4, by adding a RCPT TO:bob@bolesian.nl, thereby making sure that no matter who's the recipient, I always get a copy at my mailbox at bob@bolesian.nl as well. This also illustrates how you can send one message to multiple people all at once.

We're not done yet. There's one detail of the SMTP protocol that I don't quite understand myself. Although we specified the sender (MAIL FROM) and recipient (RCPT TO) in steps 2 and 3, we still have to repeat the From: and To: fields (and optionally Cc: as well) as the first few lines of the message body. In fact, that's also the place to put the Subject: line.

After these extra headers, we need to send a blank line, followed by the actual body of the message (stored in the MessageText property), followed by the dot on a new line, as in Listing 4.

The actual act of sending an email message is done by first assigning the MailServer name (either the full name, like smtp.gatebolsan.nl, or the resolved IP address), and then activating the internal _Socket field, which can be done with either a call to Open, or by setting the Active property to True.

The last remaining problem is to keep waiting until we've finished the last step (magic number 7) and then quit. This waiting is done in a repeat...until loop that calls Application.ProcessMessages. This means that we must include the Forms unit to be able to call Application.ProcessMessages, which might be considered a bit overkill (the reader's exercise for today is implement a waiting loop without having to use the TApplication class!).

RobotBob Version 0.1

Using the TBSMTP component and setting its properties at either design-time or runtime, we are now able to write an application that can send an email message without any further end-user interference. A small sample application that can run from a local machine having a TCP/IP connection to an SMTP server (as in the Bolesian intranet, or when you're connected to your ISP with a socket connection available), is the program AutoMail from Listing 6. Although it won't do much at this time, it can actually be considered the first draft version of RobotBob.

Now all we need to add to RobotBob (next time) is some intelligent task that produces some output, which is then in turn used as body text for the automatic email message.

Debug Logfile

You must have noticed a number of {$IFDEF DEBUG}...{$ENDIF} statements in the code I've shown you. These can be used to write to an output logfile (see the full source code on the disk for more details) to trace the connection and SMTP process. For example, sending a simple test message using the program from Listing 5 resulted in the DEBUG logfile shown in Listing 7 (the text which is shown here in red for clarity was sent by RobotBob, while the normal text is the response from the SMTP server on Bolesian's intranet WinNT web server).

Note that we didn't get an answer after the QUIT command, since RobotBob automatically terminates after sending the QUIT command itself.

➤ *Listing 4*

```
procedure TBSMTP.SocketWrite(Sender: TObject; Socket: TCustomWinSocket);
const CRLF = #13#10;
var
  i: Integer;
  Send: String;
begin
  Inc(Step);
  case Step of
    1: Send := 'HELO ' + fMessageFrom;
    2: Send := 'MAIL FROM:' + fMessageFrom;
    3: Send := 'RCPT TO:' + fMessageTo;
    4: Send := 'RCPT TO:bob@bolesian.nl'; { copy to myself }
    5: Send := 'DATA';
    6: begin
        {$IFDEF DEBUG}
         writeln(log,'From: ' + fMessageFrom);
        {$ENDIF}
         Socket.SendText('From: ' + fMessageFrom + CRLF);
        {$IFDEF DEBUG}
         writeln(log,'To: ' + fMessageTo);
        {$ENDIF}
         Socket.SendText('To: ' + fMessageTo + CRLF);
        {$IFDEF DEBUG}
         writeln(log,'Cc: bob@bolesian.nl');
        {$ENDIF}
         Socket.SendText('Cc: bob@bolesian.nl' + CRLF);
        {$IFDEF DEBUG}
         writeln(log,'Subject: ' + fMessageSubject + CRLF + CRLF);
        {$ENDIF}
         Socket.SendText('Subject: ' + fMessageSubject + CRLF + CRLF);
         for i:=0 to Pred(fMessageText.Count) do begin
           Send := fMessageText[i];
          {$IFDEF DEBUG}
           writeln(log,Send);
          {$ENDIF}
           Socket.SendText(Send + CRLF)
         end;
         Send := '.'
       end;
    7: Send := 'QUIT'
  end;
{$IFDEF DEBUG}
  writeln(log,Send);
{$ENDIF}
  Socket.SendText(Send + CRLF)
end {SocketWrite};
```

```
program AutoMail;
{$I-}
uses
  TBOBSMTP;
var
  f: Text;
  Str: String;
begin
  with TBSMTP.Create(nil) do
  try
    MailServer := 'smtp.gatebolsan.nl';
    MessageFrom := 'RobotBob';
    MessageTo := 'drbob@bolesian.nl';
    MessageSubject := 'Automatic E-mail using SMTP';
    MessageText.Add('Hi Guys,');
    MessageText.Add('');
    System.Assign(f,'c:\usr\bob\message.txt');  // the e-mail message
    Reset(f);
    if IOResult = 0 then while not eof(f) do begin
      readln(f,Str);
      MessageText.Add(Str)
    end;
    System.Close(f);
    MessageText.Add('');
    MessageText.Add('Groetjes,');
    MessageText.Add('           Bob Swart (aka Dr.Bob - www.drbob42.com)');
    SendMail;
  finally
    Free
  end
end.
```

➤ *Listing 6*

```
1998/05/22 5:38:09 PM
HELO RobotBob
220 nt01.bolesian.nl WindowsNT SMTP Server v3.02.10/1.aeue ready at Fri, 22 May
1998 7:11:42 +0100
MAIL FROM:RobotBob
250 nt01.bolesian.nl RobotBob
250 Ok.
RCPT TO:hubert@bolesian.nl
250 Ok.
RCPT TO:bob@bolesian.nl
250 Ok.
DATA
354 Start mail input, end with <CRLF>.<CRLF>.
From: RobotBob
To: hubert@bolesian.nl
Cc: bob@bolesian.nl
Subject: Automatic E-mail using SMTP

Hi Guys,

This code is useless! You have no honour!
A TRUE Klingon warrior does not comment his code!
You question the worthiness of my Code?! I should kill you where you stand!
Perhaps today IS a good day to Die! I say we ship it!

Groetjes,
           Bob Swart (aka Dr.Bob - www.drbob42.com)
.
250 Requested mail action Ok.
QUIT
```

➤ *Listing 7*

Authentication

As you can see from the logfile, I can actually pretend to be just about anyone when sending out email messages using the SMTP protocol.

And I fear that this is what a lot of the professional email spammers are doing as well (and why it is generally useless to reply to these messages anyway). But, gentle reader, you won't be using my code to start sending Spam email, will you, please?

Next Time...

Sending an email message from an SMTP client machine by using a TCP/IP connection to an SMTP server is one thing. But will this technique still work without changes if we just upload the automatic email application to a web server (which is an SMTP server as well), or do we need to modify the way we communicate with the server?

What are the practical uses of our TBSMTP component, apart from

```
procedure TBSMTP.SendMail;
begin
  Step := 0;
  _Socket.Active := False;
  _Socket.Host := fMailServer;
  _Socket.Open;
  repeat
    Application.ProcessMessages
  until Step > 7
end {SendMail};
```

➤ *Listing 5*

Spam, which is considered to be an abuse of the internet, of course?

These questions and more will be answered next time when we finally witness the birth of the real *RobotBob!*

Bob Swart (aka Dr.Bob visit www.drbob42.com) is a professional knowledge engineer technical consultant for Bolesian (www.bolesian.com), freelance technical author for *The Delphi Magazine*, co-author of *The Revolutionary Guide to Delphi 2* and the website knowledge base *Delphi Internet Solutions*. In his spare time, Bob likes to watch video tapes of Star Trek Voyager and Deep Space Nine with his 4-year old son Erik Mark Pascal and his 1.5-year old daughter Natasha Louise Delphine.