Sending mails using Lazarus

Michaël Van Canneyt

August 31, 2012

Abstract
There are various components to handle all kind of TCP/IP protocols in Lazarus.
An example is sending E-Mail with the SMTP protocol. The Synapse package offers a particularly easy way to send e-mails in Lazarus.

1 Introduction

Sometimes, one needs to send an e-mail from a program or application: a website handling on-line subscriptions for an event will send a confirmation email, or a windows service which needs to communicate an error condition to the system administrator, for example when the hard-disk threatens to run out of disk space.

There are different class libraries that can be used to send e-mails: The Indy and Synapse libraries can be used to send email. Both class libraries can be used in Delphi as well as in Lazarus.

Synapse is a collection of classes that handle various TCP/IP tasks. It is not included in the default Lazarus distribution, but can be downloaded separately from the following URL:

http://synapse.ararat.cz/

The archive contains a laz_synapse.lpk package file, which can be opened with the Lazarus package manager. It is sufficient to compile the package once. There is no need to install the package in the IDE, because the package does not contain components that can be installed on the component palette: all classes must be instantiated and used in code.

For those that prefer it, there is a separate package available, called visualsynapse. It contains a series of components that can be installed on the component palette. However, this package is not officially supported by the creator of synapse.

Once compiled, the package laz_synapse can be chosen in the list of dependencies for a project.

2 E-mail through SMTP

E-mail programs use the SMTP protocol (Simple Mail Transfer Protocol) to send e-mails. This means that they deliver the mail message to a SMTP server (a MTA or Mail Transfer Agent) which will handle further delivery of the message to its final destination. Usually, a mail server listens on TCP/IP port 25.

The SMTP protocol is simple, and sending a mail is a simple matter. In synapse, the smtpsend unit contains a function that allows to send a mail:
function SendTo(const MailFrom, MailTo, Subject, SMTPHost: string; const MailData: TStrings): Boolean;

The 5 arguments speak for themselves:

**MailFrom**  The sender’s e-mail address.

**MailTo**  The e-mail addresses of the recipients, separated by commas. This list can contain names, so things like

Michael Van Canneyt <michael@freepascal.org>

are allowed.

**Subject**  The subject of the mail. This will be added to the mail headers.

**SMTPHost**  The IP or DNS address of the SMTP server that will handle delivery of the mail. Usually, this is the outgoing mail server of the internet provider.

**MailData**  The actual e-mail message.

The SendTo function creates an instance of the TSMTPSend class, and fills in the necessary properties, after which the MailTo method is used to send the mail to all provided e-mail addresses. If everything went without errors, the function will return True. If an error occurred, it will return False.

Some mailservers require a username/password combination before they allow sending mails. The function SendToEx operates identical to SendTo, but allows to specify a username/password in addition to the other arguments:

function SendToEx(const MailFrom, MailTo, Subject, SMTPHost: string; const MailData: TStrings; const Username, Password: string): Boolean;

These functions are all that is needed to construct a simple e-mail program.

This program needs some configuration variables to contain the sender e-mail, SMTP server and username/password. These can be declared as simple form variables:

```
FSender,
FSMTPHost,
FSMTPUser,
FSMTPasswd : String;
```

A small configuration dialog allows the user to set their values, and they are saved in an .ini file and loaded when the program is started. The code for this is very simple, and will not be explained in detail.

To compose an e-mail, 2 TEdit controls for the recipients and e-mail subject are needed: (we’ll call them ETo and ESubject). Additionally, a TMemo (MMail) control is needed to enter the message text. Adding a toolbar with a ‘Send’ button completes the program. The BSend button is coupled to an action ASend, which has the following OnExecute event handler:
procedure TForm.ASendExecute(Sender: TObject);
begin
  if not CheckSettings then
    exit;
  dosendmail(ETO.Text,ESubject.Text,MMail.Lines);
end;

The CheckSettings function checks whether the form variables contain a valid sender address and SMTP server. If not, a message is shown. If everything is configured correctly, the function DoSendMail is called. This function simply calls one of the SendTo or SendToEx functions:

procedure TForm.DoSendMail(Const ATo,ASubject : String;
  Content : TStrings);

Var
  B : Boolean;
begin
  if (FSMTPUser<>'') then
    B:=SendToEx(FSender,ATo,ASubject,FSMTPHost,Content,
    FSMTPUser,FSMTPPasswd)
  else
    B:=SendTo(FSender,ATo,ASubject,FSMTPHost,Content);
  if not B then
    ShowMessage('Could not send the message!')
  else
    ShowMessage('Message sent successfully.');
end;

When everything is completed, a message reporting success of failure, is shown.

That’s all there is to sending a mail in Object Pascal.

3 Attachments

Often, it is necessary to send a HTML mail message, or an attachment with a log file, PDF or even an image. The SMTP mail protocol only supports sending basic text. To be able to send a binary attachment such as an image, MIME encoding is necessary: MIME is an acronym for Multipurpose Internet Mail Extensions. MIME encoding allows to attach an arbitrary number of files to a mail message. Every e-mail client understands this encoding, and is able to recreate the attachments from the MIME encoded mail text.

A MIME message consists of several parts. There are different kinds of parts:

Text Contains regular text, in any character encoding.
Binary Contains binary data.
Message Contains another mail message.
Multipart Contains one or more other MIME parts.

To create an e-mail with an attachment, a multipart part must be created. To this multipart part, a text part is added (for the e-mail text), and one or more binary parts. This multipart is then sent as the mail text.
Synapse offers a class to facilitate the composing of a MIME email text: T MimeMess. This class collects some parts (one or more instances of T MimePart), and then composes the MIME content of the message. This class also takes care of constructing the necessary e-mail headers, which will tell the e-mail client to treat the e-mail as a MIME encoded message.

A part is added using the AddPart function:

```pascal
function AddPart(const PartParent: T MimePart): T MimePart;
```

The T MimePart class has a multitude of properties that describe its content. Normally, all these properties must be given correct values, to enable the T MimeMess instance to correctly compose the MIME message. Luckily, there are some methods that fill out the necessary properties automatically. Adding a multipart Part to the mime message can be done using the following function:

```pascal
function AddPartMultipart(const MultipartType: String;
```

Adding a text part can be done with:

```pascal
function AddPartText(const Value: TStrings;
function AddPartTextEx(const Value: TStrings;
const PartParent: T MimePart;
PartCharset: T MimeChar; Raw: Boolean;
PartEncoding: T MimeEncoding): T MimePart;
function AddPartTextFromFile(const FileName: String;
```

And adding binary data can be done with:

```pascal
function AddPartBinary(const Stream: TStream;
const FileName: string;
function AddPartBinaryFromFile(const FileName: string;
```

All functions have as the last argument PartParent. This is the `multipart` part to which the new part must be added: only multipart parts can contain other parts. Attempting to add a binary part to another binary part will result in an error.

There are some auxiliary functions to add HTML data and complete mail messages to a MIME message.

Once all parts have been added to the message, the message must be encoded. This can be done using the EncodeMessage function.

The T MimeMess class can be demonstrated easily: The application keeps a list of file-names to attach to a message in a stringlist: (FAttachments).

The OnExecute handler of the ASend action now becomes more elaborate:

```pascal
procedure TMainForm.ASendExecute(Sender: TObject);
begin
  If not CheckSettings then
    Exit;
```
if (FAttachments.Count>0) then
  DoSendMailAndAttachments(ETo.Text,ESubject.Text,MMail.Lines)
else
  DoSendMail(ETo.Text,ESubject.Text,MMail.Lines);
end;

If there are attachments, the function DoSendMailAndAttachments is called. This function does the actual work:

procedure TMainForm.DoSendMailAndAttachments(
  Const ATo,ASubject : String;
  Content : TStrings);
Var
  Mime : TxEEMess;
  P : TxEEPart;
  I : Integer;
  B : Boolean;
begin
  Mime:=TxEEMess.Create;
  try
    // Set some headers
    Mime.Header.ToList.Text:=ATo;
    Mime.Header.Subject:=ASubject;
    Mime.Header.From:=FSender;
    // Create a MultiPart part
    P:=Mime.AddPartMultipart('mixed',Nil);
    // Add as first part the mail text
    Mime.AddPartText(Content,P);
    // Add all attachments:
    For I:=0 to FAttachments.Count-1 do
      Mime.AddPartBinaryFromFile(FAttachments[I],P);
  finally
    Mime.Free;
  end;
end;

After EncodeMessage has been called, the Lines property of the TxEEMess class contains the encoded message text, including mail headers. Because it contains the headers,
this message must be sent using SendToRaw. The SendToRaw function is defined as follows:

```pascal
function SendToRaw(const MailFrom, MailTo, SMTPHost: string;
const MailData: TStrings;
const Username, Password: string): Boolean;
```

The difference between the SendToRaw function and the SendTo or SendToEx functions is that it does not add e-mail headers to the message: the necessary e-mail headers must be present in MailData. The T MIMEMess class creates the necessary headers when the EncodeMessage method is called.

The user can manage the attachments using a listbox (LBAttachments). A pop-up menu in the listbox shows 2 menu items: one to add an attachment, the other to remove the selected attachment. The same actions can be executed using buttons in the toolbar.

The finished program is shown (with the configuration dialog) in figure 1 on page 6.

4 Send status

Even in these times of fast internet connections, sending large attachments can still take a while. During that time, the program freezes, and the user has no indication of how far the sending of an email has progressed. To remedy this, the TSMTPSend class has an event that is called at regular intervals, and in which the current status is reported. The event handler has the following signature:

```pascal
THookSocketStatus = procedure(Sender: TObject;
   Reason: THookSocketReason;
```

Figure 1: The e-mail program
const Value: string) of object;

Sender is the TCP/IP socket for which the status is being reported. Reason is a status indicator, which can have one of the following values for a client socket:

**HR_ResolvingBegin**  The server hostname is converted to an IP address.

**HR_ResolvingEnd**  The server hostname has been converted to an IP address.

**HR_SocketCreate**  The TCP/IP socket for communication with the server has been created.

**HR_SocketClose**  The TCP/IP socket for communication with the server has been closed.

**HR_Connect**  Connection has been made with the SMTP server.

**HR_WriteCount**  A data packet was sent to the server. This event is reported multiple times; Value contains the number of sent bytes in the package.

**HR_Error**  when an error has occurred.

The Value parameter contains extra information on the status, in textual form.

To be able to use this event, the functions SendTo, SendToEx and SendToRaw cannot be used. It is necessary to create an instance of the TSMTPSend class, and to call the necessary methods manually. This is not a lot of effort, and the implementation of the SendToRaw function shows how to use the class:

```
Function TMainForm.DoSendMailAndAttachmentsProgress
  (Const ATo, ASubject : String; Content : TStrings) : Boolean;
var
  SMTP: TSMTPSend;
  s, t: string;
  L : Integer;
begin
  Result := False;
  SMTP:=TSMPSTSend.Create;
  try
    SMTP.TargetHost := Trim(FSMTPHost);
    SMTP.Username := FSMTPUser;
    SMTP.Password := FSMTPPassword;
    // Set status callback:
    SMTP.Sock.OnStatus:=@ShowStatus;
    CurrentSent:=0;
    SendSize:=Length(Content.Text);

    At this point, all properties that the TSMTPSend class needs to do its work, have been set. The CurrentSent and SendSize variables are 2 form variables which keep the amount of sent bytes, and the total amount of bytes to be sent.

    Now, the actual sending of the SMTP message can start:

    // Log in to SMTP server
    if SMTP.Login then
      begin
```
// Set sender address and total send size
if SMTP.MailFrom(GetEmailAddr(FSender), SendSize) then
begin
  s:=ATo;
  // Add all recipient addresses
  repeat
    t:=GetEmailAddr(Trim(FetchEx(s, ',', '"')));
    if (t<>') then
      Result := SMTP.MailTo(t);
      if not Result then
        Break;
  until s = '';
  // Now send e-mail content
  if Result then
    Result := SMTP.MailData(Content);
  end;
  // And log out...
  SMTP.Logout;
end;
finally
  SMTP.Free;
end;
end;

The GetEmailAddr function of the synautl unit extracts the e-mail address from an address/name pair like

Michael Van Canneyt <michael@freepascal.org>

The event handler that was assigned to the TSMTPSend class still needs to be implemented. A large case statement in the event handler is used to handle all possible values of the socket status. To indicate the status, a progress bar and a status panel are added to the form. The progress bar is shown only when the connection with the server has been established, and is hidden when the connection is closed.

procedure TMainForm.ShowStatus(Sender: TObject;
  Reason: THookSocketReason;
  const Value: string);

Var
  S : String;
  d : double;

begin
  Case reason of
    HR_ResolvingBegin :
      SBMain.SimpleText:='Resolving SMTP server IP';
    HR_ResolvingEnd :
      SBMain.SimpleText:='';
    HR_SocketCreate :
      begin
        PBStatus.Visible:=True;
        PBStatus.Position:=0;
      end;
  end;
The `HR_WriteCount` status is reported after each sent data packet. The following code keeps a running total, and shows some statistics in the status panel:

```pascal
HR_WriteCount :
    begin
        CurrentSent:=CurrentSent+StrToInt(Value);
        D:=CurrentSent/SendSize;
        PBStatus.Position:=Round(D*PBStatus.Max);
        S:=Format('%d/%d bytes sent (%5.2f %%)',
                    [CurrentSent,SendSize,D*100]);
        SBMain.SimpleText:=S;
    end;
Application.ProcessMessages;
end;
```

The last statement is a call to `Application.ProcessMessages`: this will allow the form to redraw itself.

### 5 Conclusion

In today's on-line world, tasks such as sending an email is part of the standard functionality of many programs. This article has attempted to demonstrate that, using the Synapse package, Lazarus offers a simple way to send emails.