This manual is designed to give you quick start with the components included in the *Internet Mail Suite*. It will not show you the advanced features of the components, you will have to check out the help file, IMS3.HLP, and advanced demo applications which are included in the *DEMOS* folder.

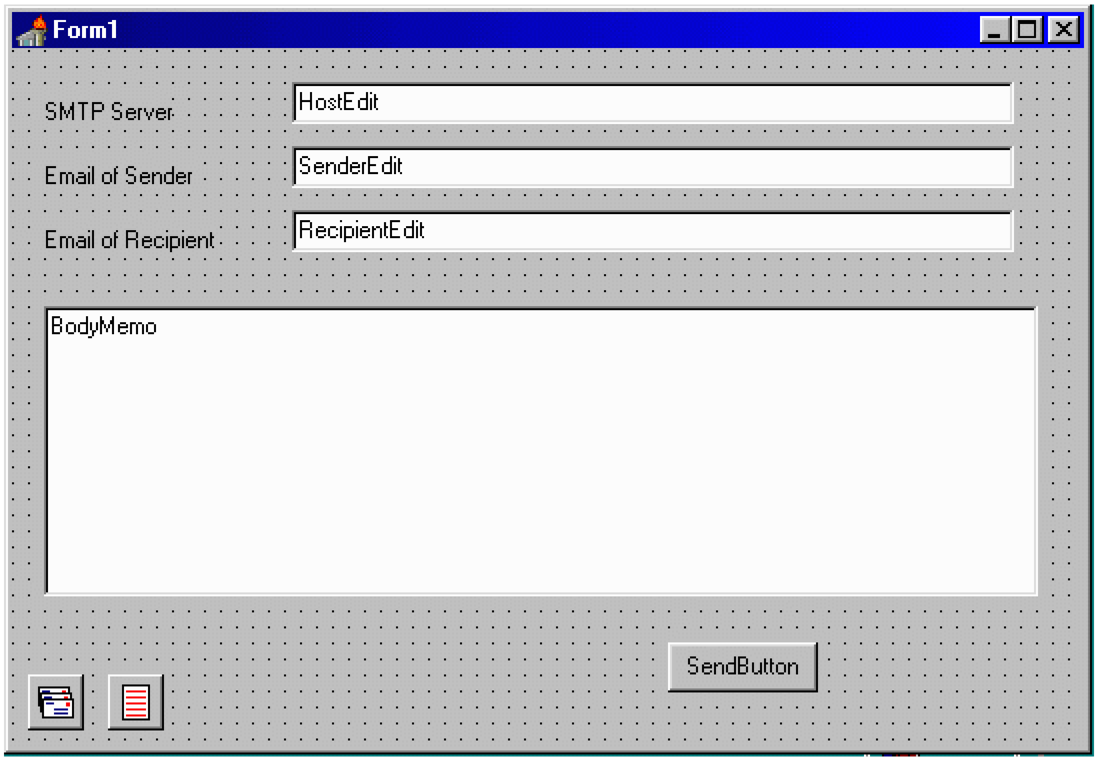
Table of Contents

How to use *TmsSMTPClient* and *TmsMessage* components

TmsSMTPClient component implements *Simple mail transfer protocol* (SMTP), defined in RFC 821. It allows you to send Internet mail messages using SMTP server. *TmsSMTPClient* component should be used in conjunction with *TmsMessage* component, which is the implementation of RFC 622 compatible email message.

Start up Delphi, and create new application. Drop on the Form1 *TmsSMTPClient* and *TmsMessage* components. Then, click *TmsSMTPClient* component on the form, go to the Object Inspector, find *MailMessage* property and set it to *msMessage1*. It will tell to the msSMTPClient1 object what exactly we are going to send.

Now, drop three TLabel, three TEdit, one TMemo, and one TButton components on the form, so that it looks like this:



Then, double click SendButton and create the event handler, which looks like the following:

```
procedure TForm1.SendButtonClick(Sender: TObject);
begin
  msSMTPClient1.Host:=ServerEdit.Text;
  msMessage1.Clear;
  msMessage1.Sender.Address:=SenderEdit.Text;
  msMessage1.Recipients.AddAddress(RecipientEdit.Text,'');
  msMessage1.Body:=BodyMemo.Lines;
  msSMTP1.Send;
end;
```

Then, compile the application and run it. In the Server box you should enter the domain address of the SMTP server of your ISP, e.g. smtp.mydomain.com, in the Sender email box – your email address, and in the recipient box – the email address of recipient. In the memo field type the message you want to send, and click Send button. If you are connected to the Internet, the message will be sent.

The application described above is included in the package. Its name is smtp.dpr.

Please take a look at the application smtpdemo.dpr, which is also included in the package. It demonstrates how to use other capabilities of *TmsSMTPClient* and *TmsMessage* components, such as using event handlers, and sending attachments. Also, take a look at the help file, which contains detailed description of important properties and methods of these components.

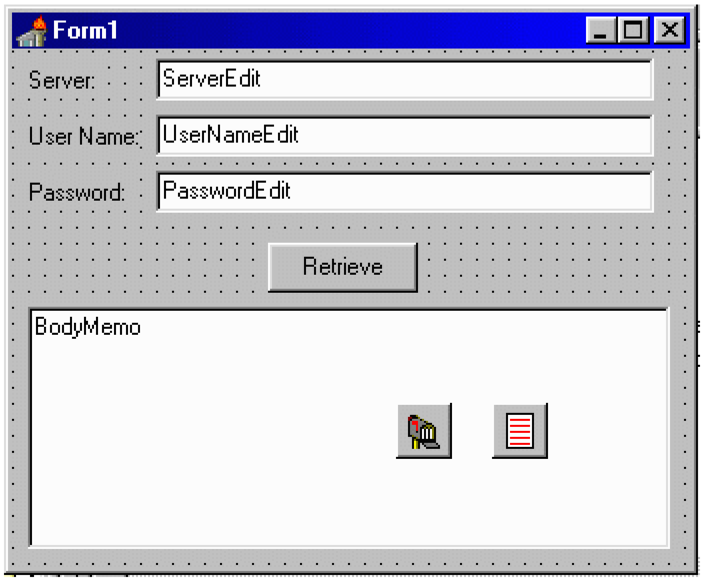
How to use *TmsPOPClient* component

TmsPOPClient component is used for receiving Internet mail using Post office protocol, version 3 (POP3), described in RFC 1939. As it's companion, TmsSMTPClient component, TmsPOPClient component should be used together with TmsMessage component.

We will create two demo applications. First will allow us to retrieve all messages from the server, second one – will let us to retrieve the messages selectively.

Start up Delphi and create new project. Drop on the form TmsPOPClient and TmsMessage components, click TmsPOPClient component, go to the Object Inspector, find MailMessage property and set it to msMessage1. It will tell TmsPOPClient component, that the messages should be retrieved into the msMessage1 component.

Then, drop three TLabel, three TEdit, one TButton, and one TMemo components to the form, name Edit components as ServerEdit, UserNameEdit, PasswordEdit, name the Button as RetrieveButton, and set it's caption as Retrieve, rename Memo to BodyMemo, so that the form looks like this:



Then, double click RetrieveButton, and create the event handler, which looks like this:

```
procedure TForm1.RetrieveButtonClick(Sender: TObject);
begin
  msPOPClient1.Host:=ServerEdit.Text;
  msPOPClient1.UserName:=UserNameEdit.Text;
  msPOPClient1.Password:=PasswordEdit.Text;
  msPOPClient1.Login;
  if msPOPClient1.TotalMessages>0 then
  begin
    msPOPClient1.CurrentMessage:=0;
    msPOPClient1.Retrieve;
    BodyMemo.Lines:=msMessage1.Body;
  end
  else
    ShowMessage('There are no messages');
  msPOPClient1.Logout;
```

```
end;
```

Then, recompile the application and run it. In the Server box you should set the domain name of the POP server you are trying to connect to. You also should set the user name and the password of your POP account. Then click Retrieve button. The application will connect to the server, retrieve first message from the server, and display it's body in the Memo. If there are no message, you will see a note about it.

The event handler, we wrote above, does the following things: in first three lines we set the Host, UserName and Password properties, then called Login method. Our application connected to the server and retrieved the information about the number of messages, which are waiting for us. Then, we are checking whether we have any messages waiting, and if it's number is greater than 0, setting the pointer of the message to the first message (message number 0), and retrieving it.

This code is contained in the application called pop1.drp, and located in the DOCS folder. You also can take a look at popdemo.dpr, which is fully functional POP3 client. It is located in the DEMOS folder.

Now, let's try to make our program more complicated, and convert it so that it retrieves the message, which contains, say, the word Camping in the subject line.

Let's go back to our OnButtonClick event handler and make several changes in it. It should look like

```
procedure TForm1.RetrieveButtonClick(Sender: TObject);
var
  i: Integer;
  Found: boolean;
begin
  msPOPClient1.Host:=ServerEdit.Text;
  msPOPClient1.UserName:=UserNameEdit.Text;
  msPOPClient1.Password:=PasswordEdit.Text;
  msPOPClient1.Login;
  if msPOPClient1.TotalMessages>0 then
  begin
    Found:=false;
    for i:=0 to msPOPClient1.TotalMessages-1 do
    begin
      msPOPClient1.CurrentMessage:=i;
      msPOPClient1.RetrieveHeaders;
      Found:=Pos('camping',LowerCase(msMessage1.Subject))>0;
      if Found then Break;
    end;
    if Found then
    begin
      msPOPClient1.CurrentMessage:=i;
      msPOPClient1.Retrieve;
      BodyMemo.Lines:=msMessage1.Body;
    end
    else
      ShowMessage('Message not found');
  end
  else
    ShowMessage('There are no messages');
  msPOPClient1.Logout;
end;
```

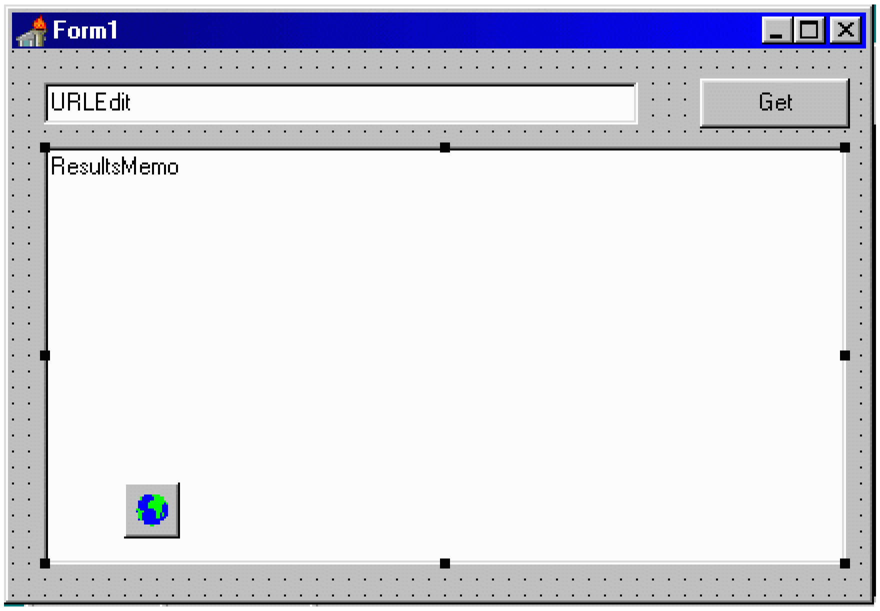
As you can see, this procedure is more complicated. Now, if the number of messages is greater than zero, we are doing the iteration through all messages, retrieving the headers, which will fill the properties of

msMessage1, and checking if the subject line contains the string we are looking for. If we find this kind of message, we are retrieving it and displaying the body of retrieved message in the Memo.
You will find this application in DOCS directory. It is called pop2.dpr. More complicated selective POP application is located in DEMOS folder. It is called spop.dpr.

How to use TmsHHTPClient Component

TmsHHTPClient component implements Hypertext Transfer Protocol (HTTP, RFC 2068), and can be used to retrieve and send the data from/to HTTP servers… You will need this component if you want to create web browser.

Start up Delphi, create new application, and drop on the form a *TmsHTTPClient* component, also, one *TEdit*, one *TButton*, and one *TMemo* components. Name *TEdit* to *URLEdit*, *TButton* – *GetButton*, *TMemo* – *ResultsMemo*. Your form should look like this:



Then, double click *GetButton*, and create the following event handler:

```
procedure TForm1.GetButtonClick(Sender: TObject);
begin
  msHTTPClient1.URL:=URLEdit.Text;
  msHTTPClient1.Get;
  msHTTPClient1.InStream.Position:=0;
  ResultsMemo.Lines.LoadFromStream(msHTTPClient1.InStream);
end;
```

In the first line of this procedure we are assigning the URL of the document we wish to retrieve. Then, we are calling *Get* method of *TmsHTTPClient* component to retrieve it, after we are done, we are "rewinding" received stream, and loading it to our *TMemo* component.

Now you can run the application and retrieve the documents from the web.

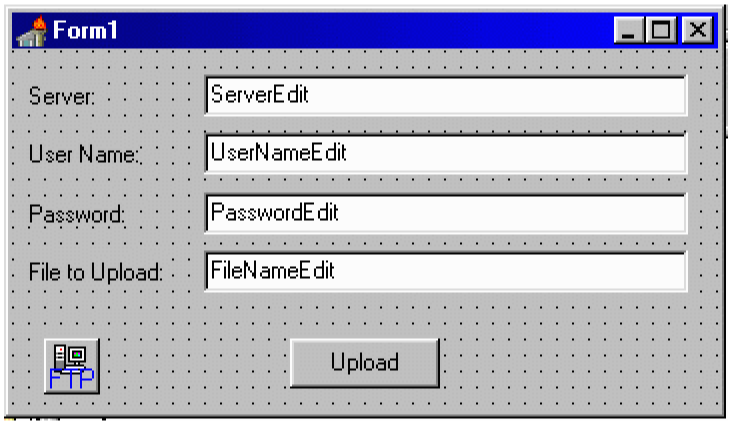
The application we described above resides in the *DOCS* folder, and is called *http.dpr*. Please, also take a look at the applications in the *DEMOS* folder, httpdemo.dpr, which is used to retrieve the resources from the Web, and *httppostdemo.dpr*, which demonstrates how to use *Post* method of *TmsHTTPClient* component.

How to use *TmsFTPClient* component

TmsFTPClient component implements File Transfer Protocol (FTP, RFC 959), and used to store and retrieve files to/from the FTP server.

Start up Delphi, and create new application. Drop on the main form *TmsFTPClient* component, also four *TLabels*, four *TEdits*, and one *TButton* components. Name *Edit* components as *ServerEdit*, *UserNameEdit*, *PasswordEdit*, and *FileNameEdit*, rename *Button* component to *UploadButton*, and set its caption as *Upload*.

Your form should look like this:



Then, double click *UploadButton* and create the event handler, which looks like this:

```
procedure TForm1.UploadButtonClick(Sender: TObject);
begin
  msFTPClient1.Host:=ServerEdit.Text;
  msFTPClient1.UserName:=UserNameEdit.Text;
  msFTPClient1.Password:=PasswordEdit.Text;
  msFTPClient1.Login;  msFTPClient1.StoreFile(FileNameEdit.Text,
    ExtractFileName(FileNameEdit.Text));
  msFTPClient1.Logout;
end;
```

First three lines – we are setting *Host*, *UserName* and *Password* properties of *TmsFTPClient* component. Then, we are logging into the server, by calling *Login* method, and uploading the file, specified in the *FileNameEdit* edit box. Full path must be specified. Last line logs us out from the server and closes the connection. The program will store out file with the same name, in the current directory on the server.

This application is in the *DOCS* folder. Name of the project is *ftp.drp*. Also, take a look at the *ftpdemo.drp*, in the *DEMOS* folder. It is the fully functional FTP client.

How to use *TmsNNTPClient* component

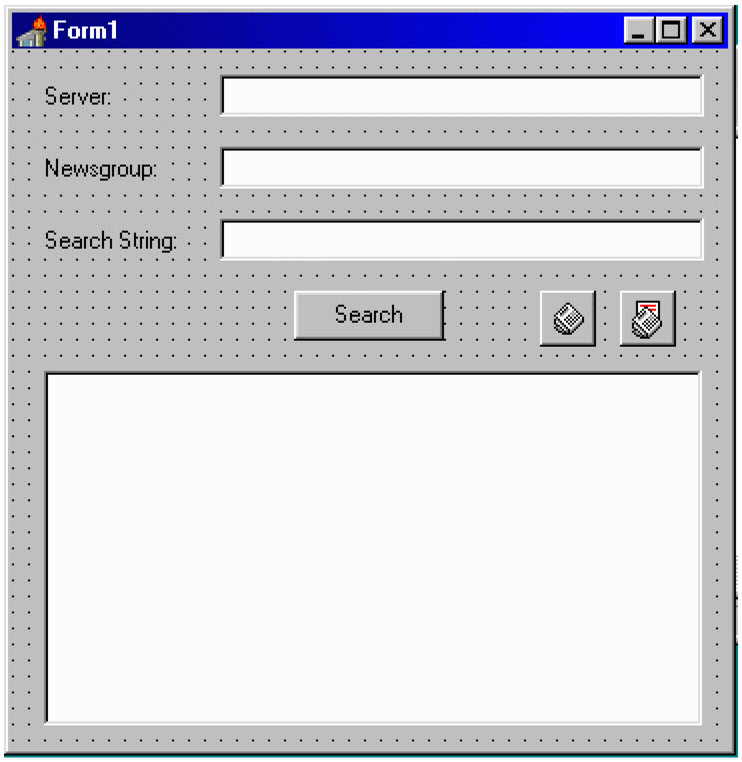
TmsNNTPClient component implements Network News Transfer Protocol (NNTP, RFC 977) and allows an access to the Usenet Newsgroups.

TmsNNTPClient component is used in conjunction with *TmsArticle* component. These two components work alike of *TmsPOPClient* and *TmsMessage* components.

Let's try to create an application, which connects to the specified newsgroup, searches for the article, the subject of which contains specified string, and, if it finds it, retrieves the article and displays it.

Start up *Delphi*. Then, drop on the *Form TmsNNTPClient* and *TmsArticle* components, then click *TmsNNTPClient* component, go to the Object inspector and set its *Article* property to *msArticle1*. Then, drop three *TLabel*, three *TEdit*, one *TButton*, and one *TMemo* components on the form, name the *Edit* components as *ServerEdit*, *NewsgroupEdit*, and *SearchEdit*, *Button – SearchButton*, and *Memo – BodyMemo*.

Your form should look like this:



Double click *SearchButton*, and create the event handler, which looks like this:

```
procedure TForm1.SearchButtonClick(Sender: TObject);
var
  i: Integer;
  Found: boolean;
begin
  msNNTPClient1.Host:=ServerEdit.Text;
  msNNTPClient1.Login;
  msNNTPClient1.Newsgroup:=NewsgroupEdit.Text;
  msNNTPClient1.GetOverview(msNNTPClient1.FirstArticle,
    msNNTPClient1.LastArticle);
  Found:=false;
  for i:=0 to msNNTPClient1.Overview.Count-1 do
  begin
    Found:=Pos(LowerCase(SearchEdit.Text),
      LowerCase(msNNTPClient1.Overview[i].Subject))>0;
    if Found then Break;
  end;
  if Found then
  begin
    msNNTPClient1.CurrentArticle:=msNNTPClient1.Overview[i].ArticleNo;
    msNNTPClient1.Retrieve;
    BodyMemo.Lines:=msArticle1.Body;
  end
  else
    ShowMessage('Cannot find the article');
  msNNTPClient1.Logout;
end;
```

In the first line we are setting the host we want to connect to, then, we are logging into the server, by calling *Login* method. In the third line we are selecting the newsgroup. After this, we are calling *GetOverview* method, which retrieves the information about the articles in the newsgroup. We are requesting overviews of all articles, starting from first one, and ending with the last.

After we are done with it, we are iterating through the *Overview*, trying to find the article, subject of which contains the search string. *Overview* property is a list of *TmsOverviewItem* objects, which contains useful information about the articles, including the subject of the article. If we find the article we are looking for, we are interrupting the loop, and setting the CurrentArticle pointer to the article we just found (*OverviewItem* also contains the number of the article, and we are using this information). Then, we are retrieving the article, and displaying its body in the BodyMemo.

If the article has not been found, we are displaying the message about it.

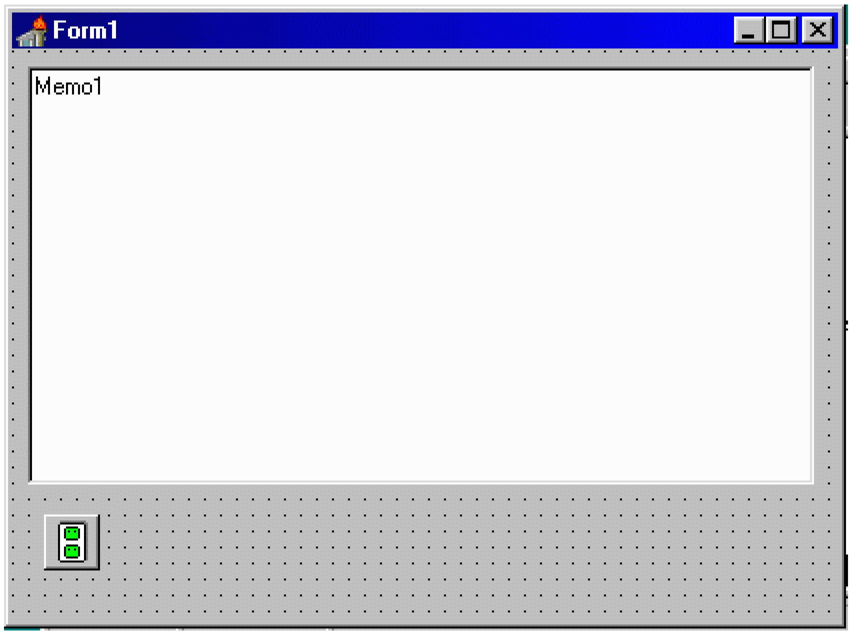
This project can be found in *DOCS* folder, its name is *nntp.drp*. You also should take a look at the project *nntpdemo.dpr*, which can be found in the *DEMOS* folder, and which is fully functional NNTP client.

How to use *TmsListenerSocket* component and *TmsServerThread* class

Use *TmsListenerSocket* component to develop server applications. It works in conjunction with *TmsServerThread* class, which is the descendant of Delphi *TThread* class, and controls the actual behaviour of the server you want to create. You should white this class yourself, and assign it to the *ServerThreadClass* property of *TmsListenerSocket* component.

Let's try to create very simple server application. It will listen on the port 1090 and, as soon as client connects to it, sends back a content of the file named *myfile.txt*.

Start Delphi IDE, and click File – New Application. On the *Form1*, drop *TmsListenerSocket* component, then, in the Object Inspector, set the *Port* property to 1090. Also, drop one TMemo component. Your form should look like this:



Now, let's write the descendant of *TmsServerThread* class, which will implement the actual behaviour of the server. We will have to create the class, which descends from *TmsServerThread*, and override the *execute* method.

Switch to the Unit1.pas by pressing F12, and, just above of the TForm1 class declaration, insert the following:

```
TMyServerThread = class(TmsServerThread)
protected
  procedure Execute; override;
end;
```

And, in the implementation section, type the following code:

```
procedure TmyServerThread.Execute;
var
  OutStream: TStream;
```

```
begin
  OutStream:=TFileStream.Create('myfile.txt',fmShareDenyWrite);
  try
    ServerSocket.SendStream(OutStream);
    ServerSocket.Disconnect;
  finally
    OutStream.Free;
  end;
end;
```

Note the usage of *fmShareDenyWrite* file open mode constant in the constructor of *OutStream*. If we use just *fdOpenWrite*, the program will crash if the server receives more than one request at the same time.

Now, let's go back to our form, and assign the OnCreate event handler:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  msListenerSocket1.ServerThreadClass:=TMyServerThread;
  msListenerSocket1.Start;
end;
```

Here we are telling to the instance of TmsListenerSocket component, which code to use when a connection request arrives from the client, and also, starting up the server.

Please note, that we are assigning the type of the server code we just wrote, not the instance of TMyServerThread class… We never create an instance of our server classes. TmsListenerSocket component does it for us when it accepts the connection.

In the OnClose event handler we will have to stop the server:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  msListenerSocket1.Stop;
end;
```

Now, let's run the program, and try to connect to it using Telnet application, which comes with Windows. Run Telnet and click Connect – Remote System, then in the Host Name box type either localhost, or 127.0.0.1, and in the Port box – 1090 (the number of the port our server is listening), and click connect. You should see the content of myfile.txt, returned by our server.

As you can see, it is very simple to create the server application using these two classes. But now let's make the server more complex.

First, let's try to add logging of received connections.

Drop the TMemo component on the form, and then add the method UpdateStatus to the TmyServerThread class definition:

```
TMyServerThread = class(TmsServerThread)
private
  FStatusStr: string;
  procedure UpdateStatus;
protected
  procedure Execute; override;
end;
```

We also added FStatusStr field, which will be used to pass the status string to the UpdateStatus method, which will be called through TThread.Synchronize method.

In the implementation section, add the following code:

```
procedure TMyServerThread.UpdateStatus;
begin
  Form1.Memo1.Lines.Add(FStatusStr);
end;
```

And modify the TMyServerThread.Execute method so that it looks like the following:

```
procedure TmyServerThread.Execute;
var
  OutStream: TStream;
begin
  FStatusStr:='Reqested connection from '+Peer;
  Synchronize(UpdateStatus);
  OutStream:=TFileStream.Create('myfile.txt',fmShareDenyWrite);
  try
    FStatusStr:='Sending data to '+Peer;
    Synchronize(UpdateStatus);
    ServerSocket.SendStream(OutStream);
    FStatusStr:='Closing connection with '+Peer;
    Synchronize(UpdateStatus);
    ServerSocket.Disconnect;
  finally
    OutStream.Free;
  end;
end;
```

Now, run the program and connect to it using Telnet application, which comes with Windows… You will see a log of connection in Memo1.

Now, let's make the server thread even more complicated. In some cases you will need to perform certain initialisation and cleanup for server threads. For example, if you wish to use some local variables inside of threads, such as other VCL components, or allocate the memory for certain variables. Since you are not creating the instances of TmsServerThread descendants, you will have to use OnServerThreadStart and OnServerThreadTerminate event handlers of TmsListenerSocket component.

Let's modify our existing server so that it OutStream is declared as a property of TMyServerThread class, not as a local variable of TMyServerThread.Execute method.

So, now, the declaration of TMyServerThread class looks like this:

```
TMyServerThread = class(TmsServerThread)
private
  FOutStream: TStream;
  FStatusStr: string;
  procedure UpdateStatus;
protected
  procedure Execute; override;
public
  property OutStream: TStream read FOutStream write FOutStream;
end;
```

Now, TMyServerThread.Execute method will look like this:

```
procedure TmyServerThread.Execute;
begin
  FStatusStr:='Reqested connection from '+Peer;
  Synchronize(UpdateStatus);
  FStatusStr:='Sending data to '+Peer;
  Synchronize(UpdateStatus);
  ServerSocket.SendStream(FOutStream);
  FStatusStr:='Closing connection with '+Peer;
  Synchronize(UpdateStatus);
  ServerSocket.Disconnect;
end;
```

Now, we have to create OnServerThreadStart and OnServerThreadTerminate event handlers for TmsListenerSocket component. Click the msListenerSocket1 on the form, then go to the Object Inspector and assign above methods.

```
procedure TForm1.msListenerSocket1ServerThreadStart(Sender: TObject;
  ServerThread: TmsServerThread);
var
  TempStream: TStream;
begin
  TempStream:=TFileStream.Create('myfile.txt',fmShareDenyWrite);
  (ServerThread as TMyServerThread).OutStream:=TempStream;
end;
```

In this event handler we are creating the TFileStream object, and assigning it to the OutStream property of the instance of TMyServerThread, which has been created by msListenerSocket1, when it received a connection request.

In the OnServerThreadTerminate event handler we have to dispose the stream we created above:

```
procedure TForm1.msListenerSocket1ServerThreadTerminate(Sender:
TObject;
  ServerThread: TmsServerThread);
begin
  (ServerThread as TMyServerThread).OutStream.Free;
end;
```

This version does not do anything different from the previous one; it just illustrates how to initialise and cleanup the server thread.

Now, we should take special care of exception handling in the server thread. If an exception raises inside of the thread, we will have to pass it's handling to the main thread, otherwise our program may crash if there is even single exception in a single connection.

The simplest way of doing it is to call *Application.HandleException* method, which will display traditional dialog box with the standard exception message. But, it will mean that the server will require the user intervention in order to close this dialog box. So, what we are going to do is – just record the information about the exception into the log.

To do it, let's change the procedure TMyServerThread.Execute to the following:

```
procedure TmyServerThread.Execute;
begin
```

```
  FStatusStr:='Reqested connection from '+Peer;
  Synchronize(UpdateStatus);
  FStatusStr:='Sending data to '+Peer;
  Synchronize(UpdateStatus);
  try
    ServerSocket.SendStream(FOutStream);
  except
    on E:Exception do
    begin
      FStatusStr:='Error '+E.Message;
      Synchronize(UpdateStatus);
    end;
  end;
  FStatusStr:='Closing connection with '+Peer;
  Synchronize(UpdateStatus);
  ServerSocket.Disconnect;
end;
```

As you can see, the creation of complex server applications, using the components included in IMS is very simple.

The code of the application we just created is in the project ls1.dpr, and is included in the IMS package.

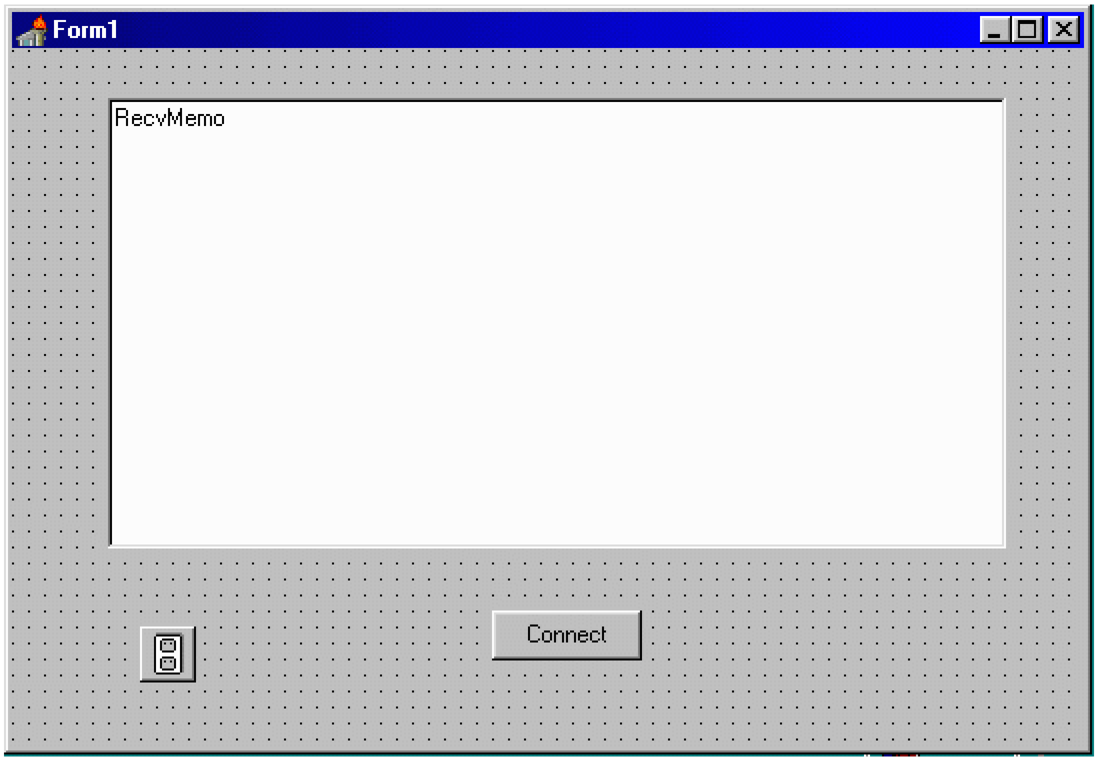
How to use TmsClientSocket component

TmsClientSocket component can be used for creating client application. It is an ancestor of all high-level client components, such as *TmsSMTPClient*, *TmsPOPClient*, *TmsHTTPClient*, *TmsNNTPClient* etc.

We will write the client application, which works with the server program, written in the chapter *How to Use TmsListenerSocket and TmsServerThread components*.

Start up Delphi, and create new application. Then, on the Form1 drop TmsClientSocket component, also one TMemo, and one TButton components. Name the TMemo as RecvMemo, and TButton - as ConnectButton, also, change the caption of the button to Connect. Then, set the port property of TmsClientSocket component to 1090, this is the port where our server will listen, and set the Host property to localhost, or 127.0.0.1, which will indicate that we will be running the server on the same computer where we are running the client.

Our form should look like this:



Then, double click the connect button and create the event handler which looks like the following:

```
procedure TForm1.ConnectButtonClick(Sender: TObject);
var
  TempStream: TStream;
begin
  TempStream:=TMemoryStream.Create;
  try
    msClientSocket1.Connect;
    msClientSocket1.RecvStream(TempStream,-1,0);
    TempStream.Position:=0;
    RecvMemo.Lines.LoadFromStream(TempStream);
    msClientSocket1.Disconnect;
  finally
    TempStream.Free;
  end;
end;
```

In this procedure, we are creating a temporary stream, which will receive the data from the server. Since we know that the server will send the data and close the connection to indicate the end of the session, we are using RecvStream method of TmsClientSocket component. Since we don't know in advance the size of the received data, we are passing minus one as a parameter. Then after the data was received, we are loading it into the Memo component on our form.

Start up the project ls1.dpr, then, recompile this application, run it, and click Connect button. You will see the received data, in memo component.