Internet Mail Suite

Internet Mail Suite (IMS) lets you do much more than send and receive Internet mail. We kept this name only for historical reasons. First version of IMS, released as early as in 1996, was handling only Internet mail related protocols. Later we added the components, implementing other standards, but decided to keep the name.

And, now, we enhanced it again. Starting from this version, you will be able to create server applications.

Internet Mail Suite contains following components:

• TmsClientSocket - multi-purpose client socket, an ancestor of other high level IMS client components;

• TmsSMTPClient - implementation of Standard Mail Transfer Protocol; Lets you send Internet mail;

• TmsPOPClient - implementation of Post Office Protocol version 3; Lets you receive Internet mail;

• TmsMessage - implementation of Internet Mail message format, according to rfc822; Lets you prepare, the email messages according to Internet standards, also parse the content of received email messages;

• TmsNNTPClient - implementation of Network News Transfer Protocol; Lets you post and retrieve the articles to/from the Usenet;

• TmsArticle - modification of TmsMessage component for NNTP;

• TmsFTPClient - implementation of File Transfer Protocol; Lets you transfer the files to/from computers running FTP server;

• TmsHTTPClient - implementation of Hypertext Transfer Protocol; Allows to retrieve the files from the Web, also post the data to HTTP servers;

• TmsSimpleListenerSocket - Allows you to write a Server, which does not require to handle multiple connections;

• TmsListenerSocket - Allows you to develop more complex servers, which can handle multiple connections in multiple threads.

And, one class, TmsWinsock. An instance of it, a global object msWinsock, always exists and is always accessible. It is declared in msWSock unit.

TmsSMTPClient component

properties methods events
unit mssmtp

declaration
```
TmsSMTPClient = class(TmsClientSocket)
```

description
Implements SMTP (Simple Mail Transfer Protocol, rfc 821) and allows you to send Internet mail. Set Host property to the address of SMTP server you wish to use for sending the data contained in the MailMessage property. Call Send method to send single message.

TmsClientSocket Component

properties methods events

unit msSocket;

declaration

```
TmsClientSocket = class(TmsSocket)
```

description

Ancestor of all IMS client socket components. Also can be used separately, to do the conversations with hosts on the Internet, which are listening to the specific port.

TmsPOPClient Component

properties methods events
unit mspop;

declaration

```
TmsPOPClient = class(TmsClientSocket);
```

description

Implements the client side of POP3 (Post Office Protocol, version 3, rfc 1939) and allows you to retrieve the messages from POP3 server, also, the information about the messages located on the server.

Set Host property to the server you wish to check the messages, you also will need to set UserName and Password properties, since they are required by POP3. Then log into the server by calling Login method, and iterate through the email messages by setting CurrentMessage property to the message number you are interested in, then retrieve entire message using Retrieve method, or, just it's headers using RetrieveHeaders method. The data will be stored in the MailMessage property, which is a TmsMessage component. You also can retrieve the information, which is normally not stored in the message headers, such as message size (by calling GetSize function), or UIDL (by calling GetUIDL function). When you are done, you can disconnect from the server, by calling Logout method.

TmsNNTPClient Component

properties methods events
unit msNNTP;

declaration
```
TmsNNTPClient = class(TmsClientSocket);
```

descritpion
Implements Network News Transfer Protocol (NNTP, rfc 977), allows you to retrieve and post articles to/from Usenet.

Set Host property to the news server you wish to use. Then log into it, calling Login method. Call RetrieveNewsgroupList method to retrieve the list of available newsgroups on the server. If you already know which newsgroup are you interested in, make it current using Newsgroup property. After doing it you the data about the numbers of available articles will be stored in FirstArticle and LastArticle properties. Then, you can get the information about available articles in the newsgroup using GetOverview method. After you retrieved the information about the articles, you can iterate through articles using First, Next, and Last methods, by setting CurrentArticle property, and calling Retrieve method, or call RetrieveArticleByNumber method to get the article into the Article property of *TmsNNTPClient* component. You also can retrieve the articles based on their IDs, without selecting the current newsgroup, using RetrieveArticleByID method. You also can prepare the article in the Article property and post it, using Post method. Call Logout method when you are done.

If your server requires authorization, you should use Authorization, UserName and Password properties of TmsNNTPClient component. This part is written by Mr. **Bruce W. Caron -** thousands of thanks to him!!!

TmsFTPClient Component

properties methods events

unit msFTP;

declaration

```
TmsFTPClient = class(TmsClientSocket);
```

description

Implements the client side of FTP (File Transfer Protocol, rfc 959). Allows you to transfer files, also, retrieve directory information from FTP server.

Set Host property to the server you wish to connect to. Set appropriately UserName and Password properties, and call Login method to connect and log into the server. After this you can use StoreFile and RetrieveFile methods, also various modifications of these methods, to transfer the files, also, GetDirList method to retrieve the directory information into the DirList property. When you are done, call Logout method to log out of the server and close the connection.

TmsHTTPClient Component

properties methods events
unit msHTTP;

declaration
```
TmsHTTPClient = class(TmsClientSocket);
```

description
Implements HyperText Transfer Protocol (HTTP, rfc 2068). Allows you to retrieve the resources from the Web, using Get method, also, post the data to HTTP servers, using Post method.

Set URL property to some document or file and call Get method. The retrieved data will be saved in the Headers and InStream properties. Then you can save OutStream into the file, calling, for example InStream.SaveToFile.

To post the data to the server, set the URL to the application which will be processing the posted data, and store the data you are going to post in OutStream, then, call Post method. The reply from the server will be stored in the Headers and InStream.

Contact

You can contact *ArGo Software Design* at:

EMail: support@argosoft.com
WWW: http://www.argosoft.com

Fax: +1-416-352-5054

Postal Mail: ArGo Software Designn
4325 Steeles Ave West, #211
North York, Ontario M3N 1V7
Canada

All comments, questions, suggestions or bug reports are welcome.

TmsSimpleListenerSocket Component

properties methods events
unit mssocket;

declaration
```
TmsSimpleListenerSocket = class(TmsListenerSocketBase);
```

description
Use this component when you wish to create simple server, which will accept only one connection at a time. The actual actions of the server will be performed in the same thread.

Set the Port property you wish your application to listen, and call Start method. Then, write OnSLSConnectionRequested event handler. In this event handler you will be able to send and receive the data using ServerSocket property of the instance of your *TmsSimpleListenerSocket*.

The following event handler will receive a string from the client, conver it to the uppercase, send it back, and close the connection:

```
procedure MyServerForm.msSimpleListenerSocket1SLSConnectionRequested(Sender:
TObject);
var
  s: string;
begin
  s:=SimpleListenerSocket1.ServerSocket.RecvLine;
  SimpleListenerSocket1.ServerSocket.SendLine(UpperCase(s));
  SimpleListenerSocket1.ServerSocket.Disconnect;
end;
```

TmsListenerSocket Component

properties methods events
unit msSocket;

declaration
```
TmsListenerSocket = class(TmsListenerSocketBase);
```

description
You should use this component when you are develiping a server which will accept mutliple connections at a time. For simpler servers use TmsSimpleListenerSocket component.

TmsListenerSocket component will accept connections, and launch a separate thread for each connection.

Set Port property to the port you wish your server to listen on, then, assign ServerThreadClass property. It must be a name of the class, which is created by you and which does actual processing of the server. Then call Start method. Your server is ready to go.

TmsMessage Component

properties methods

unit msMsg;

declaration

```
TmsMessage = class(TmsCustomMessage);
```

description

TmsMessage is a Internet-Mail oriented customized TmsCustomMessage component. Essentially, it is the same, only difference - it has the properties which are applicable only to Internet Mail (and is not applicable to Network News), such as Recipients, Priority, and ReturnReceipt.

TmsCustomMessage Component

properties methods
unit msMsg;

declaration

```
{$IFDEF VCL}
TmsCustomMessage = class(TComponent);  // default
{$ELSE}
TmsCustomMessage = class(TPersistent);
{$ENDIF}
```

description
TmsCustomMessage component is one of the key components of *The Internet Mail Suite*. This is a powerful encapsulation of rfc822 message (message in standard Internet format). You just assign the values to the properties at runtime or using the Object Inspector, then run SaveToFile or SaveToStream methods and your message is ready to go!!! Or vise versa, load the rfc822, MIME encoded message, using LoadFromFile or LoadFromStream method and it will be decoded (if it is possible) and all the properties of the component will be filled in. Now you can save the attachments calling SaveAttachment method, execute them, using ExecuteAttachment method, store the messages into the mailboxes, reply to the sender and so on.

It is an ancestor of TmsMessage and TmsArticle components, which are respectively used with the Internet mail and NNTP.

TmsArticle Component

properies methods

unit msMsg;

declaration

```
TmsArticle = class(TmsCustomMessage);
```

description

TmsArticle is a NNTP oriented enhancement of TmsCustomMessage component. It has only one additional property, Newsgroups, which contains the name of the newsgroups the article was posted in.

TmsSocket Component

properties methods events

unit msSocket;

declaration

```
TmsSocket = class(TmsSocketBase);
```

descrition

The common ancestor of all IMS client sockets. ServerSocket, associated to TmsListenerSocket and TmsSimpleListenerSocket components, is also of this type. Implements very important and useful methods for transmitting and receiving the data.

TmsSocketBase Component

properties
unit msSocket;

declaration

```
{$IFDEF VCL}
TmsSocketBase = class(TComponent)  // default
{$ELSE}
TmsSocketBase = class
{$ENDIF}
```

description

Common ancestor of all IMS components (both client and server). Contains methods, properties, and event handlers which are common for all components.

See also Using IMS in Console Applications

Using IMS in Console Applications

The compiler conditional define *VCL* controls the way *IMS* components work in the usual Windows applications, or console applications and DLLs.

It is declared in the file *msdef.inc*. By default it is enabled, and all IMS components are "real" components, in the sense of *Delphi*, since they descend from *TComponent*. They can be placed on the component palette and dropped on the form. In this case, the sockets are non-blocking, and winsock calls are asynchronous.

But, it causes Delphi to link to your application additional units, which are needed for *TComponent* class, and creates additional footprint.

If you are developing traditional Windows application, which uses forms and/or other *Delphi* components, the units which participate while you compile your program, will be used anyway, so, there is no need to change default settings.

But, if you are creating console application, or DLL, which does not use above mentioned units, then, it would be good idea to disable the conditional define VCL in the file *msdef.inc*, by editing this file. In this case IMS components will descend from *TObject*. It means, you will be no longer able to drop the components on the form (and, there is no need and way of doing it in console applications). Your programs will be smaller, but keep in mind, that the default constructors of all components will change and you will have to call them the following way:

```
MySMTP:=TmsSMTP.Create;
```

instead of

```
MySMTP:=TmsSMTP.Create(OwnerObject);
```

Also, you will be not able to use Server components. All sockets will become blocking, and some event handlers will become unavailable.

SleepTime *
Socket

SleepTime Property

unit msSocket;

applies to
TmsSocketBase class

declaration
```
SleepTime: Integer;
```

description
Helps you to control the CPU usage of computer. If it is larger than -1, IMS will call the Sleep Windows API function in the loops, with the argument specified in this property.

Not available if VCL conditional define is off.

Socket Property

unit msSocket;

applies to
TmsSocketBase class

declaration
```
property Socket: TSocket;
```

description
Handle of the socket associated with the corresponding IMS socket component.

LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

TimeOut Property

unit msSocket

applies to
TmsSocket class

declaration
```
property TimeOut : Integer;
```

description
The number of seconds TmsSocket or its descendants will be waiting for some event related to the receiving or sending data. If time expires, EmsTimedOutError exception will be raised. Default value is 60.

EmsTimedOut Exception

unit msdef

description
This exception will be raised if there was no socket activity during the TimeOut period of TmsSocket component.

Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

Cancel Method

unit msSocket

applies to
TmsSocket class

declaration
```
procedure Cancel;
```

description
Immediately closes connection with server. If the application is in the process transferring or receiving the data, EmsCanceledError exception will be raised.

EmsCanceledError Exception

unit msdef

description

This exception will be raised if the operation has been canceled.

Disconnect Method

unit msSocket

applies to
TmsSocket class

declaration
```
procedure Disconnect;
```

description
Tries to gracefully close the connection. If connection has been already closed, does not perform any action.

Read Method

unit msSocket;

applies to
TmsSocket component

declaration

```
function Read(var Buffer; Count : LongInt) : LongInt;
```

description
Receives *Count* bytes from the connected socket into the *Buffer*. If the operation was successfull, returns the number of received bytes, if not, returns SOCKET_ERROR (-1). Works the same way as Winsock recv function. It would be better if you use more high level TmsSocket methods for sending or receiving data.

Write Method

unit msSocket;

applies to
TmsSocket class

declaration

```
procedure Write(const Buffer; Count : LongInt);
```

description
Sends the Count bytes, contained in the Buffer, via connected socket. It would be better if you use more high level TmsSocket methods for sending or receiving data.

RecvChunkedStream method

unit msSocket;

applies to
TmsSocket component

declaration

```
procedure RecvChunkedStream(Stream : TStream);
```

description
Receives so called *chunked* data, which is used in HTTP. Used internally in TmsHTTPClient component and you should not have any need to call this method directly.

SendChunkedStream method

unit msSocket;

applies to
TmsSocket component

declaration

```
procedure SendChunkedStream(Stream : TStream);
```

description
Sends so called *chunked* data, which is used in HTTP. Currently is not used in IMS, but you may need to call it if you decide to create HTTP server.

RecvLine method

unit msSocket;

applies to
TmsSocket component

declaration
```
function RecvLine: string;
```

description
Receives a sequence of characters, until detects CRLF (^M^J), or LF (^J). Uses Read method.

SendLine method

unit msSocket;

applies to
TmsSocket component

declaration

```
procedure SendLine(const s: string);
```

description
Sends the characters, contained in s, via the open socket, followed by CRLF (^M^J). Uses Write method.

RecvLineStream method

unit msSocket;

applies to
TmsSocket component

declaration

```
procedure RecvLineStream(Stream: TStream; FullSize: LongInt);
```

description
Majority of standard Internet protocols frequently are sending back the data which contains multiple lines, followed by the period on the single line (^M^J'.'^M^J), which indicates the end of the data. This procedure is designed to handle these kind of data sent by servers. The data will be received in the *Stream* object, *FullSize* parameter is used to display the transfer progress, if OnTransferProgress event handler is assigned. You should pass FullSize, if you already know the size of the data which should be received, otherwise, you should pass -1.

OnTransferProgress event

unit msSocket;

applies to
TmsSocket classt

declaration
```
procedure OnTransferProgress: TmsProgressEvent;
```

description
Can be used to display the progress of the transfer. Please see TmsProgressEvent for more information.

RecvMultiLines method

unit msSocket;

applies to
TmsSocket component

declaration

```
procedure RecvMultiLines(Lines: TStrings);
```

description
Certain commands in standard Internet protocols require the server reply which contains multiple lines. This method handles this kind of replies. It is used internaly in IMS Client components.

RecvStream method

unit msSocket

applies to
TmsSocket class

declaration
```
procedure RecvStream(Stream : TStream; FullSize, StartPosition: Integer);
```

description
Receives the data from the server, and stores it into the *Stream* until connection from the server is reset, or *FullSize* bytes have arrived. If *FullSize* is -1, then it is ignored. If *StartPosition*>0, the data will be appended to the *Stream*, starting from the position indicated by this parameter.

SendStream method

unit msSocket;

applies to
TmsSocket class

declaration

```
procedure SendStream(Stream : TStream; StartPosition: Integer);
```

description
Sends the content of the *Stream*, starting from *StartPosition*.

LogFileName property

unit msSocket;

applies to
TmsSocket component

declaration
```
LogFileName: string;
```

description
If this property is assigned, the conversation between the client and the server will be stored into the file, with the name assigned to this property, after you call SaveLogFile method.

SaveLogFile method

unit msSocket;

applies to
TmsSocket component

declaration
```
procedure SaveLogFile;
```

description
If LogFileName property is not blank, the conversation between the client and server will be saved to the file. Can be used for debugging.

OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

OnConnected event

unit msSocket;

applies to
TmsSocket class

declaration
```
property OnConnected: TNotifyEvent;
```

description
This event handler will be triggered as soon as the connection has been established

OnDisconnected event

unit msSocket;

applies to
TmsSocket class

declaration
```
property OnDisconnected: TNotifyEvent;
```

description
This event handler will be triggered when connection is closed.

OnOOBData event

unit msSocket;

applies to
TmsSocket class

declaration
```
OnOOBData: TNotifyEvent;
```

description
This event handler will be triggered if out of band data arrives. This event handler is not published, and is used internally by IMS, namely, TmsFTPClient component.

OnRead and OnWrite events

unit msSocket;

applies to
TmsSocket class

declaration

```
OnRead: TNotifyEvent;
OnWrite: TNotifyEvent;
```

description
These event handlers will be triggered when the socket is ready to read (OnRead) or write (OnWrite).
These event handlers are not published and are used internally...

TmsProgressEvent type

unit msSocket;

declaration

```
TmsProgressEvent = procedure(Sender: TObject; Perc, ByteCount, LineCount:
LongInt) of Object;
```

description

This is a type of OnTransferProgress event handler of TmsSocket class. You can display the progress of the data transfer.

Meaning of parameters is as follows:

Perc - Percentage of transferred data. If it is -1, it means that the entire size of the data to be transferred is not known, and this parameter cannot be used;

ByteCount - Number of transferred bytes. Works in all cases;;

LineCount - Number of transferred lines. This parameter will work only in certain cases, e.g., when retrieving the list of newsgroups using TmsNNTPClient component. If it is -1, it means that IMS cannot determine the number of retrieved lines and this parameter should not be used.

Host
Port

inherited from TmsSocket
LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

Host Property

unit msSocket;

applies to
TmsClientSocket class

declaration
```
Host: string;
```

description
The name (e.g., *mail.domain.com*), or an IP address (e.g. *123.111.211.111*) of the computer you are trying to connect to. When you call Connect method, the TmsClientSocket component will attempt to the computer specified in this property, to the port specified in Port property.

Port property (for client components)

unit msSocket;

applies to
TmsClientSocket class

declaration
```
Port: SmallInt;
```

description
When you call Connect method, the TmsClientSocket component will attempt to connect to the computer specified in Host property, to the port specified here...

Connect method

unit msSocket;

applies to
TmsClientSocket class

declaration
```
procedure Connect; virtual;
```

description
Attempts to connect to the computer specified in the Host property, on the port specified in Port property.

Connect

inherited from TmsSocket
Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

Available
WinsockInfo
LocalName
LocalAddress

OnConnecting

inherited from TmsSocket
OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

OnConnecting event

unit msSocket;

applies to
TmsClientSocket component

declaration

```
OnConnecting: TNotifyEvent;
```

description
Will be triggered after you call Connect method. Can be used to display the information, that your program attempts to connect to the host, e.g.

```
procedure TForm1.SMTPOnConnecting(Sender: TObject);
begin
  StatusBar.SimpleText:='Connecting to '+SMTP.Host;
end;
```

TmsWinsock Class

properties methods
unit **msWSock**;

declaration

```
TmsWinsock = class
```

description

TmsWinsock class exports winsock functions from winsock.dll (wsock32.dll). They are the methods of this class, which means, if you want to call any winsock function, you can call them via *TmsWinsock*.Class.

It also has other useful methods, which can be used to retrieve useful information. Please keep in mind, that you do not need to create the instance of this object in your application, since it always exists. Then name of this global object is msWinsock. It is declared in the unit msWSock. Also, we will not be describing the functions from winsock in this help file. You will find the descriptions only of methods and properties which are unique for IMS.

Impotant: You don't have to create an instance of TmsWinsockClass. It always exists in your application, its name is msWinsock, and is declared in msWSock unit.

Available property

unit msWSock;

applies to
TmsWinsock class

declaration

```
Available: boolean;
```

description
You can use this property to find out whether Winsock implementation is available on the computer. You can check *msWinsock.Available* property and if it is false, disable winsock related items in your program.

WinsockInfo property

unit msWSock;

applies to
TmsWinsock class

declaration

```
WinsockInfo: TmsWinsockInfo;
```

description
Allows you to retrieve the Winsock information. *TmsWinsockInfo* record is declared as:

```
TmsWinsockInfo = record
  Version : Word;
  HighVersion : Word;
  Description : ShortString;
  SystemStatus : ShortString;
  MaxSockets : Word;
  MaxUdpDg : Word;
end;
```

This record is a "Pascalized" version of TWSAData structure, which is declared in winsock.pas and comes with Delphi.

LocalName property

unit msWSock;

applies to

```
TmsWinsock class
```

declaration

```
LocalName: string;
```

description

allows you to get the local name of your computer.

LocalAddress property

unit msWSock;

applies to

`TmsWinsock class`

declaration

```
LocalAddress: string;
```

description

Contains the IP address of your computer, in dot separated numeric format.

msGetHostByName
msGetHostByAddr
msGetInAddr

msGetHostByName method

unit msWSock

applies to
TmsWinsock class

declaration

```
function msGetHostByName(const AName : string) : string;
```

description
Converts the domain name into the IP address in a dot separated numeric format.

msGetHostByAddr method

unit msWSock;

applies to
TmsWinsock class

declaration

```
function msGetHostByAddr(const AnAddr : string) : string;
```

description
Attempts to convert the IP address in a dot separated numeric format into the domain name. Can be used to perform the reverse lookup of your own IP address. For example:

```
MyReverseDomainName:=msWinsock.msGethostByAddr(msWinsock.LocalAddress);
```

msGetInAddr method

unit msWSock;

applies to

`TmsWinsock class`

declaration

```
function msGetInAddr(const AnAddr : string) : TInAddr;
```

description

Converts a dot separated numeric IP address into the standard winsock TInAddr structure.

Data
EnvelopeSender
EnvelopeRecipients
MailMessage

inherited from TmsClientSocket
Host
Port

inherited from TmsSocket
LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

MailMessage property

unit msSMTP;

applies to
TmsSMTPClient and TmsPOPClient components

declaration
```
MailMessage: TmsMessage;
```

description
Holds the data which will be sent using TmsSMTPClient component, or which has been retrieved using TmsPOPClient component. See the desription of TmsMessage component.

Login
Logout
SendMail
Send

inherited from TmsClientSocket
Connect

inherited from TmsSocket
Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

OnAddressRejected event

unit msSMTP;

applies to
TmsSMTPClient component

declaration
```
OnAddressRejected: TmsAddressRejectEvent;
```

description
Triggered when the address, supplied in the MailMessage.Recipient has been rejected by the server. You have a choice either cancel the sending process, or continue with other recipients. See the description of TmsAddressRejectEvent type for more information.

Login method (TmsSMTPClient)

unit msSMTP;

applies to
TmsSMTPClient component

declaration

```
Procedure Login;
```

description
Calls Connect method, and, after the connection is established, logs into the SMTP server, sending appropriate SMTP commands.

Logout method (TmsSMTPClient)

unit msSMTP;

applies to
TmsSMTPClient component

declaration
```
procedure Logout;
```

desription
Logs out of SMTP server, by sending *QUIT* command, and closes the connection, by calling Disconnect method..

SendMail method

unit msSMTP;

applies to
TmsSMTPClient component

declaration

```
procedure SendMail;
```

description
Transfers the data, contained in MailMessage property, to the SMTP server. You must be logged into the server, by calling Login method, if you wish to use this method. This method can be useful if you want to send multiple messages during the one session, during the single connection to the server. Don't forget to call Logout after you are done. If you wish to send single message during the single connection, use Send method.

Example:

```
msSMTPClient1.Login;
for i:=1 to TotalMessageToSend do
begin
  // assign the data to the msSMTPClient1.MailMessage here
  msSMTPClient1.SendMail;
end;
msSMTPClient1.Logout;
```

Send method

unit msSMTP;

applies to
TmsSMTPClient component

declaration

```
procedure Send;
```

description
Logs into the server, sends the MailMessage, and logs out of the server. Use this method if you wish to send single message during one connection to the server. If you wish to send multiple messages, see SendMail method, along with LogIn and Logout.

OnAddressRejected
OnPreparing
OnSending
OnSent

inherited from TmsClientSocket
OnConnecting

inherited from TmsSocket
OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

TmsAddressRejectEvent type

unit mssocket;

declaration

```
TmsAddressRejectEvent = procedure(Sender: TObject; const TheAddress,
ServerReply: string; var Proceed: boolean) of Object;
```

description

This is a type of OnAddressRejectedEvent event handler of TmsSMTPClient component. Some SMTP servers do checking of the recipients. If the server does not like a recipient by any reason, it will reject the address, returning the explanation. If this situation occurs, the reply string from the server will be in the *ServerReply* parameter, the rejected address - in the *TheAddress* parameter. In this event handler you have to set the value of *Process* parameter. If you set it to *True*, the component will attempt to continue sending other addresses to the server, if it is *False*, then the message will be not sent. Default value is *False*.

OnPreparing event

unit msSMTP;

applies to
TmsSMTPClient component

declaration
```
OnPreparing: TNotifyEvent;
```

description
This event handler will be called as soon as TmsSMTPClient component starts the preparation of the MailMessage. If the attachments are large, then it may take some time, and this event handler can be useful to display the message to the user that the message has been prepared for sending.

OnSending event

unit msSMTP;

applies to
TmsSMTPClient component

declaration
```
OnSending: TNotifyEvent;
```

description
You can use this event handler to display to the user that TmsSMTPClient component currently is sending the message.

OnSent event

unit msSMTP;

applies to
TmsSMTPClient component

declaration

```
OnSent: TNotifyEvent;
```

description
Triggered as soon as the mail message has been sent. Can be used to display the message about it to the user.

CurrentMessage
MailMessage
Password
TotalMessages
TotalOctets
UserName

inherited from TmsClientSocket
Host
Port

inherited from TmsSocket
LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

RetrieveAsStream method

unit msPOP

applies to
TmsPOPClient component

declaration
```
procedure RetrieveAsStream(Stream: TStream);
```

description
Important: Recommended for advanced users. Retrieves the current message into the *Stream* object. The stream must exist before calling this method. Retrieves the message, but does not fill in the properties of MailMessage property. Can be useful if you wish to process the data, contained in the mail message by youself, bypassing TmsMessage component.

We are planning to release the advanced MIME processing components, which will be able to process much more MIME types than current TmsMessage component. This method can be used in conjunction with it. TmsSMTPClient component also has the properties and a method which would allow you to send previously prepared data, without using MailMessage property, such as EnvelopeSender, EnvelopeRecipients, and Data.

CurrentMessage property

unit msPOP;

applies to
TmsPOPClient component

declaration
```
CurrentMessage: Integer;
```

description
Has meaning only after calling Login method. Points to the current message on the server. The number is 0-based. If there are no messages on the server, it will be set to -1, and any attempt to set it to some other number will not change anything (no error will be generated). If there are the messages and you attempt to assign the number which is more than TotalMessages-1, it will be set to the last message. If you attempt to set the negative number, it will be set to 0. No error will be generated.

Data property

unit msSMTP

applies to
TmsSMTPClient component

description
```
procedure Data: TStream;
```

declaration
Contains the data which should be actually transferred to to the server. The stream must exist before you assign it to this property.

Important: should be used only by advanced users. Usually, if you fill in the MailMessage property, you don't need to use *Data*, EnvelopeSender and EnvelopeRecipients properties. But if you don't like how TmsMessage component handles the things, then you can prepare the email message by yourself, and send it using TmsSMTPClient component. When you call Send, or SendMail, TmsSMTPClient component will check whether *Data* property has the value (it is not *nil*), and if it does, the data contained in MailMessage property will be ignored.

We are planning to provide advanced MIME component. It will be able to handle more advanced MIME content types than TmsMessage component. Then these properties will become much more useful.

See also EnvelopeSender, EnvelopeRecipients properties and RetrieveAsStream method of TmsPOPClient component.

TotalMessages property

unit msPOP

applies to
TmsPOPClient component

declaration
```
TotalMessages: Integer;
```

description
Has meaning only after calling Login method, and contains the number of total messages on the server. You can iterate through the messages by their numbers, by setting CurrentMessage property from 0 to TotalMessages-1.

TotalOctets property

unit msPOP

applies to
TmsPOPClient component

declaration
```
TotalOctets: Integer;
```

description
Has meaning only after calling Login method. Contains the size of all messages on the server.

UserName property (TmsPOPClient)

unit msPOP

applies to
TmsPOPClient component

declaration
```
UserName: string;
```

description
Name of the user on the POP server. Along with Password, makes it possible to log into the server.

Password property (TmsPOPClient)

unit msPOP

applies to
TmsPOPClient component

declaration
```
Password: string;
```

description
Password of the user account on the POP server. You also should supply UserName property in order to log into the server.

Delete
GetUIDL
GetSize
Login
Logout
Retrieve
RetrieveAsStream
RetrieveHeaders

inherited from TmsClientSocket
Connect

inherited from TmsSocket
Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

Login method (TmsPOPClient)

unit msPOP

applies to
TmsPOPClient component

declaration
```
procedure Login;
```

description
Connects to the server, by calling Connect method, logs into the server, sending the UserName and Password, and fills in TotalMessages and TotalOctets properties.

Delete method

unit msPOP

applies to
TmsPOPClient component

declaration
```
procedure Delete;
```

description
Marks current message for deletion. Message will be actually deleted only after calling Logout method. That's the way POP3 protocol works.

Logout method (TmsPOPClient)

unit msPOP

applies to
TmsPOPClient component

declaration
```
procedure Login;
```

description
Logs out from the server, by sending *QUIT* command, and disconnects from the server, using Disconnect method.

EnvelopeRecipients property

unit msSMTP

applies to
TmsSMTPClient component

declaration
```
EnvelopeRecipients: TStrings;
```

description
Contains the email addresses of recipients. Used only if Data property is assigned.

Important: should be used only by advanced users. Usually, if you fill in the MailMessage property, you don't need to use Data, EnvelopeSender and *EnvelopeRecipients* properties. But if you don't like how TmsMessage component handles the things, then you can prepare the email message by yourself, and send it using TmsSMTPClient component. When you call Send, or SendMail, TmsSMTPClient component will check whether Data property has the value (it is not *nil*), and if it does, the data contained in MailMessage property will be ignored.

We are planning to provide advanced MIME component. It will be able to handle more advanced MIME content types than TmsMessage component. Then these properties will become much more useful.

See also Data, EnvelopeSender properties and RetrieveAsStream method of TmsPOPClient component.

EnvelopeSender property

unit msSMTP

applies to
TmsSMTPClient component

declaration
```
EnvelopeSender: string;
```

description
Contains the emal address of sender. Used only if Data property is assigned.

Important: should be used only by advanced users. Usually, if you fill in the MailMessage property, you don't need to use Data, *EnvelopeSender* and EnvelopeRecipients properties. But if you don't like how TmsMessage component handles the things, then you can prepare the email message by yourself, and send it using TmsSMTPClient component. When you call Send, or SendMail, TmsSMTPClient component will check whether Data property has the value (it is not *nil*), and if it does, the data contained in MailMessage property will be ignored.

We are planning to provide advanced MIME component. It will be able to handle more advanced MIME content types than TmsMessage component. Then these properties will become much more useful.

See also Data, EnvelopeRecipients properties and RetrieveAsStream method of TmsPOPClient component.

OnMessageRetrieved event

unit msPOP

applies to
TmsPOPClient component

declaration
```
OnMessageRetrieved: TNotifyEvent;
```

description
Called when the component has finished the retrieving of the current message.

GetSize method

unit msPOP

applies to
TmsPOPClient component

declaration
```
function GetSize: Integer;
```

description
Returns the size of the current message.

GetUIDL method

unit msPOP

applies to
TmsPOPClient component

declaration
```
function GetUIDL: string;
```

description
Returns the UIDL of current message. UIDL is a unique string assigned to the message by POP server. Note, that not all servers support it. In this case this function will return blank string.

Retrieve method (TmsPOPClient)

unit msPOP

applies to
TmsPOPClient component

declaration
```
procedure Retrieve;
```

description
Retrieves current message. The component will process the data and fill in the instance of TmsMessage component, which is assigned to the the MailMessage property.

RetrieveHeaders method (TmsPOPClient)

unit msPOP

applies to
TmsPOPClient component

declaration
```
procedure RetrieveHeaders;
```

description
Retrieves headers of the current message, and fills following properties of MailMessage property:

CC,, Recipients, ReturnReceipt, Priority, CharSet, ContentType, Encoding, Headers, Sender, Subject

For example, if you want to retrieve the Date header, you can do

```
MyPOPClient.RetrieveHeaders;
s:=MyPOPClient.MailMessage.Headers.GetFieldBody('Date');
```

OnRetrievingMessage
OnMessageRetrieved

inherited from TmsClientSocket
OnConnecting

inherited from TmsSocket
OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

Attachments
Body
CharSet
ContentType
Encoding
Headers
Sender
Subject

BCC
CC
Recipients
ReturnReceipt
Priority

inherited from TmsCustomMessage
Attachments
Body
CharSet
ContentType
Encoding
Headers
Sender
Subject

OnRetrievingMessage event

unit msPOP

applies to
TmsPOPClient component

declaration
```
OnRetrievingMessage: TNotifyEvent;
```

descritpion
Triggered when TmsPOPClient component starts the retrieving of the message.

Assign
Clear
ExecuteAttachment
LoadFromStream
LoadFromFile
SaveAttachment
SaveToStream
SaveToFile

Newsgroups

inherited from TmsCustomMessage
Attachments
Body
CharSet
ContentType
Encoding
Headers
Sender
Subject

Attachments property

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
property Attachments : TsmAttList;
```

description
Contains a list of attachments. Use Attachments property to access the individual attachment. For example, the following code:

```
s:=MyMessage.Attachments[i].ContentType;
```

will assign to the string variable *s* the content type of the attachment no *i*.

TmsAttList type

unit msMsgCls;

declaration

```
  TmsAttList = class(TPersistent)
  private
    FList : TList;
    function GetItem(Index : Integer) : TmsAttItem;
    procedure SetItem(Index : Integer; Value : TmsAttItem);
    function GetCount : Integer;
    procedure ReadData(Reader : TReader);
    procedure WriteData(Writer : TWriter);
  protected
    procedure DefineProperties(Filer : TFiler); override;
  public
    constructor Create;
    destructor Destroy; override;

    function Add(Value : TmsAttItem) : Integer;
    function AddFile(const FileName : string) : Integer;
    procedure Clear;
    procedure Delete(Index : Integer);
    procedure Assign(Source : TPersistent); override;
    procedure Exchange(Index1,Index2 : Integer);
    property Count : Integer read GetCount;
    property Items[Index : Integer] : TmsAttItem read GetItem write SetItem;
default;
  end;
```

description

The list of TmsAttItem objects. This is the type of the Attachments property of TmsCustomMessage component.

TmsAttItem type

properties

unit

msMsgCls

declaration

```
TmsAttItem = class(TPersistent)
```

description

Encapsulates the Internet mail attachment. The contents of the attachment is contained in the Contents property, FileName property is the name of the file attachment. ContentType is the MIME type, it is assigned automatically, based on the extension of the FileName, and the data contained in the Windows registry, but you can still change it, if you assign it after assigning the *FileName*. The same applies to the ContentTransferEncoding property, which indicates the method of the encoding of the attachment.

Contents
ContentType
ContentTransferEncoding
FileName

Contents property

unit
msMsgCls;

applies to
TmsAttItem class

declaration
```
property Contents : TMemoryStream;
```

description
Contains the actual contents of the attachment. It can binary data, such as a contents of the jpeg picture, or text, if you are sending the text file.

ContentType property (TmsAttItem)

unit
msMsgCls;

applies to
TmsAttItem class;

declaration
```
property ContentType : ShortString;
```

description
This is a MIME type of the attachment. TmsAttItem retrieves the data from the registry, when you set the FileName property. This will be the default MIME type. But you can change it before sending the message. This value will appear in the *Content-Type* header when you are sending the message.

When receiving the message, *ContentType* property is set to the one contained in the header of the received message.

ContentTransferEncoding property

unit
msMsgCls;

applies to
TmsAttItem class

declaration
```
property ContentTransferEncoding : TEncoding;
```

description
Usually you don't have to use this property. Before sending the message TmsAttItem will try to figure out which value to choose to set this property based on the ContentType property. The following rules apply: if ContentType contains string 'TEXT', then *ContentTransferEncoding* will be etQP, otherwise - etBase64. In other words, for text attachments IMS will use Quoted-Printable encoding methods, for all other cases - Base64.

TEncoding type is declared in *msMsgCls* unit as follows:

```
TEncoding = (etNone, etBase64, etQP);
```

FileName property

unit
msMsgCls;

applies to
TmsAttItem class

declaration
```
property FileName : ShortString;
```

description
File name of the attachment. Presently the Internet Mail Suite supports only file attachments, if while decoding the TmsCustomMessage won't find the file name for the attachment, it will assign one itself. On the other hand, while sending the messages, you cannot add the attachment to the TmCustomMessage component without specifying the file name.

Body property

unit
msMsg

Applies to:
TmsCustomMessage component;

declaration
```
property Body : TStrings;
```

description
This is a body of the message itself. If typed the message in memo1 and want to send its contents as email message, or the Usenet article, the following code will do it:

```
msMessage.Body:=Memo1.Lines;
```

CharSet property

unit
msMsg;

applies to:
TmsCustomMessage component;

declaration
```
property CharSet : TmsCharSet;
```

description
This property will be useful for European and Japanese users. If this property is set to csISO8859, then the body will be encoded using Quoted-Printable method, if it is set to csISO2022jp (Japanese character set), the message headers, such as a subject and the names of the senders and recipients, will be encoded using Base64 method, as described in rfc1522.

TmsCharSet type

Unit
msMsgCls

Declaration
```
TmsCharSet = (csUSASCII,csISO8859,csISO2022jp);
```

Description
a type of CharSet property of the TmsCustomMessage component. Indicates which character set is to be used when encoding the message body.

ContentType property (TmsCustomMessage)

unit
msMsg;

applies to:
TmsCustomMessage component;

declaration:
```
property ContentType : ShortString;
```

description
This property makes sense only if Encoding property is set to meMIME.

You should not modify this property directly, it will be set automatically by TmsCustomMessage component. We did not make it read only because we wanted it to appear in the Object Inspector. It will help the users to understand better how it works.

But here is the additional information:

It will be auto generated based on the Body and Attachments properties. There are the following rules:

a) If the body of the message is empty:
if there is only one attachment, then *ContentType* is the same as the ContentType property of this attachment, if there are more than one attachments, then *ContentType* is *multipart/mixed*.

b) if the body of the message is not empty:
if there are no attachments, then *ContentType* is *text/plain*, otherwise - *multipart/mixed*. If you wish to send the html message, then set the property to *text/html* after you assigned to the Body property html content.

c) If the body of message is not empty, and the attachments are named as follows: _alt.xxx.. _alt.yyy where xxx and yyy are extensions (for example rtf, htm) ContentType will be *multipart/alternative*. This content type requires more explanation and you can read the special section dedicated to the multipart/alternative messages.

multipart/alternative messages

multipart/alternative type is relatively new and is used only by advanced mail programs, such as the latest versions of Microsoft Outlook/Outlook Express, and Netscape mail.

Here is how rfc1522 explains this content type::

```
The multipart/alternative type is syntactically identical to multipart/mixed,
but the semantics are different.  In particular,  each of the parts is an
"alternative" version of the same information.

Systems should recognize that the content of the various parts are
interchangeable.  Systems should choose the "best" type based on the local
environment and preferences, in some cases even through user interaction.  As
with multipart/mixed, the order of body parts is significant.  In this case,
the alternatives appear in an order of increasing faithfulness to the
original content. In general, the best choice is the LAST part of a type
supported by the recipient system's local environment.

Multipart/alternative may be used, for example, to send mail in a fancy text
format in such a way that it can easily be displayed anywhere
```

You can generate *multipart/alternative* messages using TmsCustomMessage component

1. Assign the *text/plain* version of the message to the Body property of TmsCustomMessage component;

2. Create alternative version(s) as file(s). Name files as _alt.xxx, where xxx can be, for example, html, rtf etc. The file name must but _alt, otherwise TmsCustomMessage component will generate multipart/mixed message; The file names in the encoded messages will be suppressed, so you don't have to worry about ugly file names. This name is used internally by TmsMessage components, just to guess that you are trying that to generate multipart/alternative message.

3. Attach _alt.xxx files as attachments. These files must follow the message body, i.e. the 0th attachment must be _alt.xxx file. Otherwise multipart/mixed file will be generated;

4. If you want to attach another files, do it after attaching all _alt.xxx files. In this case the Content-Type of the message itself will be multipart/mixed, but the alternative parts will be included as a multipart/alternative subsection;

PLEASE NOTE: The Internet Mail Suite does not provide any methods to create html or rtf files. It is your, programmer's responsibility to handle this.

How to decode multipart/alternative messages

1. Save the received message into the stream/file and call LoadFromStream or LoadFromFile method;

2. If the message does not contain attachments, first section of the message (TmsCustomMessage assumes that first section is text/plain) will be stored in the Body property, the alternative sections will be stored as attachments. The ContentType will be multipart/alternative; Names will be _alt.# where # is the zero-based number of the message; It is your responsibility to look up the Attachments[i].ContentType property and

according to its value (e.g. text/html, text/rtf) choose the section you wish to display;

3. If the message contains attachments, the ContentType will be *multipart/mixed*. The first section of the

multipart/alternative section (again, TmsCustomMessage will assume that first section is text/plain) will be stored into the Body property.The following alternative sections will be stored as attachments with names _alt.#, where # is the zero-based number of the message; Again, it is your responsibility to look up the Attachments[i].ContentType property and choose the section you wish to display; The 'real' attachments will follow the alternative sections.

Please use this feature with caution, if you need assistance or more information, send mail to archie@argosoft.com.

Encoding property

unit
msMsg;

applies to:
TmsCustomMessage component;

declaration
```
property Encoding : TmsMsgEncoding;
```

description
indicates which encoding method (MIME/Base64 or UUCode) to use when sending the messages with attachments. Default value is meMIME.

Please note, that if this property is set to meMIME the attachments will be encoded according to the MIME standards, if it is set to meUU, all attachments, including non-binary ones will be encoded using UUEncode method.

TmsEncoding type is delcared in the msMsgCls.pas unit as follows:

```
TmsMsgEncoding = (meMIME, meUU);
```

Headers property

unit
msMsg;

applies to:
TmsCustomMessage component;

declaration
```
property Headers : TmsHeaders;
```

description
A list of headers of the mail message or usenet article. Items should be in the format requested by rfc822. TmsCustomMessage component does not do any verification, except when sending a message. If you insert the headers which are automatically generated by TmsCustomMessage, they will be discarded Use this property only to set additional headers, which are not generated by TmsCustomMessage component.

For example, the following code will add the X-Mailer header to your outgoing message:

```
MyMessage.Headers.Add('X-Mailer: My Email Client');
```

Following headers are automatically generated by TmsCustomMessage component:

From: based on Sender property
Mime-Version: 1.0 hardcoded
Content-Type: based on ContentType property
Content-Transfer-Encoding: based on CharSet property, also on Body properties (see discussion about ContentType property)

Here are the headers which are generated, in addition to the ones above, by TmsMessage component:

To: based on Recipients property;
CC: based on CC property
Priority: based on Priority property
Return-Receipt: based on ReturnReceipt and Sender properties

And, one header, which is generated by TmsArticle component:
Newsgroups: based on Newsgroups property.

Sender property

unit
msMsg

applies to:
TmsCustomMessage component;

declaration
```
property Sender : TmsMailAddress;
```

description
Contains the address and name of sender. If sending the message, you have to fill in at least the Address of *Sender*, otherwise the message will be not sent.

Recipients property

unit
msMsgCls

applies to
TmsMessage component

declaration
```
property Recipients : TmsAddressList;
```

description
A list of recipients of mail message. Probably you will be setting only one recipient, but if you wish to address the message to many people, you can set multiple addresses. You can also use CC and BCC properties for these purposes. All three properties work the same way, because they all are of the same type: TmsAddressList. When sending the message, at list one address field either *Recipients* or *CC* lists must be set.

Example:
The following code adds the address 75231.330@compuserve.com to the list of recipients. There are two ways to do it:

```
procedure TForm1.Button1Click(Sender : TObject);
begin
  msMessage1.Recipients.AddAddress('75231.330@compuserve.com','');
end;
```

or

```
procedure TForm1.Button2Click(Sender : TObject);
var
  Addr : TmsMailAddress
begin
  Addr:=TmsMailAddress.Create;
  Addr.Address:='75231.330@compuserve.com';
  msMessage1.Add(Addr);
end;
```

CC property

unit
msMsg

applies to
TmsMessage component

declaration
```
property CC : TmsAddressList;
```

description
Contains a list of Carbon Copy recipients of the email message. The names listed here will be included in the outgoing message. If you want to send a message to someone, but not to include his/her name and address in the headers of the message, use BCC property.

Priority property

unit
msMsg

applies to
TmsMessage component;

declaration
```
property Priority : TPriority;
```

description
Indicates the priority of message. It will work only if receiver agent supports 'X-priority' header. ptLow priority corresponds to X-priority=5, ptNormal - X-priority=2, ptHigh - X-Priority=1. Default is ptNormal.

TPriority type is declared in msMsgCls.pas unit as follows:

```
TPriority = (ptLow,ptNormal,ptHigh);
```

ReturnReceipt property

unit
msMsg

applies to
TmsMessage component

declaration
```
property ReturnReceipt : boolean;
```

description
Is set to *true* if the message delivery receipt has been requested. Please keep in mind, that majority of server do not honor this request. Default is *false*.

Newsgroups property

unit msMsg;

applies to
TmsArticle component;

declaration

```
property NewsGroups : string;
```

description
Lists newsgroups where the article will be posted if you call msNNTP.PostArticle method, or, if the article has been retrieved from the server, contains the list of newsgroups where the message was cross posted. The names of the newsgroups would be comma-separated.

TmsHeaders class

methods

unit msMsgCls;

declaration

```
TmsHeaders = class(TStringList)
```

description

Implements Internet message headers. This is the type of the Headers property of TmsCustomMessage component. Contains the methods for parsing and retrieving field values. You can use these methods when you are looking for some specific values in the message headers.

Contains
GetFieldBody
GetMultilineFieldBody
Remove

Contains method

unit msMsgCls;

applies to
TmsHeaders class

declaration
```
function Contains(const ss : string; MatchCase : boolean) : boolean;
```

description
performs simple parsing of the headers and returns true if at least one header contains the string ss
If *MatchCase* is true, then search is case sensitive.

GetFieldBody method

unit
msMsgCls

applies to
TmsHeaders class

declaration
```
function GetFieldBody(FieldName : string) : string;
```

description
Returns the body of the header field, without the Field name itself, for example, if one of the lines in TmsHeaders class is

```
Return-Receipt-To: jdoe@domain.com
```

the function
```
GetFieldBody('Return-Receipt-To')
```

will return

```
jdoe@domain.com
```

The search is not case sensitive, you also don't have to pass ':', *GetFieldBody* function checks for it and adds ':' if it not present.

To retrieve multiline fields, use GetMultiLineFieldBody method.

GetMultiLineFieldBody method

unit
msMsgCls

applies to
TmsHeaders class

declaration
```
procedure GetMultiLineFieldBody(FieldName : string; FieldBody : TStringList);
```

description
Parses the strings contained in the instance of TmsHeaders class and returns the lines it finds related to the *FieldName* string. For example:

assume the message contains the following headers:

```
CC:  John Doe <jdoe@domain.com>, Argosoft <argosoft@interlog.com>
Return-Receipt-To: ao149@torfree.net
CC: janedoe@aol.com (Jane Doe)
```

the call
```
GetMultiLineFieldBody('CC',TempStrings);
```

will fill the Strings property of TempStrings by the following information:

Strings[0] John Doe <jdoe@domain.com>
Strings[1] Argosoft <argosoft@interlog.com>
Strings[2] janedoe@aol.com (Jane Doe)

i.e. all separators will be removed and the method will return the date you can process using the the procedures from msUtils unit.

Remove method

unit
msMsgCls

applies to
TmsHeaders class

declaration
```
procedure Remove(FieldName : string);
```

description
Locates and removes all headers *FieldName* from the TmsHeaders class.

TmsMailAddress class

unit msMsgCls;

declaration

```
TmsMailAddress = class(TPersistent)
private
  FName : ShortString;
  FAddress : ShortString;
public
  procedure Assign(Value : TmsMailAddress);
  procedure Edit; virtual;
  procedure Clear;
published
  property Name : ShortString read FName write FName;
  property Address : ShortString read FAddress write FAddress;
end;
```

description

TmsMailAddress contains just two properties: *Address* and *Name*, where *Address* is the Internet email address and *Name* is the name of a sender, recipient or the member of CC or BCC list.

Usually, when assigning the values, *Name* can be omitted, but *Address* has to be entered.

Subject property

unit
msMsg;

applies to
TmsCustomMessage component

declaration
```
property Subject: string;
```

description
A subject line of the mail message or newsgroup article.

Assign method

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure Assign(Value : TPersistent);
```

description
If *Value* is not TmsCustomMessage, raises exception, if *Value* is TmsCustomMessage then copies full contents of *Value* to the current instance.

Clear method (TmsCustomMessage)

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure Clear;
```

description
Clears all the contents of the TmsCustomMessage component, resets the counts of all lists to zero and disposes all allocated memory. If you wish to send different messages in a row, you have to call this method, otherwise, if you add new objects to the list properties, old values will be not cleared. For example, if you are sending the message with the file attached, then you want to send another message to another recipient with another attachment, you have to call this method after sending first message, otherwise, if you use Attachments.AddFile method, old attachment will be not discarded and both attachments will be sent to second recipient.

ExecuteAttachment method

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure ExecuteAttachment(Index : Integer);
```

description
Executes the attachment # Index. Calls *ShellExecute* windows API function, which checks if there is some action associated with 'open'-ing of the attached file and executes it. The files are temporarily saved in the temporary directory and are deleted when destroying TmsCustomMessage component.

LoadFromStream method

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure LoadFromStream(AStream : TStream);
```

description
Loads the raw email message, received from server, decodes attachments and fills in the fields of TmsCustomMessage object. If an error occurs while decoding the data, the entire original message will be saved as the next available attachment and in the body of message will appear the warning. If you have the raw email message in the file, use LoadFromFile method, which, internally uses *LoadFromStream* method.

LoadFromFile method

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure LoadFromFile(const FileName : string);
```

description
Loads the raw email message received from server, decodes attachments and fills in the fields of TmsCustomMessage object. If an error occurs while decoding the data, the entire original message will be saved as the next available attachment and in the body of message will appear the warning. If you have the raw email message in the stream, use LoadFromStream method.

SaveAttachment method

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure SaveAttachment(Index : Integer; const Path : string);
```

description
Saves the attachment # Index (zero based) into the specified directory. If attachment # Index does not exist, EListError exception will be raised. To the file will be assigned the name Attachments[Index].FileName.

SaveToStream method

unit
msMsg

applies to
TmsCustomMessage component

declaration
```
procedure SaveToStream(AStream : TStream);
```

description
prepares rfc822 (ready-to-go) message, i.e. encodes all attachments, prepares all necessary mail headers and saves the output into the stream. If you wish to save the message into the file, use SaveToFile method.

SaveToFile method

unit
msMsg;

applies to
TmsCustomMessage component;

declaration
```
procedure SaveToFile(const FileName : string);
```

description
prepares rfc822 (ready-to-go) message, i.e. encodes all attachments, prepares all necessary mail headers and saves the output into the file with name *FileName*. If you wish to save the message into the stream, use SaveToStream method.

inherited from TmsCustomMessage
Assign
Clear
ExecuteAttachment
LoadFromStream
LoadFromFile
SaveAttachment
SaveToStream
SaveToFile

BCC property

unit
msMsg

applies to
TmsMessage component

declaration
```
property BCC : TmsAddressList;
```

description
Contains a list of Blind Carbon Copy recipients of the message. Blind Carbon Copy recipients are almost the same as Carbon Copy recipients (property CC), only difference is the names listed in *BCC* will not appear in the message. Because of this, *BCC* property of received messages is always empty.

TmsAddressList class

unit msMsgCls;

declaration

```
  TmsAddressList = class(TPersistent)
  private
    FList : TList;
    function GetItem(Index : Integer) : TmsMailAddress;
    procedure SetItem(Index : Integer; Value : TmsMailAddress);
    function GetCount : Integer;
    procedure ReadData(Reader : TReader);
    procedure WriteData(Writer : TWriter);
  protected
    procedure DefineProperties(Filer : TFiler); override;
  public
    constructor Create;
    destructor Destroy; override;
    function Add(Value : TmsMailAddress) : Integer;
    function AddAddress(const TheAddress, TheName : string) : Integer;
    procedure Clear;
    procedure Delete(Index : Integer);
    procedure Assign(Value : TmsAddressList);
    property Count : Integer read GetCount;
    property Items[Index : Integer] : TmsMailAddress read GetItem write
SetItem; default;
  end;
```

description

List of TmsMailAddress objects. Contains methods and properties for handling this list. To add new mail address to the list use *Add* or *AddAddress* method, also contains routines for clearing, deleting and maintaining the addresses. This is the type of Recipients, CC and BCC properties of TmsMessage component.

inherited from TmsCustomMessage
Assign
Clear
ExecuteAttachment
LoadFromStream
LoadFromFile
SaveAttachment
SaveToStream
SaveToFile

Authorization
Boundary
FileName
Headers
InStream
OutStream
Password
PostContentType
Proxy
ProxyAuthorization
ProxyPassword
ProxyUserName
URL
UserAgent
UserName

Inherited from TmsClientSocket
Host
Port

inherited from TmsSocket
LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

Authorization property (TmsHTTPClient)

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
Authorization: boolean;
```

description
Some pages on the web require authorization. You have to set this property to *True*, also set UserName and Password properties in order to access these resources.

TmsHTTPClient component supports only basic authorization.

UserName property (msHTTPClient)

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
UserName: string;
```

description
Makes sense only of Authorization is *True*, and contains the user name which will be used to access the web resource. Password property aslo must be set.

Password property

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
Password: string;
```

description
Makes sense only of Authorization is *True*, and contains the password which will be used to access the web resource. UserName property aslo must be set.

Login method (TmsFTPClient)

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure Login;
```

description
Connects to the server, by calling Connect method, and logs into it by sending the data contained in UserName and Password properties.

Boundary property

unit msHTTP;

applies to
TmsHTTPCleint component

declaration
```
Boundary: string;
```

description
Should be used only when you are posting the data with the PostContenType *multipart/form-data*. Since the data is multipart, TmsHTTPClient component must transmit it with the *Content-Type* header. Note, that IMS does not provide the tools for generating the *multipart/form-data* content.

FileName property (TmsHTTPClient)

unit msHTTP;

applies to
TmsHTTPClient component

declaration
```
FileName: string;
```

description
This is a read-only property, which will get it's value after you set URL property. It will contain the File Name of the retrieved resource, to simplify you the way of saving it.

PostContenType property

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
PostContentType: string;
```

description
Content Type of the posted data. It can have two values: *application/x-www-form-urlencoded*, which is the default and is used for posting simple data, such as a form submissions, or *multipart/formdata*, which is used for more complex HTTP operations, such as, file uploads and so on.

URL property

unit
msHTTP

applies to
TmsHTTPClient component

declaration
```
property URL : string;
```

description
URL of the document you want to retrieve using TmsHTTPClient component. It must start with *http://*. After you specify URL you can call Get method and retrieve the document or file. URL is also an address of a resource you want to apply the posted data, if you are planning to use Post method.

examples of URL are:
http://www.microsoft.com
http://www.interlog.com/~argosoft/ims.gif

Get method

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
function Get: Integer;
```

description
Connects to the server, sends GET HTTP request, retrieves the data sent from the server, and closed the connection. Received data will be saved in InStream property. Return value contains the reply code sent by server.

Post method

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
function Post: Integer;
```

description
Applies HTTP *Post* method to the resource indicated in URL. Can be used for posting form data or sending the files to the server, or for uploading files, if it is supported by servers.

You should supply the URL to which the *Post* method will be applied and save the data to be transmitted to OutStream property.

Content-type of data by default is *application/x-www-form-urlencoded*. If you wish to send more complex data, you should generate it yourself, store into the OutStream and change the PostContentType property to *multipart/form-data*, as described in rfc1867.

For example, to send the form data:

```
procedure SendFormData;
var
  Buf : array[0..255] of Char;
begin

StrCopy(@Buf,'email=archie@argosoft.com&realname=Archie&subscribe=subscribe')
;
  agHTTP1.OutStream.Write(Buf,StrLen(@Buf));
  agHTTP1.URL:='http://www.argosoft.com/cgi-bin/guestbook.cgi';
  agHTTP1.Post;
end;
```

Headers property (TmsHTTPClient)

unit msHTTP;

applies to
TmsHTTPClient component

declaration
```
Headers: TmsHeaders;
```

description
Contains the headers for the data returned by server. Since it's type is TmsHeaders, you can easily get the values of different headers. For example, if you wish to get Server header, you can do

```
s:=MyHTTPClient.Headers.GetFieldBody('Server');
```

UserAgent property

unit
msHTTP

applies to
TmsHTTPClient component

declaration
```
property UserAgent : string;
```

description
Will be sent to the server, as an *User-Agent* header, when calling the methods of TmsHTTPCleint components. This property is not required, but allows you to identify your program to the server.

Proxy property (TmsHTTPClient)

unit
msHTTP

applies to
TmsHTTPClient component

declaration
```
property Proxy : string;
```

description
Specifies an address of proxy server. If your application does not use proxy server, you should leave this property blank. If you use proxy server, you also must set a Port property to the port of the proxy server. **Attention users of IMS 1.XX** - this is one of the major changes, you should not specify the proxy server in the domain:port format anymore. You are setting *Proxy* to the address of the proxy server, *Port* - to the port of the proxy server (not to the port of actual host).

If proxy server requires authorization, you should set ProxyAuthorization property to True, also, ProxyUserName and ProxyPassword properties.

ProxyUserName property

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
ProxyUserName: string;
```

description
Used only if Proxy property is not blank, and ProxyAuthorization property is True. You should use this property if you are using proxy server which requries authorization. ProxyPassword property should be also set. IMS supports only basic authorization.

ProxyAuthorization property

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
ProxyAuthorization: boolean;
```

description
Should be set to *True* if you are using proxy server which requires authorization. ProxyUserName and ProxyPassword properties must be also set.

ProxyPassword property

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
ProxyPassword: string;
```

description
Used only if Proxy property is not blank, and ProxyAuthorization property is True. You should use this property if you are using proxy server which requries authorization. ProxyUserName property also must be set.

InStream property

unit
msHTTP

applies to
TmsHTTPClient component

declaration
```
InStream: TStream;
```

description
That's where the received data will be stored if you use Get, or Post methods of TmsHTTPClient component. You can save the retrieved data to file, e.g. calling MyHTTPClient.InStream.SaveToFile() method...

OutStream property

unit
msHTTP

applies to
TmsHTTPClient component

declaration
```
OutStream: TStream;
```

description
Contains the data to be sent to the HTTP server using Post method. You have to store data you are going to post in this stream, prior to calling Post method.

For example:

```
const
  p : PChar = 'firstname=john&lastname=doe';
.
  MyHTTPClient.OutStream.Write(p^,StrLen(p));
  MyHTTPClient.Post;
```

Head
Get
Post

inherited from TmsClientSocket
Connect

inherited from TmsSocket
Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

Head method

unit msHTTP

applies to
TmsHTTPClient component

description
```
function Head: Integer;
```

declaration
The same as Get method, but it will not retrieve the data itself, the data will be returned only into the Headers property. This method sends HEAD request, instread of GET request. And, as a reply, the server returns only headers. The function will return the reply code from the server.

OnSendingRequest
OnRequestSent

inherited from TmsClientSocket
OnConnecting

inherited from TmsSocket
OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

OnRequestSent event

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
OnRequestSent: TNotifyEvent;
```

description
Is called after connecting to the server, and sending the http request, such as GET, HEAD, or POST plus full post data.

OnSendingRequest event

unit msHTTP

applies to
TmsHTTPClient component

declaration
```
OnSendingRequest: TNotifyEvent;
```

description
Is called after connecting to the server, but before sending HTTP request.

Login method (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure Login;
```

description
Connects to the server, specified in Host, using Connect method, also, attempts to retrieve the overview format, which will be later used when you call GetOverview method.

RetrieveNewsgroupList method

unit msNNTP

applies to
TmsNNTPClient component

declaration

```
procedure RetrieveNewsgroupList;
```

description
Retrieves a list of available newsgroup from the server. You must be logged into the server before calling this method. Fills the NewsgroupList property. Since the operation of the retrieving the newsgroup lists can be lengthy, retrieved data can be saved into the file using SaveNewsgroupList method, and retrieved later using LoadNewsgroupList method.

Newsgroup property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
Newsgroup: string;
```

description
Set this property to the name of the newsgroup you are interested in. You must be logged onto the server, using Login method. After setting this newsgroup, FirstArticle, LastArticle, and TotalArticles properties will get the values.

FirstArticle property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
FirstArticle: Integer;
```

description
Read-Only property. Has meaning only after logging into the server, using Login method, and selecting the current newsgroup, by setting Newsgroup property. Contains the number of first available article in the newsgroup.

LastArticle property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
LastArticle: Integer;
```

description
Read-Only property. Has meaning only after logging into the server, using Login method, and selecting the current newsgroup, by setting Newsgroup property. Contains the number of last available article in the newsgroup.

GetOverview method

unit
msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure GetOverview(FirstMsgNo, LastMsgNo: Integer);
```

description
You should be logged into the server, and the newsgroup must be selected before calling this method. It retrieves the information of messages in the *FirstMsgNo* - *LastMsgNo* range and fills in the Overview property,

RetrieveArticleByNumber method

unit msNNTP

applies to
TmsNNTPClient component

declaration

```
procedure RetrieveArticleByNumber(Number: Integer);
```

description
Works the same way as Retrieve method, but instead of current article it retrieves the article specified in the *Number* parameter.

Article property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
Article: TmsArticle;
```

description
The usage of this property is analogous to MailMessage property of TmsSMTPClient and TmsPOPClient properties.

This structure will be filled in automatically after you call Retrieve, RetrieveArticleByNumber, or RetrieveArticleByID method. You also have to fill in the properties of this structure before calling Post method, for posting the article.

RetrieveArticleByID method

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure RetrieveArticleByID(const ID: string);
```

description
Retrieves the article by it's unique MessageID. You must be logged into the server, but you don't have to select a newsgroup, since ID is unique.

Post method (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure Post;
```

description
Post the article, which has been previously prepared using Article property to the newsgroups specified in the Newsgroups property of the Article. You must be logged onto the server in order to call this function.

Logout method (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure Logout;
```

description
Logs out from the NNTP server, by sending *QUIT* command, and disconnects from it, by calling Disconnect method.

Authorization property (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
Authorization: boolean;
```

description
Certain servers require the authorization to access certain information. You should set this property to *True*, also, set UserName and Password properties properly if you are dealing with the server which requires authorization.

UserName property (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
UserName: string;
```

description
Certain servers require the authorization to access certain information. If you are dealing with this kind of server, you should set Authorization property to true, also, assign to this property, also to Password property, proper values.

Password property (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
UserName: string;
```

description
Certain servers require the authorization to access certain information. If you are dealing with this kind of server, you should set Authorization property to true, also, assign to this property, also to UserName property, proper values.

Article
Authorization
BOA
CurrentArticle
EOA
FirstArticle
LastArticle
Newsgroup
NewsgroupList
Overview
Password
TotalArticles
UserName

inherited from TmsClientSocket
Host
Port

inherited from TmsSocket
LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

Retrieve method (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

description
```
procedure Retrieve;
```

declaration
Retrieves current article. You must be logged onto the server, and Newsgroup must be selected. Retrieves the article, number of which is set in CurrentArticle property. The article will be placed in Article property.

BOA property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
BOA: boolean;
```

description
Read-Only property. It has meaning after you have logged into the server, calling Login method, and selected the newsgroup, by setting Newsgroup property. It is true if CurrentArticle is set to the first article in the newsgroup. This property allows you to iterate between the articles using First, Next, Last methods.

CurrentArticle property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
CurrentArticle: Integer;
```

description
Has meaning only after logging on on the server (Login method) and selecting the newsgroup (Newsgroup property). Points to the current selected article in the newsgroup. It is between FirstArticle and LastArticle... Keep in mind, that, usually, not all articles between the *FirstArticle* and *LastArticle* range is avaliable. Call of Next method will ensure that the pointer is reset to the next available article.

First method

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure First;
```

description
Has meaning only after logging on on the server (Login method) and selecting the newsgroup (Newsgroup property). It sets the CurrentArticle pointer to the first available article in the newsgroup.

Next method

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure Next;
```

description
Has meaning only after logging on on the server (Login method) and selecting the newsgroup (Newsgroup property). Resets the CurrentArticle property to the next available article in the selected newsgroup.

Last method

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure Last;
```

description
Has meaning only after logging on on the server (Login method) and selecting the newsgroup (Newsgroup property). Resets the pointer to the current article in the selected newsgroup (CurrentArticle) to the last available article in the newsgroup.

TotalArticles property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
TotalArticle: Integer;
```

description
Read-Only property. Has meaning only after logging into the server, using Login method, and selecting the current newsgroup, by setting Newsgroup property. Contains the number of available articles in the newsgroup. Keep in mind, that does not always equal to LastArticle-FirstArticle+1, because usually not all articles with the numbers between *FirstArticle*...*LastArticle* are available in the newsgroup.

EOA property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
EOA: boolean;
```

description
Read-Only property. It has meaning after you have logged into the server, calling Login method, and selected the newsgroup, by setting Newsgroup property. It is true if CurrentArticle is set to the last article in the newsgroup. This property allows you to iterate between the articles using First, Next, Last methods.

TmsNewsgroupItem class

unit msNNTP

declaration

```
TmsNewsgroupItem = class
protected
  procedure ParseString(const s: string);
  function ToString: string;
public
  Name: string;
  FirstArticle: LongInt;
  LastArticle: LongInt;
  Flag: Char;
end;
```

description

Each item of TmsNewsgroupList class is of this type. *Name* property contains the name of the newsgroup, *FirstArticle* and *LastArticle* - numbers of first and last available articles in the newsgroup, *Flag* can have following values:

y - posting to the newsgroup is allowed
n - posting to the newsgroup is not allowed
m - the newsgroup is moderated.

NewsgroupList property

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
NewsgroupList: TmsNewsgroupList;
```

description
Contains the list of available newsgroups on the server. You should call RetrieveNewsgroupList method to retrieve this list. You also can save the list to the file by calling SaveNewsgroupList method for later retrieval it using LoadNewsgroupList method.

See the topic related to TmsNewsgroupList for more information.

Overview property

unit
msNNTP

applies to
TmsNNTPClient component

declaration
```
Overview: TmsOverviewList;
```

description
Contains a list of the information related to the articles in the newsgroup. You have to log into the server, calling Login method, then slect a newsgroup, by setting Newsgroup property, and call GetOverview method. Overview list will be filled with the useful information, containing a list of article-related data. The demo program, nntpdemo, demonstrates the usage of this property, see also the description of TmsOverviewList and TmsOverviewItem classes.

GetOverview method uses extended NNTP commands, which are not a part of standard required NNTP implementation, so, some servers may not support it. In this case you should use GetHeaders method to get the information about the messages.

Example:

Say you retrieved the *Overview* after calling GetOverview method, and wish to find the article which with the subject containing the string *Honda Civic*. Let's attempt to find out whether any of the articles contain this string:

```
var
  Found: boolean;
  i: Integer;
begin
  Found:=false; i:=0;
  while (not Found) and (i<MyNNTPClient.Overview.Count) do
  begin
    Found:=LowerCase(MyNNTPClient.Overview[i].Subject)='honda civic';
    if not Found then Inc(i);
  end;
```

If after completing this loop *Found* is *true*, then there is an article cotaining the string we were looking for. Now we can retrieve it, either by setting the CurrentArticle and calling Retrieve method, like this

```
MyNNTPClient.CurrentMessage:=MyNNTPClient.Overview[i].ArticleNo;
MyNNTPClient.Retrieve;
```

or by calling

```
MyNNTPClient.RetrieveMessageById(MyNNTPClient.Overview[i].MessageId);
```

SaveNewsgroupList method

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure SaveNewsgroupList(const FileName: string);
```

description
Saves the content of NewsgroupList property into the file named *FileName*. The list can be retrieved later using LoadNewsgroupList method.

LoadNewsgroupList method

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure LoadNewsgroupList(const FileName: string);
```

description
Loads the list of the newsgroups from the file named *FileName*. The list must be previously saved using SaveNewsgroupList method. Loaded data will be saved in NewsgroupList property.

TmsNewsgroupList class

unit msNNTP

declaration

```
  TmsNewsgroupList = class
  private
    FList : TList;
    function Get(Index : Integer) : TmsNewsgroupItem;
    procedure Put(Index : Integer; Value : TmsNewsgroupItem);
    function GetCount : Integer;
  protected
    function PickAPart(s : string; Position : Integer) : string;
  public
    constructor Create;
    destructor Destroy; override;
    function Add(Value : TmsNewsgroupItem) : Integer;
    procedure Delete(Index: Integer);
    procedure SaveToFile(const FileName : string);
    procedure SaveToStream(Stream : TStream);
    procedure LoadFromFile(const FileName : string);
    procedure LoadFromStream(Stream : TStream);
    procedure Clear;
    property Items[Index : Integer] : TmsNewsgroupItem read Get write Put;
default;
    property Count : Integer read GetCount;
  end;
```

description

Contains a list of TmsNewsgroupItem objects. Each item of it contains the information about the name of the newsgroup, also number of the first and last available articles in the group, also flag, which indicates whether posting is allowed, and if the group is moderated.

This is a type of NewsgroupList property of TmsNNTPClient component.

TmsOverviewList class

unit msNNTP

declaration

```
  TmsOverviewList = class
  private
    FList: TList;
    function Get(Index: Integer): TmsOverviewItem;
    procedure Put(Index: Integer; Value: TmsOverviewItem);
    function GetCount: Integer;
  public
    constructor Create;
    destructor Destroy; override;
    procedure Clear;
    function Add(Value: TmsOverviewItem): Integer;
    property Count: Integer read GetCount;
    property Items[Index: Integer]: TmsOverviewItem read Get write Put;
default;
  end;
```

description

This is a type of the Overview property of TmsNNTPClient component. It is a list of TmsOverviewItem objects.

TmsOverviewItem class

unit msNNTP

declaration

```
  TmsOverviewItem = class
  private
    FFmt: TStringList;
    function PickData(const Header,Data: string): string;
    function PickMsgNo(const s: string): Integer;
    procedure ParseString(const s: string);
  public
    ArticleNo: LongInt;
    Subject: string;
    SenderName: string;
    SenderAddress: string;
    Date: TDateTime;
    TimeZone: ShortString;
    MessageID: string;
    References: string;
    Bytes: LongInt;
    Lines: Integer;
    constructor Create(Fmt: TStringList);
    destructor Destroy; override;
  end;
```

declaration

Each item of TmsOverviewList class, *TmsOverviewItem* object contains following useful information about the article:

ArticleNo - Article number. Can be used for retrieving the article by setting CurrentArticle to this number and calling Retrieve;

Subject - Subject line of the article;

SenderName - Name of the sender;

SenderAddress - Email address of the sender;

Date - Date when the message was posted, in Delphi TDateTime format;

TimeZone - String representation of the time zone of the sender, according to rfc 822;

MessageID - Unique string which identifies the article. Can be used for retrieving the article using RetrieveArticleByID;

References - MessageIDs of the articles the current article refers to;

Bytes - Size of the article in bytes;

Lines - Number of lines in the article

Note, that some of these values can be not available. In this case string values will be blank, and numeric values will be set to -1.

First
GetOverview
Last
LoadNewsgroupList
Login
Logout
Next
Post
Retrieve
RetrieveArticleByID
RetrieveArticleByNumber
RetrieveHeader
RetrieveHeaders
RetrieveNewsgroupList
SaveNewsgroupList

inherited from TmsClientSocket
Connect

inherited from TmsSocket
Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

RetrieveHeader method

unit msNNTP

applies to
TmsNNTPClient component

declaration

```
procedure RetrieveHeader(const Header: string): string;
```

description
Retrieves the data of the header, specified in the *Header* parameter, of the CurrentArticle.

Example

If you wish to retrieve the subject of the current article, the following should do it:

```
TheSubject:=MyNNTPClient.RetrieveHeader('Subject');
```

RetrieveHeaders method (TmsNNTPClient)

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
procedure RetrieveHeaders;
```

description
Retrieves headers of the current article, and fills following properties if Article property:

Newsgroups, CharSet, ContentType, Encoding, Headers, Sender, Subject

For example, if you want to retrieve the Organization header, you can do

```
MyPOPClient.RetrieveHeaders;
s:=MyPOPClient.MailMessage.Headers.GetFieldBody('Oragnization');
```

OnOverviewItemRetrieved
OnNewsgroupItemRetrieved

inherited from TmsClientSocket
OnConnecting

inherited from TmsSocket
OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

OnNewsgroupItemRetrieved event

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
OnNewsgroupItemRetrieved: TmsNewsgroupItemRetrievedEvent;
```

description
Triggered after next item of the list of the newsgroup has been retrieved, when RetrieveNewsgroupList method is running. It will allow you to display the list of retrieved newsgroups as the information arrives. But, keep in mind, that it will slow down your program.

example

```
procedure TNNTPForm.msNNTPClient1NewsgroupItemRetrieved(Sender: TObject;
   NewsgroupItem: TmsNewsgroupItem);
begin
  GroupsListBox.Items.Aded(NewsgroupItem.Name);
end;
```

TmsNewsgroupItemRetrievedEvent type

unit msNNTP

declaration

```
  TmsNewsgroupItemRetrievedEvent = procedure(Sender: TObject;
     NewsgroupItem: TmsNewsgroupItem) of Object;
```

description

This is a type of OnNewsgroupItemRetrieved event handler of TmsNNTPClient component.

OnOverviewItemRetrieved event

unit msNNTP

applies to
TmsNNTPClient component

declaration
```
OnOverviewItemRetrieved: TmsOverviewItemRetrievedEvent;
```

description
Triggered after the next item of the overview has been retrieved, when GetOverview method is running. It will allow you to display the information about retrieved items information arrives.

TmsOverviewItemRetrievedEvent type

unit msNNTP

declaration

```
  TmsOverviewItemRetrievedEvent = procedure(Sender: TObject;
     OverviewItem: TmsOverviewItem) of Object;
```

description

This is a type of OnOverviewItemRetrieved event of TmsNNTPClient component.

UserName property (TmsFTPClient)

unit msFTP

applies to
TmsFTPClient component

declaration
```
UserName: string;
```

description
Name of the user on the FTP server. Along with the Password, makes it possible to log into the server.

Password property (TmsFTPClient)

unit msFTP

applies to
TmsFTPClient component

declaration
```
Password: string;
```

description
Contains the password for the user specified in UserName property.

Logout method (TmsFTPClient)

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure Logout;
```

description
Logs out of the FTP server, by sending *QUIT* FTP command, and closes the connection to the server by calling Disconnect method.

StoreFile method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure StoreFile(const LocalFilePath, RemoteFilePath: string);
```

description
Sends the local file, named *LocalFilePath* to the server, and names it according to the parameter specified in *RemoteFilePath*. You must be logged into the server, by calling Login method, in order to use this method.

RetrieveFile method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure RetrieveFile(const RemoteFilePath, LocalFilePath: string);
```

description
Retrieves the file, named *RemoteFilePath*, from the remote FTP server and stores it as *LocalFilePath*. You mus be logged into the server, using Login method, in order to use this method.

GetDirList method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure GetDirList;
```

description
Retrieves the information about the current directory on the FTP server, and fills in DirList property. You must be logged onto the server, using Login method, before calling this method.

DirList property

unit msFTP

applies to
TmsFTPClient component

declaration
```
DirList: TmsFTPDirList;
```

description
Contains the data about the the structure of the directory on the FTP server. Gets filled in after calling GetDirList method.

Each item member of the DirList property corresponds to either file, directory name, or, so called *link* on the FTP server. Link is kind of pointer either to the file, or directory in the different directory. *Kind* field of this object indicates whether it is a file *(fkFile*), Directory (*fkDirectory*) or Link (*fkLink*). *FileName* contains the name of the file or directory and is blank for the link, *Size* contains the size of the file, and has no meaning when *Kind* is directory or link. *Date* - date/time stamp of the directory or the file, *LinkPtr* - contains the name of the linkedh file or directory.

In some cases TmsFTPClient component cannot fill in *DirList* property, since, as mentioned in the discussion of ServerType property, the output sent by the server is not standardized, you still can get the file names and file sizes, using GetFileList and GetFileSize methods, or, if you wish to retrieve the directory structure 'as is', you can use GetDirectoryOutput method.

Here is an example, which generates the report based on the *DirList* property:

```
procedure MakeReport;
var
  i : Integer;
  f : TextFile;
begin
  AssignFile(f,'report.txt');
  Rewrite(f);
  for i:=0 to msFTPClient1.DirList.Count-1 do
  begin
    case msFTPClient1.DirList[i].Kind of
      fkFile : WriteLn(f,'File');
      fkDirectory : WriteLn(f,'Directory');
      fkLink : WriteLn(f,'Link');
    end;
    WriteLn(f,'Name: ',msFTPClient1.DirList[i].FileName);
    WriteLn(f,'Size: ',msFTPClient1.DirList[i].Size);
    WriteLn(f,'Date: ',DateTimeToStr(msFTPClient1.DirList[i].Date);
    if msFTPClient1.DirList[i].Kind=fkLink then
      WriteLn(f,'Link points to: ',msFTPClient1.DirList[i].LinkPtr);
  end;
  CloseFile(f);
end;
```

CurrentDirectory
DirList
PassiveMode
Password
Proxy
ProxyType
ServerType
TransferType
UserName

inherited from TmsClientSocket
Host
Port

inherited from TmsSocket
LogFileName
TimeOut *

inherited from TmsSocketBase
SleepTime *
Socket

CurrentDirectory property

unit msFTP

applies to
TmsFTPClient component

declaration
```
CurrentDirectory: string;
```

description
Contains the name of the current directory on the server. If you assign to this property another directory, current directory will be changed. The "setter" of this property calls ChangeDirectory method.

ChangeDirectory method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure ChangeDirectory(const Path: string);
```

description
Selects the current directory on the FTP server. You must be logged into the server before calling this method. You also can use CurrentDirectory property to change the current directory.

PassiveMode property

unit msFTP

applies to
TmsFTPClient component

declaration
```
PassiveMode: boolean;
```

description
Usually, when FTP client (in this case, *TmsFTPClient* component) initiates the data transfer with the server, it are sending the information about the address and the availabe port of the local computer, and then waits until the server connects to the specified address. This is the normal mode and is called active mode. If you select passive mode (set the property PassiveMode to true), then the server will not initiate the connection itself, it will wait until the client does it. This option is sometimes required by certain proxy servers. Default value of this property is *False* (active mode).

Proxy property (TmsFTPClient)

unit msFTP

applies to
TmsFTPClient component

declaration
```
Proxy: string;
```

description
Specifies an address of proxy server. If your application does not use proxy server, you should leave this property blank. If you use proxy server, you also must set a Port property to the port of the proxy server. **Attention users of IMS 1.XX** - this is one of the major changes, you should not specify the proxy server in the domain:port format anymore. You are setting *Proxy* to the address of the proxy server, *Port* - to the port of the proxy server (not to the port of actual host).

In FTP, you also will need to set ProxyType property. Check out the proxy server documentation to find out which method to use.

How to Register

If you live in Canada, please visit http://www.argosoft.com/delphi/canreg.html for registration information.

The retail price of the *Internet Mail Suite 2.0* is **US$129**

If you are the registered user of *IMS 1.XX*, you can upgrade for **US$35.00**.

Registered users will receive the complete source code of all units included in the *Internet Mail Suite*, technical support by email, bug fixes and free updates until the next major version release. Usage of the Internet Mail Suite is **royalty free**.

You can use the service, provided by **DigiBuy** to register IMS. You will have to go to the order web pages, and submit your order. You can use the credit card, or check. *DigiBuy* also accepts purchase orders.

If you use your credit card and order the IMS 2.0 (not the upgrade), after submitting the form, *DigiBuy* will instantly verify your credit card, and send you the email with the link to the registed version. So, you can download the registered version **almost instantly**.

If you want to order by phone, fax, send to DigiBuy a check, or purchase order, you will still have to submit the order form, select the payment method, and you will see the appropriate information, how exactly to order, on their pagest.

Here are the links to the order forms:

IMS 2.0:

http://www.digibuy.com/cgi-bin/order.html?220379+93362900250

Upgrade from IMS 1XX to IMS 2.0:

http://www.digibuy.com/cgi-bin/order.html?220379+93304780544

You also can bypass DigiBuy, and mail us a check directly. Just complete the form contained in the file regform.doc (it is located in the IMS directory), and mail it, along with a check, payable to ArGo Software Design, to the following address:

ArGo Software Design
4325 Steeles Ave West, #211
North York, Ontario M3N 1V7
Canada

As soon as we get your check, we will email you the registered version.

ProxyType property

unit msFTP

applies to
TmsFTPClient component

description
```
ProxyType: TmsProxyType;
```

declaration
Currently TmsFTPClient component supports two kinds of FTP proxy servers: so called User with No Logon (used e.g. by *Wingate* proxy server), and Proxy Open (used e.g., by *CProxy*). We will be adding more proxy types as far as users request it. You have to make sure that your proxy type is set properly, otherwise your application will not work correctly.

TmsProxyType type

unit msFtpCls

declaration

```
TmsProxyType = (fpUserNoLogon, fpProxyOpen);
```

description

This is a type of the ProxyType property of TmsFTPClient component.

ServerType property

unit msFTP

applies to
TmsFTPClient component

declaration
```
ServerType: TmsServerType;
```

description
Contains the information about the server type. We had to introduce this property due to the following problem: unfortunately the directory data sent by the server after *LIST* FTP command is not standardized. TmsFTPClient component tries to take into the account most common output formats and parse the data in order to fill in the DirList property. We extensively tested the component with UNIX and DOS standard types, which are most common, but there are also another server types which, probably we cannot handle.

But if you don't need to retrieve the directory data from the server, if you just need to run RetreiveFile and/or StoreFile methods and their modifications, then you should not have any problems with the server types.

TmsServerType type

unit msFtpCls

declaration

```
TmsServerType = (stAuto, stUnix, stDos, stHP3000, stAS400);
```

description

This is a type of the ServerType property of TmsFTPClient component.

TransferType property

unit msFTP

applies to
TmsFTPClient component

declaration
```
TransferType: TmsTransferType;
```

description
Transfer type for files to or from the FTP server. Should be set to *ftBinary* for binary files and to *ftAscii* for text files.

TmsTransferType type

unit msFtpCls

declaration

```
TmsTransferType = (ttBinary, ttASCII);
```

description

This is a type of TransferType property of TmsFTPClient component.

AppendStoreFile
CancelDataTransfer
ChangeDirectory
ChangeToUpperDirectory
DeleteDirectory
EraseFile
GetDirectoryOutput
GetDirList
GetFileList
GetFileSize
Login
Logout
MakeDirectory
RenameFile
ResumeRetrieveFile
ResumeStoreFile
RetrieveFile
StoreFile

inherited from TmsClientSocket
Connect

inherited from TmsSocket
Cancel
Disconnect
Read
RecvChunkedStream
RecvLine
RecvLineStream
RecvMultiLines
RecvStream
SaveLogFile
SendChunkedStream
SendLine
SendStream
Write

TmsFTPDirEntry class

unit msFTPCls

declaration

```
type
  TmsFTPDirEntry=class
    Kind : TmsFileKind;
    FileName : ShortString;
    Size : LongInt;
    Date : TDateTime;
    LinkPtr : ShortString;
  end;
```

description

A type of each item of TmsFTPDirList object, which is a type of DirList property of TmsFTPClient component.

TmsFTPDirList class

unit msFTPCls

declaration

```
type.
  TmsFTPDirList = class(TPersistent)
  private
    FList : TList;
    FServerType : TmsServerType;
    function Get(Index : Integer) : TmsFTPDirEntry;
    procedure Put(Index : Integer; Value : TmsFTPDirEntry);
    function GetCount : Integer;
  public
    constructor Create;
    destructor Destroy; override;
    function Add(Value : TmsFTPDirEntry) : Integer;
    procedure Assign(Source : TPersistent); override;
    function AddString(const s : string) : Integer;
    procedure Clear;
    property Items[Index : Integer] : TmsFTPDirEntry read Get write Put;
default;
    property Count : Integer read GetCount;
    property ServerType: TmsServerType read FServerType write FServerType;
  end;
```

declaration

Is a type of the DirList property of TmsFTPClient component. A list of TmsFTPDirEntry items.

TmsFileKind type

unit
msFTPCls

declaration
```
type
  TmsFileKind = (fkUnknown, fkFile, fkDirectory, fkLink);
```

description
Type of the Kind field of TmsFTPDirEntry class.

GetFileList method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure GetFileList(Strings: TStrings);
```

description
Retrieves the list of the files in the current directory. You must be logged into the server, also, *Strings* object must exist before you call this method. *Strings* will be filled with the names of the files and directories.

example
```
procedure Form1.GetListOfFiles(const Path: string);
begin
  msFTPClient1.Login;
  msFTPClient1.CurrentDirectory:=Path;
  msFTPClient1.GetFileList(Memo1.Lines);
  msFTPClient1.Logout;
end;
```

GetFileSize method

unit msFTPCls

applies to
TmsFTPClient component

declaration

```
function GetFileSize(const FileName: string): LongInt;
```

description
Returns the size (in bytes) of the specified file. You must be logged into the server before calling this method.

GetDirectoryOutput method

unit
msFTP

applies to
TmsFTPClient component

declaration
```
procedure GetDirectoryOutput(Stream: TStream);
```

description
Allows you to retrieve the directory structure as it was sent by the server, without parsing it. If you want to retrieve the data and have it analized by TmsFTPClient component, use GetDirList method.

You must be logged into the server, also, the *Stream* must exist before you call this method.

Example
```
procedure Form1.ShowCurrentDirectoryOutput;
var
  Stream: TStream;
begin
  Stream:=TMemoryStream.Create;
  try
    MyFTPClient.Login;
    MyFTPClient.GetDirectoryOutput(Stream);
    Stream.Position:=0;
    Memo1.Lines.LoadFromStream(Stream);
    MyFTPClient.Logout;
  finally
    Stream.Free;
  end;
end;
```

AppendStoreFile method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure AppendStoreFile(const LocalFilePath, RemoteFilePath: string);
```

description
Works the same way as StoreFile method, but instead of creating, or rewriting the file on the remote server, appens the sent data, contained in the *LocalFilePath* file, to the file stored on the remote server stored with the name *RemoteFilePath*.

CancelDataTransfer method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure CancelDataTransfer;
```

description
Cancels data transfer, if one is in progress, by sending ABOR command to the server, via control connection. If you wish to cancel control connection, call Cancel method.

ChangeToUpperDirectory method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure ChangeToUpperDirectory;
```

description
Requests from server to change to the parent directory by sending CDUP command through control channel. You have to be logged into the server to use this command.

DeleteDirectory method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure DeleteDirectory(const Path: string);
```

description
Requests from server to delete the directory indicated in *Path* parameter, by issuing *RMD* command. You have to be logged into the server to use this method.

EraseFile method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure EraseFile(const FileName: string);
```

description
Requests from server to delete the file indicated in the *FileName* parameter, by issuing *DELE* command. You have to be logged into the server to use this method.

MakeDirectory method

unit msFTP

applies to

`TmsFTPClient component`

declaration

```
procedure MakeDirectory(const Path: string);
```

description

Requests from server to create a directory indicated in *Path* parameter, by issuing *MKD* command. You have to be logged into the server to use this method.

RenameFile method

unit msFTP

applies to
TmsFTPClient component

declaration
```
procedure RenameFile(const OldFilePath, NewFilePath: string);
```

description
Requests from server to rename the file from *OldFileName* to *NewFileName*, by issuing *RNFRO* and *RNTO* commands. You have to be logged into the server to use this method.

ResumeRetrieveFile method

unit msFTP

applies to
TmsFTPClient component

declaration

```
procedure ResumeRetrieveFile(const RemoteFilePath, LocalFilePath: string;
Marker: LongInt);
```

description
Resumes file retrieval. You have to pass the local and remote file names, also a marker, where you wish to resume the transfer. The marker is a position in the file stream, where the transfer should be resumed. Some servers will not accept this method, but most of them will. If you will get a EmsServerError exception when calling this method, it means the server does not support resuming, and you have to use RetrieveFile method. You have to be logged into the server to use this method.

EmsServerError type

unit msDef

declaration

```
type
  EmsServerError = class(Exception);
```

description

This is the exception which will be raised if the server returns negative reply.

ResumeStoreFile method

unit msFTP

applies to
TmsFTPClient component

declaration

```
procedure ResumeStoreFile(const LocalFilePath, RemoteFilePath: string;
Marker: LongInt);
```

declaration

Resumes file upload. You have to pass the local and remote file names, also a marker, where you wish to resume the transfer. The marker is a position in the file stream, where the transfer should be resumed. Some servers will not accept this method, but most of them will. If you will get a EmsServerError exception when calling this method, it means the server does not support resuming, and you have to use StoreFile method. You have to be logged into the server to use this method.

OnDataTransferProgress
OnDataTransferStart
OnDataTransferTerminate
OnLineReceived
OnLineSent

inherited from TmsClientSocket
OnConnecting

inherited from TmsSocket
OnConnected
OnDisconnected
OnOOBData
OnRead
OnTransferProgress
OnWrite

TmsDataTransferProgressEvent type

unit
msFTP

declaration
```
type
  TmsDataTransferProgressEvent = procedure(Sender: TObject; ByteCount:
LongInt) of Object;
```

description
This is a type of OnDataTransferProgress event handler of TmsFTPClient component.

OnDataTransferProgress event

unit msFTP

applies to
TmsFTPClient component

declaration
```
OnDataTransferProgressEvent: TmsDataTransferProgressEvent;
```

description
Can be used to display the data transfer progress. *ByteCount* parameter contains the number of transferred bytes. If you wish to display the percentage of the transferred data, you will have to calculate it yourself. For sending the files, you can find out the entire size of the file you are sending using e.g., FileSize function. When retrieving, you can use GetFileSize method.

OnDataTransferStart event

unit msFTP

applies to
TmsFTPClient component

declaration
```
OnDataTransferStart: TNotifyEvent;
```

description
Is called just before the data transfer starts. Can be used to display the message, that the data transfer is just about to commence.

OnDataTransferTerminate event

unit msFTP

applies to
TmsFTPClient component

declaration
```
OnDataTransferTerminate: TNotifyEvent;
```

description
Called after the data connection is closed. Can be used to display the message about it.

OnLineReceived event

unit msFTP

applies to
TmsFTPClient component

declaration
```
OnLineReceived: TmsLineTransferEvent;
```

description
Is called when a line has been received from the server. Can be used to log the conversation with the server, along with OnLineSent event handler.

OnLineSent method

unit msFTP

applies to
TmsFTPClient component

description
```
OnLineSent: TmsLineTransferEvent;
```

declaration
Is called when a line has been received from the server. Can be used to log the conversation with the server, along with OnLineReceived event handler.

TmsLineTransferEvent type

unit msSocket

declaration

```
type
  TmsLineTransferEvent = procedure(Sender: TObject; const TheLine: string) of
Object;
```

description

This is the type of OnLineSent and OnLineReceived event handlers of TmsFTPClient component. Parameter *TheLine* contains the line which has been received or sent via the control connection.

TmsListenerSocketBase component

properties methods events
unit msSocket

declaration

```
TmsListenerSocketBase = class(TmsSocketBase);
```

description

An ancestor of TmsSimpleListenerSocket and TmsListenerSocket components. Contains properties, methods and events which are common these components.

Port
ServerSocketTimeOut

inherited from TmsSocketBase
SleepTime *
Socket

Port property (for listener components)

unit msSocket

applies to
TmsListenerBase component

declaration
```
Port: SmallInt;
```

description
Your server application, which uses TmsListenerSocket or TmsSimpleListenerSocket components, will start listening on this port after you call Start method.

Start method

unit msSocket

applies to
TmsListenerSocketBase component

declaration
```
procedure Start;
```

description
Starts up the server, by listening to the port specified in the Port property.

ServerSocketTimeOut property

unit msSocket

applies to
TmsListenerSocketBase component

declaration
```
ServerSocketTimeOut: Integer;
```

description
Sets the time out of the server socket. Server socket itself is TmsSocket, and this value will be passed to the TimeOut property of the instance of the server socket.

Start
Stop

Stop method

unit msSocket

applies to
TmsListenerSocketBase component

declaration
```
procedure Stop;
```

description
Stops listening to the specified Port, or, in other words, stops the server. If there are any connections, call of this method will not disconnect them.

OnStart
OnStop

OnStart event

unit msSocket

applist to
TmsListenerSocketBase component

desclaration
```
OnStart: TNotifyEvent;
```

description
Is called when you call Start method.

OnStop event

unit msSocket

applist to
TmsListenerSocketBase component

desclaration
```
OnStop: TNotifyEvent;
```

description
Is called when you call Stop method.

ServerSocket

inherited from TmsListenerSocketBase
Port
ServerSocketTimeOut

inherited from TmsSocketBase
SleepTime *
Socket

OnSLSConnectionRequested

inherited from TmsListenerSocketBase
OnStart
OnStop

ServerSocket property

unit msSocket

applies to
TmsSimpleListenerSocket component

description
```
ServerSocket: TmsSocket;
```

description
This is the socket you should use when you are writing the handler for doing whatever your server must do. You can receive and send the data using the methods of TmsSocket component.

inherited from TmsListenerSocketBase
Start
Stop

OnSLSConnectionRequested event

unit msSocket

applies to
TmsSimpleListenerSocket component

declaration
```
OnSLSConnectionRequested: TNotifyEvent;
```

description
You must assign this method to make your server functional. Use the methods of ServerSocket property of your instance of TmsSimpleListenerSocket component. Also, see the example in the TmsSimpleListenerSocket topic.

ConnectionList
ConnectionCount
ServerThreadClass
SuspendedServer

inherited from TmsListenerSocketBase
Port
ServerSocketTimeOut

inherited from TmsSocketBase
SleepTime *
Socket

ServerThreadClass property

unit msSocket

applies to
TmsListenerSocket component

declaration
```
ServerThreadClass: TmsServerThreadClass;
```

description
Very important property of TmsListenerSocket component. You must create a class, which descends from TmsServerThread, and assign it's name to this property. You must override *Execute* method of TmsServerThread descendant by putting in your Execute method all the work you wish your server to perform.

example
Let's create a server which will send the line 'Hi There' as soon as a client connects to it, and then closes the connection. First we should create the descendant of TmsServerThread class, which, by itself, descends from TThread class...

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

procedure THiThereServer.Execute;
begin
  ServerSocket.SendLine('Hi There');
  ServerSocket.Disconnect;
end;
```

Now, we can do:

```
procedure MyForm.StartButtonClick(Sender: TObject);
begin
  msListenerSocket1.ServerThreadClass:=THiThereServer;
  msListenerSocket1.Port:=1090;  // or whatever
  msListenerSocket1.Start;
end;
```

TmsServerThread class

unit msSocket

declaration

```
type
  TmsServerThread=class(TThread)
  public
    Peer: ShortString;
    ServerSocket: TmsSocket;
    procedure Cancel; virtual;
  end;
```

description

You must override this class, by creating your own class, also override *Execute* method. This thread will be launched whenever connection from the client is requested.

Peer field contains the IP address (in dot separated numeric format) of the client, *ServerSocket* is an instance of TmsSocket component, which is created with the *TmsServerThead* class, procedure *Cancel* let's you to write the procedure which would cancel the connection, if it is necessary.

TmsServerThreadClass type

unit msSocket

declaration

```
TmsServerThreadClass=class of TmsServerThread;
```

description

This is a type of ServerThreadClass property of TmsListenerSocket component.

SuspendedServer property

unit msSocket

applies to
TmsListenerSocket component

declaration
```
SuspendedServer: boolean;
```

description
Must be set to *true* if you wish to start the execution of the ServerThread manually. By default is set to *false*.

CancelAllConnections

inherited from TmsListenerSocketBase
Start
Stop

ConnectionList property

unit msSocket

applies to
TmsListenerSocket component

declaration
```
ConnectionList: TmsConnList;
```

description
Thread safe list of TmsServerThread objects which are currently running, i.e., active connections. You can use this property to track the connections, but, we found, that it would be more useful if you could do it more easily using OnServerThreadStart and OnServerThreadTerminate event handlers, e.g., to display the connection infos in the ListBox, or ListView. It would be better, since TmsConnList is not visual.

TmsConnList class

unit msSocket

declaration

```
  TmsConnList = class
  private
    FCS: TRTLCriticalSection;
    FList: TList;
    function GetCount: Integer;
    function Get(Index: Integer): TmsServerThread;
    procedure Put(Index: Integer; Value: TmsServerThread);
  protected
    procedure LockList;
    procedure UnlockList;
  public
    constructor Create;
    destructor Destroy; override;
    function Add(Item: TmsServerThread): Integer;
    procedure Remove(Item: TmsServerThread);
    procedure Delete(Index: Integer);
    function IndexOf(Value: TmsServerThread): Integer;
    procedure Clear;
    property Count: Integer read GetCount;
    property Items[Index: Integer]: TmsServerThread read Get write Put;
default;
  end;
```

description

This is a type of ConnectionList property of TmsListenerSocket component.

OnServerThreadStart event

unit msSocket

applies to
TmsListenerSocket component

declaration
```
OnServerThreadStart: TmsServerThreadEvent;
```

description
Triggered just before server thread (descendant of TmsServerThread object) starts the execution. The parameter *ServerThread* (see TmsServerThreadEvent type) contains the pointer to it's instance. You can use this event handler to do the initialization of the thread, e.g., if you have declared some objects in the server thread, you can create and initialize them here, rather than writing your own constructor of the server thread, it might be even safer, then doing it in the thread (Don't forget to clean up them in the OnServerThreadTerminate event handler). You also can add use this event handler to display the information about the connection.

example
Say you decided to write a server, which will read the content of somefile.txt and send it to the client, then disconnects, and declared the descendant of TmsServerThread like this:

```
TMyServerThread=class(TmsServerThread)
protected
  procedure Execute; override;
public
  SL: TStrings;
end;

procedure TMyServerThead.Execute;
var
  i: Integer;
begin
  SL.LoadFromFile('somefile.txt');
  for i:=0 to SL.Count-1
    ServerSocket.SendLine(SL[i]);
  ServerSocket.Disconnect;
end;
```

Then, you assigned this type to the ServerThreadClass property of TmsListenerSocket component:

```
TmsListenerSocket1.ServerThreadClass:=TMyServerThread;
```

As you can see, *SL* object is not created in the thread. You can create it in *OnServerThreadStartEvent*:

```
procedure Form1.msListenerSocket1ServerThreadStart(Sender: TObject;
ServerThread: TmsServerThread);
begin
  (ServerThread as TMyServerThread).SL:=TStringList.Create;
end;
```

You must free *SL* object in the OnServerThreadTerminate event handler:

```
procedure Form1.msListenerSocket1ServerThreadTerminate(Sender: TObject;
```

```
ServerThread: TmsServerThread);
begin
  (ServerThread as TMyServerThread).SL.Free;
end;
```

OnServerThreadTerminate event

unit msSocket

applies to
TmsListenerSocket component

declaration
```
OnServerThreadTerminate: TmsServerThreadEvent;
```

description
Called after the connection has been terminated. *ServerThread* parameter (see TmsServerThreadEvent type) contains a pointer to the thread which is about to be terminated. You can use this event hanlder to cleanup the data you created in OnServerThreadStart event handler, and so on. See OnServerThreadStart topic for more discussion.

ConnectionCount property

unit msSocket

applies to
TmsListenerSocket component

declaration
```
ConnectionCount: Integer;
```

description
Count of active connections, or active threads. This is a read-only property.

CancelAllConnections method

unit msSocket

applies to
TmsListenerSocket component

declaration
```
procedure CancelAllConnections;
```

description
Cancels all active connections. Calls *Cancel* method of the descendants of the TmsServerThread object. In TmsServerThread *Cancel* is declared as virtual, so you must override this method if you wish to do some other cleanup work in your thread, othewise, the default method will be called, which just closes the socket.

OnConnectionRequested
OnServerThreadStart
OnServerThreadTerminate

inherited from TmsListenerSocketBase
OnStart
OnStop

OnConnectionRequested event

unit msSocket

applies to
TmsListenerSocket component

declaration
```
OnConnectionRequested: TmsRequestEvent;
```

description
Triggered when the listener socket detects the connection request, just before of creating the server thread. Parameter *Peer* (see TmsRequestEvent type) contains the IP address of the computer which is trying to connect, in dot separated numeric format. You can decide whether you wish to accept the connection, or deny it, by setting *Allow* parameter appropriately. Default value is *true*.

TmsRequestEvent type

unit msSocket

declaration

```
TmsRequestEvent = procedure(Sender: TObject; const Peer: string; var Allow:
boolean) of Object;
```

description

This is a type of OnConnectionRequested event handler of TmsListenerSocket component.

TmsServerThreadEvent type

unit msSocket

declaration

```
TmsServerThreadEvent = procedure(Sender: TObject; ServerThread:
TmsServerThread) of Object;
```

description

This is a type of OnServerThreadStart and OnServerThreadTerminate event handlers of TmsListenerSocket component.

What's New in IMS 2

We have rewritten all core units. Now they should work more efficiently.

TmsSMTP component - Everything is almost the same. It has been renamed to TmsSMTPClient. We just added three properties, which are reserved for the future use, for handling more advanced MIME content types. Advanced MIME component will be relased later;

TmsPOP component - *TmsRemotePOP* component does not exist anymore. Now single POP component, called TmsPOPClient, does all the job. *MessageList* property is gone, since it was causing memory problems when users were retrieving large messages. Now, instead of the list, we are using the MailMessage property. The property can be filled in after retrieving the single message. Also, *RemoteInfo* classes do not exist, now you can retrieve all the information into the MailMessage headers, using RetrieveHeader method, and, we have additional methods for the values which are usually not included in the headers, such as message size (GetSize) and UIDL (GetUIDL);

TmsMessage and **TmsArticle** components - no changes;

TmsNNTP component - New component, called TmsNNTPClient, now supports extended NNTP commands, it allows to retrieve the information about the articles much faster then before.

TagFTP component - Now it is called TmsFTPClient. It supports passive transfers (PassiveMode property), also has new methods, such as GetFileList, GetFileSize, also, lets you to retrieve the directory information as it was sent by server, using GetDirectoryOutput method.

TagFinger, **TagWhois**, **TagTime** components - we removed these components from IMS, since, after the introduction of TmsClientSocket component, there was really no use to keep the components which deal with such simple protocols. We are providing demo programs (see DEMOS directory) which demonstrate how to use TClientSocket component to implement each of these protocols;

TagRas component - we removed it from IMS. It will be distributed as freeware. Main reason - unability to support it. RAS appears to behave differently on different Windows versions, even, on different computers with the same Windows versions. Another reason - more and more people are getting connection to the Internet using cable and ASDL...

New components - TmsListenerSocket, TmsSimpleListenerSocket - for creating server applications.
New class TmsWinsock - lets you to call winsock functions directly, also has couple of useful methods, which lets you to get the IP address and the domain name of your computer, and more...

More features - you can create CGI applications more efficiently, since simple procedures allow you to get rid of Delphi footprint. See Using IMS in Console Applications.

Example Applications

In the *DEMOS* folder you will find the applications, which demonstrate the usage of the components included in the *Interner Mail Suite*:

TmsWinsock
wsInfoDemo.drp - usage of msWinsock object;

TmsClientSocket:
FingerDemo.drp - Finger client
WhoIsDemo.drp - WhoIs client
TimeDemo.drp - Network Time client

TmsSMTPClient, TmsMessage:
smtpdemo.dpr - SMTP client

TmsPOPClient, TmsMessage:
popdemo.drp - POP3 client
spop.dpr - Advanced usage of TmsPOPClient component. Let's you to preview the messages before you decide what to do with them;

TmsNNTPClient, TmsArticle:
nntpdemo.drp - NNTP client;

TmsFTPClient
ftpdemo.drp - FTP client
ftpresumedemo.dpr - Shows how to resume file upload/download using TmsFTPClient component

TmsHTTPClient
httpdemo.drp - HTTP client. Demonstrates the usage of Get method;
httpPosDemo.dpr - Shows how to post the data to the server, using TmsHTTPClient component.

Installation

Delphi 2
Click *Component* - *Install* and install the file **msreg.pas**

Delphi 3
Click *Component* - *Install Packages*, and install the package **ims2d3.dpl**. You also will need to add the path to IMS to the Library path. To do it, click *Tools* - *Environment Options*, then select *Library* tab, and in the *Library Path* box add the path to the IMS files, separated by a ";".

Delphi 4
Start up Delphi 4, click *Component* - *Install Packages*. In the *Project Options* dialog box, click *Add* button, and browse until you find **ims2.bpl** file. Select it and click *Open*. IMS components should appear in the *Internet Mail Suite* tab of your component palette.

Now you will need to add the path to IMS in the Library search path. Click *Tools* - *Environment Options*, select *Library* tab, click small button next to the *Library Path* edit box, In the *Directories* box, in the smaller edit box, type the path where you placed IMS files (e.g. *c:\Ims2*), click *Add*, then click *OK*, once more - *OK*, and you are done.

Special Thanks

Special thanks to **Mr. Brian Milburn**, **Solid Oak Software, Inc.**, for extensive tests of the *The Internet Mail Suite* and very useful advises.

Also to **Bruce W. Caron**, **Luc Wuyts**, **Martin Baur**, **Paul Stohr**, **John Taylor**, **Jayson Minard**, **David Sherman**, **Alexander Lucke** and everyone, who helped to locate bugs in our older, TSMTP, TPOP3 components and in the Internet Mail Suite, also helping us to make out products much more productive and stable.