TMapiSession Component

Properties Methods

Unit
Mapi

Description
The TMapiSession component is used to create a valid MAPI session for the TMapiMessage component. It handles logging in and logging out of the MAPI system.

You determine the parameters for the new session by setting the MAPIName, MAPIOptions, and MAPIPwd properties. To establish the new session you call OpenSession. The session is terminated by calling CloseSession.

The session handle may be obtained through the Session property.

Properties
The following properties are unique to the TMapiSession component:

MAPIName
MAPIOptions
MAPIPwd
Session

MAPIOptions Property

Example

Declaration
property TMAPIOptions: TMAPISesOptions;

Description
The MAPIOptions property determines what will happen when the OpenSession method is called.   Valid options are:

soLogonUI	Setting this option will display the standard mail logon dialog if the system is unable to login with the credentials provided by MAPIName and MAPIPwd.
soNewSession	Setting this option will create a new MAPI session, even if one already exists. If this option is not set, an existing session (if available) will be used.
soDownloadMail	Setting this option will download all new mail when the session is opened. This may significantly increase processing time.

Example
The following will use an existing session (if possible) and prompt the user for login credentials if the login attempt fails:

```
MapiSession1.Options := [soLogonUI];
```

TMAPISesOptions Type

Declaration
TMAPISesOption = (soLogonUI, soNewSession, soDownloadMail);
TMAPISesOptions = **set of** TMAPISesOption;

Description
The TMAPISesOptions type is a set of options the MAPIOptions property can assume.

TMAPIName Property

Example

Declaration
property MAPIName: string;

Description
The MAPIName property is used to set the user name for the MAPI session.

Example
The following will set the login parameters for user **John Smith** with a password of **doggie**:

```
MapiSession1.MAPIName := `John Smith`;
MapiSession1.MAPIPwd := `doggie`;
```

TMAPIPwd Property

Example

Declaration
property MAPIPwd: string;

Description
The MAPIPwd property is used to set the user password for the MAPI session.

Session Property (TMapiSession)

Example

Declaration
property Session: LongInt;

Description
The Session property is available only at run-time and is read only.   It will return the MAPI session handle if a valid MAPI session has been established.   If the OpenSession method has not yet been called, or if the call failed, then it will return zero.

Example
The following code checks for a valid session handle and then assigns it to the Session property of a TMAPIMessage component:

```
procedure SetSession;
var
  SessionCheck: LongInt;
begin
  SessionCheck := MapiSession1.Session;
  if SessionCheck <> 0 then
    MapiMessage1.Session := SessionCheck
  else
    ShowMessage(`No MAPI session exists`);
end;
```

Methods
The following methods are supported by TMapiSession component:

OpenSession
CloseSession

OpenSession Method

Example

Declaration
function OpenSession: TMAPIResults

Description
The OpenSession method opens a MAPI session for use.   It uses the MAPIName and MAPIPwd properties for login credentials.

If the MAPIOptions property includes soNewSession then a new MAPI session will be created.   If this option is not set, the TMapiSession component will attempt to use an existing session (if one exists).

Any return other than mrSuccess indicates the call failed and no session was opened.   The Session property will be set to zero.

A return of mrSuccess indicates that the cail was successful.   The Session property will contain the value of the MAPI session.

Example
The following code opens a new MAPI session and then closes it:

```
procedure CreateSession;
var
  TheResult: TMAPIResults;
begin
  with MapiSession1 do
    begin
      MAPIName := `John Smith`;
      MAPIPwd := `doggie`;
      MAPIOptions := [soNewSession];
    end;

  TheResult := MapiSession1.OpenSession;
  if TheResult = mrSuccess then
    ShowMessage(`Successful opened session #` + IntToStr(MapiSession1.Session)
  else
    ShowMessage(`Unable to open session.   Error #` + IntToStr(Ord(TheResult)));
end;
```

TMAPIResults Type

Declaration

```
TMAPIResults = (mrSuccess, mrUserAbort, mrFailure, mrLoginFailure, mrDiskFull, mrInsufficientMemory,
                mrAccessDenied, mrTooManySessions, mrTooManyFiles, mrTooManyRecips,
                mrAttachmentNotFound, mrAttachmentOpenFailure, mrAttachmentWriteFailure,
                mrUnknownRecip, mrBadRecipType, mrNoMessages, mrInvalidMessage,
                mrTextTooLarge, mrInvalidSession, mrUnsupportedType, mrAmibiguousRecip,
                mrMessageInUse, mrNetworkFailure, mrInvalidEdits, mrInvalidRecips, mrUnsupported);
```

Description

A set of possible results returned from MAPI calls.   Meanings are indicated by the error constant, i.e. mrUserAbort indicates that the user aborted the login process.   Most results are handled internally in the TMapiSession and TMapiMessage components; mrSuccess, mrFailure, and mrLoginFailure are the only errors likely to require the programmer`s attention.

For a complete description, please consult the Microsoft MAPI reference.

CloseSession Method

Example

Declaration
function CloseSession: TMAPIResults

Description
The CloseSession method closes the open session specified by the Session property.

A return of mrSuccess indicates that the session was closed.   The Session property will be set to zero.

Any return other than mrSuccess indicates that the component failed to close the session.

MAPI Components

TMapiMessage
TMapiSession

TMapiMessage Component

Properties		Methods

Unit
Mapi

Description
The TMapiMessage component is used to create and retrieve MAPI messages.

To send a message you obtain a valid Session from a TMapiSession component.   Fill in the Files, NoteText, Recips, and Subject properties.   You then call the SendMail method.

The SendOptions property effects how the message is sent.   The SendDocuments method provides a simplified way to send data files.

To read a message   you obtain a valid Session from a TMapiSession component.   Set the ReadOptions property as desired, and then call then GetMail method.   The DeleteMail method allows you to delete messages.

Properties
The following properties are unique to the TMapiMessage component:

ClearOnSend
DateReceived
Files
NoteText
ReadOptions
Recips
SendOptions
Session
Subject

Methods
The following methods are supported by TMapiMessage component:

DeleteMail
FreeRawMail
GetMail
GetRawMail
SendDocuments
SendMail
SendRawMail

ClearOnSend Property

Example

Declaration
property ClearOnSend: Boolean;

Description
The ClearOnSend property determines whether or not the TMapiMessage component resets itself after a call to the SendMail and SendDocuments methods.   If set to True, the DateReceived, Files, NoteText, Recips, SendOptions, and Subject properties are cleared after either of the methods are called

Example
The following code sets the property:

```
        MapiMessage1.ClearOnSend := True;
```

DateReceived Property

Example

Declaration
property DateReceived: **string**;

Description
The DateReceived property is run-time and read-only.   It is meaningful only after a call to the GetMail method, when it will contain the date the the message was received formated as YYYY/MM/DD HH:MM.

This property is automatically assigned by the MAPI system for outgoing mail.

Example
The following code shows how to use the property:

```
procedure CheckMail;
var
  MapiResult: TMAPIResults;
begin
  MapiMessage1.Session := MapiSession1.Session;
  MapiResult := MapiMessage1.GetMail;

  if MapiResult = mrSuccess then
    ShowMessage(`This mail was received on ` + MapiMessage1.DateReceived);
end;
```

Files Property

Example

Declaration
property Files: TStringList;

Description
The Files property contains the list of file attachments and and their positions for a message.

When sending mail, you fill the Strings property with the full pathname of the attachment, and the Objects property with the file position. The file position must be between -1 and StrLen(NoteText.GetText). Invalid or unassigned positions default to the beginning of the message (value of -1). For example, the lines

```
MapiMessage1.NoteText.Add(`Hello World!!!`);
NewIndex := MapiMessage1.Files.Add(`C:\DOS\README.TXT`);
MapiMessage1.Files.Objects[NewIndex] := TObject(7);
```

would add the attachment `C:\DOS\README.TXT` in the 3rd position of the message text, replacing the \`W\` in \`World\` (characters are not replaced when the position is -1). Note that you should cast the position from an integer to the type TObject.

When reading mail, the Strings and Objects properties will be filled with the same information as listed above. For a message with two attachments (`README.TXT` and `SMILE.BMP`)

```
for L := 0 to (MapiMessage1.Files.Count -1) do
  begin
  FullPath := Files.Strings[L)];   {Returns \TEMP\README.TXT, \TEMP\SMILE.BMP}
  Position := Integer(Files.Objects[L])
  ShowMessage(FullPath + `, ` + IntToStr(Position));
  end
```

would retreive the pathnames and positions of both attachments. Note that the paths returned are in the system`s TEMP directory, where the MAPI system copies any file attachments.

The TMapiMessage component will delete any file attachments copied to the TEMP directory when the GetMail or DeleteMail methods are called. Otherwise, it is the programmer`s responsibility to make sure that any temporary files are deleted before changing the contents of the Files property.

The Files property behaves like a standard TStringList in all other ways.

Example
The following code attaches files to an outgoing message and then displays the names of the attachments from a retrieved message:

```
procedure FilesTest;
var
  MapiResult: TMapiResults;
  NewIndex: Integer;
  Loop: Integer;
begin
  { Add some Text with underscore to be replaced by a file }
  MapiMessage1.NoteText.Add(`Here are some files: _`)

  { Attach C:\AUTOEXEC.BAT to beginning of file }
  NewIndex := MapiMessage1.Files.Add(`C:\AUTOEXEC.BAT`);
  MapiMessage1.Files.Objects[NewIndex] := TObject(-1);

  { Attach C:\CONFIG.SYS to end of note text }
  NewIndex := MapiMessage1.Files.Add(`C:\CONFIG.SYS`);
  MapiMessage1.Files.Objects[NewIndex] := TObject(22);

  { Send the message; get a new one}
  MapiMessage1.SendMail;
  MapiResult := MapiMessage1.GetMail;

  If MapiResult = mrSuccess then
    begin
      { Display the names and positions of all attachments }
      Loop := Files.Count
      while Loop > 0 do
      begin
        ShowMessage(`File: ` + Files.Strings[Loop -1]);
        ShowMessage(`Position ` + IntToStr(Integer(Files.Objects[Loop - 1])));
        Dec(Loop);
      end;
    end;

end;
```

NoteText Property

Example

Declaration
property NoteText: TStringList;

Description
The NoteText property holds the body text of the message. When sending a message, fill in the Strings property with the desired text. Separate paragraphs with a blank line. For example,

```
MapiMessage1.NoteText.Add(`This is the first paragraph!`);
MapiMessage1.NoteText.Add(``);
MapiMessage1.NoteText.Add(`This is the second paragraph`);
MapiMessage1.NoteText.Add(`This will be the part of the second paragraph`);
```

creates a message with two paragraphs of text. It is frequently convenient to use a TMemo to fill the NoteText property, as in:

```
MapiMessage1.NoteText.AddStrings(Memo1.Lines);
```

When reading a message, this property is filled with the text from the message in the same manner.

The NoteText property behaves like a standard TStringList in all other ways.

Example
The following code assigns text to the message and then reads the text from a retrieved message:

```
procedure NoteTextTest;
var
  MapiResult: TMapiResults;
begin
  { Put text in the message from a memo control}
  MapiMessage1.NoteText.AddStrings(Memo1.Lines);
  MapiMessage1.SendMail;

  { Now read a message and assign the text to a memo control }
  MapiResult := MapiMessage1.GetMail;
  if MapiResult = mrSuccess then
    Memo1.Lines.AddStrings(MapiMessage1.Strings);

end;
```

ReadOptions Property

Example

Declaration
property ReadOptions: TMapiMsgReadOptions;

Description
The ReadOptions property determines what will happen when the GetMail and GetRawMail methods are called.   Valid options are:

moEnvelopeOnly	Supresses the retrieval of the file attachments (Files) and note text (NoteText) of the message. All other information is retrieved normally. Setting this option usually reduces the processing time required for getting mail.
moSuppressAttach	Supresses the retrieval of the file attachments (Files) of the message. All other information is retrieved normally. This option is ignored if moEnvelopeOnly is set. Setting this option usually reduces the processing time required for getting mail.
moPeek	Instructs the mail system not to mark retrieved messages as read.
moUnreadOnly	Instructs the mail system to only retrieve messages that are not marked as read. If this option is not set, all messages in the inbox will be fetched.

Example
The following code retrieves message headers without marking the mail as read:

```
procedure CheckHeader;
var
  MapiResult: TMapiResults;
begin
  { Set options }
  MapiMessage1.ReadOptions := [moEnvelopeOnly, moPeek];

  { Check for mail }
  MapiResult := MapiMessage1.GetMail;
  while MapiResult = mrSuccess do
    begin
      ShowMessage(`New message subject: ` + MapiMessage1.Subject);
      MapiResult := MapiMessage1.GetMail;
    end;

end;
```

TMAPIMsgReadOptions Type

Declaration
TMAPIMsgReadOption = (moEnvelopeOnly, moSuppressAttach, moPeek, moUnreadOnly);
TMAPIMsgReadOptions = **set of** TMAPIMsgReadOption;

Description
The TMAPIMsgReadOptions type is a set of options the ReadOptions property can assume.

Recips Property

Exampletmapimessage_recips_example

Declaration
property Recips: TStringList;

Description
The Recips property contains the list of recipients and their type for a message.

When sending mail, you fill the Strings property with the Name of the recipient, and the Objects property with the recipient type. The recipient type must be of the type TMAPIRecipTypes. Invalid or unassigned types default to rtTo. For example, the lines

```
NewIndex := MapiMessage1.Files.Add(`John Smith`);
MapiMessage1.Files.Objects[NewIndex] := TObject(rtTo);
```

would add the recipeint `John Smith` on the `To:` line. You must have at least one recipient of the type rtTo. The MAPI system automatically assigns the originator of the message; it is not necessary to provide this information. Accordingly, rtOriginator is an invalid type for sending mail.

When reading mail, the Strings and Objects properties will be filled with the same information as listed above. The originator of the message is always the first recipient in the list.

The Files property behaves like a standard TStringList in all other ways.

Example
The following code assigns recipients to outgoing mail and then display recipients for retrieved mail:

```
procedure RecipsTest;
var
  MapiResult: TMapiResults;
  NewItem: Integer;
  Loop: Integer;
begin
  { Assign recipients }
  with MapiMessage1 do
    begin
      NewItem := Recips.Add(`John Smith`);
      Recips.Objects[NewItem] := TObject(mrTo);
      NewItem := Recips.Add(`Jane Doe`);
      Recips.Objects[NewItem] := TObject(mrCc);
      NewItem := Recips.Add(`Bob Jones`);
      Recips.Objects[NewItem] := TObject(mrBcc);
      SendMail;
    end;

  { Now get recipients from new mail }
  MapiResult := MapiMessage1.GetMail;
  if MapiResult = mrSuccess then
    begin
      for Loop := 0 to (MapiMessage1.Recips.Count - 1) do
        ShowMessage(`Recipient: ` + MapiMessage1.Recips.Strings[Loop]);
    end;

end;
```

TMAPIRecipTypes Type

Declaration

```
TMAPIRecipTypes = (rtOriginator, rtTo, rtCc, rtBcc);
```

SendOptions Property

Example

Declaration
property SendOptions: TMAPIMsgSendOptions;

Description
The SendOptions property determines what will happen when the SendtMail and SendRawMail methods are called.   Valid options are:

moReturnReceipt	Requests a return receipt when the message is received.
moShowDialog	Shows the send mail dialog when sending the mail. This will allow the user to add additional recipients, text, etc. to the message before it is submitted to the MAPI system for delivery.
moShowAddressBook	Shows the MAPI address book when the message is sent. This will allow the user to add addtional recipients to the message.

Example
The following requests a return receipt when the message is sent:

```
MapiMessage1.SendOptions := [moReturnReceipt]
```

TMAPIMsgSendOptions Type

Declaration
TMAPIMsgSendOption = (moReturnReceipt, moShowDialog, moShowAddressBook);
TMAPIMsgSendOptions = **set of** TMAPIMsgSendOption

Description
The TMAPIMsgSendOptions type is a set of options the MAPISendOptions property can assume.

Session Property (TMapiMessage)

Example

Declaration
property Session: LongInt;

Description
The Session property contains the handle of a valid MAPI session that is used by all of the methods of TMapiMessage. Reading this value will tell you the current session in use; since TMapiMessage does not maintain its own MAPI session, however, there is no guarantee that session is still valid.

The Session property should be assigned to the Session property of a TMapiSession component. There should not be a need for further manipulation as long as that session remains valid.

All the methods of TMapiMessage except the SendDocuments method require a valid session handle in this property. If any of the methods are called before this property is assigned then they will fail, returning mrInvalidSession.

Example
The following code gets a valid session handle and assigns it to the property:

```
procedure CreateSession;
var
  MapiResult: TMapiResults
begin
  { Open a new session }
  MapiResult := MapiSession1.OpenSession;

  { If it`s valid, assign it to MapiMessage1 }
  if MapiResult = mrSuccess then
    MapiMessage1.Session := MapiSession1.Session;

end;
```

Subject Property

Example

Declaration
property Subject: **string**;

Description
The Subject property contains the subject line for the message. When sending mail you can fill this property with the message subject. It will be ignored if left blank.

When reading mail, this property is set to the subject of the current message.

Example
This code sets the subject and then reads it from a retrieved message:

```
procedure SubjectTest;
var
  MapiResult: TMapiResults
begin
  { Set the subject }
  MapiMessage1.Subject := `This is a test...`;

  { Send the message }
  MapiMessage1.SendMail;

  { Get any new messages }
  MapiResult := MapiMessage1.GetMail;
  if MapiResult = mrSuccess then
    ShowMessage(`New message subject: ` + MapiMessage1.Subject);

end;
```

DeleteMail Method

Example

Declaration
function DeleteMail: TMapiResults

Description
The DeleteMail methods deletes the current message stored in the TMapiMessage component.

You must have a valid message in order to call DeleteMail.   DeleteMail should most commonly be used soon after a call to GetMail.

Example
The following code deletes all messages in the inbox:

```
procedure ClearInBox;
var
  MapiResult: TMapiResultstmapiresults
begin
  { Loop through all messages and trash them! }
  repeat
    MapiResult := MapiMessage1.GetMail;
    if MapiResult = mrSuccess then
      begin
        ShowMessage(`Deleting message: ` + MapiMessage1.Subject);
        MapiMessage1.DeleteMail;
      end;
  until MapiResult <> mrSuccess;

end;
```

FreeRawMail Method

Example

Declaration
function FreeRawMail(TheMessage: TMAPIAMessagePtr): TMAPIResults

Description
The FreeRawMail method is used to free memory used after a call to GetRawMail.

This method is the only way to free the memory used by the GetRawMail method; failure to do so can result in lost resources, system instability, or general protection faults.

Example
The following code obtains, manipulates, and disposes a message:

```
procedure CheckSubject;
var
  MapiResult: TMapiResults;
  TheMessage: TMAPIAMessagePtr;
begin
  { Get a message }
  New(TheMessage);
  MapiResult := MapiMessage1.GetRawMail(TheMessage);
  if MapiResult = mrSuccess then
    begin
      ShowMessage(`Retrieved message: ` + StrPas(TheMessage^.Subject));
      MapiMessage1.FreeRawMail(TheMessage);
    end;

end;
```

TMAPIAMessage, TMAPIFileDesc, TMAPIRecipDesc Types

Declarations

```
TMAPIRecipDescPtr = ^TMAPIRecipDesc;
TMAPIRecipDesc = record
  Reserved, RecipClass: LongInt;
  Name, Address: PChar;
  EIDSize: LongInt;
  EntryID: Pointer;
  end;

TMAPIFileDescPtr = ^TMAPIFileDesc;
TMAPIFileDesc = record
  Reserved, Flags, Position: LongInt;
  PathName, FileName: PChar;
  FileType: Pointer;
  end;

TMAPIAMessagePtr = ^TMAPIAMessage;
TMAPIAMessage = record
  Reserved: LongInt;
  Subject, NoteText, MessageType, DateReceived, ConversationID: PChar;
  Flags: LongInt;
  Originator: TMAPIRecipDescPtr;
  RecipCount: LongInt;
  Recips: TMAPIRecipDescPtr;
  FileCount: LongInt;
  Files: TMAPIFileDescPtr;
  end;
```

Description

These types are used by the FreeRawMail, GetRawMail, and SendRawMail methods.   Programmers not using those methods do not need to deal with these types.

The programmer is responsible for all memory allocations, releases, and data validation.   For a complete description, please consult the Microsoft MAPI reference.

GetMail Method

Example

Declaration
function GetMail: TMAPIResults

Description
The GetMail method retrieves the next available message in the inbox.

You must have a valid Session before calling this method.   GetMail will then retrieve the next available message in the inbox (according to the options set in the ReadOptions property) and place its contents into the DateReceived, Files, NoteText, Recips, and Subject properties.   The originator of the message will be first entry in the Recips list.

Repeated calls to GetMail will eventually yield a return of mrNoMessages, indicating that all messages have been retrieved.   Further calls will start the cycle again, i.e. the first message will be retrieved, then the second, and so on until mrNoMessages.

Although these properties will remain stable until the next call to GetMail, messages retrieved by GetMail are not guaranteed to remain valid because other applications may delete the message.   Applications should be able to handle failed calls to methods such as DeleteMail (which will return mrInvalidMessage).

Example
The following code displays the subject of all new messages in the inbox:

```
procedure CheckInBox;
var
  MapiResult: TMAPIResults;
begin
  { Don`t mark messages as read }
  MapiMessage1.ReadOptions := [moEnvelopeOnly, moPeek, moUnreadOnly];

  { Loop through message }
  repeat
    MapiResult := MapiMessage1.GetMail;
    if MapiResult = mrSuccess then
      ShowMessage(`New message about: ` + MapiMessage1.Subject);
  until MapiResult = mrNoMessages;

end;
```

GetRawMail Method

Exampletmapimessage_freerawmail_example

Declaration
function GetRawMail(TheMessage: TMAPIAMessagePtr): TMapiResults;

Description
The GetRawMail property is provided for programmers who need to manipulate MAPI message structures directly.   It fills a user-supplied pointer with a pointer to a TMAPIAMessage structure.   The user is responsible for calling the FreeRawMail method in order to free the memory used by GetRawMail.

Because the GetRawMail method requires considerable knowledge of both low-level MAPI and dynamic memory allocation and disposal, you should use the GetMail method instead of GetRawMail whenever possible.

SendDocuments Method

Exampletmapimessage_senddocuments_example

Declaration
function SendDocuments: TMAPIResults

Description
The SendDocuments method provides a simplified method for sending data files via MAPI.

When calling SendDocuments, all properties are ignored except the Files property, which should contain the full path names of any files you wish to attach to the message (positions are ignored).   A send note dialog is always displayed, allowing the user to assign recipients and note text.

The SendDocuments method is the **only** method of TMapiMessage that **does not** require a valid session handle in the Session property.   Accordingly, there is no need to use a TMapiSession component if you will only be using the SendDocuments method.

Example
The following code will send the files `C:\AUTOEXEC.BAT` and `C:\CONFIG.SYS`:

```
procedure DoMail;
var
  TMapiResult: TMAPIResults;
begin
  { Load up the files to send }
  MapiMessage1.Files.Add(`C:\AUTOEXEC.BAT`);
  MapiMessage1.Files.Add(`C:\CONFIG.SYS`);

  { Send the files }
  MAPIResult := MapiMessage1.SendDocuments;
  if MapiResult = mrSuccess then
    ShowMessage(`Documents Send!`);

end;
```

SendMail Method

Example

Declaration
function SendMail: TMAPIResults;

Description
The SendMail method creates and sends a new message using the contents of the Files, NoteText, Recips, and Subject properties.   The SendOptions and ClearOnSend properties determine how the message is sent.

You must have a valid Session and at least one recipient (or have moShowAddressBook in the SendOptions property set) in order to call SendMail.   If any of the recipient`s names are amibiguous, a dialog will be displayed to let the user choose the appropriate name.

Example
The following code sends a new message

```
procedure SayHi;
var
  MapiResult: TMAPIResults;
begin
  { Create message contents }
  with MapiMessage1 do
    begin
      Subject := `Hello World!`;
      Recips.Add(`John Smith`);
      NoteText.Add(`Just wanted to say hi...`);
    end;

  { Now send the message }
  MapiResult := MapiMessage1.SendMail;
  if MapiResult <> mrSuccess then
    ShowMessage(`You better call John instead!`);

end;
```

SendRawMail Method

Example

Declaration
function SendRawMail(TheMessage: TMAPIAMessagePtr): TMAPIResults;

Description
The SendRawMail method sends message using a user-supplied pointer to a TMAPIAMessage structure. The user is responsible for creating and destroying the message structure.

Because the SendRawMail method requires considerable knowledge of both low-level MAPI and dynamic memory allocation and disposal, you should use the SendMail method instead of SendRawMail whenever possible.

Example
The following creates a new message and sends it:

```
procedure MakeMail;
var
  TheMessage: TMAPIAMessagePtr;
  Recips: TMAPIRecipDescPtr;
begin
  { Create new message and fill it in }
  New(TheMessage);
  with TheMessage^ do
    begin
      Reserved := 0;
      Subject := StrNew(`Test`);
      NoteText := StrNew(`Hello world!`);
      MessageType := PChar(0);
      DateReceived := PChar(0);
      ConversationID := PChar(0);
      Originator := TMAPIRecipDescPtr(0);
      FileCount := 0;
    end;

  { Now add recipients }
  New(Recips);
  Recips^.Name := StrNew(`John Smith`);
  TheMessage^.Recips = Recips;

  { Send the message }
  MapiResult := MapiMessage1.SendRawMail(TheMessage);

  { Free memory }
  StrDispose(Recips^.Name);
  StrDispose(TheMessage^.Subject);
  StrDispose(TheMessage^.NoteText);
  Dispose(Recips);
  Dispose(TheMessage);

end;
```