

Envoyer sous Windows un message en utilisant la bibliothèque CDO de Microsoft

Auteur : Christophe PIQUER

Publié le 29/06/2024

Ce tutoriel est destiné à apprendre à réaliser avec LAZARUS une application qui permet sous Windows et sans utilisation d'un client de messagerie, l'envoi d'un e-mail avec ou sans pièce jointe.

Sommaire

I Introduction.....	2
II Le formulaire.....	3
III Les composants et propriétés.....	3
IV Le code de l'application.....	5
IV.A Les clauses uses.....	5
IV.B La fonction d'envoi d'e-mail par CDO suite à création des objets OLE et paramétrage.....	5
IV.C La procédure de paramétrage et de constitution du message pour l'envoi.....	7
IV.D Les procédures de vérification des adresses e-mail.....	8
IV.D.1 La fonction de vérification de composition de l'adresse e-mail du destinataire.....	9
IV.D.2 La fonction de vérification des adresses des destinataires.....	9
IV.E Les boutons : Envoyer, Quitter et Joindre.....	11

V Utilisation du HTML.....	12
V.A Les messages.....	13
V.B Les liens hypertextes.....	13
VI Le message permanent.....	13
VI.A Le champ message (Message :).....	14
VI.B Le champ expéditeur (De :).....	14
VI.C Le champ destinataire (Pour :).....	14
VI.D Le champ sujet (Sujet :).....	14
VI.E Les pièces jointes	15
VI.F Dans le code.....	15
VII Conclusion.....	16
VIII Remerciements.....	16

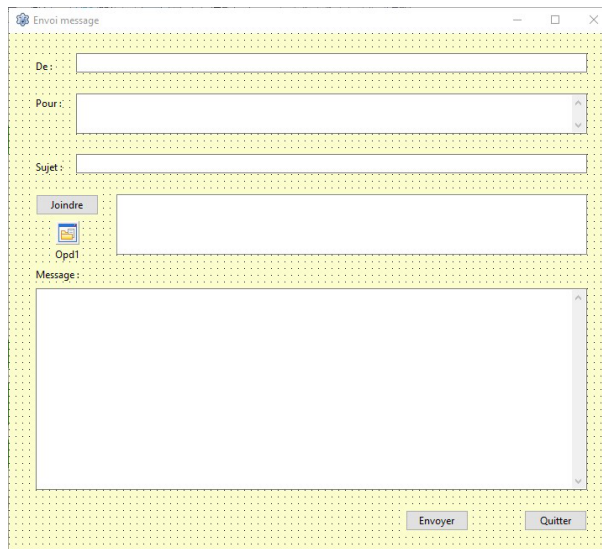
I Introduction

Ce tutoriel inspiré de « [Application et méthode d'envoi de lettres d'information en CDO](#) » de **Jean-Damien GAYOT**, amène à créer un formulaire interface utilisateur dans lequel seront saisies les données nécessaires pour l'envoi d'un e-mail. L'application s'appuie sur la bibliothèque CDO (Collaboration Data Objects) de Microsoft qui permet d'envoyer des messages sans client de messagerie installée sur son PC. Le code de l'application proposé dans ce tutoriel est une adaptation de celui publié par **samiechikh** dans son article « [Envoyer un Email avec CDO Message](#) ».

II Le formulaire

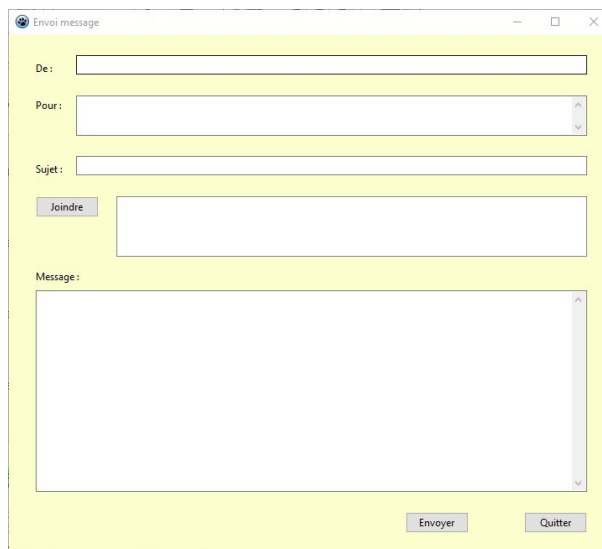
Ce formulaire est élaboré avec les éléments classiques constitutifs d'un client de messagerie pour envoyer des messages, c'est-à-dire les champs : expéditeur, destinataire, sujet, pièce jointe et corps de texte.

Vue en conception :



The screenshot shows a window titled "Envoi message" with a dotted background. It contains several input fields: "De:", "Pour:", "Sujet:", and a large "Message:" text area. There is a "Joindre" button with a paperclip icon and a label "Opd1" below it. At the bottom right, there are "Envoyer" and "Quitter" buttons.

Vue en production :



The screenshot shows the same "Envoi message" window but with a solid light yellow background. The layout and elements are identical to the design view, including the "De:", "Pour:", "Sujet:", "Message:" fields, the "Joindre" button, and the "Envoyer" and "Quitter" buttons.

III Les composants et propriétés

Seules les propriétés essentielles sont définies dans le tableau.

L'informatique de Chrispi

Composants	Propriétés
TForm	Caption : Envoi message Color : \$00CDFEFC Name : Formess Position :poScreenCenter
Tlabel (<i>De :</i>)	Caption : De : Name : Lb1
Tlabel (<i>Pour :</i>)	Caption : Pour : Name : Lb2
Tlabel (<i>Sujet :</i>)	Caption : Sujet : Name : Lb3
Tlabel (<i>Message :</i>)	Caption : Message : Name : Lb4
TEdit (<i>De :</i>)	Name : Ed1 Text : ' <i>Vide</i> '
TEdit (<i>Sujet :</i>)	Name : Ed3 Text : ' <i>Vide</i> '
TMemo (<i>Message :</i>)	Lines : ' <i>Vide</i> ' Name : Mm1 ScrollBars : ssVertical WordWrap : True
TMemo (<i>Pour :</i>)	Lines : ' <i>Vide</i> ' Name : Mm2 ScrollBars : ssVertical WordWrap : True
Tbutton (<i>Envoyer</i>)	Caption : Envoyer Name : Btn1
Tbutton (<i>Quitter</i>)	Caption : Quitter Name : Btn2
Tbutton (<i>Joindre</i>)	Caption : Joindre Name : Btn3
TListBox	Name : Lbx1
TOpenDialog	Name : Opd1

- Dans le champ correspondant à **De** : => saisie d'un nom d'expéditeur.
- Dans le champ correspondant à **Pour** : => saisie des adresses e.mail des destinataires.

- Dans le champ correspondant à **Sujet** : => saisie du sujet du message.
- Dans le champ à côté du bouton **Joindre** : => saisie des pièces jointes.
- Dans le champ sous **Message** : => saisie du corps du message.

❗ Les adresses des destinataires doivent être saisies séparées d'un point-virgule suivies d'un espace.

IV Le code de l'application

IV.A Les clauses uses

Voici les clauses **uses** nécessaires au fonctionnement de l'application et qui permettent la création et le paramétrage des objets utilisant la **bibliothèque CDO**.

```
interface
uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls,
  Variants, ComObj;
```

IV.B La fonction d'envoi d'e-mail par CDO suite à création des objets OLE et paramétrage

Afin de pouvoir utiliser dans votre programme la bibliothèque CDO, il est nécessaire de créer la fonction **ConfMail**. Celle-ci permet de créer et de configurer l'ensemble des éléments qui sont utiles à l'envoi de messages. Elle se termine d'ailleurs par la tentative d'envoi du message et enfin la libération des ressources mobilisées.

La fonction **ConfMail** :

```
//Fonction d'envoi d'e-mail par CDO suite à création des objets OLE et
paramétrage
function ConfMail(Name, Emailfrom, EmailTo, password, Subject, body,
smtp: String;
  ListeFichier:TStrings; port:Integer): Boolean;
const
  _Cdo='http://schemas.microsoft.com/cdo/configuration/';
```

```
var
  _Message: OleVariant;
  _Config: OleVariant;
  i: Integer;
begin
  try
    //Création des objets CDO.Message et CDO.Configuration
    _Message:= CreateOleObject('CDO.Message');
    _Config:= CreateOleObject('CDO.Configuration');
    //Configuration des paramètres SMTP
    _Config.Fields(_Cdo+'sendusername'):= Emailfrom; //
Expéditeur/adresse mail utilisateur smtp
    _Config.Fields(_Cdo+'sendpassword'):= password; // Mot de passe
mail utilisateur smtp
    _Config.Fields(_Cdo+'sendusing'):= 2; // Envoyer en utilisant CDO
    _Config.Fields(_Cdo+'sendemailaddress'):= Name; // Nom de
l'expéditeur
    _Config.Fields(_Cdo+'smtpserver'):= smtp; // Serveur smtp
    _Config.Fields(_Cdo+'smtpserverport'):= port; // Port du serveur
    _Config.Fields(_Cdo+'smtpconnectiontimeout'):= 60; // Time out de
connexion
    _Config.Fields(_Cdo+'smtpauthenticate'):= 1; // 1 pour
authentification sinon 0
    _Config.Fields(_Cdo+'smtpusesssl'):= True; //'True pour SSL/TLS
    //Configuration pour TLS 1.2

{ _Config.Fields('http://schemas.microsoft.com/cdo/configuration/tlsproto
ocol'):= 1; // 1 pour TLS 1.2

    _Config.Fields('http://schemas.microsoft.com/cdo/configuration/smtpuses
ssl'):= True; // True pour SSL/TLS

    _Config.Fields('http://schemas.microsoft.com/cdo/configuration/smtpauth
enticate'):= 1; // 1 pour authentification sinon 0 }
    //Actualisation des champs
    _Config.Fields.Update;
    //Liaison de la configuration au message
    _Message.Configuration:= _Config;
    //Définition des détails du message
    _Message.From:= Name; // Nom choisi par l'expéditeur
    _Message.&To:= EmailTo; // Adresse mail destinataire
    _Message.Subject:= Subject; // Sujet du message
    _Message.TextBody:= body; // Corps de texte du message
    _Message.HtmlBody:= body; //Corps de texte du message HTML
    //Si ajout de pièces jointes
    if ListeFichier.Count > 0 then
      begin
        for i:= 0 to ListeFichier.Count-1 do
          _Message.AddAttachment(ListeFichier[i]);
        end;
    //Tentative d'envoi du message
    try
      _Message.Send;
      Result:= True;
```

```
Except
  Result:= False;
end;
//Libération des ressources
finally
  VarClear(_Message);
  VarClear(_Config);
end;
end;
```

❶ Pour utiliser la configuration TLS 1.2, il faut activer les lignes de code relatives à ce protocole et désactiver les lignes précédentes correspondantes aux champs.

IV.C La procédure de paramétrage et de constitution du message pour l'envoi

Pour pouvoir envoyer un message, il est nécessaire de créer la procédure **EnvMail** qui sollicite la fonction **ConfMail** élaborée précédemment. Dans celle-ci, on paramètre les éléments relatifs au **serveur smtp** et on définit l'ensemble des composants qui recueillent le contenu d'un message.

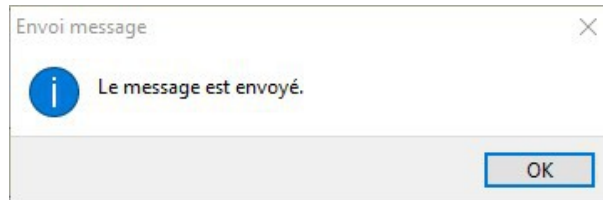
La procédure **EnvMail** :

```
//Procédure de paramétrage et de constitution du message pour l'envoi
procedure TFormess.EnvMail(Sender: TObject);
var
  Exp, Pswd, Serv, Nom, Dest, Subj, Mess: String;
  Prt, c, d: Integer;
  Pjte: TStrings;
begin
  //Paramétrage du serveur
  Exp:= 'monadresse@email.com'; //Adresse mail du compte de messagerie
  pswd:= 'monmotdepasse'; //Mot de passe du compte de messagerie
  Serv:= 'smtp.email.com'; //Serveur smtp
  Prt:= 465; //Port du serveur smtp
  //Contenu du message
  Nom := Ed1.Text+'<monadresse@email.com>'; //Nom choisi par
  l'expéditeur pour envoyer le message
  Dest:= Mm2.Text; //Adresse mail du ou des destinataires
  Subj:= Ed3.Text; //Sujet du message
  Mess:= Mm1.Text; //Corps de texte du message
  Pjte:= Lbx1.Items; //Pièces jointes
  //Envoi du message
  if ConfMail(Nom, Exp, Dest, Pswd, Subj, Mess, Serv, Pjte, Prt)= True then
  //Fonction configuration et envoi du message
  begin
    MessageDlg('Envoi message', 'Le message est envoyé.',
    mtInformation, [mbOk], 0);
```

```
for c:= 0 to Formess.ComponentCount - 1 do
  if Formess.Components[c] is TEdit then
    TControl(Formess.Components[c]).Caption:= ''; //Efface les
champs réalisés avec des composants TEdit
  for d:= 0 to Formess.ComponentCount - 1 do
    if Formess.Components[d] is TMemo then
      TControl(Formess.Components[d]).Caption:= ''; //Efface les
champs réalisés avec des composants TMemo
    Lbx1.Items.Clear; //Efface la liste des pièces jointes
  end;
end;
```

Dans cette procédure, on constate qu'il est possible de saisir un simple nom dans le champ (expéditeur) **De** : constitué par le composant **Ed1**. Si rien n'y est renseigné, le destinataire recevra le message de la part de l'adresse e-mail inscrite dans la procédure alors qu'à l'inverse, à la première lecture il verra le nom de l'expéditeur.

Lorsque le message est envoyé l'utilisateur en est informé :



Suite à l'envoi d'un message, les champs du formulaire sont effacés.

⚠ Du fait d'un renforcement de la sécurité réalisé par **Google**, aujourd'hui les utilisateurs d'une messagerie **Gmail** doivent utiliser un « **mot de passe d'application** » pour la configuration du serveur SMTP.

[Google : Se connecter avec des mots de passe d'application](#)

IV.D Les procédures de vérification des adresses e-mail

Avant qu'un message soit envoyé, l'ensemble des adresses e-mail des destinataires doivent être vérifiées. Pour faire cela, il y a deux fonctions. **IsValidEmail** permet de s'assurer que la composition d'une adresse e-mail est correcte et **VerifEmail** contrôle l'ensemble des adresses destinataires. Ces deux fonctions sont déclarées comme suit :

```
private
  function IsValidEmail(const Email: string): Boolean;
  function VerifEmail(Mm: TMemo): Boolean;
```


IV.D.1 La fonction de vérification de composition de l'adresse e-mail du destinataire

Cette fonction vérifie que le contenu du champ (destinataire) **Pour** : constitué par le composant **Mm2** possède les caractéristiques d'une adresse e-mail. Ainsi, nous élaborons la fonction **IsValidEmail** qui permet de contrôler la validité de l'adresse du destinataire.

Préalablement, afin de vérifier la composition d'une expression telle une adresse e-mail, il faut déclarer la clause **RegExpr** de la manière suivante :

```
implementation
{$R *.lfm}

uses
  RegExpr;
```

La fonction **IsValidEmail** :

```
//Fonction de vérification de composition de l'adresse e-mail du
destinataire
function TFormess.IsValidEmail(const Email: string): Boolean;
var
  Expr: TRegExpr;
begin
  Result:= False;
  Expr:=TRegExpr.Create;
  Expr.Expression := '^([A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]
{2,})$';
  Result:= Expr.Exec(Email);
end;
```

IV.D.2 La fonction de vérification des adresses des destinataires

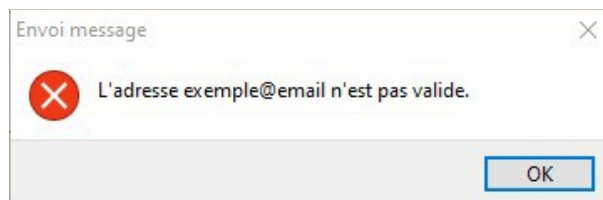
Comme il est possible d'envoyer un même message simultanément à plusieurs destinataires, il est alors nécessaire de vérifier l'ensemble des adresses e-mail de ces derniers. Pour faire cela, la fonction **VerifEmail** est apte à contrôler chacune des adresses et à les reconstruire après une éventuelle correction ou modification.

La fonction **VerifEmail** :

```
//Fonction de vérification des adresses des destinataires
function TFormess.VerifEmail(Mm: TMemo): Boolean;
var
  EmailList: TStringList;
  i, j: Integer;
```

```
EmailLine: string;
EmailsInLine: TStringList;
begin
  Result:= False;
  EmailList := TStringList.Create;
  try
    //Séparation les lignes du TMemo
    EmailList.Text := Mm.Text;
    //Création une liste pour stocker les adresses e-mail dans chaque
    ligne
    EmailsInLine := TStringList.Create;
    try
      for i := 0 to EmailList.Count - 1 do
        begin
          // Séparation des adresses e-mail de la ligne en utilisant le
          point-virgule comme séparateur
          EmailsInLine.Delimiter := ';';
          EmailsInLine.DelimitedText := EmailList[i];
          // Vérification de chaque adresse e-mail dans la ligne
          for j := 0 to EmailsInLine.Count - 1 do
            begin
              // Nettoyage des adresses e-mail en supprimant les espaces
              EmailLine := Trim(EmailsInLine[j]);
              // Vérification de la validité des adresses e-mail
              if IsValidEmail(EmailLine) then
                Result:= True
              else
                if MessageDlg('Envoi message','L'adresse e-mail ' +
                EmailLine + ' n'est pas valide.', mtError, [mbOk], 0) = mrOk then
                  abort;
            end;
          end;
        //Libération de la liste d'adresses
        finally
          EmailsInLine.Free;
        end;
      finally
        EmailList.Free;
      end;
    end;
end;
```

Pour chaque adresse e-mail défectueuse, l'utilisateur est prévenu par un message l'informant de l'adresse incriminée :



Dans ce message d'avertissement, on remarque qu'il manque quelque chose du même type que .fr

ou .com à la fin de l'adresse « exemple@email » pour qu'elle soit valide.

⚠ Il est possible de saisir l'adresse de plusieurs destinataires séparées par un point-virgule suivi d'un espace. Cependant, dans le cas d'une seule adresse saisie ou bien la dernière d'une liste, elles ne doivent pas être suivies d'un point-virgule sous peine de survenue d'un message de non validité d'adresse.

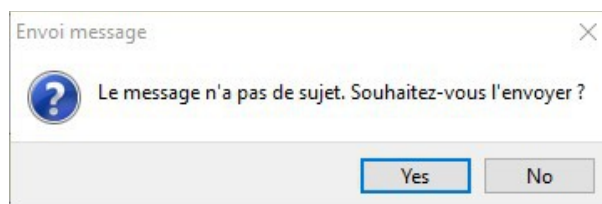
IV.E Les boutons : Envoyer, Quitter et Joindre

Un clic sur le bouton **Envoyer** sollicite d'abord la fonction **VerifEMail** pour vérifier les adresses destinataires, puis la vérification de la présence d'un sujet avant de mettre en œuvre **EnvMail** la procédure d'envoi du message.

Code du bouton **Envoyer** :

```
//Procédure pour envoyer le message
procedure TFormess.Btn1Click(Sender: TObject);
begin
  if VerifEMail(Mm2)= False then //Lance la vérification des adresses
e-mail
    Exit;
  if Ed3.Text = '' then //Teste si sujet du message absent
    if MessageDlg('Envoi message','Le message n''a pas de sujet.
Souhaitez-vous l''envoyer ?', mtConfirmation,[mbYes, mbNo],0) = mrNo
then
      Exit;
  EnvMail(Self); //Envoi du message
end;
```

On note que l'utilisateur peut accepter l'envoi du message même sans sujet. D'ailleurs, cela lui est demandé :



Si l'utilisateur clique sur **No**, la procédure est stoppée. Celui-ci peut alors saisir le sujet du message ou quitter l'application. S'il clique sur **Yes**, le message est envoyé sans sujet.

Le bouton **Quitter** sert à fermer l'application.

Code du bouton **Quitter** :

```
//Procédure pour fermer l'application
procedure TFormess.Btn2Click(Sender: TObject);
begin
  Close;
end;
```

La fonction **ConfMail** sollicitée dans la procédure **EnvMail** offre la possibilité d'expédier des pièces jointes. Un clic sur le bouton **Joindre** ouvre la boîte de dialogue **Opd1** pour choisir les fichiers à mettre en pièces jointes. Une fois sélectionnés, ceux-ci apparaîtront dans la liste **Lbx1**.

Code du bouton **Joindre** :

```
//Procédure pour ajouter des pièces jointes
procedure TFormess.Btn3Click(Sender: TObject);
begin
  if Opd1.Execute then //Ouverture de la boîte de dialogue pour
sélection des pièces jointes
    Lbx1.Items.Add(Opd1.FileName); //Ajout de pièces jointes
end;
```

A la fermeture de l'application :

```
//Procédure pour quitter l'application
procedure TFormess.FormClose(Sender: TObject; var CloseAction:
TCloseAction);
begin
  Application.Terminate; //Quitte "proprement" l'application
end;
```

V Utilisation du HTML

Définition du langage HTML (Source : [Wikipédia](#)) :

« Le **HyperTextMarkup Language**, généralement abrégé **HTML** ou, dans sa dernière version, **HTML5**, est le langage de balisage conçu pour représenter les pages web.

Ce langage permet d'écrire de l'hypertexte (d'où son nom), de structurer sémantiquement une page web, de mettre en forme du contenu, de créer des formulaires de saisie ou encore d'inclure des ressources multimédias dont des images, des vidéos, et des programmes informatiques. Le HTML offre également la possibilité de créer des documents interopérables avec des équipements très variés et conformément aux exigences de l'accessibilité du web. »

Pour plus d'informations, voir : <https://xhtml.developpez.com/>.

V.A Les messages

La configuration proposée dans la fonction **ConfMail** permet l'envoi de messages au format **HTML**. Pour essayer, faites un copier-coller dans le champ **Message** : du texte suivant codé en **HTML**, puis envoyez le à un destinataire de votre choix.

```
<p style = "font-family: arial, Helvetica, sans-serif;">
  <span style = "font-size: small;">Ce</span>
  <span style = "font-size: large; color: #339966; text-decoration:
underline;">texte</span>
  <span style = "font-size: small;">re&ccedil;u est un</span>
  <span style = "font-size: large; color:
#0000ff;"><em>exemple</em></span>
  <span style = "font-size: small;">de <strong>message transmis
en</strong></span>
  <span style = "font-size: large; color:
#ff0000;"><strong>HTML</strong></span><span style = "font-size: large;
color: #000000;"><span style = "font-size: small;">.</span></span>
</p>
```

Le message reçu par le destinataire est le suivant :

Ce texte reçu est un *exemple* de message transmis en **HTML**.

V.B Les liens hypertextes

Pour saisir les **liens hypertextes**, il vaut mieux le faire en **HTML** sinon il parviendront au destinataire en **texte brut**.

Pour essayer, faire un copier-coller dans le champ **Message** : du lien suivant, puis l'envoyer à un destinataire.

```
<a href="https://www.developpez.net/forums/f189/autres-
langages/pascal/lazarus/" target="_blank">Voir sur Developpez.com</a>
```

Le lien cliquable reçu par le destinataire est le suivant :

[Voir sur Developpez.com](https://www.developpez.net/forums/f189/autres-langages/pascal/lazarus/)

VI Le message permanent

L'objectif n'est pas nécessairement de créer un client de messagerie, mais plutôt de permettre à

une application d'expédier un message (peut-être toujours le même) par exemple à la suite ou en fin d'un traitement. Alors, il faut adapter ce qui a été précédemment étudié à un besoin spécifique autre. Les paragraphes suivants apportent un éclairage pour aider à réaliser cette adaptation.

VI.A Le champ message (Message :)

Dans le cadre d'une application qui doit toujours envoyer le même message, il est possible de créer un message permanent. Celui-ci peut être intégré dans le composant **TMemo : Mm1** au niveau de la propriété **Lines**. La possibilité de modifier ou d'enrichir le message peut-être conservée, sinon il faut faire passer la propriété **ReadOnly** du composant à **True**. Mais on peut aussi rendre **Mm1** invisible en passant la propriété **Visible** à **False**.

Il est aussi possible d'inscrire le message directement dans le code. Alors, il n'y a pas lieu de conserver **Mm1** et il convient de supprimer dans le code tout ce qui s'y réfère. Dans cette configuration, le message n'est pas modifiable par l'utilisateur.

VI.B Le champ expéditeur (De :)

En conservant l'expéditeur apparent, il est possible d'inscrire le nom au niveau de la propriété **Text** du composant **Tedit : Ed1** rendu non modifiable en passant la propriété **ReadOnly** à **True**, sinon le rendre invisible en passant la propriété **Visible** à **False**.

Il est aussi imaginable de ne pas avoir besoin de faire figurer au niveau de l'application le nom de l'expéditeur et seulement l'inscrire dans le code. Dans ce cas, il faut enlever **Ed1** et bien sûr tout ce qui y est lié.

VI.C Le champ destinataire (Pour :)

Dans le composant **TMemo : Mm2**. Ici encore la propriété **Lines** est utilisée pour inscrire la liste des adresses qui recevront le message. La possibilité de la modifier ou de l'enrichir peut-être conservée, sinon il faut passer la propriété du composant **ReadOnly** à **True**. Si on souhaite l'invisibilité de **Mm2**, il suffit de passer la propriété **Visible** à **False**.

Les destinataires peuvent aussi être fixés dans le code. Il n'est plus nécessaire de conserver le composant **Mm2** et ce qui s'y réfère doit être supprimé. Dans cette configuration, la liste des destinataires n'est pas modifiable par l'utilisateur.

VI.D Le champ sujet (Sujet :)

Pour le champ sujet, il faut procéder de la même manière que pour celui de l'expéditeur.

VI.E Les pièces jointes

On doit supprimer le bouton **Joindre** et le composant **TopenDialog: Opd1**. Il est possible de saisir les pièces jointes au niveau de la propriété **Items** du composant **TListBox : Lbx1**. Si on veut rendre **Lbx1** invisible, il suffit de passer la propriété **Visible** à **False**.

Les fichiers mis en pièces jointes peuvent aussi être prédéterminés dans le code.

VI.F Dans le code

Au niveau de la procédure **EnvMail**, il faut procéder comme suit pour passer la composition du message par le code.

```
//Contenu du message
Nom := 'Mon nom'+ '<monadress@email.com>'; //Nom choisi par
l'expéditeur pour envoyer le message
Dest:= 'adress.dest1@email.com; adress.dest2@mail.fr'; //Adresse e-
mail du ou des destinataires
Subj:= 'Mon sujet'; //Ed3.Text; //Sujet du message
Mess:= 'Mon message'; //Corps de texte du message
Lbx1.Items.Add(ExtractFilePath('Chemin du fichier 1')+ 'Nom du fichier
1 avec son extension');
Lbx1.Items.Add(ExtractFilePath('Chemin du fichier 2')+ 'Nom du fichier
2 avec son extension');
Lbx1.Items.Add(ExtractFilePath('Chemin du fichier 3')+ 'Nom du fichier
3 avec son extension');
Pjtc:= Lbx1.Items; //Pièces jointes
```

Au niveau du bouton **Envoyer**, les conditions de vérification des adresses des destinataires et de présence d'un sujet doivent être supprimées.

```
//Procédure pour envoyer le message
procedure TForm1.Btn1Click(Sender: TObject);
begin
  EnvMail(Self); //Envoi du message
end;
```

❗ Une configuration essentiellement basée sur le code comme abordée dans ce dernier paragraphe nécessitera de revisiter ce qui a été vu jusque là. Cela conduira à un allègement lié à la suppression de certains composants, procédures et fonctions.

VII Conclusion

Vous êtes parvenu au terme de ce tutoriel.

Certes la solution présentée ressemble à la partie d'un quelconque client de messagerie destinée à envoyer des messages, mais l'intérêt de ce tutoriel est plutôt de montrer comment mettre en œuvre la bibliothèque CDO de Microsoft dans une application pour Windows conçue avec Lazarus.

Vous voilà maintenant prêt pour intégrer au besoin dans vos créations ce code qui permettra l'envoi de message.

VIII Remerciements

Merci à [laurent_ott](#) et [f-leb](#) pour l'intérêt porté à ce tutoriel et leurs remarques constructives.