

Envoi et réception de mails avec le framework 2.0

par MORAND Louis-Guillaume ([Page perso de Louis-Guillaume MORAND](#))

Date de publication : 21/08/2005

Dernière mise à jour :

Ce tutoriel a pour but de vous présenter comment envoyer et recevoir des mails à l'aide du Framework 2.0

Introduction

- 1 - Envoi de mail
 - 2 - Réception de mail
 - 2.1 - Principe
 - 2.2 - Connexion
 - 2.3 - Réception
 - 3 - Téléchargements
- Conclusion
Remerciements

Introduction

La gestion de l'envoi et de la réception de mails sont des besoins récurrents dans bon nombre d'applications. Si vous n'y avez jamais été confronté, il y a de fortes chances que vous le soyez un jour, et dans ce cas, vous serez heureux d'apprendre que le framework .Net inclus des classes qui vous permettront d'implémenter facilement cette fonctionnalité.

Le but de ce tutoriel est donc de vous présenter les nombreuses possibilités offertes par .Net

1 - Envoi de mail

Dans cette première partie, nous allons voir ce que j'estime être le plus intéressant avec les mails: l'envoi. L'envoi de mail peut être utilisé de diverses manières: notification, envoi de rapport ou mail "personnel".

L'équipe Microsoft a bien tenu compte de ce besoin et ils ont développé pour nous, deux nouveaux espaces de nom qui facilitent l'envoi de mail.

Ces deux espaces de noms sont **System.Net.Mail** et **System.Net.Mime** qui permettent respectivement, de créer un objet Mail (*MailMessage*) et de gérer les entêtes MIME des mails.

Commençons donc par créer l'objet représentant notre mail:

Création de l'objet représentant le mail

```
MailMessage monMail;
```

Cette classe contient un nombre conséquent de propriétés permettant de personnaliser le mail comme vous le feriez avec votre client de messagerie. Commençons par définir l'auteur du mail. Ceci se réalise à l'aide de la classe *MailAddress*.

Ajout de l'auteur du mail

```
monMail.From = new MailAddress("pharaonix@developpez.com");
```

La deuxième chose à définir sont le(ou les) destinataire(s).

Ajout de destinataires

```
// ajout de deux destinataires principaux
monMail.To.Add(new MailAddress("morpheus@developpez.com"));
monMail.To.Add(new MailAddress("neo.51@developpez.com"));

// ajout d'un destinataire CC
monMail.Cc.Add(new MailAddress("titi@developpez.com"));

// ajout d'un destinataire CCI
monMail.CCi.Add(new MailAddress("webman@developpez.com"));
```

Comme vous le voyez, il est très facile de spécifier les destinataires du mail et surtout d'en définir plusieurs tout en spécifiant à quel groupe de réception ils appartiennent.

Spécifions maintenant le sujet(titre) du mail ainsi que son contenu

Ajout du sujet et du contenu

```
monMail.Subject = "Mail de test";
monMail.Body = "Hello les gens";
```

Votre mail est alors prêt à être envoyé: ce que nous allons faire tout de suite.

Créons un objet *SMTPClient* et configurons le:

Création/Personnalisation du SMTPClient

```
// création du SMTPClient
SmtpClient client = new SmtpClient();

// définition du serveur smtp
client.Host = "smtp.developpez.com";

// définition des login et pwd si smtp sécurisé
client.Credentials = new NetworkCredential("pharaonix", "mon mot de passe");
```

Il ne nous reste plus qu'une chose à faire: envoyer le mail.

Envoi du mail

```
client.Send(monMail);
```

Je vous avais promis précédemment qu'il était possible d'inclure des pièces jointes à vos mails. vous allez constater que leur utilisation est tout aussi simple.

Chaque pièce jointe est représentée par un objet *Attachment*

Pièces jointes

```
// création de la pièce jointe
Attachment maPieceJointe = new Attachment(@"c:/fichier.zip"); // chemin de la pièce jointe

// ajout de la pièce jointe au mail
monMail.Attachments.Add(maPieceJointe);

/*
// suppression de la pièce jointe
monMail.Attachments.RemoveAt(0);
*/
```

Vous ne pouvez qu'admirer le travail fait par les développeurs Microsoft et apprécier la facilité avec laquelle vous pourrez dorénavant envoyer n'importe quel type de mail dans vos applications .Net



2 - Réception de mail

2.1 - Principe

Je n'ai pas trouvé sur la MSDN de classe spécifique pour la réception de mail, mais il existe des composants tiers sur le net (payants) qui proposent des classes facilitant cette tâche. Nous allons voir comment se passer de ces classes tierces et comment recevoir les mails le plus simplement possible.

La récupération de mail en POP3 se fait à l'aide d'une communication client/serveur. En effet, le client(logiciel de messagerie) et le serveur s'envoient une suite bien définie de commandes et/ou de réponses correspondantes.

Qui	Action
Client	connexion avec le socket
Serveur	+OK connected to pop3
Client	USER pharaonix
Serveur	+OK name is a valid mailbox
Client	PASS mon_mot_de_passe
Serveur	+OK user exist with that password
Client	STAT
Serveur	+OK [Nombre de Messages] [Taille]
Client	RETR [Numero Du Mail]
Serveur	+OK
	Données du mail (entête+contenu)
Client	QUIT
Serveur	+OK

Notez que j'indique le serveur renvoie "+OK" ainsi qu'un message de réponse. Dans le cas d'une erreur, le serveur renverra "-ERR" et un message de réponse. Toutes ces commandes, ce fonctionnement, cette nomenclature de messages, sont définis dans la  [norme rfc1939](#) ou la  [norme rfc1725](#).

Et dans la pratique? Et bien, nous allons devoir suivre ces étapes une par une.

2.2 - Connexion

Commençons par nous connecter au serveur:

```
private TcpClient monSocket;
monSocket = new TcpClient("pop.developpez.com", 110); // 110 est le port par défaut pour les
serveurs POP3
```

La communication client/serveur se fera à l'aide d'un flux (stream):

```
private NetworkStream monStream;
```

Nous initialisons notre *Stream* à l'aide du stream de notre socket. Profitons en pour créer un objet *StreamReader* qui nous permettra de lire le contenu de notre Stream, tout du long de la communication avec le serveur POP3. C'est dans ce dernier que nous recevrons les messages de réponse du serveur ainsi que les mails.

```
// initialisation du Stream
monStream = monSocket.GetStream();

// création du streamreader
private StreamReader monStreamReader;
monStreamReader = new StreamReader(monStream);
```

Maintenant que nous avons une "connexion" avec le serveur, nous allons nous y authentifier. Nous allons préparer les commandes et les envoyer au serveur.

```
// Préparation de la commande USER avec le nom d'utilisateur
String Commande = "user " + _Login + "\r\n";

// Envoi de la commande au serveur
Byte[] bCommande = System.Text.ASCIIEncoding.ASCII.GetBytes(Commande);
monStream.Write(bCommande, 0, bCommande.Length);
```

Lorsque nous nous sommes connecté au serveur, celui nous a renvoyé la réponse suivante: "+OK connected to pop3 on 1012", puis nous lui avons envoyé la commande USER à laquelle il a répondu (car l'utilisateur existe vraiment): +OK name is a valid mailbox.

L'étape suivante consiste à envoyer le mot de passe du compte pour être reconnu et authentifié sur le serveur:

```
// Préparation de la commande PASS avec le mot de passe utilisateur
String Commande = "pass " + _Pwd + "\r\n";

// Envoi de la commande au serveur
Byte[] bCommande = System.Text.ASCIIEncoding.ASCII.GetBytes(Commande);
monStream.Write(bCommande, 0, bCommande.Length);
```

Si le login et surtout le mot de passe sont corrects, alors vous recevrez le message: "+OK user exist with that password". Vous pouvez dorénavant travailler comme bon vous semble avec le serveur.

2.3 - Réception

Nous allons maintenant aborder la réception des mails. Celle-ci se fait en deux principales étapes. La première consiste à demander au serveur s'il contient des mails. Nous pouvons la réaliser à l'aide de la commande STAT, puis nous parserons la réponse du serveur

```
/// <summary>
/// Récupère le nombre de mails sur le serveur, en parsant la réponse du serveur du type "+OK Nbre
/// Taille"
/// </summary>
/// <param name="mode"></param>
/// <param name="index"></param>
/// <returns></returns>
private int NombreDeMessages(int mode, params int[] index)
{
    String sOutStream = "";
    if (mode == 1)
```

```
sOutputStream = "stat\r\n";
else if (mode == 2)
    sOutputStream = "list " + index[0].ToString() + "\r\n";

String[] tempS = { "0" };
try
{
    EnvoiCommande(sOutputStream);
    String tempLog = sr.ReadLine();
    tempS = tempLog.Split(' ');
}
catch (Exception e) {
    MessageBox.Show("erreur" + e);
}
return int.Parse(tempS[1]);
}
```

La méthode précédente nous renvoyait donc le nombre de mails qui se trouvent sur le serveur et la taille totale prise par ceux-ci, puis nous la parsions pour ne récupérer que le nombre de mails

Avant de démarrer, il faut bien prendre en compte le fait suivant : le traitement de réception peut être assez complexe, pour les raisons suivantes :

- même s'il existe des normes, tous les mails ne les respectent pas forcément,
- les possibilités de contenu d'un mail sont très vastes.

Ne vous étonnez donc pas que votre réception de mail ne soit pas aussi parfaite que celle d'un client de messagerie comme Microsoft Outlook ou Thunderbird.

Dans cette partie, je vais donc vous présenter une manière simple de récupérer les mails et les informations associées. Il serait bien entendu possible de développer cette approche d'une manière beaucoup plus exhaustive, mais ce n'est pas le but de cette présentation.

Nous allons maintenant recevoir les mails. Cela ne se fait qu'en les recevant **un par un**, en les "appelant" par leur numéro.

Récupération des messages un par un

```
#region Méthode : RecupereListeMessages
private void RecupereListeMessages()
{
    for (int i = 1; i <= inbMsg; i++)
    {
        lblStatus.Text = "Retrieve Message " + i.ToString() + " ...";
        int intSizeMsg = NombreDeMessages(2, i);
        EnvoiCommande("retr " + i.ToString() + "\r\n");
        parseMail(intSizeMsg);
    }
}
#endregion
```

Nous bouclons donc pour récupérer les mails les uns après les autres. Voyons maintenant comment récupérer chaque mail:

```
#region Méthode : ParseMail
/// <summary>
/// Rempli la listview listant la liste détaillée des mails
/// </summary>
/// <param name="size"></param>
private void parseMail(int size)
{
    string szTemp = sr.ReadLine();
    string szSubject = "";
    string szDate = "";
    int temp;

    if (szTemp != "-")
    {
        szTemp = sr.ReadLine();
        string szSender;
        if ((temp = szTemp.IndexOf("<")) != -1)
            szSender = szTemp.Substring(temp);
        else
            szSender = "Inconnu|Erreur";

        System.Windows.Forms.ListViewItem curItem = lvMails.Items.Add("");
        szTemp = sr.ReadLine();
        while (szTemp != ".")
        {
            if (szTemp.IndexOf("Date:") != -1)
                szDate = szTemp.Substring(5, szTemp.Length - 5);
            if (szTemp.IndexOf("Subject:") != -1)
                szSubject = szTemp.Substring(8, szTemp.Length - 8);
            szTemp = sr.ReadLine();
        }
        curItem.SubItems.Add(szSender.Replace("<", "").Replace(">", ""));
        curItem.SubItems.Add(szSubject);
        curItem.SubItems.Add(Convert.ToDateTime(szDate).ToShortDateString());
        curItem.SubItems.Add(Convert.ToDateTime(szDate).ToShortTimeString());
    }
}
#endregion
```

Le code ci-dessus, appelé pour chaque rapatriement de mail (commande RETR), lit le mail ligne par ligne pour en ressortir certaines informations. Celles-ci sont contenues dans l'entête du mail. Or celui-ci est accolé au texte du mail lui-même. Nous devons donc détecter les mots clés (Subject, Date, etc.) afin d'en récupérer l'information correspondante.

Cette méthode permet simplement de remplir une listview, en affichant la liste des mails et quelques informations sur ceux-ci:

Dans ma source (fournit en fin de cet article), je ne "récupère" (et ne traite) les mails que lorsque l'utilisateur sélectionne un mail dans la liste des mails. En effet, n'ayant pas de base de données où les stocker, je dois les redemander au serveur mail, pour les traiter et les afficher dans la partie visualisation (une simple TextBox pour l'exemple):

```
#region Méthode : RecupereMail
/// <summary>
/// Récupère le "source" de l'email sélectionné
/// </summary>
/// <param name="NumMail"></param>
private void RecupereMail(int NumMail)
{
    EnvoiCommande("retr " + NumMail.ToString() + "\r\n");
    string szTemp = sr.ReadLine();
}
```

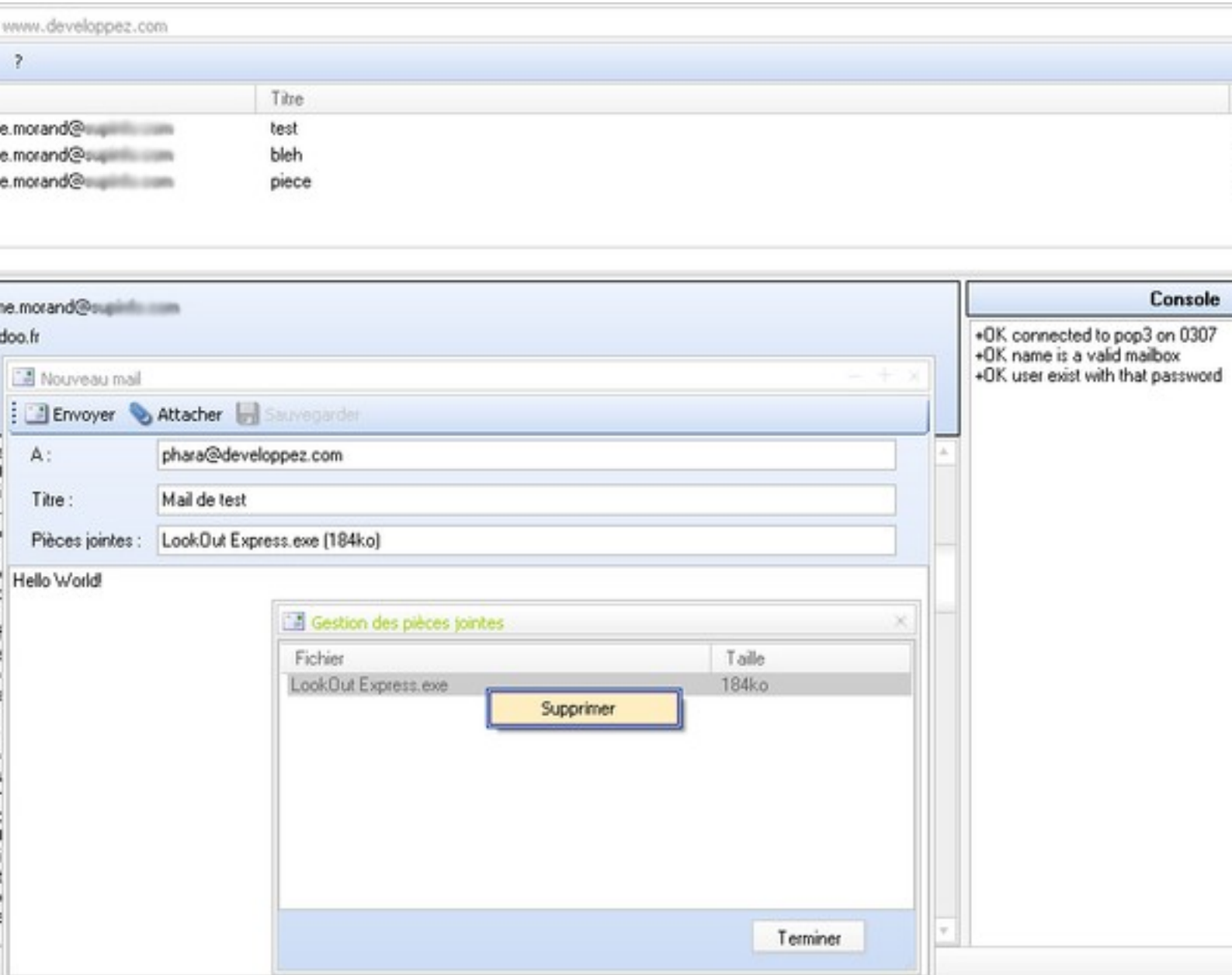
```
while (szTemp != ".")// fin du mail
{
    if (szTemp.Length != 0)// on ne travail par sur les lignes vides (sauts de ligne)
    {
        // on ajoute la ligne dans la textbox d'affichage du mail.
        // Si le dernier caractère est le symbole "=" alors cela représente un saut de ligne que l'on
        // doit rajouter.
        if (szTemp[szTemp.Length - 1].Equals('='))
            txtDetails.AppendText(szTemp.Substring(0, szTemp.Length - 1));
        else txtDetails.AppendText(szTemp + "\r\n");
    }
    szTemp = sr.ReadLine(); // lit la ligne suivante
}
```

En utilisant la méthode précédente, vous afficherez tel quel le mail que vous renvoi le serveur mail. En regardant de plus près vous vous rendrez compte qu'il contient l'entête plus le contenu du mail plus les pièces jointes et dans certains cas, il contient une version HTML et une version Text du mail.

Les clients de messagerie comme Microsoft Outlook ou Mozilla Thunderbird doivent alors faire un gros traitement dessus pour ne vous afficher que le contenu du mail dans un seul format (html ou text), en vous séparant les informations d'entête et les pièces jointes. Dans la source que je vous propose, je fais la même chose sans avoir un résultat parfait, car le traitement de l'entête diffère selon le client messagerie de l'émetteur du mail.

Comme dit plus haut, un mail est contenu dans un seul flux mais a un architecture bien particulière. Il contient l'entête et le contenu du mail **sans** séparation stricte, et il est donc difficile de ne récupérer que le corps du mail. Pis encore, un mail peut contenir plusieurs versions du contenu: la version texte et la version HTML. Et pour finir, les pièces jointes sont intégrées au mail mais encodés en base-64. Autant dire que le traitement du mail peut être compliqué et que le traitement "à la volée" (en une seule fois pendant la récupération) peut devenir extrêmement lourd. C'est pourquoi, je vous conseille de récupérer le mail, de le stocker en local (dans un fichier ou une base de données) et de "travailler" dessus pour en récupérer les informations importantes (contenu, pièce jointe, entête).

3 - Téléchargements



Afin de voir le code en "état de marche", je vous propose un client mail "complet":

- réception des mails
- récupération des informations du mail
- récupération de l'entête du mail (bêta)
- envoi de mail avec pièces jointes
- configuration des serveurs (pop, smtp)

Télécharger LookOut Express (200ko)

Conclusion

Comme vous avez pu le voir, la réception des mails reste une problématique non résolue dans cette seconde version du Framework (ou alors je ne l'ai pas trouvé :)), par contre, l'envoi de mail devient une facilité incomparable grâce au travail des développeurs Microsoft, et vous n'aurez plus aucune excuse de ne pas l'intégrer dans vos applications.

Remerciements

Un rôh merci à **Oliver Delmotte (titi)** qui m'a beaucoup aidé à analyser et debugguer les différents mails des différents clients/webmail, et également merci à **Xavier Vlieghe (Xo)** pour ses corrections/conseils.

