Les contrôles

ActiveX

NB : ce fascicule fait partie d'un travail de diplôme sur le sujet « Technologie ActiveX & Visual Basic 6 ».
Les autres fascicules peuvent être demandés à fcomte@caramail.com.

Critiques, remarques, questions ? fcomte@caramail.com

1 - Caractéristiques

D'un point de vue général, un contrôle ActiveX a les caractéristiques suivantes :

- Il peut réaliser les mêmes choses qu'un serveur DLL in-process.

- Il peut être contenu dans un conteneur. Cela signifie que le contrôle possède sa propre interface utilisateur contenue dans la fenêtre d'une application, dans une feuille de Visual Basic, ou dans une page Web. Le contrôle est capable d'interagir avec l'application qui le contient de plusieurs manières.

- Il supporte les caractéristiques de programmation en cours de conception, à savoir :

    - supporter les pages de propriétés,

    - être habilité à conserver des valeurs de propriétés réglées en cours de conception à une localisation spécifiée par le conteneur,

    - être habilité à interagir avec les propriétés du conteneur si il y en a un.

En d'autres termes, schématiquement :

Un contrôle ActiveX = un composant de code + une interface utilisateur + les caractéristiques de son conteneur.

1.1 Avantages des contrôles ActiveX :

→ De bonnes performances. L'exécution se fait « in-process ».
→ Les contrôles sont compatibles avec de multiples conteneurs, telles les applications Microsoft Office ou les navigateurs Internet.
→ Le développement des contrôles est simplifié dans l'environnement de Visual Basic.

1.2 Inconvénients des contrôles AcrtiveX :

→ Les contrôles sont bien plus rapides que les ActiveX EXE, mais de loin pas aussi rapides que les DLL.
→ La création d'un contrôle de qualité peut s'avérer plus complexe qu'on ne le pensait au départ.
→ Les contrôles augmentent la complexité du déploiement d'une application.
→ L'enregistrement, la vérification de version, ainsi que la vérification des composants sont requis pour une distribution sûre du produit.

2 - Limitations des contrôles ActiveX

Les contrôles ActiveX créés avec Visual Basic s'exécutent sous les plates-formes Windows 95, 98, et 2000, NT 3.51 (SP 5 et plus), et NT 4.0. Voyons ce que cela signifie au niveau du choix de cette technologie.

Les contrôles ActiveX seront appropriés si :

→ On désire développer une application s'exécutant sur les plates-formes énoncées, grâce à Visual Basic ou à tout autre conteneur (ou encore une application VBA) qui peut supporter des contrôles ActiveX.

→ On désire créer un site Web orienté vers les utilisateurs de ces plates-formes, en admettant que ceux qui ne le font pas seront lésés au niveau des fonctionnalités du site.

→ On désire créer un site Web pour un intranet d'une société utilisant les technologies Microsoft.

Par contre, les contrôles ActiveX ne seront pas appropriés si :

→ Notre application doit supporter un mode d'exécution 16-bits.

→ Nous désirons créer un site Web multi-plates-formes, incluant Apple, UNIX, et Windows NT (en dessous de la version 3.51 avec SP 5). Dans ce cas là, il vaut bien mieux songer à développer sous JAVA.

3 - Une cacophonie d'objets...

Un grand nombre d'objets est utilisé durant le processus de conception et d'exécution d'un contrôle ActiveX, nous allons parcourir les principaux, à savoir :

→ votre objet Control
→ l'objet UserControl
→ l'objet Ambient
→ l'objet Extender
→ les objets constitutifs du contrôle (Constituent objects)

Nous allons créer un contrôle simple, ce qui nous permettra de suivre le processus d'utilisation de ces différents objets. Ce contrôle sera un contrôle privé qui sera inclus dans une application EXE standard. On ne pourra donc pas compiler de fichier .OCX public[1] (c'est d'ailleurs la seule différence entre un contrôle public et un contrôle privé).

Créons un projet standard EXE, et ajoutons-y un contrôle utilisateur. Changeons le nom du projet en `PrjCtl`, et celui du contrôle `UserControl1` en `MonControle`.

[1] Lorsque le projet a été compilé, les contrôles privés ne peuvent pas être utilisés par d'autres applications. Ils ne peuvent être utilisés qu'à l'intérieur du projet dans lequel ils ont été compilés.

Dans la fenêtre du code de `MonControle`, saisissons le code suivant :

```
Dim m_MaProp As String

Public Property Get MaProp() As String
      MaProp = m_MaProp
End Property

Public Property Let MaProp(vValeur As String)
    m_MaProp = vValeur
End Property
```

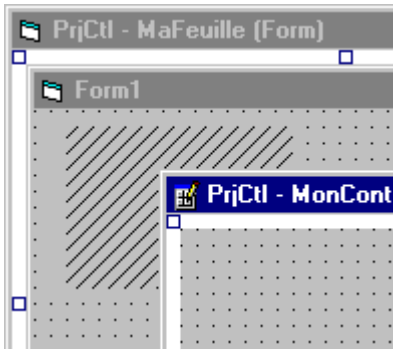
Et changeons le nom de la feuille du projet en `MaFeuille` ; ce sera la feuille qu'on utilisera pour tester le contrôle.

Si on regarde dans la boîte à outils, on découvre un icône grisé associé à notre contrôle.

Lorsque nous fermons la fenêtre du contrôle, cet icône n'est plus grisé. Créons donc une instance de notre contrôle sur la feuille `MaFeuille`. Le nom de cet instance est déterminé par le nom que nous avions donné au contrôle, soit `MonControle1`.

Voyons les propriétés de cette instance : `MaProp` figure dans la liste, ainsi qu'une quantité d'autres propriétés. Celles-ci sont nommées propriétés « étendues » (extender properties), car elle sont implémentées dans un objet externe fourni par le conteneur (dans notre cas, Visual Basic).

Sans fermer la fenêtre de conception de la feuille, ouvrons le fenêtre de conception du contrôle, et observons la feuille : l'emplacement du contrôle sur celle-ci est couvert de lignes diagonales indiquant que le contrôle n'est pas disponible.



Saisissons le code événementiel suivant dans la fenêtre du code du contrôle :

```
Event Click()
Private Sub UserControl_Click()
    RaiseEvent Click
End Sub
```

Fermons la fenêtre de code ainsi que la fenêtre de conception du contrôle, et ajoutons dans le code de la feuille :

```
Private Sub MonControle1_Click()
    MsgBox "MonControle a été cliqué !"
End Sub
```

Exécutons maintenant le projet, en vérifiant que la feuille `MaFeuille` est la feuille de démarrage. Cliquons sur la zone où le contrôle a été placé (sur la feuille `MaFeuille`), une boîte de dialogue apparaît, telle que celle de la figure ci-contre.

Stoppons le projet et ramenons la fenêtre de conception du contrôle au premier plan. Ajoutons maintenant un bouton au contrôle :

```
Event ClicBouton()
Private Sub MonBouton_Click()
    RaiseEvent ClicBouton
End Sub
```

Ainsi que le code suivant dans la feuille `MaFeuille` :

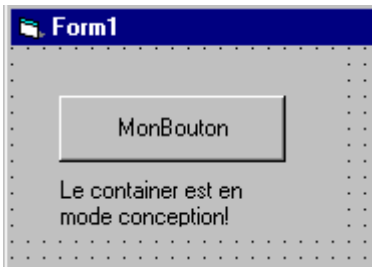
```
Private Sub MonControle1_ClicBouton()
    MsgBox "MonBouton a été cliqué !"
End Sub
```

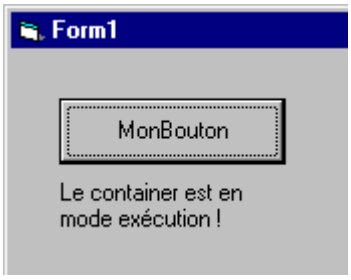
Exécutons le projet, et cliquons aussi bien sur la zone du contrôle que sur le bouton. Du point de vue du programmeur, on reçoit alors deux événements `Click` gérés par deux interfaces événementielles COM, une depuis l'objet `UserControl`, et l'autre depuis le bouton, qui est un contrôle constitutif.

Ajoutons une étiquette au contrôle ainsi que le code suivant dans l'événement `UserControl_Paint()` :

```
Private Sub UserControl_Paint()
    If Ambient.UserMode Then
        Lbl1.Caption = "Le container est en mode exécution !"
    Else
        Lbl1.Caption = "Le container est en mode conception!"
    End If
End Sub
```

Fermons le concepteur et voyons la feuille en mode conception.





Exécutons le projet et voyons à nouveau la feuille.

Ceci illustre bien le comportement adaptatif d'un contrôle ActiveX en mode de conception ou en mode d'exécution. Il utilise pour cela l'objet `Ambient`, qui contient des informations à propos de l'environnement du conteneur.

Voyons maintenant les cinq types d'objets que nous avons utilisé (quelques-uns sans le savoir).

3.1 L'objet `Control`

La seule partie du contrôle que l'on implémente en tant qu'auteur du contrôle est l'objet contrôle lui-même, dans notre cas, l'objet `MonControle`. Cet objet a deux buts : il définit les méthodes publiques, propriétés et événements disponibles à ceux qui utilisent le contrôle, et il définit le comportement du contrôle.
En plus des méthodes et propriétés que l'on implémente, nous avons accès aux propriétés des objets `UserControl`, `Extender`, `Ambient` et de tous les contrôles constitutifs que l'on ajoute à notre contrôle.

3.2 L'objet `UserControl`

Du point de vue du programmeur, l'objet `UserControl` est un sous-objet de notre contrôle. On accède aux méthodes de l'objet `UserControl` en utilisant la syntaxe `UserControl.method()`. L'objet `UserControl` peut lever des événements dans notre objet `Control` auquel on peut répondre dans le code de celui-ci[2].

Du point de vue d'une application Visual Basic, l'objet `Control` est en quelque sorte un sous-objet de l'objet `UserControl`. Ceci parce que Visual Basic sait comment placer et gérer l'objet `UserControl` sur un conteneur. Visual Basic – avec Windows – dirige la souris et les entrées du clavier vers l'objet `UserControl`, qui envoie ainsi les événements vers le code du contrôle.

L'objet `UserControl` est responsable de la création de la fenêtre du contrôle, des activations/désactivations des éléments du contrôle (focus), et tout ce qui concerne les interactions de l'interface utilisateur. Toutes ces activités sont ensuite exposées à l'objet `Control` au travers des événements et des propriétés.

L'objet `UserControl` est l'objet par défaut du contrôle. Pour bien le saisir, voyons un exemple concret : admettons que l'on travaille avec la propriété `Caption` d'une feuille. Admettons maintenant que nous définissons une nouvelle propriété nommée `Caption` sur cette même feuille, cette nouvelle propriété va cacher la propriété `Caption` par défaut fournie par la feuille : les variables définies localement cachent les variables de même nom définies globalement. Tout comme les propriétés et méthodes locales cachent leur homologues définies globalement. Mais il est toujours possible d'accéder à la variable globale `Caption`, en la préfixant explicitement : `Form.Caption`.

[2] Notez que les événements ne sont pas levés dans les instances du contrôle que les développeurs créeront par la suite, à partir du contrôle. Si vous désirez exposer aux développeurs un événement tel que `UserControl_Click`, vous devrez définir un nouvel événement dans votre objet `Control` et lever l'événement vous-même.

3.3 L'objet `Ambient`

Cet objet est utilisé pour obtenir des informations du conteneur. Les propriétés de cet objet permettent de déterminer quelles sont les possibilités offertes par le conteneur et adapter en conséquence le contrôle pour les utiliser. Par exemple, nous avons utilisé la propriété `UserMode`, qui permettait de déterminer si le conteneur était en mode de conception ou d'exécution `(Ambient.UserMode)`. Un autre exemple est la propriété `BackColor`, qui permet d'obtenir la couleur de fond du conteneur. Ceci peut être utile dans le cas où on désire obtenir depuis un contrôle cette couleur de fond (afin de l'ajuster par exemple). L'objet `UserControl` fournit l'événement `AmbientChanged`, qui permet ainsi d'être informé lorsque la propriété `BackColor` est modifiée.

Les conteneurs peuvent définir leurs propres propriétés ambiantes et ne sont pas forcés de les implémenter (y compris les propriétés des conteneurs visibles dans l'explorateur d'objets de Visual Basic ainsi que ceux de la documentation de VB). Pour faciliter la vie du programmeur, l'objet `Ambient` détecte les propriétés du conteneur manquantes et retourne simplement la valeur par défaut de la propriété. Cela signifie-t-il donc qu'un conteneur n'est pas obligé d'implémenter une propriété de base telle que `UserMode` ? Hé non. Si elle n'est pas implémentée, comment le contrôle peut-il alors se rendre compte si il se trouve en mode de conception ou pas ? Il ne le peut pas. Certains conteneurs de toute façon ne distinguent pas de différence entre le mode de conception et le mode d'exécution. Ceux-ci n'ont pas besoin d'exposer la propriété `UserMode`, auquel cas l'objet `Ambient` retournera par défaut la valeur `true`.

3.4 L'objet `Extender`

Il existe des propriétés associées aux contrôles qui n'ont absolument rien à voir avec le fonctionnement de ceux-ci. Par exemple, chaque contrôle dans Visual Basic possède une propriété `Name` : celle-ci est gérée par le conteneur. D'autres propriétés qui tombent dans le même panier sont les propriétés `Visible`, `Left`, ou `Right`.
Lorsqu'un développeur utilise notre contrôle, le conteneur ajoute automatiquement ces propriétés à l'interface de notre contrôle afin que le développeur puisse y accéder. Ce sont elles, les propriétés `Extender`. Le développeur est incapable de faire la différence entre les propriétés fournies par le conteneur et celles fournies par le contrôle. Par contre, l'auteur du contrôle si. De plus, pas besoin d'implémenter les propriétés `Extender` (en fait, on ne peut pas le faire). Mais il est possible d'accéder aux propriétés `Extender` fournies par le conteneur en utilisant l'objet `Extender`.

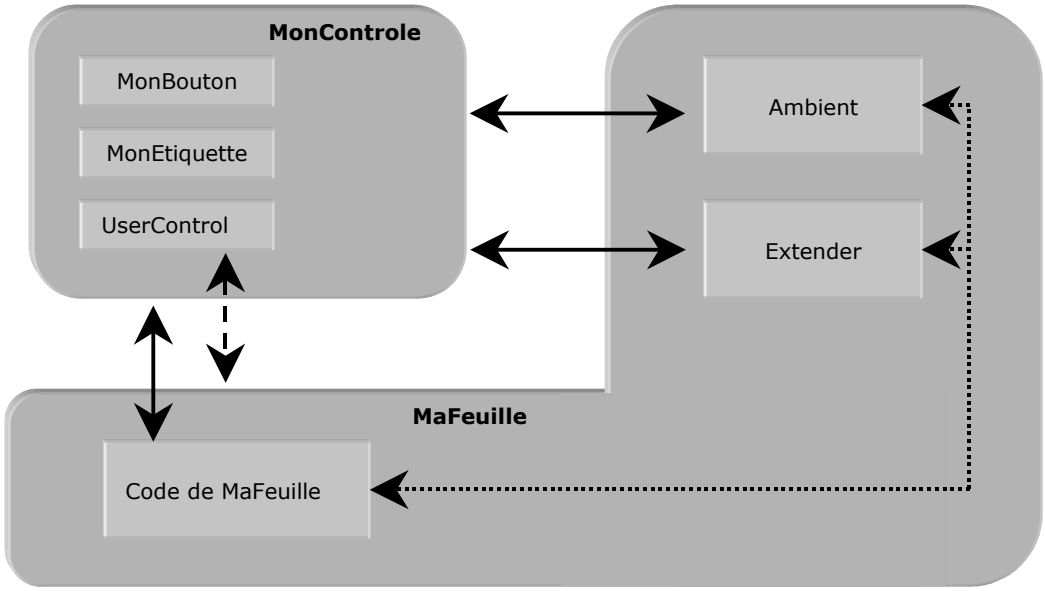
Comme il n'est pas possible de connaître les propriétés d'un conteneur à l'avance, l'accès à celles-ci se fait toujours avec par liaison tardive.

3.5 Les objets constitutifs d'un contrôle

D'une certaine manière, ce sont les objets les plus faciles à manipuler. Ils fonctionnent exactement de la même manière qu'un contrôle placé sur une feuille, et sont placés sur un contrôle comme un contrôle l'est sur une feuille. On y accède de la même manière, et les événements sont levés de manière identique. Mais attention : il faut se souvenir que les développeurs qui utiliseront notre contrôle n'auront pas accès aux propriétés, méthodes et événements de ces contrôles constitutifs, à moins que nous ne les exposions explicitement.

3.6 Résumé des objets d'un contrôle ActiveX

La figure suivante résume le modèle objet du contrôle ActiveX que nous avons réalisé précédemment. Le module de gauche représente le contrôle, incluant l'objet `UserControl` et les contrôles constitutifs, et celui de droite indique les objets en relation avec le conteneur.



Relation inter-objets dans un contrôle ActiveX

4 Événements clés de l'objet `UserControl` (cycle de vie)

La signification des événements clés dans la vie d'un objet `UserControl` est la suivante :

- Vous créez une instance d'un contrôle en double-cliquant sur la boîte à outils ou en ouvrant une feuille sur laquelle une instance du contrôle a été préalablement placée. Les contrôles constitutifs (si il y en a) sont créés. L'objet `UserControl` est créé, les contrôles constitutifs y sont situés.

- L'événement `Initialize` se produit à chaque fois qu'une instance de votre contrôle est créée ou recréée. C'est toujours le premier événement de la vie d'une instance de contrôle.

- L'événement `InitProperties` se produit seulement lors de la première apparition d'une instance de contrôle, lorsqu'une instance du contrôle est placée sur une feuille. Dans cet événement, vous définissez les valeurs initiales des propriétés du contrôle.

- L'événement `ReadProperties` se produit lors de la création de la deuxième instance du contrôle, et lors de chacune des nouvelles créations suivantes. Dans cet événement, vous récupérez les valeurs de propriétés de l'instance du contrôle dans la copie mémoire du fichier .frm appartenant à la feuille sur laquelle le contrôle a été placé.

- L'événement `Resize` se produit à chaque fois qu'une instance d'un contrôle est recréée, et à chaque fois qu'elle est redimensionnée soit en mode conception, par le développeur d'une feuille, soit dans le code, au moment de l'exécution. Si votre objet `UserControl` contient des contrôles constitutifs, vous les organisez dans la procédure relative à cet événement, ce qui définit l'apparence de votre contrôle.

- L'événement `Paint` se produit à chaque fois que le conteneur demande au contrôle de se dessiner lui-même. Cela peut intervenir à tout moment, même avant la réception par le contrôle de son événement `Show` (par exemple, si une feuille cachée s'imprime). Pour les contrôles personnalisés, l'événement `Paint` se produit lorsque vous dessinez l'apparence de votre contrôle. S'il n'y a aucun contrôle constitutif, l'objet `UserControl` se dessine lui-même.

- Vous lancez l'exécution (F5). Visual Basic ferme la feuille.

- L'événement `WriteProperties` se produit lorsqu'une « instance-création » de votre contrôle est détruite, dès lors qu'au moins l'une des valeurs de propriétés a changé. Dans cet événement, vous sauvegardez toutes les valeurs de propriétés définies par un développeur pour l'instance du contrôle. Les valeurs sont écrites dans la copie mémoire du fichier .frm.

- L'événement `Terminate` se produit lorsque le contrôle est sur le point d'être détruit. L'objet `UserControl` et ses contrôles constitutifs sont détruits.

5 – Exemple de contrôle ActiveX

5.1 Introduction

L'auteur voulait créer un programme permettant d'envoyer des mails ultra-rapidement depuis le bureau Windows. C'est chose faite avec Mail Now !, une sorte d'Outlook accéléré. L'idée était de limiter la quantité d'informations à saisir au strict minimum, soit une adresse de serveur SMTP, une adresse mail d'émetteur, une autre de destinataire, le sujet du mail et le corps. Ces paramètres sont les paramètres minimaux de SMTP, comme nous le verrons plus loin. Le contrôle MailNow.ocx utilise un contrôle constitutif fourni par Microsoft dans VB, Winsock, et il est testé dans un projet qui charge l'application dans le systray (la partie de droite de la barre des tâches), afin de pouvoir y accéder en tout temps.

5.2 Incursion dans l'univers des télécommunications…

Le contrôle que nous allons développer nécessite de comprendre certains mécanismes de télécommunications. Nous allons donc quitter momentanément l'univers microsoftien de COM, et se pencher sur les protocoles TCP et SMTP.

Utilisation du contrôle Winsock 

Le contrôle Winsock – qui n'a pas d'interface visible à l'exécution – permet d'accéder facilement aux services de réseau TCP (*Transfer Control Protocol*) et UDP (*User Datagram Protocol*). Il suffit, sans avoir à connaître tous les détails des connexions, de définir les propriétés et d'invoquer les méthodes du contrôle pour assurer la connexion vers une machine distante et pouvoir échanger des données dans l'une ou l'autre des directions.

L'application que nous créerons nécessitant une connexion, nous utiliserons le protocole TCP, dont nous allons parler maintenant.
TCP établit et de maintient une connexion vers un ordinateur distant, permettant ainsi aux deux ordinateurs d'échanger des données. Pour créer une application cliente, nous devons connaître le nom de l'ordinateur serveur (ou son adresse IP, propriété *RemoteHost*), ainsi que le port (propriété *RemotePort*) sur lequel il sera « à l'écoute », puis invoquer la méthode *connect*. Nous ne nous occuperons pas de l'application serveur, notre application se limitant à envoyer des mails.
Une fois la connexion réalisée, chacun des ordinateurs peut envoyer (méthode *SendData*) et recevoir (méthode *GetData*, qui suit l'événement *DataArrival*) des données.

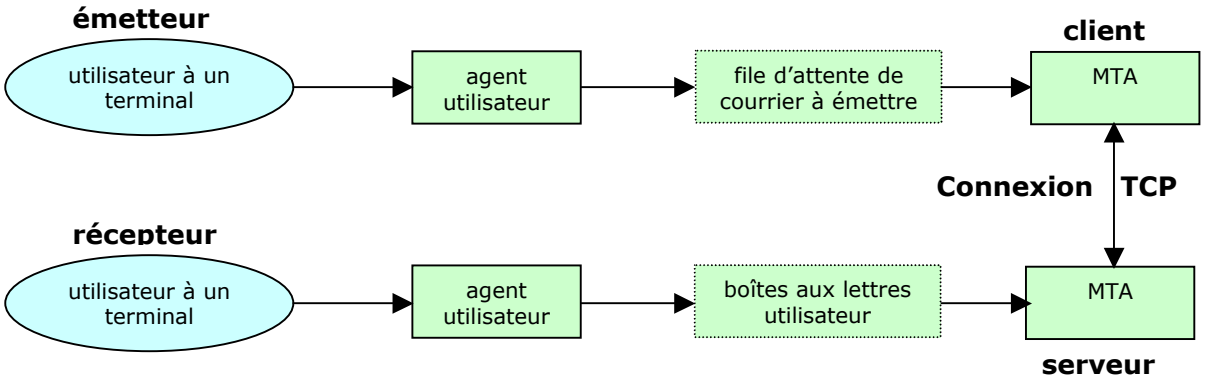
Introduction à SMTP (Simple Mail Transfer Protocol)

TCP est un protocole de niveau 4 dans le modèle OSI, c'est-à-dire qu'il prend en charge le transport de l'information en gérant le flux de données entre 2 machines. La couche applicative quand à elle s'occupe des détails de communications d'une application particulière. Plusieurs implémentations de TCP/IP regroupent les applications suivantes :

- Telnet pour la prise de contrôle à distance
- FTP (*File Transfer Protocol*), le protocole de transfert de fichiers
- SNMP (*Simple Network Management Protocol*), permettant l'administration de réseaux
- SMTP (*Simple Mail Transfer Protocol*), pour le courrier électronique.

C'est sur cette dernière que nous nous attarderons.

La figure suivante montre un schéma d'échange par e-mail utilisant TCP/IP.



Il existe une multitude d'implémentations d'agent d'utilisateur (MH, Elm, ou encore Mush sous Unix pour les plus connus). L'échange de courrier sous TCP est effectué par un agent de transfert de message (MTA), le plus répandu sous Unix étant Sendmail. Nous n'entrerons pas plus dans les détails de MTA, néanmoins citons l'ouvrage « TCP/IP, règles et protocoles », de W.R. Stevens (voir bibliographie), pour une explication complète du sujet.

Il y a un petit nombre de commandes que le client peut envoyer au serveur : moins d'une douzaine (en comparaison, FTP a plus de 40 commandes). Nous allons décrire par un exemple (fictif), effectué depuis une station Unix, lesquelles seront utilisées dans notre programme pour montrer ce qui se passe lorsqu'on envoie du courrier. Les caractères en gras sont ceux saisis par l'auteur, le reste étant généré.

```
// nous invoquons l'agent utilisateur
mail –v rstevens@noao.edu
// ceci est affiché par l'agent
To : rstevens@noao.edu
// on nous demande ensuite le sujet
Subject : test
// l'agent ajoute une ligne blanche entre les entêtes et le corps

// le corps de notre message, termine par un point signifiant la fin du msg
a dimanche pour notre partie de golf.
.
// affichage de l'agent utilisateur
```

```
Sending letter . . . rstevens@noao.edu . . .

// l'affichage qui suit (et jusqu'au bout) est la sortie du MTA (Sendmail
dans notre cas)

Connecting to mailhost via ether . . .
Trying 140.253.1.34 . . . connected.

// le client effectue une ouverture active du port TCP 25. Quand c'est
fait, le client attend un message de bienvenue (code de réponse 220) du
serveur. Cette réponse du serveur doit commencer avec le nom entièrement
qualifié et conforme du domaine de la machine serveur noao.edu

220 noao.edu Sendmail 4.1/SAG-Noao.G89 ready at Mon, 20 Oct 01 15:34:22 MST

// le client s'identifie avec la commande HELO, dont l'argument est le nom
totalement qualifié du nom de domaine de la machine cliente

>>> HELO sun.tuc.noao.edu.
250 noao.edu Hello sun.tuc.noao.edu., pleased to meet you

// identification de l'expediteur du mail
>>> MAIL From :<rstevens@sun.tuc.noao.edu>
250 <rstevens@sun.tuc.noao.edu> . . . Sender OK

// identification du destinataire
>>> RCPT TO : <rstevens@noao.edu> . . . Recipient OK

// envoi des données
>>> DATA
354 Enter mail, end with « . » on a line by itself

>>>.
250 Mail accepted

>>> QUIT
221 noao.edu delivering mail

rstevens@noao.edu . . . Sent
sent.
```

Comme on le constate, seules 5 commandes sont exécutées : dans l'ordre, HELO, MAIL FROM, RCPT TO, DATA, et QUIT.

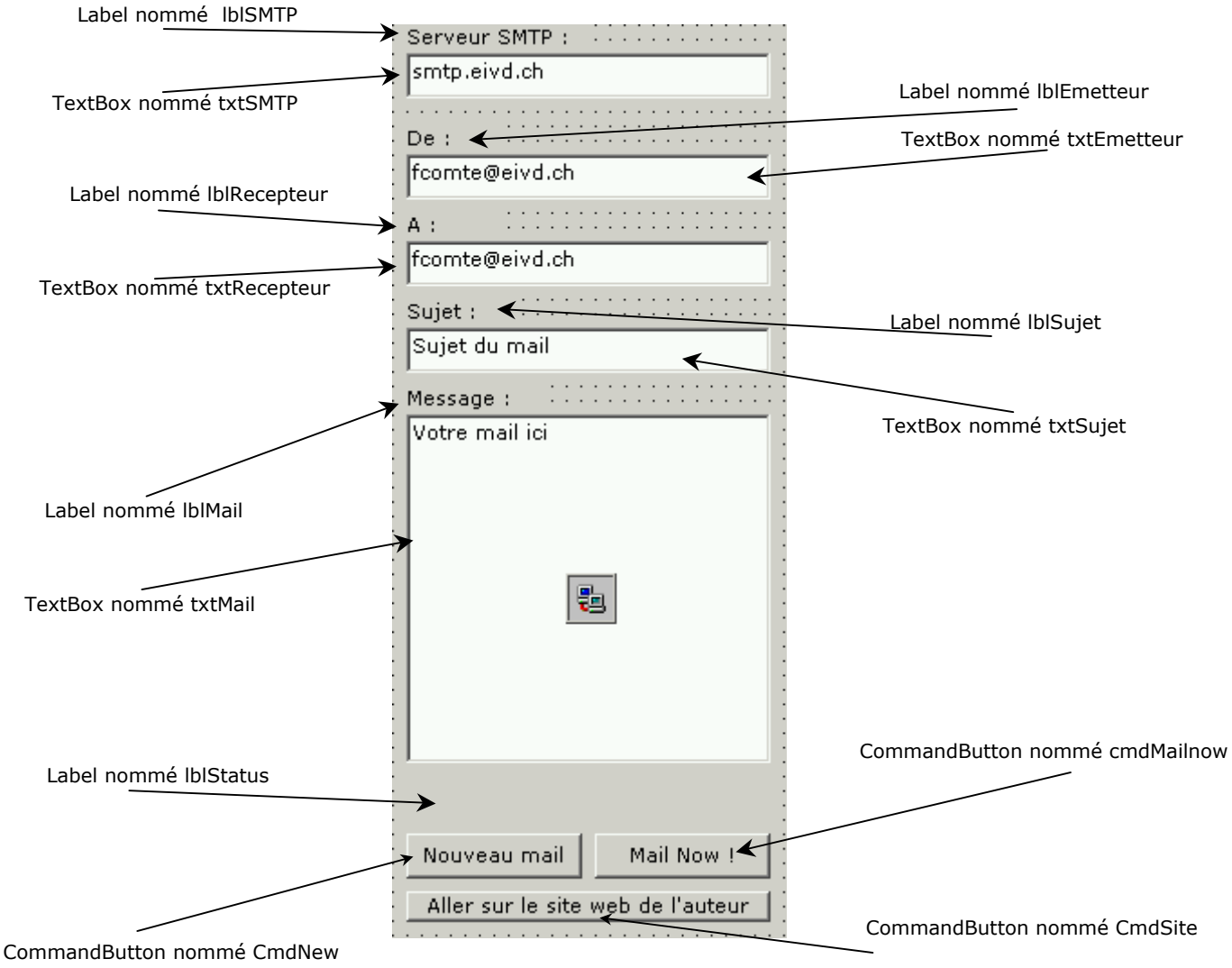
Revenons du côté de Microsoft pour concevoir notre contrôle.

Nota Bene
Autant avertir le lecteur tout de suite : l'auteur du contrôle Mail Now ! a réalisé deux versions. La première fonctionnait (plus ou moins bien) jusqu'à ce qu'une nouvelle installation sur le serveur de mail de l'EIVD (mail relay) ne vienne mettre son grain de sel… Une deuxième version, allégée, a du être trouvée en urgence. Celle-ci ne gère pas les exceptions aussi bien que la précédente (les codes d'erreurs retournés par le serveur SMTP sont simplement retournés à l'utilisateur), mais elle a le mérite de fonctionner. C'est donc cette deuxième version dont nous parlerons dans ce rapport (les sources de la première version tout comme ceux de la seconde sont fournis en annexe ainsi que sur le CD Rom).

5.3 Conception du contrôle

Maintenant que nous avons vu comment fonctionnait le protocole SMTP, nous allons l'utiliser dans notre contrôle.

Notre contrôle, nommé MailNow, permettra d'envoyer des mails à partir d'une interface simplifiée à l'extrême. Les seuls paramètres dont nous avons besoin sont l'adresse du serveur SMTP, l'adresse email de l'émetteur et celle du récepteur.
Commençons donc par définir l'interface graphique du contrôle. Après avoir sélectionné Contrôle ActiveX dans la fenêtre de nouveau projet, nous allons remplir l'objet `UserControl1` avec les mêmes objets que sur la figure suivante.



Ajoutons le contrôle `Winsock` (que nous aurons obtenu depuis Projet → Composants…) n'importe où sur `UserControl`.

Apposons le code suivant sur le bouton `CmdNew`, de telle manière qu'un clic sur le bouton efface les champs « adresse du destinataire », « sujet », et « mail ». Ainsi, un utilisateur désirant envoyer plusieurs mails n'aura pas à effacer lui-même ces champs.

```
' bouton nouveau mail
Private Sub CmdNew_Click()
    TxtRecepteur = ""
    TxtSujet = ""
    TxtMail = ""
End Sub
```

Créons un type énumérant tous les messages possibles lors de la communication entre le serveur SMTP et le client :

```
Private Enum SMTP_State
    MAIL_CONNECT
    MAIL_HELO
    MAIL_FROM
    MAIL_RCPTTO
    MAIL_DATA
    MAIL_DOT
    MAIL_QUIT
End Enum

Private m_State As SMTP_State
```

La partie importante du code se place dans l'événement `Winsock1_DataArrival`, qui est levé une fois la connexion réalisée. Voyons comment cette connexion s'effectue :

```
' bouton envoyer mail
Private Sub CmdMailnow_Click()
    Winsock1.Connect Trim$(txtSMTP), 25
    m_State = MAIL_CONNECT
End Sub
```

Un clic sur le bouton Mail Now ! exécute la connexion au serveur SMTP (`txtSMTP`). Puis l'événement `DataArrival` est levé. Les données (code de réponse à 3 chiffres) sont alors extraites du tampon et passent par un test `if` qui va – selon ces valeurs – réagir de manière adéquate.

```
Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)

    Dim strServerResponse   As String
    Dim strResponseCode     As String
    Dim strDataToSend       As String

    'Sortir les données du tampon de la socket
    Winsock1.GetData strServerResponse
    Debug.Print strServerResponse

    'Reception du code de reponse du serveur
    strResponseCode = Left(strServerResponse, 3)
    '
    'traitements a effectuer en fonction du code
    'de reponse envoye par le serveur
    If strResponseCode = "250" Or _
       strResponseCode = "220" Or _
       strResponseCode = "354" Then

        Select Case m_State
```

```
'…
'… ICI SELON LE CODE A 3 CHIFFRES RECU !
'…
        End Select
    Else
        'En cas de reponse du serveur ne figurant
        'pas dans la liste requise, la connexion
        'est fermee, et l'utilisateur recoit le
        'code d'erreur
        Winsock1.Close

        If Not m_State = MAIL_QUIT Then
            MsgBox "SMTP Error: " & strServerResponse, _
                    vbInformation, "SMTP Error"
        Else
           ' affichage d'envoi
            lblStatus.Caption = "Votre mail est envoyé !"
            lblStatus.Refresh
        End If

    End If

End Sub
```

Ainsi, voyons comment est gérée la connexion (nous allons uniquement détailler `MAIL_CONNECT`, soit le premier cas rencontré par le contrôle : la connexion au serveur[3]) :

```
        Select Case m_State
            Case MAIL_CONNECT

                lblStatus.Caption = "Connexion..." ' affichage de l'etat
                lblStatus.Refresh
                'changement de l'etat de la session
                m_State = MAIL_HELO
                'suppression des espaces blancs
                strDataToSend = Trim$(TxtEmetteur)
                'extraction de l'adresse email
                strDataToSend = Left$(strDataToSend, _
                                InStr(1, strDataToSend, "@") - 1)
                'envoi de la commande HELO au serveur
                Winsock1.SendData "HELO " & strDataToSend & vbCrLf
                Debug.Print "HELO " & strDataToSend
                ' affichage de connexion effectuee
                lblStatus.Caption = "Connecté !"
                lblStatus.Refresh
```

Les affichages dans le Debug sont ici pour garder une trace dans la fenêtre d'exécution des actions effectuées par le programme. Rappelons qu'il n'est pas possible de déboguer notre projet étant donné qu'il utilise le contrôle `Winsock`, et qu'une socket ne peut être « stoppée » en exécution[4].

Ajoutons le code du bouton `CmdSite`. Celui-ci permettra d'ouvrir une fenêtre Internet Explorer (IE dans ce cas étant le conteneur du contrôle) en spécifiant l'url d'un site (en l'occurrence celui de l'auteur) :

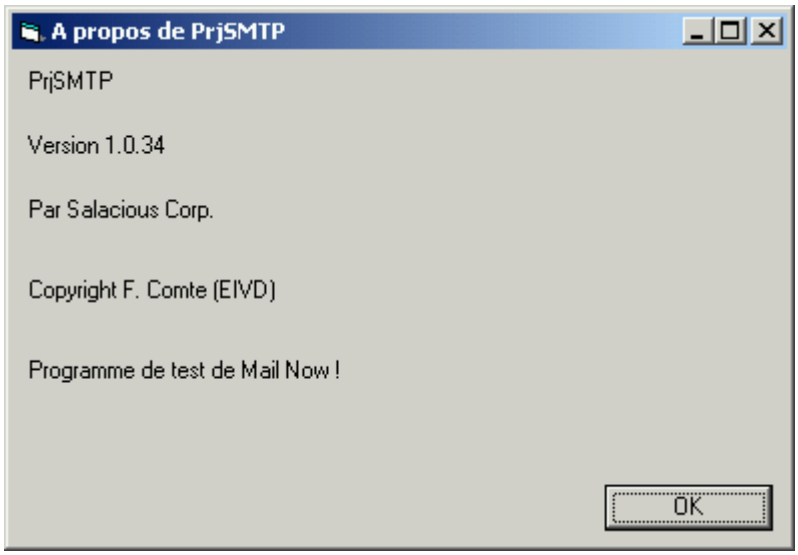
[3] Se référer aux annexes codes source pour le code complet.
[4] C'est un petit détail qui a fait perdre pas mal de temps à l'auteur…

```
' bouton permettant de se déplacer sur le site web de l'auteur
Private Sub CmdSite_Click()
 Hyperlink.NavigateTo Target:="http://www.chez.com/fcomte"
End Sub
```

Une fois le code terminé, notre contrôle est prêt. Ajoutons un projet standard EXE (nommé `PrjSMTP`) afin de tester le contrôle, et créons une instance de celui-ci en le déposant sur la feuille de `PrjSMTP`.

Créons l'exécutable PrjSMTP.exe (menu Fichier) et testons-le.
Comme nous l'avons noté auparavant, il est possible qu'il ne fonctionne pas. Non pas que le code comporte des fautes, mais ceci est dû au serveur de mails qui n'accepte pas la demande. Le lecteur trouvera dans le CD en annexe (répertoire contrôle ActiveX) un fichier .txt recensant une liste de serveurs anonymes, qui acceptent des utilisateurs non enregistrés.

L'auteur fournit également un projet de test complet, qui inclut les fichiers permettant de loger l'application dans le systray et d'afficher une fenêtre "à propos de…" sur l'événement clic droite sur l'icône de Mail Now ! depuis le systray.



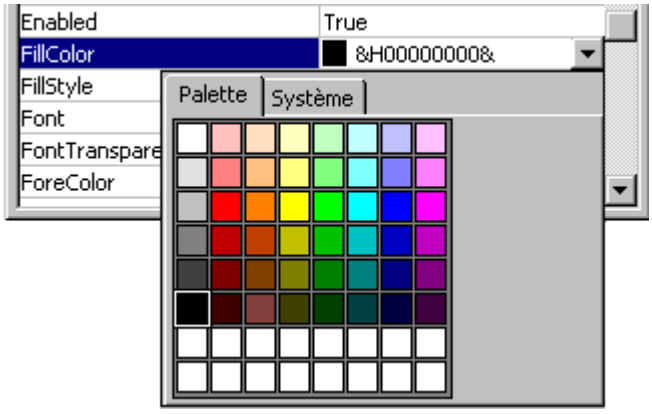
Le programme souffre cependant de petits bugs : l'application ne se quitte pas correctement (dans la mesure où son processus tourne encore dans le gestionnaire des tâches) lorsqu'elle est quittée par l'article "Quitter" du menu déroulant. De plus, si l'utilisateur omet de saisir un des champs nécessaires, l'application occupe le 100% du CPU au moment de l'envoi du mail.

6 - Les pages de propriétés

6.1 Introduction

Les pages de propriétés vous offrent une alternative à la fenêtre Propriétés pour afficher les propriétés des contrôles ActiveX. Vous pouvez regrouper sur une page plusieurs propriétés en relation les unes avec les autres, ou utiliser une page pour doter une propriété trop complexe pour la fenêtre Propriétés d'une interface ayant l'aspect d'une boîte de dialogue.

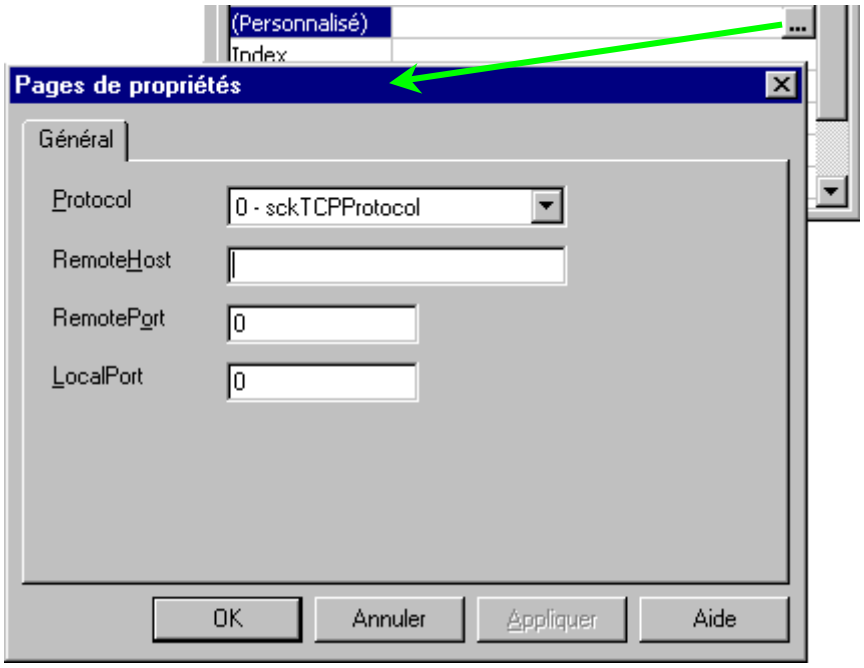
Visual Basic fournit 3 pages de propriétés standards : `StandardFont`, `StandardColor`, et `StandardPicture`. Si vous déclarez des propriétés de type `Font`, `OLE_COLOR` ou `Picture`, la fenêtre Propriétés de Visual Basic associera automatiquement ces propriétés à la page de propriétés standard appropriée. La figure suivante montre un exemple de page de propriétés - qui comporte deux onglets - fournie par Visual Basic. Chaque onglet de la page de propriété représente un objet `PropertyPage`. Cette page permet de choisir une couleur en déroulant un menu. L'avantage est que le choix de cette couleur s'effectue rapidement, sans avoir à ouvrir une nouvelle fenêtre qui prendrait de la place. Mais aussi de pouvoir saisir plus facilement des informations relatives au contrôle que si cela devait être fait dans la fenêtre de propriétés. Dans certains cas, le seul moyen d'entrer une valeur pour une propriété est d'utiliser une page de propriétés[5]. Mais nous verrons que ce ne sont pas les seuls buts des pages de propriétés.



Une page de propriétés peut également être ouverte, comme une feuille (bien que sa conception ressemble à celle d'une feuille, nous verrons que son fonctionnement en est très éloigné). C'est le cas de la page de propriétés du contrôle Winsock, comme le montre la figure suivante.

[5] Ouvrez donc la page de propriétés du contrôle Coolbar (disponible depuis l'article Composant... du menu Projet, sous le nom Microsoft Windows Common Controls-3 6.0) pour constater le nombre de propriétés disponibles.

Les pages de propriétés s'ouvrent en cliquant sur le bouton ...



Lors de la conception d'une telle fenêtre, l'objet `PropertyPage` n'affiche pas d'onglet. Il n'affiche pas non plus les boutons OK, Annuler, et Appliquer (lorsqu'ils existent, ce qui n'est pas le cas dans notre exemple). Ceux-ci sont fournis automatiquement par la boîte de dialogue Pages de propriétés, et ne font partie d'aucun objet individuel `PropertyPage`. La boîte de dialogue Pages de propriétés utilise comme texte pour l'onglet la propriété `Caption` de l'objet `PropertyPage`.

Nous allons maintenant voir comment fonctionnent les pages de propriétés, en créer une, la connecter à un contrôle, puis utiliser celles qui sont fournies par Visual Basic.

6.2 Fonctionnement des pages de propriétés

Nous avons déjà averti le lecteur que le fonctionnement d'une page de propriétés était différent de celui d'une feuille. Par exemple, quand la boîte de dialogue Pages de propriétés crée une instance d'une page de propriétés, l'événement `Initialize` est le premier que reçoit l'objet `PropertyPage` – exactement comme avec une feuille. Toutefois, contrairement à une feuille, l'objet `PropertyPage` ne reçoit pas d'événement `Load`.
L'événement clé pour l'objet `PropertyPage` est l'événement `SelectionChanged`.

Les trois actions essentielles que doit exécuter l'objet `PropertyPage` sont :

→ Dans l'événement `SelectionChanged`, obtenir les valeurs de propriétés à éditer.

→ Définir la propriété `Changed` de l'objet `PropertyPage` chaque fois que l'utilisateur édite une valeur de propriété.

→ Dans l'événement `ApplyChanges`, recopier sur le contrôle (ou les contrôles) sélectionné(s) les valeurs des propriétés éditées.

Voyons ces actions les unes après les autres.

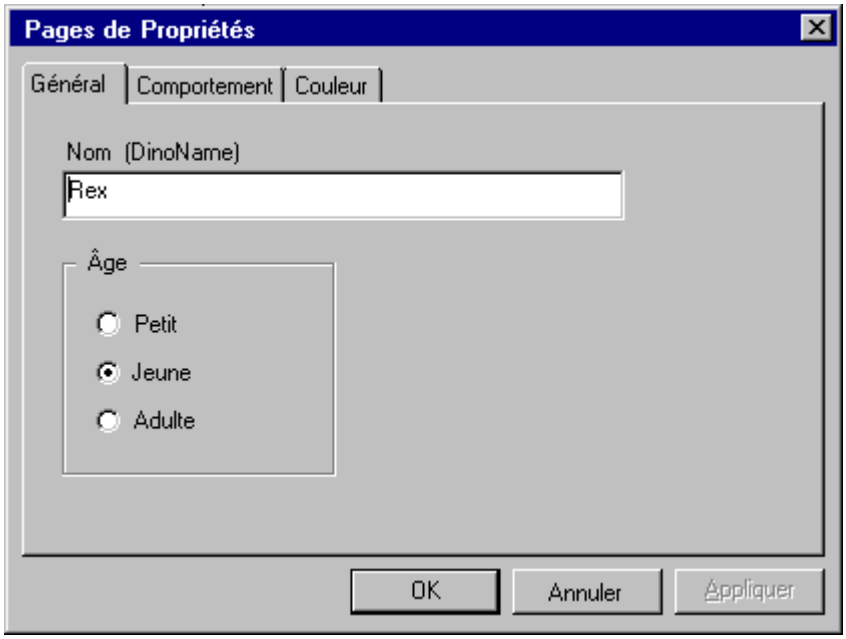
L'événement SelectionChanged

Cet événement se produit quand la page de propriétés est affichée et quand un changement se produit dans la liste des contrôles sélectionnés[6].

Vous devez toujours traiter cet événement comme si votre page de propriétés était chargée pour la première fois. Comme vous le verrez, un changement de sélection apporte un changement fondamental de l'état de la page de propriétés.

Programmation de l'événement `SelectionChanged` pour un contrôle unique

La chose la plus importante que vous aurez à faire dans l'événement `SelectionChanged` est de définir les valeurs des contrôles qui affichent les valeurs de propriétés pour édition. Considérez par exemple la page Général pour un contrôle nommé Velociraptor :



Supposez que la propriété `Age` du contrôle Velociraptor utilise l'instruction `Enum` Publique suivante :

```
Public Enum DinoAge
      vvPetit
      vvJeune
      vvAdult
End Enum
```

[6] Par exemple, après avoir sélectionné une instance de votre contrôle et avoir ouvert la boîte de dialogue Pages de propriétés, un développeur pourrait s'apercevoir qu'il aurait dû changer les propriétés de deux instances de votre contrôle. En cliquant sur la seconde instance tout en maintenant la touche Ctrl enfoncée, il peut ajouter une seconde instance à la liste des contrôles sélectionnés. Chacune de vos pages de propriétés recevrait alors un événement `SelectionChanged`.

L'événement `SelectionChanged` de la page de propriétés pourrait ressembler à ceci :

```
Private Sub PropertyPage_SelectionChanged()
      ' place la valeur de la propriété DinoName pour le premier
      ' contrôle sélectionné dans la zone de texte txtDinoName
' pour affichage et édition.
TxtDinoName = SelectedControls(0).DinoName
' utilise la valeur de la propriété Age du premier contrôle
' sélectionné pour sélectionner l'option appropriée dans le
' cadre d'options Age.
optAge(SelectedControls(0).Age).Value = True
' Le code ci-dessus résulte du fait que les éléments de l'instruction
' Enum DinoAge ont les valeurs 0, 1, et 2.
End Sub
```

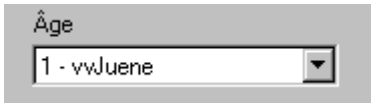
La collection `SelectedControls`

La collection `SelectedControls` contient tous les contrôles qui se trouvent sélectionnés dans le conteneur sur lequel travaille le développeur. La collection peut contenir plusieurs instances de votre contrôle. Si votre page de propriétés est partagée par plus d'un contrôle dans votre composant, alors la collection peut contenir plusieurs types de contrôles[7].

Pour le moment, ignorez la possibilité que de multiples contrôles puissent être sélectionnés. Ce que fait le code montré ci-dessus dans l'événement `SelectionChanged` consiste à prendre la valeur de chaque propriété du premier contrôle de la collection et à l'assigner au contrôle approprié sur la page de propriétés.
Dans le cas d'un unique contrôle sélectionné, toutes les valeurs des propriétés du contrôle se trouvent placées dans des champs où l'utilisateur peut les éditer.

Différentes manières d'éditer des propriétés

Au lieu de montrer les valeurs de la propriété Age comme un ensemble d'options, vous pouvez utiliser une liste déroulante faisant apparaître les éléments de l'opération. Cette liste prend moins de place que les options (avantage qui grandit avec le nombre de valeurs possibles), et elle fait apparaître les noms des constantes qui seraient utilisées dans le code. Le fragment de code suivant vous montre comment vous pouvez établir une telle liste.




```
Private Sub PropertyPage_SelectionChanged()
      TxtDinoName = SelectedControls(0).DinoName
      ' crée une liste déroulante contenant les valeurs et les noms des
' éléments Enum pour la propriété Age, et sélectionne celle
' qui correspond à la valeur courante de la propriété Age.
cboAge.AddItem vvPetit & «  - vvPetit »
cboAge.AddItem vvJeune & «  - vvJeune »
cboAge.AddItem vvAdulte & «  - vvAdulte »
' l'index de chaque élément Enum dans la liste déroulante est
' identique à la valeur de l'élément
End Sub
```

[7] Vous n'avez pas à vous inquiéter du fait que la collection contienne d'autres contrôles que le vôtre (par exemple des contrôles `TextBox`) car la boîte de dialogue Pages de propriétés n'affiche que les pages qui sont utilisées par tous les contrôles se trouvant sélectionnés.

Puisque vous pouvez choisir n'importe quelle représentation modifiable qui ait un sens pour une propriété, gardez à l'esprit que plus une propriété prend de place, plus il vous faudra d'onglets. La minimisation du nombre d'onglets rend la page de propriétés de votre contrôle plus facile à utiliser. Pour la plupart des énumérations, c'est une liste déroulante qui fera l'usage le plus efficace de l'espace disponible.

Programmation de l'événement `SelectionChanged` pour des contrôles multiples

Pour déterminer si de multiples contrôles se trouvent sélectionnés, vous pouvez tester la propriété `Count` de la collection `SelectedControls` en vérifiant si elle est supérieure à 1.

Pour traiter de multiples instances de contrôles sélectionnées, il est utile de diviser en deux groupes les propriétés de votre contrôle :

→ Les propriétés qui peuvent recevoir sensiblement la même valeur pour de multiples contrôles. Par exemple, il est très pratique de pouvoir donner la même valeur à la propriété `BackColor` de plusieurs contrôles `Label`.

→ Les propriétés auxquelles il n'y aurait pas de sens à donner la même valeur pour de multiples contrôles. Par exemple, il n'est pas particulièrement utile de donner la même valeur à plusieurs contrôles `Label` pour la propriété `Caption`. Cela pourrait même être embarrassant pour l'utilisateur si cela se produisait.

L'une des approches que vous pourriez adopter pour votre événement `SelectionChanged` consiste à désactiver les champs pour les propriétés de la seconde catégorie, à chaque fois que des contrôles multiples sont sélectionnés[8].

Activation du bouton Appliquer en affectant la valeur `True` à la page de propriété `Changed`

Pour dire à Visual Basic que l'utilisateur a édité une propriété ou plus sur la page de propriétés, vous devez donner la valeur `True` à la propriété `Changed` de l'objet `PropertyPage`. Comme il n'existe aucun moyen de savoir quelle propriété un utilisateur pourrait décider de changer, vous devez faire de même pour chaque propriété affichée par page.

Par exemple, pour notifier à l'objet `PropertyPage` des changements dans les propriétés `DinoName` ou `Age` de l'exemple Velociraptor, vous utiliserez le code suivant :

```
Private Sub txtDinoName_Changed()
      Changed = True
End

Private Sub cboAge_Changed()
      Changed = True
End Sub
```

Notez que ceci revient exactement au même que d'écrire `PropertyPage.Changed = True`[9].

[8] Si vous avez des contrôles multiples dans votre projet et que deux d'entre eux partagent une même page de propriétés, assurez-vous de fournir une récupération d'erreurs pour le code qui lit les valeurs de propriétés. Si le premier contrôle sélectionné n'inclut pas toutes les propriétés apparaissant sur la page, une erreur se produira quand vous essaierez de lire cette valeur de propriété.

[9] L'opération `Changed = True` remplit deux fonctions. Tout d'abord, elle valide à nouveau le bouton Appliquer. Ensuite, elle évite la fermeture de la boîte de dialogue quand l'utilisateur clique sur OK. C'est la seule manière d'éviter la fermeture de la boîte de dialogue.

La notification adressée à l'objet PropertyPage pour lui indiquer que des valeurs ont changé active le bouton Appliquer de la boîte de dialogue Pages de propriétés, et déclenche l'événement ApplyChanges quand l'utilisateur clique dessus, change les onglets, ou ferme la boîte de dialogue.

L'événement ApplyChanges

Le second événement par ordre d'importance dans un projet PropertyPage est l'événement ApplyChanges. C'est avec cet événement que vous recopiez sur les contrôles qui se trouvent sélectionnés les valeurs des propriétés éditées.

L'événement ApplyChanges se produit quand l'utilisateur :

→ Clique sur le bouton OK pour fermer la boîte de dialogue
→ Clique sur le bouton Appliquer
→ Sélectionne un autre onglet dans la boîte de dialogue Pages de propriétés.

Le code suivant pour l'événement ApplyChanges suppose que l'événement SelectionChanged a été programmé pour utiliser une liste déroulante pour la propriété Age comme montré plus haut.

```
Private Sub PropertyPage_ApplyChanges()
      Dim vv As Velociraptor
      ' définit la propriété DinoName du premier
' contrôle sélectionné seulement
      SelectedControls(0).DinoName = txtDinoName
      For Each vv In SelectedControls
            ' transfère la valeur se trouvant sélectionnée
' dans la liste déroulante pour la propriété DinoAge
' à tous les contrôles sélectionnés
vv.DinoAge = cboAge.ListIndex
' le code ci-dessus fonctionne parce que la valeur
' de chacun des éléments de l'instruction Enum
' est égale à celle de son index dans cboAge.
      Next
End Sub
```

Comme il n'y a généralement aucun sens à donner le même nom à tous les vélociraptors, la propriété DinoName n'est appliquée qu'au premier contrôle sélectionné. La propriété Age, quant à elle, est appliquée à tous les contrôles sélectionnés.

Traitement des erreurs dans l'événement ApplyChanges

Dans le cas présenté ci-dessus, il n'y a aucun risque d'erreur dans l'événement ApplyChanges. La propriété Text est une simple chaîne, et la liste déroulante limite les entrées de l'utilisateur pour la propriété Age aux seules valeurs valides.

Si votre page de propriétés permet à l'utilisateur de taper des valeurs qui peuvent être rejetées par la procédure Property Let (ou Property Set), vous devez utiliser la récupération d'erreur dans l'événement ApplyChanges. Le schéma le plus simple consiste à utiliser On Error Resume Next, et à tester Err.Number après chaque propriété susceptible de causer une erreur.

Quand une erreur se produit :

→ Interrompez l'événement ApplyChanges
→ Affichez un message d'erreur, de façon que l'utilisateur comprenne ce qui s'est produit

→ Placez le focus sur la propriété qui a causé l'erreur
→ Donnez la valeur `True` à la propriété `Changed` de l'objet `PropertyPage`.

6.3 Connexion d'une page de propriétés à un contrôle ActiveX

Une fois que vous avez ajouté des pages de propriétés à votre projet de contrôle ActiveX, vous pouvez utiliser la boîte de dialogue Connexion des pages de propriétés pour établir une connexion entre un contrôle et les pages de propriétés que vous voulez lui faire utiliser.

Quand un utilisateur fait apparaître la boîte de dialogue Pages de propriétés comme une instance d'un de vos contrôles, chacune des pages que vous avez connectées au contrôle apparaît comme un onglet dans la boîte de dialogue.

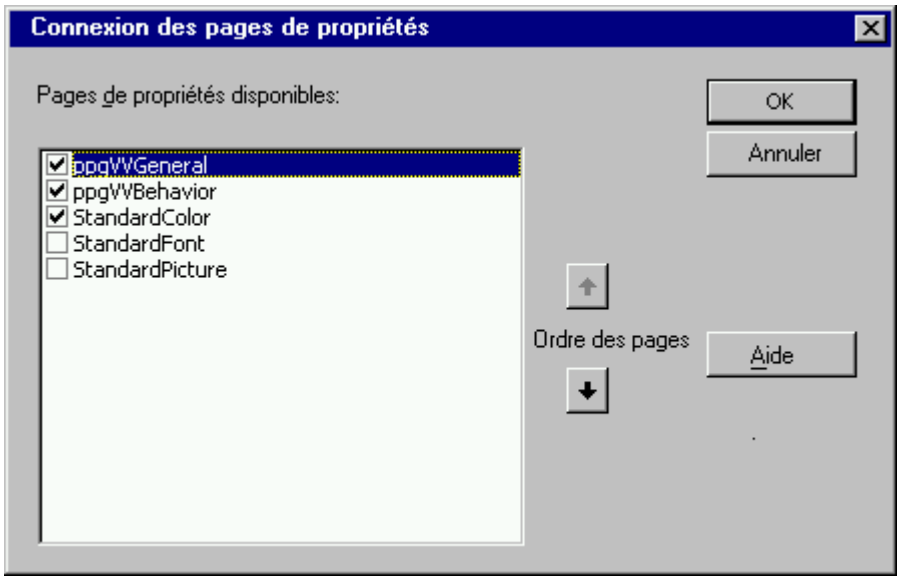
1/ Dans la fenêtre Explorateur de projet, double-cliquez sur le contrôle pour ouvrir sa fenêtre de conception.

2/ Appuyez sur F4 pour ouvrir la fenêtre Propriétés.

3/ Double-cliquez sur la propriété `PropertyPages` (ou un simple clic sur la propriété puis sur le bouton Sélection) pour ouvrir la boîte de dialogue Connexion des pages de propriétés.

4/ Cochez chaque page de propriétés que vous voulez faire apparaître lorsque le développeur, qui utilise votre contrôle, ouvre la boîte de dialogue Pages de propriétés.

5/ Utilisez les boutons Ordre des pages pour mettre les pages dans l'ordre où voulez les voir apparaître dans la boîte de dialogue Pages de propriétés, puis cliquez sur OK.



6.4 Associer une page de propriétés à une propriété

Une propriété est parfois trop compliquée pour être définie à partir de la fenêtre Propriétés. Une propriété peut être un objet, comme l'objet `Fonts`, doté de ses propres propriétés, comme nous l'avons vu précédemment. Elle peut aussi consister en un tableau de valeurs, ou même en une collection d'objets, comme une collection de boutons de barre d'outils.

Si vous déclarez une propriété de type `Font`, `OLE_COLOR`, ou `Picture`, Visual Basic l'associera automatiquement à une page `StandardFont`, `StandardColor`, ou `StandardPicture`.

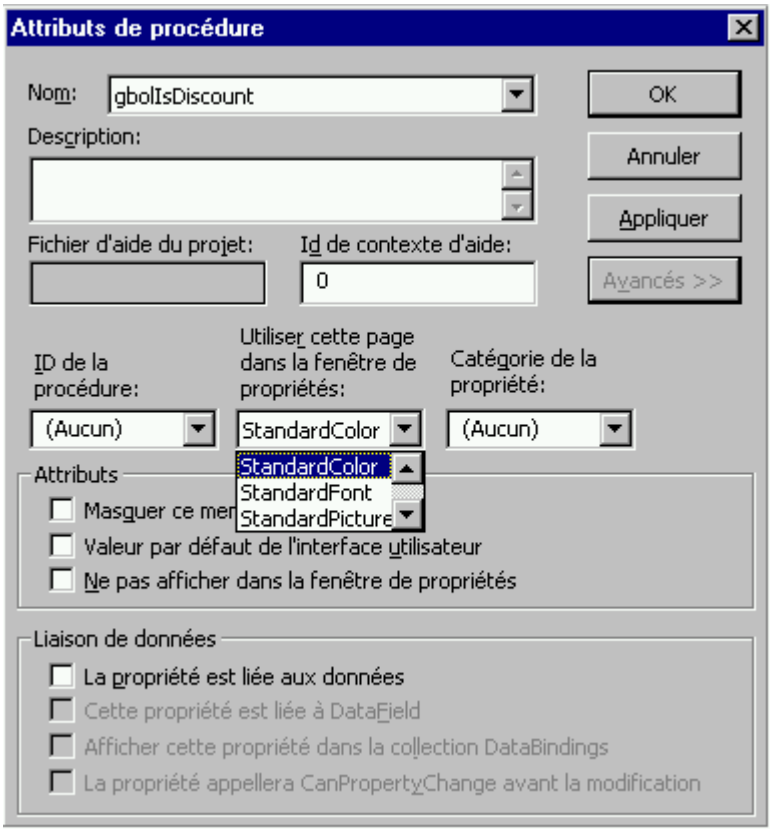
Pour associer une page de propriétés à une propriété individuelle

1/ Depuis la fenêtre Explorateur de projet, cliquez avec le bouton droit de la souris sur la fenêtre de conception `UserControl` pour ouvrir le menu contextuel, et cliquez sur Code pour ouvrir la fenêtre de code.

2/ Dans le menu Outils, cliquez sur Attributs de procédure pour ouvrir la boîte de dialogue Attributs de procédure, et cliquez sur Avancés pour développer la boîte de dialogue.

3/ Dans la zone Nom, cliquez sur la propriété que vous voulez associer à une page de propriétés.

4/ Sélectionnez la page de propriétés voulue dans la liste Utiliser cette page dans la fenêtre Propriétés, comme il est montré ci-dessous, puis cliquez sur Appliquer ou sur OK.



Evénement `EditProperty`

Visual Basic vous permet d'associer plusieurs propriétés à une même page de propriétés. Vous voudrez peut-être le faire si votre contrôle a plus d'une propriété qui utilise la même disposition de page de propriétés ou si une propriété utilise en partie la disposition d'une autre page de propriétés. Dans ce dernier cas, vous pouvez utiliser l'événement `EditProperty` pour ne valider que les parties nécessaires de la page de propriétés.

Quand l'utilisateur clique sur le bouton Sélection pour une propriété qui est associée à une page de propriétés, la page reçoit l'événement `EditProperty` en plus de celui qu'elle reçoit normalement. Vous pouvez utiliser l'argument `PropertyName` de l'événement `EditProperty` pour identifier la propriété dont le bouton Sélection a été cliqué.

L'événement `EditProperty` vous permet de faire subir différentes opérations à la page de propriétés, selon la nature de votre contrôle et la complexité de la propriété qui se trouve éditée. Vous pourriez par exemple :

- Si la propriété qui se trouve éditée est affichée dans un seul contrôle sur la page de propriété, placer le focus sur le contrôle.
- Activer et désactiver des contrôles sur la page de propriétés, de façon que seuls les champs applicables à la propriété spécifiée soient activées.

Masquer des propriétés dans la fenêtre Propriétés

Il peut arriver que vous ne vouliez pas afficher une propriété dans la fenêtre Propriétés. Cela peut se produire si l'affichage d'une valeur de propriété demande des calculs prenant beaucoup de temps et que le développeur utilisant le contrôle est irrité par le temps d'accès à la fenêtre Propriétés.

Dans la dialogue Attributs de procédure, accessible à partir du menu Outils, cliquez sur la propriété que vous voulez masquer. Cliquez ensuite sur le bouton Avancées, cochez l'option Ne pas afficher dans la fenêtre Propriétés, puis sur Appliquer.

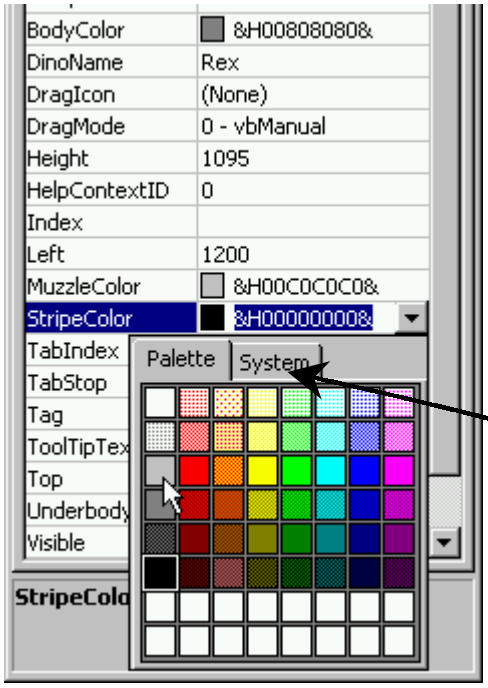
6.5 Utiliser les pages de propriétés standard

Visual Basic fournit trois pages de propriétés standard : `StandardFont`, `StandardColor`, et `StandardPicture`. Si vous déclarez des propriétés de type `Font`, `OLE_COLOR`, ou `Picture`, la fenêtre Propriétés de Visual Basic associera automatiquement ces propriétés à la page de propriétés standard appropriée.
Toutefois, Visual Basic n'associera pas automatiquement ces pages à la boîte de dialogue Pages de propriétés. Utilisez la procédure abordée au point 3 (connexion d'une page de propriétés à un contrôle ActiveX) pour ajouter des pages de propriétés standard à la liste des pages qui seront affichées dans la boîte de dialogue Pages de propriétés.

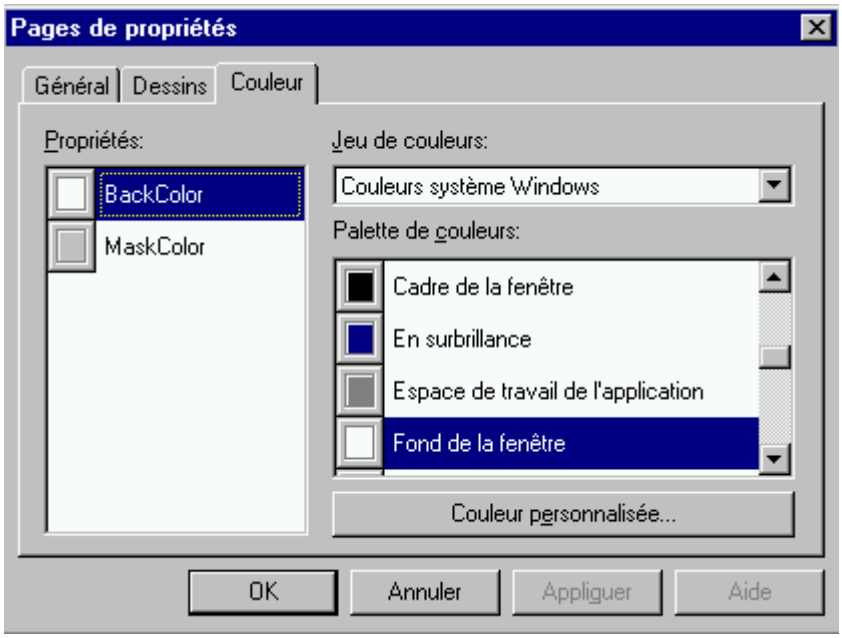
Pages de propriétés standard et propriétés multiples

Si votre contrôle a plus d'une propriété qui utilise une page de propriétés standard et que vous ajoutez cette page à la propriété `PropertyPages` de votre contrôle, la page standard inclura une liste de toutes les propriétés que peut sélectionner l'utilisateur.

Par exemple, la figure suivante montre la fenêtre `Propriétés` et la boîte de dialogue `Pages de propriétés` pour l'hypothétique contrôle Velociraptor, qui a plusieurs propriétés de type `OLE_COLOR` :



Comme le montre la figure suivante, la page `Color` affichée par la boîte de dialogue Pages de propriétés pour le contrôle Velociraptor comporte une zone de liste contenant les quatre propriétés de couleur du contrôle.



Cette figure montre aussi que la boîte de dialogue Pages de propriétés utilise un format très différent de celui utilisé par la fenêtre Propriétés.

La portion de code suivante montre comment la propriété `StripeColor` de l'hypothétique contrôle Velociraptor doit être déclarée de manière à fonctionner avec la fenêtre Propriétés et la boîte de dialogue Pages de propriétés :

```
Private mStripeColor As OLE_COLOR

Public Property Get StripeColor() As OLE_COLOR
   StripeColor = mStripeColor
End Property

Public Property Let StripeColor( _
      ByVal NewColor As OLE_COLOR)
   mStripeColor = NewColor
End Property
```

Bibliographie critique et liens Internet

- Visual Basic developer's guide to COM and COM+, Wayne S. Freeze, Sybex 2000, 460 pages.
  L'ouvrage qui permet le mieux (parmi ceux cités dans cette bibliographie) de comprendre les technologies de composants de Microsoft, et met en relation simplement et efficacement les différents concepts. La partie consacrée à DCOM est malheureusement très courte, et n'offre qu'une petite introduction au protocole. L'auteur traite également COM+, MSMQ, et IMDB.

- Developping COM/ActiveX components with Visual Basic 6, Dan Appleman, Sams 1999, 860 pages.
  Cet ouvrage s'adresse avant tout aux développeurs professionnels sur VB, et montre les meilleures techniques pour développer des composants COM et DCOM. D'un niveau très nettement plus élevé que le titre précédent, cette « bible » (presque 900 pages !) passe en revue la conception d'objets COM mais aussi et surtout les mécanismes internes lors de l'exécution des composants. C'est à l'aide de cet ouvrage que j'ai réalisé une grande partie des explications techniques.

- TCP/IP, règles et protocoles, W.R. Stevens, Addison Wesley 1995, 600 pages.
  Cet ouvrage m'a permis de comprendre le protocole SMTP, utilisé pour un des exemples ActiveX.

- http://msdn.microsoft.com, le lien INCONTOURNABLE pour développer sous plate-forme Windows.

- http://www.microsoft.com/com/tech/com.asp, la page de Microsoft pour tout ce qui concerne la technologie COM.