Un formulaire auto-extensible pour Access

par Charles A.

Date de publication : 31/03/2006

Dernière mise à jour : 31/03/2006

Thèmes abordés : · description et limites des propriétés auto extensible / auto réductible · mise en oeuvre d'une solution de contournement · application à un cas pratique : un formulaire d'information Niveau requis : avancé / expert

I - introduction

Le but de ce tutoriel est de permettre de réaliser un formulaire dont la taille s'adapte automatiquement à son contenu.

Derrière l'apparente trivialité du sujet, il s'avère hélas que MS Access ne permet pas d'obtenir ce type de fonctionnalité de manière native.

Dans le cas où nous souhaiterions réaliser un formulaire de type info-bulle, aucun outil ne nous permet d'obtenir directement le résultat attendu.

D'aucun me parleront des propriétés **Auto extensible** (*CanGrow*) et **Auto réductible** (*CanShrink*), mais nous verrons que cela n'est pas aussi simple.

Nous trouverons alors des solutions de contournement avant d'en sélectionner une et de la mettre en oeuvre sur un cas pratique : un formulaire d'information.

II - les limites des propriétés Auto extensible / Auto réductible

Notre objectif est de réaliser un formulaire d'information.

Chaque information à afficher est constituée d'une ou plusieurs phrases dont la longueur est indéfinie.

Ce qui fait qu'il n'est possible de déterminer à l'avance le rapport Hauteur / Largeur du texte.

Nous pourrions bien sûr nous contenter de mettre en place un ascenseur de déplacement, mais cela rend la lecture d'une information beaucoup moins visuelle et conviviale.

II-A - Quelques propriétés de formulaire

- **Taille ajustée (*AutoResize*)**

C'est une propriété du **formulaire**.

Elle permet lors de l'**ouverture** d'ajuster la taille du formulaire à son contenu.

Cependant, si la longueur du texte excède la taille du TextBox prévue en mode conception, la taille du formulaire ne sera pas affectée.

- **Fenêtre indépendante (*PopUp*)**

Permet de placer le formulaire au dessus de tous les autres, ce qui sera utile pour notre formulaire d'information.

Mais là encore, aucun impact sur un redimensionnement dynamique.

- **Fenêtre modale (*Modal*)**

Permet de placer le formulaire au-dessus de tous les autres, toutefois, la propriété ***PopUp*** prend l'exclusivité de l'affichage et interdit l'accès aux autres formulaires tant que celui qui a été ouvert en mode ***modal*** reste ouvert.

Cette propriété peut être utile pour notre projet mais là aussi pas d'impact sur les dimensions.

II-B - les propriétés Auto extensible / Auto reductible

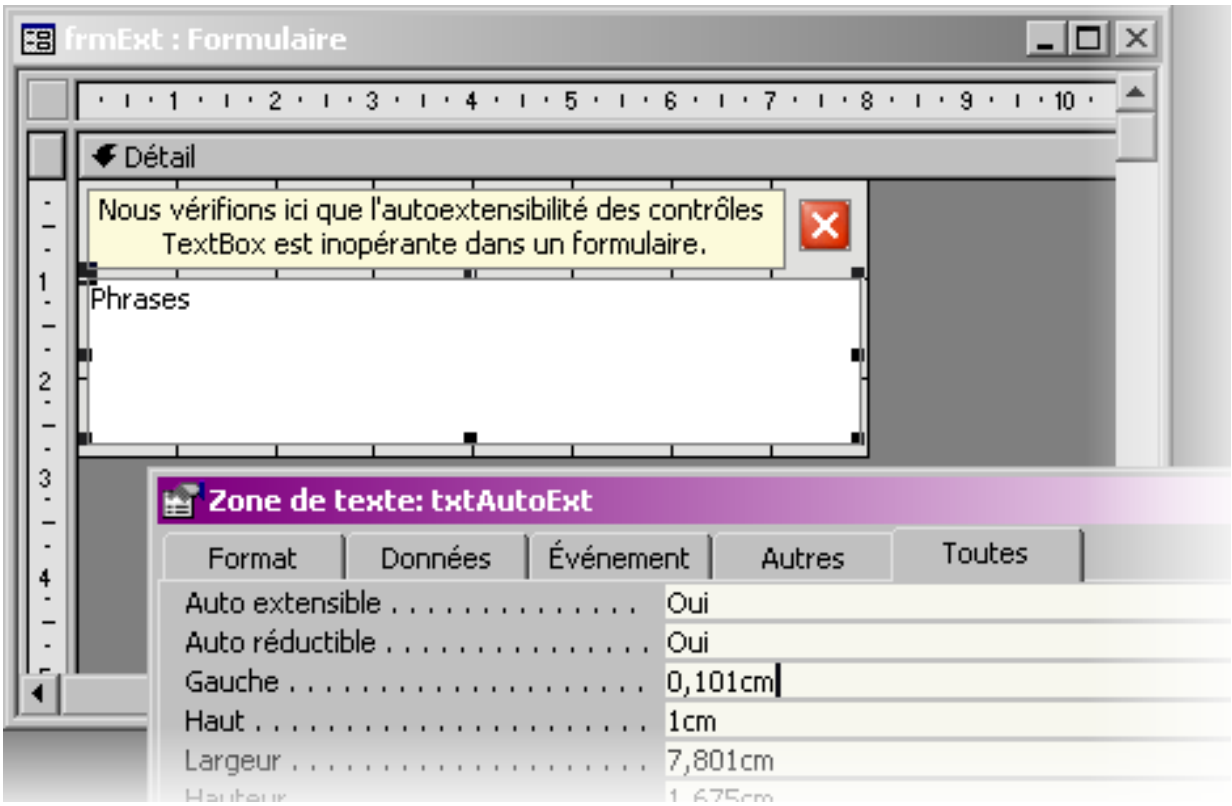
Ces propriétés sont disponibles à la fois pour les sections de formulaires et pour les contrôles de formulaire.

Auto extensible (*CanGrow*) et **Auto réductible** (*CanShrink*) permettent d'adapter les dimensions d'un contrôle à son contenu.

Ce sont théoriquement et intuitivement les propriétés idéales mais l'aide Access ne laisse pas longtemps planer de doute sur la question :

> ***Aide Access***
>
> *Vous pouvez utiliser les propriétés **AutoExtensible** (CanGrow) et **AutoRéductible** (CanShrink) pour contrôler l'apparence de sections et des contrôles dans les formulaires et états que vous **imprimez ou affichez en aperçu avant impression**.*
>
> *Par exemple, si vous attribuez la valeur Oui aux deux propriétés, une section ou un contrôle s'ajustent automatiquement verticalement pour imprimer ou afficher l'ensemble des données qu'ils contiennent.*
>
> *Notes*
>
> *· Les propriétés **AutoExtensible** (CanGrow) et **AutoRéductible** (CanShrink) ne s'appliquent pas aux sections en-tête de page et pied de page d'un état ou d'un formulaire, bien qu'elles s'appliquent aux contrôles de ces sections.*
>
> ***· Ces propriétés affectent l'affichage des sections et des contrôles de formulaire uniquement lorsque le formulaire est imprimé ou affiché en aperçu avant impression, et non lorsqu'il est affiché en mode Formulaire, en mode Feuille de données ou en mode Création.***

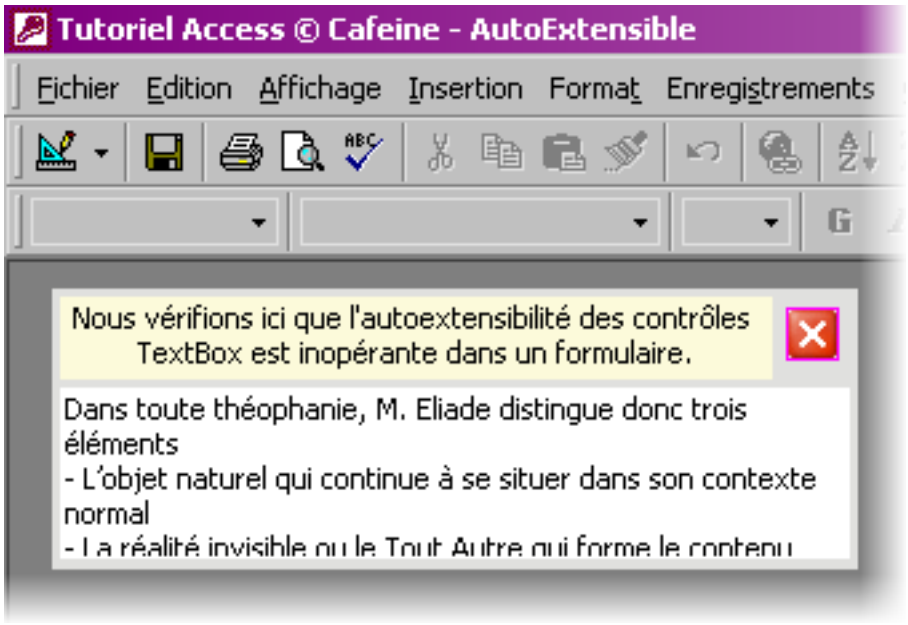
En travaillant avec ces propriétés dans un formulaire, cela donne :



Les propriétés CanGrow et CanShrink sur Oui

Nous mettons ces propriétés sur **Oui**.

En passant en mode formulaire, nous vérifions hélas, ce qu'affirme l'aide Access.

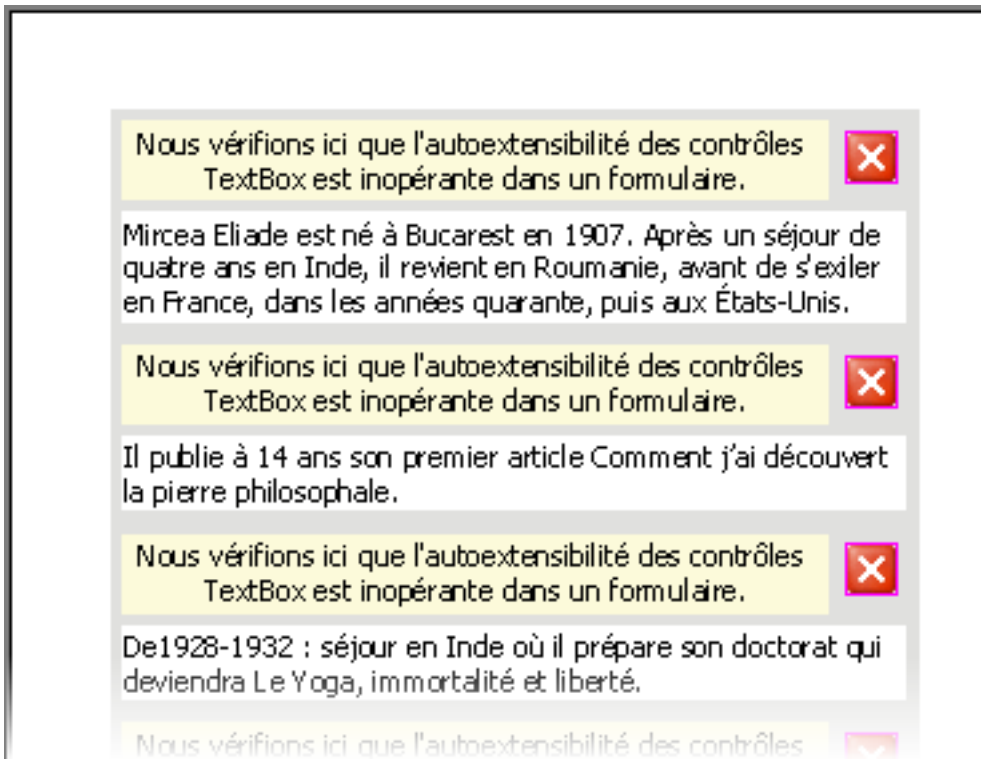


Echec des propriétés

Le redimensionnement dynamique n'est accessible qu'à l'impression, soit de l'état, soit du formulaire.

Par conséquent, il nous sera impossible de réaliser notre fonctionnalité de dimensionnement dynamique de formulaire avec ces propriétés.

En passant en mode aperçu avant impression, nous vérifions que le contexte fonctionnel de ces propriétés est l'impression.



Application en mode aperçu avant impression

III - Une solution de contournement

Dans la section précédente, nous il vous a été démontré que Microsoft Access ne dispose pas des fonctions appropriéés pour adapter la taille d'un formulaire à son contenu.

Nous allons donc chercher une solution de contournement.

Afin de déterminer la hauteur qu'occupe un texte en tenant compte bien entendu de sa police, sa taille, sa casse (gras, italique, soulignement etc.), voici comment nous allons procéder :

III-A - les APIs

Nous pouvons effectivement envisager d'utiliser les API de Windows pour calculer la hauteur d'un texte, ces dernières retournant la taille en Pixels d'une chaîne de caractères.

Elles restent cependant relativement lourdes à mettre en oeuvre lorsque la chaîne contient des sauts de ligne mais on pourrait pour y parer coder des fonctions de césure et de calcul de marges gauche et droite.

Cela dit, il faut bien avouer que c'est tout sauf simple.

A titre d'exemple : deux fonctions de calcul.

III-A-1 - l'API GetTextExtentPoint32 (gdi32)

Fonction avec l'API GetTextExtentPoint32

```
Private Type POINTAPI
    X As Long
    Y As Long
End Type

' retourne le point maximal d'un affichage d'une chaîne
Private Declare Function GetTextExtentPoint32 Lib "gdi32" Alias _
                         "GetTextExtentPoint32A" (ByVal hDC As Long, ByVal lpsz As String, _
                                                  ByVal cbString As Long, lpSize As POINTAPI) As
Long

'   retourne le handle (poignée) d'un contexte d'affichage ou device context
Private Declare Function apiGetDC Lib "user32" _
    Alias "GetDC" _
    (ByVal hWnd As Long) _
    As Long

'   libère un contexte d'affichage
Private Declare Function apiReleaseDC Lib "user32" _
    Alias "ReleaseDC" _
    (ByVal hWnd As Long, _
    ByVal hDC As Long) _
    As Long

Public Function GetTextWidth(ByVal strExp As String) As Long
```

Fonction avec l'API GetTextExtentPoint32

```
Dim myPoint As POINTAPI
Dim hDC As Long

' ouverture du Device Context courant
hDC = apiGetDC(0&)

' l'API attribue les coordonnées maximales au point API passé en argument
GetTextExtentPoint32 hDC, strExp & Chr(0), Len(strExp), myPoint

GetTextWidth = myPoint.X

' libération des ressources mémoire
apiReleaseDC 0&, hDC

End Function
```

Utilisation : Fonction avec l'API GetTextExtentPoint32

Fenêtre exécution

```
?gettextwidth("Utilisation d'une API pour calculer la longueur d'un texte")
 364
```

III-A-2 - l'API DrawText (user32)

Une autre fonction avec gestion du retour à la ligne :

Calcul avec l'API DrawText

```
Private Type RECT
    Left As Long
    Top As Long
    Bottom As Long
    Right As Long
End Type

'   retourne le handle (poignée) d'un contexte d'affichage ou device context
Private Declare Function apiGetDC Lib "user32" _
    Alias "GetDC" _
    (ByVal hWnd As Long) _
    As Long

'   libère un contexte d'affichage
Private Declare Function apiReleaseDC Lib "user32" _
    Alias "ReleaseDC" _
    (ByVal hWnd As Long, _
    ByVal hDC As Long) _
    As Long

' dessine un texte sur un device context
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" ( _
        ByVal hDC As Long, ByVal lpStr As String, ByVal nCount As Long, _
        lpRect As RECT, ByVal wFormat As Long) As Long

Public Function GetTextWidthMultiline(ByVal strExp As String, _
                        Optional ByVal lngWidthMax As Long = 0) As Long

Dim myRect As RECT
Dim hDC As Long

' ouverture du Device Context courant
hDC = apiGetDC(0&)

' Le paramètre DT_CALCRECT indique à l'API DrawText de ne pas
'  dessiner le texte mais de renvoyer la taille du rectangle
'  contenant le texte dans le rectangle myRect passé en argument
If lngWidthMax <> 0 Then
    ' cas où nous spécifions une largeur maximale
    '  nous ajoutons le paramètre DT_WORDBREAK pour forcer
    '  les retours à la ligne pour les chaînes qui dépassent
    '  la largeur maximale
```

Calcul avec l'API DrawText

```
        myRect.Bottom = lngWidthMax
        DrawText hDC, strExp, Len(strExp), myRect, _
                      DT_CALCRECT Or DT_TOP Or DT_LEFT Or DT_WORDBREAK
        GetTextWidthMultiline = myRect.Right
    Else
        DrawText hDC, strExp, Len(strExp), myRect, _
                      DT_CALCRECT Or DT_TOP Or DT_LEFT
        GetTextWidthMultiline = myRect.Bottom
    End If

    ' libération des ressources mémoire
    apiReleaseDC 0&, hDC

    End Function
```

Utilisation :

- en cas de monoligne renvoie la longueur en pixels

- en cas de multiligne renvoie la hauteur en pixels, il faut qu'une largeur maximale ait été précisée.

fenêtre exécution

```
?gettextwidthMultiline("Utilisation d'une API pour calculer la longueur d'un texte")
 364

?gettextwidthMultiline("Utilisation d'une API pour calculer la longueur d'un texte", 200)
 32
```

III-B - Les fonctions non documentées d'Access

Je remercie au passage Arkham46 de m'avoir découvrir ces fonctions plus ou moins cachées.

Il s'agit d'une bibliothèque de fonctions intégrée à Access mais inaccessible pour le commun des utilisateurs.

Au milieu de la vingtaine de fonctions, se trouve : **TwipsFromFont** qui renvoie la taille en Twips d'une chaîne de caractères.

Cette fonction est plus pratique et rapide à mettre en oeuvre que les APIs car elle renvoie directement des Twips et non des pixels et ne nécessite pas de déclarations préalables.

Cependant, elle comporte un inconvénient, tout comme les API vues ci-avant, elle ne gère pas très bien la césure et le renvoi à la ligne.

Dans le code suivant nous utilisons TwipsFromFont pour calculer la longueur de la chaîne.

Lorsque, quand celle-ci dépassera la largeur du contrôle, nous poserons un retour à la ligne.

La fonction non documentée écrite par Arkham46

```
Public Function GetTextLength(pCtrl As Control, ByVal str As String, Optional ByVal Height As
Boolean = False)
    Dim lx As Long, ly As Long

    ' initialisation du WizHook
    WizHook.Key = 51488399

    ' la fonction attribue des valeurs de largeur / hauteur à lx et ly
    '  en fonction de la police, taille, gras, italique, souligné ...
    WizHook.TwipsFromFont pCtrl.FontName, pCtrl.FontSize, pCtrl.FontWeight, _
                          pCtrl.FontItalic, pCtrl.FontUnderline, 0, _
                          str, 0, lx, ly

    If Not Height Then
        GetTextLength = lx
    Else
        GetTextLength = ly
    End If
End Function
```

La fonction est performante mais ne parvient à calculer les longueurs et largeurs en Twips que pour un texte d'une seule ligne.

Il nous manque une fonction de césure capable d'estimer le nombre lignes potentielles en fonction de sa longueur pour que nous puissions le découper convenablement.

Pour ce faire, nous allons écrire une fonction récursive, c'est à dire qui s'appelle elle-même (**Cesure()**).

L'algorithme est le suivant :

- parcours de la chaîne caractère par caractère et calcul de la longueur par la fonction **GetTextWidth()**

- dès que la largeur calculée dépasse celle de notre contrôle nous découpons la chaîne au dernier mot et appelons à nouveau la fonction de césure sur le reste de la chaîne.

Voici le code :

La fonction de césure

```
Private intCounter As Integer
Public CtlHeight As Long

Function Cesure(ByVal ctl As Control, _
                ByVal str As String, _
                ByVal lngMaxWidth As Long, _
                ByVal lngSideMargin As Long, _
                Optional ByVal ReInit As Boolean = False) As Integer

' argument  :   ctl (Contrôle sur lequel nous calculons les dimensions)
' argument  :   str (Chaîne du texte dont on calcule la longueur)
' argument  :   lngMaxWidth (Largeur du contrôle maximum pour compter les retours à ligne)
' argument  :   lngSideMargin (marge gauche et droite entre le texte et son contrôle)
' argument  :   ReInit (initialisation du compteur de ligne, passé à True lors du premier
```

La fonction de césure

```
    '                       lancement et par défaut à False)

    Dim strSplitB As String
    Dim strSplitA As String
    Dim strSplitI As String
    Dim strTmp As String
    Dim strTab() As String
    Dim i As Long
    Dim lngType As Long
    Dim lngLastBreak As Long
    Dim iCar As String * 1

    ' caractères sur lesquels nous faisons un découpage après (After)
    strSplitA = ",?;.:!})]"""
    ' caractères sur lesquels nous faisons un découpage avant (Before)
    strSplitB = "{([=+-/*\"
    ' caractères sur lesquels nous faisons un découpage en ignorant le caractère (Ignore)
    strSplitI = " "

    i = 1

    If ReInit Then
        intCounter = 1
    Else
        intCounter = intCounter + 1
    End If

    ' cas de retours chariots inclus dans la chaîne
    If InStr(str, vbCrLf) Then
        strTab = Split(str, vbCrLf)
        intCounter = intCounter - 1
        For i = 0 To UBound(strTab)
            Cesure ctl, strTab(i), ctl.Width, lngSideMargin
        Next i
        Exit Function
    End If


    If str & "" <> "" Then
        Do While i <= Len(str)
            iCar = Mid(str, i, 1)

            If InStr(strSplitA & strSplitB & strSplitI, iCar) Then
                lngLastBreak = i

                If InStr(strSplitA, iCar) Then
                    lngType = 0
                End If

                If InStr(strSplitB, iCar) Then
                    lngType = 1
                End If

                If InStr(strSplitI, iCar) Then
                    lngType = -1
                End If

            End If
            strTmp = strTmp & iCar
            If GetTextLength(ctl, strTmp) >= lngMaxWidth - (lngSideMargin * 2) Then
                Cesure ctl, Trim(Right(str, Len(str) - lngLastBreak - lngType)), lngMaxWidth,
    lngSideMargin
                Exit Function
            End If
        i = i + 1
        Loop
    End If

    CtlHeight = intCounter * (GetTextLength(ctl, "|", True) + 30)

    End Function
```

III-C - Un formulaire caché

Certains d'entre vous plus aguerris que d'autres ont peut-être déjà rencontré une méthode intéressante : **.SizeToFit**

Elle est accessible en mode **conception** et par le menu : **Format > Taille > Au contenu**

Cette fonctionnalité permet d'adapter à son contenu une **étiquette** (*Label*), ce qui répond à notre objectif.

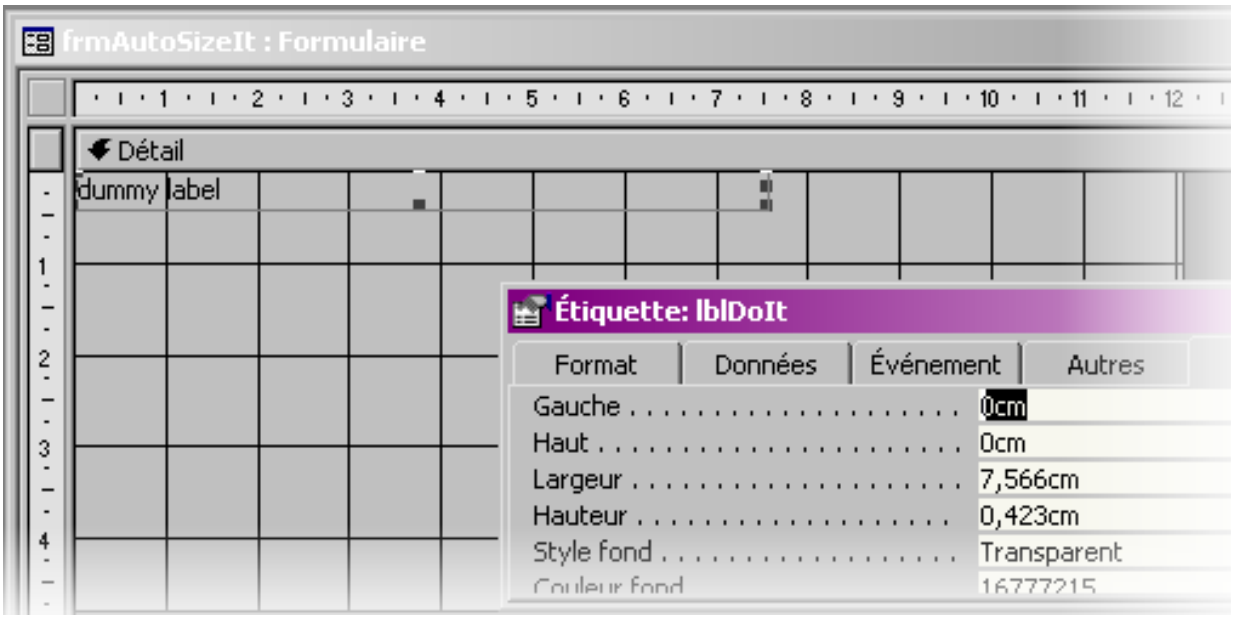
L'inconvénient est que cette méthode n'est disponible qu'en mode conception.

Or, il nous est impossible de l'utiliser directement dans notre formulaire puisque celui-ci sera ouvert en mode formulaire.

Il nous reste une alternative, celle d'utiliser cette méthode dans un autre formulaire.

Il serait plus intéressant de ne pas afficher le second formulaire, nous allons donc l'ouvrir en le masquant.

Nous créons un formulaire appelé **frmAutoSizeIt** dans lequel nous nous contentons de ne créer qu'un seul contrôle, une étiquette : **lblDoIt**



le formulaire qui restera caché

Nous pouvons définir un algorithme :

- lors de l'**ouverture** du formulaire principal, nous ouvrons le second en **mode conception** et **caché**.

Evénements ouverture et fermeture du formulaire

```
Private Sub Form_Load()
    ' lors du chargement du form
    '  nous ouvrons le form frmAutoSizeIt en mode conception (acDesign)
    '  et en mode caché (acHidden)

    DoCmd.OpenForm "frmAutoSizeIt", acDesign, , , , acHidden

End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' lors du déchargement du form
    '  nous fermons le form frmAutoSizeIt sans le sauvegarder (acSaveNo)

    DoCmd.Close acForm, "frmAutoSizeIt", acSaveNo

End Sub
```

- lors de l'**affichage** d'un enregistrement, il nous faut recalculer la hauteur du texte à afficher au moyen de la fonction **GetCtlSize()**

Evenement Current

```
Private Sub Form_Current()
    ' événement provoqué pour chaque affichage d'un nouvel enregistrement

    If Not IsNull(Me.txtAutoExtView) Then
        If Len(Me.txtAutoExtView) > 0 Then
            '   appel de la fonction de calcul de la hauteur d'un contrôle
            Me.txtAutoExtView.Height = GetCtlSize("txtAutoExtView")

            '   affectation de la hauteur calculée
            Me.Section("Détail").Height = Me.txtAutoExtView.Top + Me.txtAutoExtView.Height

            '   marge pour la partie basse du formulaire
            Me.InsideHeight = Me.Section("Détail").Height + 50
        End If
    End If

End Sub
```

- la fonction **GetCtlSize()** utilise la méthode **.SizeToFit** sur l'étiquette du formulaire caché pour en récupérer la hauteur, le formatage de l'étiquette aura préalablement été copié sur celui du texte à afficher

Application de la méthode SizeToFit

```
Private Function GetCtlSize(ByVal ctl As String) As Long
    ' fonction permettant de trouver la hauteur du contrôle pour un texte donné
    '  nous passons les caractéristiques de mise en forme
    '  puis nous utilisons la méthode .SizeToFit pour trouver la bonne hauteur

    ' argument  :   ctl As String (Chaîne nom du contrôle qui doit subir ce traitement)

' Le formulaire frmAutoSizeIt est celui qui est ouvert en mode conception et caché
'  rappelons que l'étiquette lblDoIt est celle sur qui va s'appliquer la méthode
'  SizeToFit pour calculer la hauteur du texte
With Forms("frmAutoSizeIt").lblDoIt
    '   attribution du texte du contrôle avec un renvoi à la ligne
```

Application de la méthode SizeToFit

```
        .Caption = Me.Controls(ctl).Value & vbCrLf & " "

        '   attribution des enrichissements de forme
        '       Police de caractères
        .FontName = Me.Controls(ctl).FontName

        '       Taille de caractères
        .FontSize = Me.Controls(ctl).FontSize

        '       Gras
        .FontBold = Me.Controls(ctl).FontBold

        '       Italique
        .FontItalic = Me.Controls(ctl).FontItalic

        '       Epaisseur
        .FontWeight = Me.Controls(ctl).FontWeight

        '       Largeur du contrôle
        .Width = Me.Controls(ctl).Width

        '   application de la méthode .SizeToFit qui ajuste automatiquement
        '       la largeur du contrôle au contenu
        .SizeToFit

        '   on répète l'opération sans le retour charriot
        .Caption = Me.Controls(ctl).Value
        .Width = Me.Controls(ctl).Width
        .SizeToFit

        '   on renvoie la hauteur du contrôle
        GetCtlSize = .Height
    End With

    End Function
```

- nous mettons à jour la **hauteur** du TextBox et du premier formulaire.

Cette solution outre son originalité s'avère très simple et fiable dans sa mise en oeuvre, c'est celle que nous allons retenir dans notre étude de cas.

IV - cas pratique : un formulaire d'information

IV-A - Principes

Nous allons donc mettre en oeuvre la solution vue dans la section précédente pour concevoir un formulaire d'information.

Pour que ce formulaire soit encore plus performant, il convient de lui adjoindre une fonctionnalité de déplacement flottant.

A l'image des formulaires de type "Tip of the Day" ou info-bulle, il peut être intéressant de le déplacer.

Ayant masqué les barres de titres et boutons de contrôle pour plus de simplicité dans l'affichage, le déplacement de fenêtre n'est plus, a priori, accessible.

Là encore, les APIs viennent à notre rescousse : nous créons une étiquette (Label : **lblBarreTitre**) qui va **émuler** le comportement d'une barre de titre.

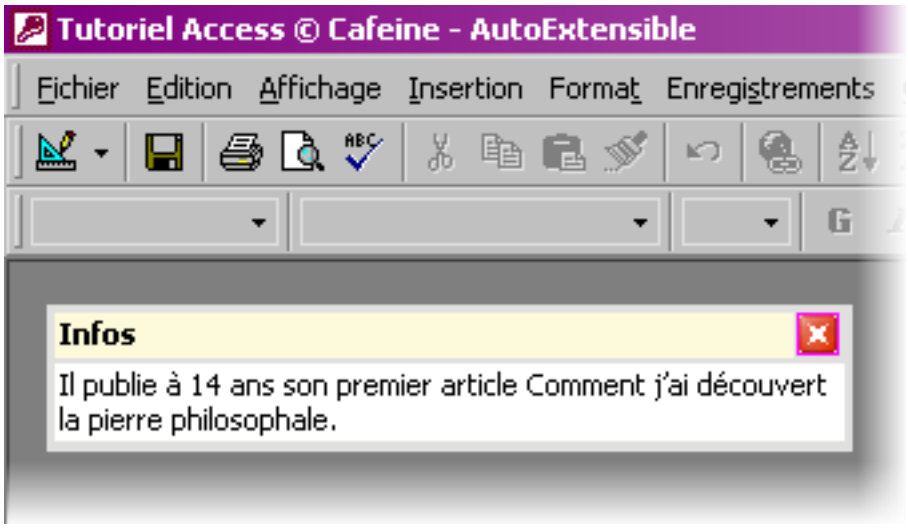
L'**API SendMessage** va envoyer à Windows le **même message système** que lorsqu'un utilisateur effectue un **clic sur la barre de titre**.

API de déplacement du formulaire

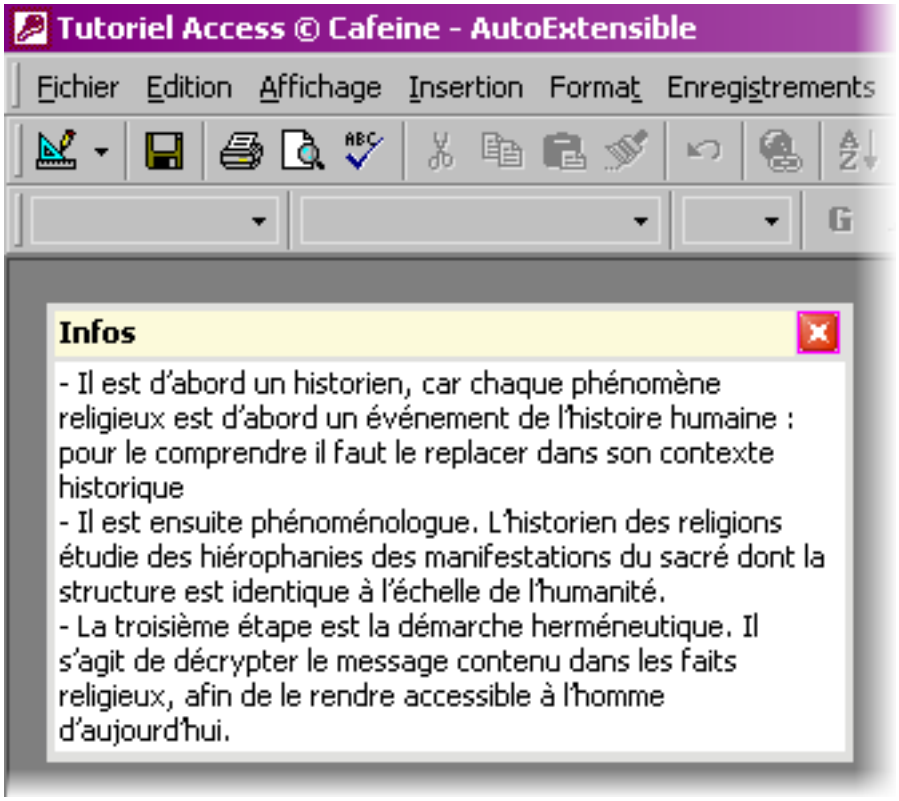
```
Private Declare Function ReleaseCapture Lib "user32" () As Long
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" ( _
            ByVal hwnd As Long, _
            ByVal wMsg As Long, _
            ByVal wParam As Long, _
            lParam As Any) As Long

Private Const WM_NCLBUTTONDOWN = &HA1
Private Const HTCAPTION = 2

Private Sub lblBarreTitre_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)

    ReleaseCapture
    SendMessage Me.hwnd, WM_NCLBUTTONDOWN, HTCAPTION, 0&

End Sub
```

IV-B - Utilisation du formulaire

Nous voyons le fonctionnement de notre formulaire d'information qui fonctionne correctement.



Auto réduction du formulaire



Auto extension du formulaire

V - Téléchargement

Vous pouvez télécharger l'application exemple en cliquant ici :

- **version 97**

- **version 2000**

VI - Conclusion

Ce court tutoriel nous a permis de réaliser un formulaire d'information dont la taille s'adapte parfaitement au contenu.

Nous avons dû contourner une limitation d'Access qui ne sait pas gérer le redimensionnement dynamique en mode formulaire.

J'espère que cet article vous aura familiarisé avec quelques techniques de codage de contrôles ou vous aura donné quelques idées.

Merci à Arkham46 pour son ouverture vers les fonctions non documentées qui feront sans doute l'objet d'un prochain article.

Merci aux membres de la rédaction pour leur relecture attentive et leurs remarques constructives.