Personnaliser vos barres de commandes dans Access

par Philippe JOCHMANS (home page de Starec)

Date de publication : 08/02/2008

Dernière mise à jour : 08/02/2008

Cet article a pour but de vous expliquer comment créer et personnaliser vos barres de commandes dans Access.

I - INTRODUCTION

Pour personnaliser vos applications, vous pouvez créer vos propres barres de menus, barres d'outils et menus contextuels.

Cet article a pour but de vous montrer comment réaliser, grâce à VBA, vos propres barres de commandes. Nous n'étudierons que la création des barres par VBA, car le code nous offre beaucoup plus de possibilités que la réalisation de celles-ci à partir des menus personnalisés.

Tous les tests ont été réalisés sous Access 2002. Cependant, comme il s'agit d'une bibliothèque Office que nous allons utiliser, tout ce que vous verrez dans cet article est applicable à Access, Excel et Word.

Téléchargement : Vous pouvez télécharger une base Access, avec des listings, et la possibilité de visualiser toutes les icônes des barres de commandes.
Vous trouverez la base à télécharger en bas de cet article.

Ce article sera constitué de plusieurs parties

- Une partie théorique : Vous verrez les différentes propriétés des barres de commandes et de leurs composants.
- Une partie pratique : Nous y verrons comment créer ses propres barres de commandes.
- Une partie bonus : Avec des exemples et des trucs et astuces.

II - PREREQUIS

Une bonne connaissance de VBA est nécessaire pour la lecture de cet article, car comme il a été stipulé dans l'introduction nous traiterons uniquement de la création de barres de commandes par le code.

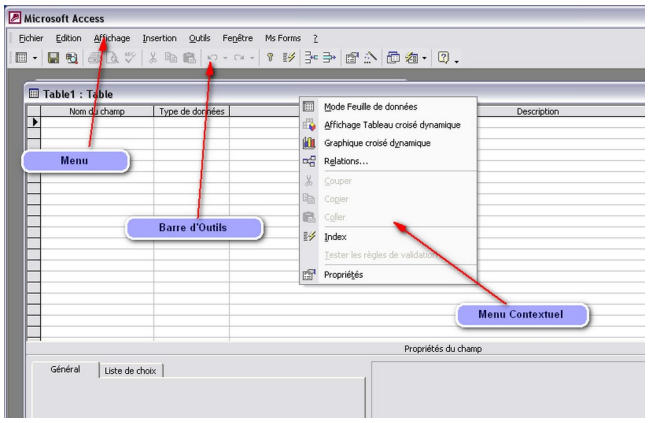
Je vous invite à lire cet article de Tofalu : **Ajouter un menu dans un formulaire**

Pour réaliser toutes les manipulations sur le code il faut référencer la bibliothèque Microsoft Office 10.0 Object Library (10, car cela correspond à la version de l'office 2002, 9 pour 2000 et 11 pour 2003).

Nous ne verrons pas toutes les propriétés, mais celles qui à mon avis sont les plus intéressantes pour la création de ses propres barres de commandes. C'est pourquoi l'utilisation de l'aide (F1) et de l'explorateur d'objet (F2) est indispensable (personnellement je m'en sers tous les jours).

III - DEFINITION

Il est important de savoir de quoi on parle, nous allons donc définir les différents types de barres de commandes représentées sur l'image ci-dessous.



III-A - Les barres de menus

Les barres de menus sont celles que vous trouverez dans toutes les applications Office, vous les connaissez car l'homogénéité de toutes les applications qui fonctionnent sous Windows fait que les barres de menus ont souvent la même structure :

Fichier - Edition - Affichage #..

Cela est devenu un standard, il faudra en tenir compte lorsque vous réaliserez vos propres barres de menus, cela évitera de dérouter les utilisateurs de vos applications.

Vous constaterez que souvent celles-ci sont évolutives en fonction de l'action que vous êtes en train d'effectuer (par exemple sous Access, la barre évolue lorsque vous êtes en mode création de requête).

III-B - Les barres d'outils

Les barres d'outils sont principalement constituées d'icônes, il s'agit en fait (la plupart du temps) de raccourcis que vous trouverez dans les menus.

III-C - Les menus contextuels

Les menus contextuels apparaissent lorsque vous faites un clic sur le bouton droit de votre souris (bien sûr si celle-ci est configurée en droitier), on les appellent aussi menus flottants.

IV - CONNAITRE LES BARRES DE COMMANDES

Avant d'attaquer la création des barres de commandes, nous allons voir la collection CommandBars qui contient l'ensemble des barres de commandes.

IV-A - Apprendre à connaître les barres de commandes

IV-A-1 - ActiveMenuBar

Cette propriété permet de connaître quelle barre est active, elle renvoie l'identifiant d'un objet CommandBar.

Le code suivant renvoie le nom de la barre de commandes active.

VBA

```vba
MsgBox Application.CommandBars.ActiveMenuBar.Name
```

IV-A-2 - DisableCustomize

Cette propriété permet de bloquer la possibilité de personnaliser une barre de commandes.

Cela peut-être utile lors de la création de vos barres de commandes, car en effet si vous distribuez un fichier mdb, un utilisateur peut vouloir ajouter un bouton n'importe où. Hors cela peut faire planter votre application, au niveau de la gestion de vos boutons sur vos barres, car ceux-ci sont repérés par rapport à une position.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Bloque la possibilité de personnaliser une barre de commandes
- **False (Faux)** : Offre la possibilité de personnaliser une barre de commandes

*Cette propriété est valable pour l'ensemble des barres de commandes de votre application. Cependant, il est possible d'instaurer une protection barre par barre, pour cela aller jeter un coup d'oeil à la propriété **Protection** de l'objet **CommandBar**.*

IV-B - Aspect visuel des barres de commandes

Nous pouvons changer l'aspect visuel des barres de commandes, cependant il faut prendre en compte deux choses :

- Comme il s'agit de la collection des barres de commandes, l'aspect s'appliquera à toutes les barres de commandes de votre application.
- Comme nous utilisons des barres de commandes Office, certaines actions seront répercutées dans les autres applications Office (Excel, Word).

IV-B-1 - AdaptiveMenus

Cette propriété permet de modifier l'aspect des menus pour savoir s'ils sont adaptables ou non. Cela correspond à la propriété que l'on trouve dans le menu contextuel lorsque l'on veut personnaliser les menus et qui se trouve dans l'onglet options à "Afficher les menus dans leur intégralité".

Le terme est un peu barbare, en fait c'est quand un menu est entièrement déroulé ou non en fonction des dernières actions de l'utilisateur. L'image ci-dessous vous permettra de mieux comprendre ce phénomène.


AdaptiveMenus = True


AdaptiveMenus = False

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Le menu est adaptable.
- **False (Faux)** : Le menu n'est pas adaptable.

IV-B-2 - DisableAskAQuestionDropdown

Cette propriété permet d'activer ou de désactiver le menu déroulant de l'aide intuitive qui se trouve dans la partie de droite de votre barre de commandes.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : L'aide est désactivée.
- **False (Faux)** : L'aide est active.

IV-B-3 - DisplayFonts

Cette propriété a pour but de modifier l'aspect de la liste déroulante des polices. En effet, vous avez la possibilité de voir le nom des polices avec leur fonte ou avec la fonte système.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Le nom des polices apparaissent avec leur propre fonte.
- **False (Faux)** : Le nom des polices apparaît avec la fonte système.

IV-B-4 - DisplayKeysInTooltips

Cette propriété permet d'activer ou désactiver la visualisation des raccourcis clavier dans les infos bulles.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Les raccourcis claviers apparaissent dans les infos bulles.
- **False (Faux)** : Les raccourcis claviers n'apparaissent pas dans les infos bulles.

IV-B-5 - DisplayTooltips

Cette propriété permet ou non l'affichage des infos bulles sur les barres d'outils lors du survol de la souris sur les boutons.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Les infos bulles apparaissent.
- **False (Faux)** : Les infos bulles n'apparaissent pas.

IV-B-6 - LargeButtons

Cette propriété permet de déterminer l'aspect des boutons de commandes des barres d'outils (ne s'applique pas aux barres de menus et aux menus contextuels).

Les images des boutons peuvent prendre deux tailles suivant la valeur de la propriété.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Les boutons des barres d'outils sont agrandis.

- **False (Faux)** : Les boutons des barres d'outils sont réduits.

Cela ne concerne que l'image qui apparaît sur les boutons et non le texte qui peut l'accompagner.

IV-B-7 - MenuAnimationStyle

Cette propriété permet de déterminer le style de l'animation des menus et des menus contextuels.

Cette propriété peut prendre 4 valeurs :

- msoMenuAnimationNone (valeur 0).
- msoMenuAnimationRandom (valeur 1).
- msoMenuAnimationSlide (valeur 3).
- msoMenuAnimationUnfold (valeur 2).

Testez ces différentes valeurs pour voir les effets possibles.

IV-B-8 - Interférence avec les autres applications Office

Le fait de changer l'aspect visuel des barres de commandes peut toucher les autres applications du pack Office (Word, Excel et Outlook), il faut donc prendre en considération ce phénomène pour ne pas perturber les utilisateurs qui peuvent à la fois travailler sur une application Access et en même temps travailler sur Word.

Le tableau suivant vous permet de savoir si l'utilisation de ces propriétés a un effet ou non sur les autres applications Office.

Propriété	Interférence
AdaptiveMenus	OUI
DisableAskAQuestionDropdown	NON
DisplayFonts	OUI
DisplayKeyInTooltips	OUI
DisplayTooltips	OUI
LargeButtons	OUI
MenuAnimationStyle	OUI

IV-C - L'objet CommandBar

L'objet **CommandBar** représente chaque barre de commandes de votre application.

Il existe donc trois types de barres de commandes.

- Les barres de menus : **msoBarTypeMenuBar** (valeur = 1).
- Les barres d'outils : **msoBarTypeNormal** (valeur = 0).
- Les menus contextuels : **msoBarTypePopup** (valeur = 2).

Nous allons donc voir les quelques propriétés pour connaître les barres de commandes (l'utilisation de l'explorateur d'objet vous permettra de consulter les propriétés non présentes dans ce chapitre).

IV-C-1 - Type

Il y a donc trois types de barres de commandes, le code suivant vous permet d'avoir la liste des barres de commandes par type.

VBA

```
Public Sub NbrParBarre()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Cette procédure va vous permettre d'afficher dans une msgbox le nombre
    '                 de barres de commandes par type dans Access
    '================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim lngCpteMenu As Long
    Dim lngCpteOutil As Long
    Dim lngCpteContextuel As Long
    Dim strMessage As String

    ' ===== boucle =====
    For Each cmb In CommandBars
        If
 cmb.BuiltIn = True Then ' pour ne prendre que les barres de commandes définies par l'application
            If cmb.Type = msoBarTypeMenuBar Then
                lngCpteMenu = lngCpteMenu + 1
            End If
            If cmb.Type = msoBarTypeNormal Then
                lngCpteOutil = lngCpteOutil + 1
            End If
            If cmb.Type = msoBarTypePopup Then
                lngCpteContextuel = lngCpteContextuel + 1
            End If
        End If
    Next cmb

    ' ===== préparation du message =====
    strMessage = "Comptage des barres de commandes par type :" & vbCrLf & vbCrLf & _
                 "Barre de menu : " & lngCpteMenu & vbCrLf & _
                 "Barre d'outils : " & lngCpteOutil & vbCrLf & _
                 "Menu Contextuel : " & lngCpteContextuel & vbCrLf & vbCrLf & _
                 "Soit un total de " & lngCpteMenu + lngCpteOutil +
 lngCpteContextuel & " barres de commandes."

    ' ===== affichage du message =====
    MsgBox strMessage, vbOKOnly + vbInformation, "Comptage des barres de commandes"


End Sub
```

Ce qui nous donne le résultat suivant :

Le nombre peut varier en fonction de la version d'Access, sauf pour la barre de menu, car il y en a toujours une.
Vous pouvez bien sûr lancer ce code dans les autres applications Office, et vous aurez alors des résultats différents.

IV-C-2 - Position

Les barres de commandes peuvent avoir plusieurs positions :

- **msoBarTop** : la barre est positionnée en haut de l'écran
- **msoBarBottom** : la barre est positionnée en bas de l'écran
- **msoBarLeft** : la barre est positionnée sur la gauche de l'écran
- **msoBarRight** : la barre est positionnée sur la droite de l'écran
- **msoBarFloating** : la barre est une barre de menu contextuel

Vous verrez lors de la création des barres de commandes, que cette propriété et une autre vont vous permettre de définir le type de barre que vous allez créer.

IV-C-3 - Name et NameLocal

Ces deux propriétés vous permettent de connaître le nom (**Name**) en Anglais d'une barre de commandes et dans la langue de l'application (**NameLocal**) le nom de cette barre.

Vous trouverez dans la base à télécharger en cliquant sur le bouton " Tous les Menus Access " la liste complète des barres de commandes de votre version d'Access.

IV-C-4 - Index

En plus d'être repérée par son nom, une barre de commandes peut-être repérée par son Index. Il s'agit d'une valeur numérique de type long.

Vous retrouverez cette valeur dans la liste des barres de commandes des fichiers à télécharger.

*Vous pouvez faire appel à une barre de commandes par son Nom (**Name**) ou son **Index** (la propriété **NameLocal** ne peut-être utilisé).*

Les deux instructions suivantes sont équivalentes :

VBA
```
MsgBox Application.CommandBars("DataBase").Index
MsgBox Application.CommandBars(2).Index
```

Elles vous renvoient la valeur 2 qui est l'index de la barre de commandes DataBase (Base de données).

IV-C-5 - BuiltIn

Cette propriété vous permet de savoir si votre barre de commandes est définie par votre application ou s'il s'agit d'une barre de commandes personnalisée (en lecture seule).

Celle-ci peut prendre deux valeurs :

- **True (Vrai)** : Si la barre de commandes est définie par votre application.
- **False (Faux)** : Si la barre de commandes est une barre de commandes personnalisée.

Exemple :

A force de faire des tests pour la rédaction de cet article, ma base Access était recouverte d'une multitude de petites barres. Le code suivant m'a permis de me débarrasser de tous ces parasites.

VBA
```
Public Sub SupBarrePerso()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Cette routine va permettre de supprimer toutes les barres personnalisées
    '================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar

    ' ===== suppression =====
    For Each cmb In Application.CommandBars
        If cmb.BuiltIn = False Then
            ' si la barre de commandes est une barre personnalisée
            Application.CommandBars(cmb.Name).Delete
        End If
    Next cmb

End Sub
```

IV-C-6 - Controls

Cette propriété permet de connaître les contrôles d'une barre de commandes.

Le code suivant va vous permettre d'afficher dans la fenêtre exécution (Ctrl + G) la liste des contrôles d'une barre de commandes.

VBA
```
Public Sub ListeControls(IdIndex As Long)
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Cette routine permet d'avoir la propriété Caption d'une barre de commande
    '                 passée en paramètre
    '================================================================================

    Dim ctrl As Office.CommandBarControl

    For Each ctrl In Application.CommandBars(IdIndex).Controls
        Debug.Print ctrl.Caption
    Next ctrl
End Sub
```

Il suffit de passer en paramètre l'index de la barre de commandes.

Vous trouverez la liste des contrôles dans les fichiers à télécharger en cliquant sur le bouton " Tous les Détails des Barres "

IV-C-7 - Enabled

Cette propriété va vous permettre de déterminer ou de décider si une barre de commandes sera active ou non.

Celle-ci peut prendre deux valeurs :

- **True (Vrai)** : la barre de commandes est active.
- **False (Faux)** : la barre de commandes est inactive.

Le code suivant va vous permettre de désactiver toutes les barres de commandes d'une application Office.

VBA
```
Dim cmb As Office.CommandBar
For Each cmb In Application.CommandBars
    cmb.Enabled = False
Next cmb
```

IV-C-8 - Delete

Cette propriété permet de détruire une barre de commandes.

L'utilisation la plus courante que je fais avec cette propriété, c'est lorsque je crée des barres de commandes dynamiques lors de l'exécution d'une application.

En effet, si vous créer une barre de commandes et que celle-ci existe déjà, votre application va générer une erreur. Pour cela il faut détruire la barre, mais si c'est la première fois que le code tourne, il y aura une erreur parce que la barre n'existe pas.

La solution est de mettre ces deux lignes de codes au début de la routine de création de barres.

VBA
```
On Error Resume Next
Application.CommandBars("MaBarre").Delete
```

IV-C-9 - Height

Cette propriété va permettre de connaître ou d'affecter une hauteur à une barre de commandes.

La ligne suivante renvoie la hauteur en pixels de la barre d'outil DATABASE.

VBA
```
MsgBox Application.CommandBars("DATABASE").Height
```

*Contrairement aux dimensions des contrôles en VBA, la propriété **Height** est un entier de type long en **Pixels** et **Non en Twips**.*

IV-C-10 - Width

Cette propriété va permettre de connaître ou d'affecter une largeur à une barre de commandes (en **Pixels**).

IV-C-11 - Left

Cette propriété permet de récupérer ou de positionner une barre de commandes par rapport au bord gauche de la fenêtre de l'application (Cette distance est exprimée en **Pixels**).

IV-C-12 - Top

Cette propriété permet de récupérer ou de positionner une barre de commandes par rapport au bord haut de la fenêtre de l'application (Cette distance est exprimée en **Pixels**).

IV-C-13 - Protection

Il est possible d'assurer une protection à une barre de commandes pour éviter certaines manipulation malheureuses d'un utilisateur.

Cette propriété peut prendre plusieurs valeurs :

Propriété	Commentaires
msoBarNoChangeDock	Votre barre de commandes est positionnée en haut de la fenêtre (Position : **msoBarTop**), vous ne pouvez pas la déplacer sur le reste de la fenêtre. Votre barre de commandes est une barre d'outils flottante (Position : **msoBarFloating**) , vous ne pouvez pas mettre votre barre de commandes sous la barre de menus, elle restera toujours flottante.
msoBarNoChangeVisible	Vous ne pouvez fermer la barre de commandes (la croix que apparaît sur une barre d'outils flottante disparaît).
msoBarNoCustomize	Vous ne pourrez personnaliser votre barre de commandes (même si vous pouvez afficher la fenêtre de personnalisation des barres, aucun contrôle ne pourra se positionner sur celle-ci).
msoBarNoHorizontalDock	Vous ne pourrez pas positionner une barre de commandes (en général une barre d'outils) horizontalement.
msoBarNoMove	Vous ne pourrez pas déplacer la barre de commandes.
msoBarNoProtection	Il n'y a aucune protection sur votre barre de commandes, l'utilisateur peut en faire ce qu'il veut donc DANGER !
msoBarNoResize	Empêche le redimensionnent d'une barre de commandes.
msoBarNoVerticalDock	Vous ne pourrez pas positionner une barre de commandes (en général une barre d'outils) verticalement.

Pour cumuler les protections, il suffit d'ajouter les paramètres de la même manière que pour une MsgBox.
Le code suivant empêche le déplacement d'une barre et de la fermer.

VBA

```
cmb.Protection = msoBarNoMove + msoBarNoChangeVisible
```

IV-C-14 - Reset

Cette méthode a pour but de réinitialiser les propriétés d'une barre de commandes.

Cette propriété est très utile lorsque vous avez effectué des manipulations sur les barres de commandes définies par votre application. Cela va vous permettre de les retrouver dans leur état d'origine.

IV-D - Les éléments composants les barres de commandes

Les barres de commandes sont composées de différents types de contrôles qui se décomposent en 3 catégories :

CommandBarButton : Il s'agit d'un simple bouton, où un clic enclenche une action.

CommandBarComboBox : Cette catégorie regroupe plusieurs styles de listes déroulantes.

CommandBarPopup : Cette catégorie regroupe plusieurs styles de menus contextuels.

Nous allons voir en détail ces différentes catégories et les propriétés communes. Les instructions seront survolées, elles seront abordées en détail lors des créations des barres de commandes.

IV-D-1 - CommandBarControl

Les propriétés que nous allons voir ici sont communes à toutes les catégories.

IV-D-1-a - BeginGroup

Cette propriété permet de déterminer si un contrôle est le premier d'un groupe de contrôles.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : C'est le premier contrôle du groupe de contrôles.
- **False (Faux)** : Ce n'est pas le premier contrôle du groupe de contrôles.

L'utilisation de cette instruction vous permettra de positionner un séparateur avant le contrôle, ainsi vous pourrez visuellement séparer des groupes de contrôles de barres de commandes par thème.

IV-D-1-b - BuiltIn

Cette propriété permet de savoir s'il s'agit d'un contrôle définie par l'application ou d'un contrôle personnalisé.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Il s'agit d'un contrôle définie par l'application.
- **False (Faux)** : Il s'agit d'un contrôle personnalisé.

IV-D-1-c - Caption

Cette propriété définie le texte qui se trouve sur un contrôle de barre de commandes. Celui-ci apparaît aussi dans l'info bulle si aucun texte n'est défini pour celui-ci.

Le fait de mettre un texte dans la propriété **Caption** ne veut pas dire que celui-ci s'affichera, il faudra aussi jouer avec la propriété **Style** (qui sera vue plus loin).

IV-D-1-d - TooltipText

Cette propriété permet d'afficher un message dans l'info bulle lors du survol du contrôle par la souris.

*Si vous ne spécifiez aucune chaîne de texte pour cette propriété, c'est la valeur de la propriété **Caption** qui sera affichée.*

IV-D-1-e - Enabled

Cette propriété active ou désactive un bouton de barre de commandes. Cela a pour effet de le griser ou de le dégriser en fonction de la valeur de cette propriété.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Le contrôle est actif.
- **False (Faux)** : Le contrôle est inactif.

IV-D-1-f - Id

Cette propriété permet de connaître l'action du contrôle, celle-ci ne s'applique qu'aux contrôles définis par l'application, car les contrôles que vous créez auront la valeur 1. (Vous retrouverez les Id pour chaque contrôle dans les fichiers en téléchargement.)

Dans la partie Bonus, vous trouverez des utilisations de cette propriété.

IV-D-1-g - Index

Cette propriété représente la position du contrôle dans une barre de commandes.

*Ne pas confondre **Index** avec **Id**, car **Id** représente une valeur qui détermine l'action du contrôle dans l'application.*

IV-D-1-h - Parent

Cette propriété (en lecture seule) permet de connaître des informations sur la barre de commandes à laquelle appartient le contrôle (son nom, etc #)

IV-D-1-i - OnAction

Cette propriété permet de définir l'action d'un bouton de barre de commandes.

Il peut s'agir d'une macro (Access), d'une routine ou d'une fonction, voici les différentes utilisation possibles de cette propriété.

Lancer une macro (Access) :

Le code suivant va permettre de lancer une macro nommée MaMacro, qui est affectée au 1er bouton de la barre de commandes Test.

```VBA
Application.CommandBars("Test").Controls(1).OnAction = "MaMacro"
```

Ici je parle de macro Access, et non de macro Excel ou Word, car les termes peuvent prêter à confusion.

Lancer une fonction (Function) :

Le code suivant dans la même lignée va lancer une fonction MaFonction.

```VBA
Application.CommandBars("Test").Controls(1).OnAction = "=MaFonction()"
```

Lancer une routine (Sub) :

Le code suivant va de la même manière lancer une routine MaRoutine.

```VBA
Application.CommandBars("Test").Controls(1).OnAction = "MaRoutine"
```

L'utilisation des macros (Access) est déconseillée, en effet elles ne permettent pas de mettre en place une gestion d'erreur.
Les fonctions (Function) et les routines (Sub) doivent être déclarées en Public et dans un module, faites un module comportant toutes les instructions de vos menus, vous pourrez ainsi les retrouver plus facilement.
Vous trouverez un exemple d'utilisation dans la partie Bonus, dans la création de lettres d'un répertoire.

IV-D-1-j - Execute

Cette méthode va vous permettre de lancer l'action définie par la propriété **OnAction**, vous en trouverez un exemple dans la partie Bonus qui traite de l'utilisation des fonctions des barres de commandes.

IV-D-1-k - Delete

Cette instruction permet de supprimer un contrôle de barre de commandes.

IV-D-1-l - Visible

Cette propriété va permettre de rendre visible ou invisible un bouton de barre de commandes.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : Le contrôle est visible.
- **False (Faux)** : Le contrôle est invisible.

IV-D-1-m - SetFocus

Cette méthode donne le focus au bouton spécifié.

IV-D-2 - CommandBarButton

Le contrôle bouton de commandes est le bouton standard des barres de commandes. Il reprend les instructions ci-dessus et de nouvelles propres à ce contrôle.

IV-D-2-a - BuiltInFace

Cette propriété permet de définir si l'image du bouton de commandes, est son image par défaut. Cela s'applique aux barres de commandes prédéfinies par l'application.

Cette propriété peut prendre deux valeurs :

- **True (Vrai)** : L'image est celle définie par défaut.
- **False (Faux)** : L'image n'est pas celle définie par défaut.

IV-D-2-b - Copy

Cette propriété permet de copier un contrôle existant dans une nouvelle barre de commandes. Nous aborderons son utilisation lors de la création des barres de commandes personnalisées.

IV-D-2-c - CopyFace

Cette propriété permet de copier l'image du bouton pour la mettre sur un autre.

Contrairement à l'instruction **Copy** qui elle copie le contrôle et toutes ses propriétés, avec **CopyFace**, nous ne récupérons que l'image.

IV-D-2-d - FaceId

Cette propriété nous donne le numéro de l'image d'un bouton d'une barre de commandes. Pour avoir les numéros de toutes les images, essayez le formulaire qui se trouve dans la base en téléchargement.

IV-D-2-e - Style

Sur les boutons de commandes nous pouvons avoir différents styles en combinant le texte et les icônes.

Ceux-ci sont :

- **msoButtonAutomatic**
- **msoButtonCaption**
- **msoButtonIcon**
- **msoButtonIconAndCaption**
- **msoButtonIconAndCaptionBelow**
- **msoButtonIconAndWrapCaption**

- **msoButtonIconAndWrapCaptionBelow**
- **msoButtonWrapCaption**

Le code suivant va vous afficher une barre d'outils avec les différents styles, ce qui vous permettra de mieux vous rendre compte de l'impact visuel.

VBA

```
Public Sub StyleCommandButton()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Le code suivant va vous permettre d'afficher différents styles de boutons
    '================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton

    ' ===== suppression de la barre existante =====
    On Error Resume Next
    Application.CommandBars("StyleButton").Delete

    ' ===== création de la barre =====
    Set cmb = Application.CommandBars.Add("StyleButton", msoBarFloating)
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonAutomatic"
        .FaceId = 25
        .Style = msoButtonAutomatic
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonCaption"
        .FaceId = 25
        .Style = msoButtonCaption
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonIcon"
        .FaceId = 25
        .Style = msoButtonIcon
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonIconAndCaption"
        .FaceId = 25
        .Style = msoButtonIconAndCaption
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonIconAndCaptionBelow"
        .FaceId = 25
        .Style = msoButtonIconAndCaptionBelow
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonIconAndWrapCaption"
        .FaceId = 25
        .Style = msoButtonIconAndWrapCaption
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
```

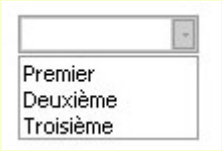
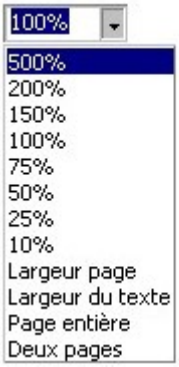
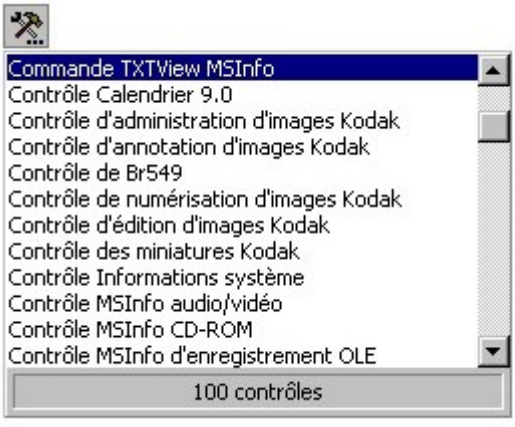
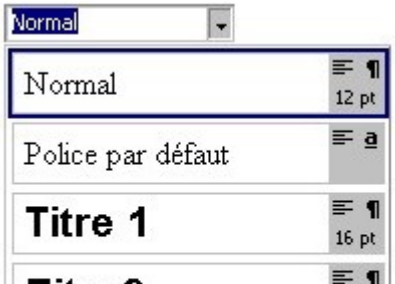
VBA
```
        .Caption = "msoButtonIconAndWrapCaptionBelow"
        .FaceId = 25
        .Style = msoButtonIconAndWrapCaptionBelow
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .BeginGroup = True
        .Caption = "msoButtonWrapCaption"
        .FaceId = 25
        .Style = msoButtonWrapCaption
    End With

    ' ===== rendre la barre visible =====
    cmb.Visible = True

End Sub
```

Le résultat est le suivant.

IV-D-3 - CommandBarComboBox

Il s'agit des boutons de commandes qui ont l'aspect d'une liste déroulante. il y'en a plusieurs, en voici la liste et leur aspect.

Cependant, ne rêvez pas en voyant ces multiples graphiques, car il n'est pas possible de tous les utiliser.

Contrôle	Image
msoControlButtonDropdown	
msoControlComboBox	
msoControlDropdown	
msoControlEdit	
msoControlSplitDropdown	
msoControlOCXDropDown	
msoControlGraphicCombo	

De tous ces contrôles nous ne pourrons en utiliser que 3 pour la création de barres de commandes personnaliséses : **msoControlEdit, msoControlDropdown et msoControlComboBox**.

Les listes déroulantes ont d'autres propriétés que nous allons évoquer.

IV-D-3-a - AddItem

Cette méthode va nous permettre de rajouter des éléments à la liste.

IV-D-3-b - Clear

Cette méthode va nous permettre de supprimer tous les éléments d'une zone de liste.

IV-D-3-c - DropDownLines

Va nous permettre de déterminer le nombre de ligne d'une zone de liste déroulante.

IV-D-3-d - DropDownWidth

Permet de déterminer la largeur de la liste (en pixels).

IV-D-3-e - List

Permet de récupérer la valeur de la liste qui a été sélectionnée par rapport à sa position donnée par la propriété **ListIndex**.

IV-D-3-f - ListCount

Permet de récupérer le nombre de lignes de la liste.

IV-D-3-g - ListHeaderCount

Cette propriété va permettre de réaliser un trait de séparation dans une liste déroulante. Le chiffre que vous allez affecter à cette propriété déterminera le nombre de lignes avant cette séparation.

IV-D-3-h - ListIndex

Cette propriété va permettre de récupérer le numéro de la ligne sélectionnée.

IV-D-3-i - RemoveItem

Permet de supprimer une ligne de la liste.

IV-D-3-j - Style

Une liste peut avoir deux styles :

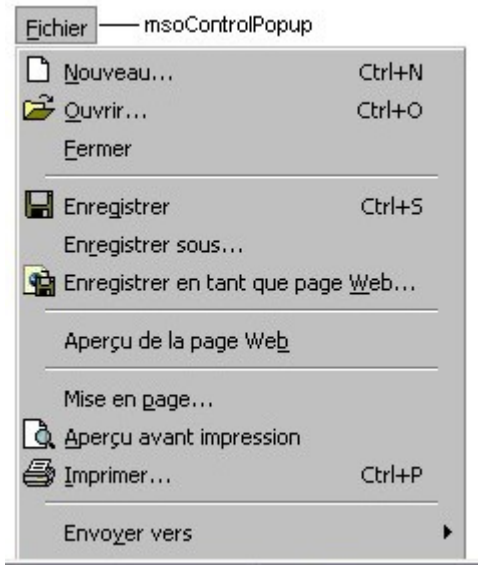
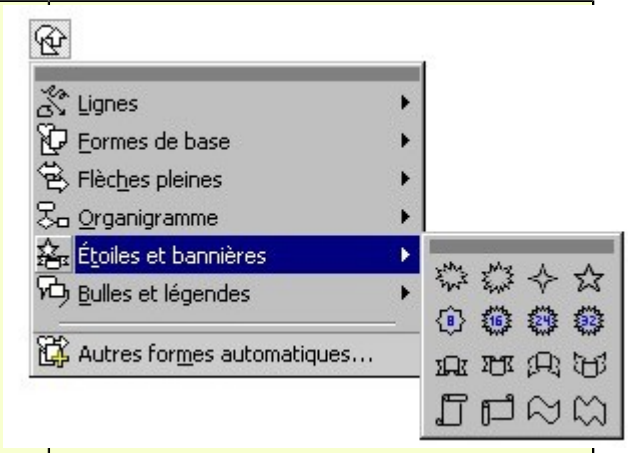
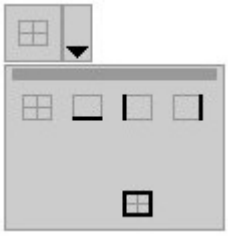
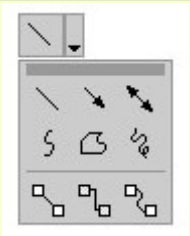
- **msoComboLabel** ; Une étiquette est affectée à la liste, et est remplie par la propriété **Caption**.

- **msoComboNormal** : Il n'y a pas d'étiquette.

IV-D-4 - CommandBarPopup

Ce contrôle permet d'ajouter un sous-menu à une barre de commandes.

Voici quelques exemples de type de **CommandBarPopup**.

msoControlPopup	
msoControlButtonPopup	
msoControlSplitButtonPopup	
msoControlSplitButtonMRUPopup	

De toutes ces contrôles nous ne pourrons en utiliser qu'un seul, le contrôle **msoControlPopup**

V - CREATION DE BARRES DE COMMANDES PERSONNALISEES

Nous allons maintenant rentrer dans le vif du sujet après toute cette théorie en réalisant nous-mêmes nos barres de commandes, nous allons mettre en pratique tout ce qui a été vu précédemment, et découvrir d'autres propriétés.

V-A - Créer un Menu

V-A-1 - Présentation

Il est important que votre menu ait une présentation similaire aux menus que vous trouvez dans toutes les applications Windows. Le but est de ne pas perturber les utilisateurs de vos applications, il faut garder une certaine logique de présentation.

- **Fichier** : Avec au minimum le bouton pour quitter l'application.
- **Edition** : Si vous avez des opérations de copier coller.
- **Outils** : Si par exemple vous avez des paramétrages personnalisés.
- **?** : Si vous avez décider de joindre à votre application un fichier d'aide.

V-A-2 - Ajouter une barre de menu

Lorsque l'on ajoute une barre de menus, c'est pour remplacer la barre de menus d'Access par la sienne.

Pour ajouter une barre de menus nous allons utiliser la méthode **Add** de la collection **CommandBars**.

Syntaxe : Add(Name, Position, MenuBar, Temporary)

- **Name** : Valeur de type texte pour nommer votre menu
- **Position** : Position de la barre de menus. Ce paramètre peut prendre plusieurs valeurs, pour une barre de menus nous allons prendre la valeur : **msoBarTop**, car une barre de menu se positionne toujours en haut.
- **MenuBar** : Ce paramètre permet de déterminer si la barre de menus doit remplacer la barre de menus existante, nous allons mettre cette valeur à True (Vrai).
- **Temporay** : Pour déterminer si cette barre est temporaire, cela signifie que celle-ci sera détruite lors de la fermeture de l'application.

Le code suivant va créer une barre de menus nommée " MaBarre "

VBA
```
Public Sub CreerMenu()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Création d'une barre de menu
    '================================================================================

    ' ==== déclaration =====
    Dim cmb As Office.CommandBar

    ' ===== affectation =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True, True)

    ' ===== rendre la barre visible =====
    cmb.Visible = True
End Sub
```

Nous avons maintenant une barre de menus vide qui a remplacé la barre de menus d'Access.

Si vous lancer ce code, vous verrez donc votre menu Access disparaître.
Fermez votre base et ré ouvrez là, vous verrez votre barre de menus Access ré apparaître, cela justifie le caractère temporaire, car votre barre a été détruite lorsque vous quittez l'application.
N'oubliez pas de rendre votre barre visible.

*Lorsque l'on déclare l'objet cmb, on peut se passer du rappel de la bibliothèque Office, il est préférable de la garder, car si vous faites appel à une autre bibliothèque qui a un objet **CommandBar**, Access risque de "perdre les pédales".*

V-A-3 - Ajouter des boutons à un menu

Nous allons maintenant ajouter des boutons à notre menu.

Un bouton ne permettra pas d'ajouter un sous-menu, un click sur celui-ci lancera une action.

Pour ajouter un bouton de commande nous allons utiliser la méthode **Add**, mais qui cette fois va ajouter un bouton à la collection des contrôles d'une barre de commandes.

Syntaxe : Add(Type, Id, Parameter, Before, Temporary)

- **Type** : Il s'agit du type de bouton, comme nous créons un bouton de commande , nous utiliserons le type **msoControlButton**
- **Id** : Détermine l'Id du bouton de commande, comme nous allons créer notre propre bouton, nous allons utiliser la valeur 1 (Ce paramètre est facultatif).
- **Parameter** : Va nous permettre de passer un paramètre à l'action qui sera déclenchée par le clic sur le bouton.
- **Before** : Indique la position du contrôle, en fait il l'intercale avant le contrôle qui avait cette position. (Ce paramètre est facultatif, et sans indication, le contrôle se positionnera à la fin de la barre de commandes).
- **Temporary** : Indique si le contrôle est temporaire ou non (un peu comme les barres de commandes), si sa valeur est **True**, celui-ci sera détruit à la fermeture de l'application, si c'est **False** (valeur par défaut et faculative), le contrôle sera conservé.

Le code suivant ajoutera deux boutons à notre menu.

VBA

```
Public Sub CreerMenu()
    ' ============================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Création d'une barre de menu
    '============================================================================================

    ' ==== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton

    On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== affectation =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True, True)

    ' ===== ajout de boutons =====
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "Bouton 1"
        .Style = msoButtonCaption
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "Bouton 2"
        .Style = msoButtonCaption
    End With

    ' ===== rendre la barre visible =====
    cmb.Visible = True
```

VBA
```
End Sub
```

Vous avez sans doute noté que nous avons ajouté les deux lignes suivantes :

VBA
```
On Error Resume Next
Application.CommandBars("MaBarre").Delete
```

En effet, le but de cet article est de créer des menus, mais ceux-ci peuvent être dynamiques en fonction de l'utilisateur. Hors lorsque vous relancer la routine de création de menus, il faut détruire le précédent. Mais si c'est la première fois, le code ne trouvera pas la barre de commandes et générera une erreur.
Ces deux lignes permettent de lever cette erreur, et le processus peut se dérouler correctement.

V-A-4 - Ajouter un sous-menu à un menu

Notre menu comporte que deux boutons, nous allons donc ajouter un sous-menu, et dans ce sous menu un autre sous-menu.

Nous en profiterons pour ajouter des images.

VBA
```
Public Sub CreerMenuSousMenu()
    ' ================================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Création d'une barre de menu
    '================================================================================================

    ' ==== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton
    Dim subCmb As Office.CommandBarPopup
    Dim subCmb1 As Office.CommandBarPopup

    On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== affectation =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True, True)

    ' ===== ajout de boutons =====
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "&Bouton 1"
        .Style = msoButtonCaption
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "B&outon 2"
        .Style = msoButtonCaption
    End With

    ' ===== ajout du sous-menu =====
    Set subCmb = cmb.Controls.Add(msoControlPopup)
    subCmb.Caption = "So&us-Menu"
    ' ===== on ajoute 1 bouton au sous-menu avec une image =====
    Set btn = subCmb.Controls.Add(msoControlButton)
```

```VBA
    With btn
        .FaceId = 25
        .Caption = "&Recherche"
        .Style = msoButtonIconAndCaption
    End With
    ' ==== on ajoute un sous-menu au sous-menu avec deux boutons =====
    Set subCmb1 = subCmb.Controls.Add(msoControlPopup)
    subCmb1.Caption = "Sous-Menu2"
    Set btn = subCmb1.Controls.Add(msoControlButton)
    With btn
        .Caption = "&toto"
        .Style = msoButtonCaption
    End With
    Set btn = subCmb1.Controls.Add(msoControlButton)
    With btn
        .Caption = "t&ruc"
        .Style = msoButtonCaption
    End With


    ' ===== rendre la barre visible =====
    cmb.Visible = True
End Sub
```

C'est cette ligne **Dim subCmb As Office.CommandBarPopup** qui va déterminer le type de barre.

*Nous avons mis devant une lettre l'esperluette **(&)**, cela nous permettra de sélectionner les menus à l'aide du clavier en utilisant la combinaison Alt+LaLettreSouligné.*

Comme nous le voyons, avec quelques lignes on peut créer un sous-menu rapidement. Sur celui-ci vous pouvez ajouter bien sûr l'action à effectuer à l'aide de la propriété **OnAction**.

V-A-5 - Ajouter une liste déroulante à un menu

Nous allons voir maintenant comment ajouter une liste déroulante à un menu.

Le code suivant permet de créer une liste avec trois choix, et nous récupérons ce choix dans une routine.

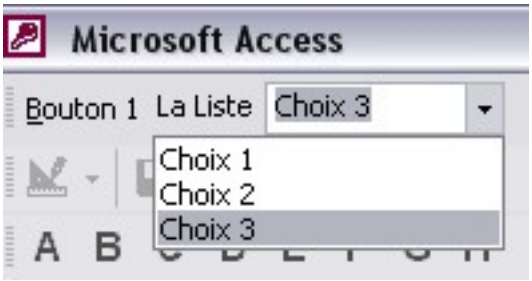
Création de la barre :

```VBA
Public Sub MenuAvecListe()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Ce code va nous permettre de créer une liste déroulante dans un menu
    '================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton
```

```VBA
    Dim btnList As Office.CommandBarComboBox

     On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== affectation =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True, True)

    ' ===== ajout de boutons =====
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "&Bouton 1"
        .Style = msoButtonCaption
    End With

    ' ===== ajout de la liste =====
    Set btnList = cmb.Controls.Add(msoControlComboBox)
    With btnList
        .Caption = "La Liste"
        .Style = msoComboLabel
        .AddItem "Choix 1"
        .AddItem "Choix 2"
        .AddItem "Choix 3"
        .OnAction = "RecupListe"
    End With

    ' ===== rendre la barre visible =====
    cmb.Visible = True
End Sub
```

La routine :

```VBA
Public Sub RecupListe()
    Dim btn As CommandBarComboBox
    Set btn = CommandBars("MaBarre").Controls(2)
    MsgBox btn.List(btn.ListIndex)
End Sub
```



V-A-6 - Ajouter des sous-menus prédéfinis

Il est inutile de créer des menus, alors que ceux-ci existent dans les menus prédéfinis. Nous allons voir comment ajouter ceux-ci à notre barre de menus.

Dans cet exemple, nous allons créer une barre de menus avec un sous-menu Fichier, nous allons donc reprendre le menu Fichier d'Access.

```VBA
Public Sub CreerMnuAvecFichier()
    ' ================================================================================
```

VBA
```
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Ce code va permettre de créer une barre de menu avec le menu Fichier
    '=============================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim cmbFichier As Office.CommandBarPopup

    ' ===== destruction =====
    On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== création =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True)

    ' ajout du menu popup fichier
    Set cmbFichier = Application.CommandBars("Menu Bar").Controls(1)
    ' on copie le menu popup dans la nouvelle barre de menu.
    cmbFichier.Copy cmb


    ' ===== affichage =====
    cmb.Visible = True
End Sub
```

Vous avez dorénavant dans votre menu un menu Fichier qui vous permet d'avoir les mêmes fonctions que la barre de menus d'Access, et sans coder chaque bouton, puisque l'on hérite des propriétés et actions de la barre d'origine.

Cependant, il y a des boutons dont nous n'avons pas besoin, nous allons désactiver les contrôles, en utilisant la propriété **visible**, ce qui nous donne le code suivant :

VBA
```
Public Sub CreerMnuAvecFichier()
    ' =============================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Ce code va permettre de créer une barre de menu avec le menu Fichier
    '=============================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim cmbFichier As Office.CommandBarPopup
    Dim cmbNewFichier As Office.CommandBarPopup

    ' ===== destruction =====
    On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== création =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True)

    ' ajout du menu popup fichier
    Set cmbFichier = Application.CommandBars("Menu Bar").Controls(1)
    ' on copie le menu popup dans la nouvelle barre de menu.
    cmbFichier.Copy cmb

    ' on va enlever des contrôles du menu fichier
    ' on affecte d'abord la barre à une autre variable objet
    Set cmbNewFichier = cmbFichier
    ' on va enlever les contrôles dont on a pas besoin
    With cmbNewFichier
        .Controls(1).Visible = False
        .Controls(2).Visible = False
        .Controls(3).Visible = False
        .Controls(4).Visible = False
        .Controls(5).Visible = False
        .Controls(6).Visible = False
        .Controls(7).Visible = False
```

```VBA
        .Controls(8).Visible = False
    End With

    ' ===== affichage =====
    cmb.Visible = True
End Sub
```

On peut également utiliser la méthode Delete.

L'inconvénient de cela, c'est que le menu Fichier d'Access est aussi touché.
En effet, si vous ouvrez une autre base Access à coté, vous verrez que le menu Fichier est similaire à votre base.
C'est pourquoi, si vous avez à amputer un menu prédéfini, il faut mieux ajouter les boutons de commandes un à un, c'est ce que nous verrons dans la suite.

*Pour réinitialiser votre barre, il faut utiliser la méthode **Reset**, ainsi le code suivant va nous permettre de retrouver notre barre dans son état initial.*
Cela est également valable pour les barres d'Access.

```VBA
Application.CommandBars("MaBarre").Controls(1).Reset
```

V-A-7 - Ajouter des boutons prédéfinis

Comme on l'a vu, l'utilisation d'une barre de commandes prédéfinie peut poser des problèmes lorsque vous voulez supprimer des éléments.

Il est dans ce cas préférable de créer un sous-menu et d'y mettre les contrôles de barres de commandes voulus.

Dans cet exemple nous allons créer une barre de menus avec un menu fichier, et les boutons de commande Imprimer et Fermer

D'abord il nous faut repérer l'identifiant de ces boutons de commandes, c'est ce que va nous donner le code suivant :

```VBA
Public Sub RecupId()
    ' ================================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Ce code va nous permettre de récupérer des informations sur les boutons
    '                 Imprimer et quitter du menu fichier
    '================================================================================================

    ' ===== declaration =====
    Dim cmbPop As Office.CommandBarPopup
    Dim btn As Office.CommandBarControl

    Set cmbPop = Application.CommandBars("Menu Bar").Controls(1)
    For Each btn In cmbPop.Controls
        If btn.Type = msoControlButton Then
            Debug.Print btn.Id & "-" & btn.Caption
        End If
    Next btn

End Sub
```

Le résultat est le suivant :

- 2936-&Nouvelle base de données...
- 2937-&Ouvrir...
- 106-&Fermer
- 3-Enre&gistrer
- 748-En&registrer sous...
- 2938-&Exporter...
- 5905-Rec&hercher...
- 3850-Conne&xion...
- 3965-Re&venir
- 1791-&Charger à partir d'une requête...
- 1790-&Enregistrer en tant que requête
- 1792-&Enregistrer en tant que texte
- 3655-Aperçu de &la page Web
- 247-Mise en &page...
- 109-&Aperçu avant impression
- 4-&Imprimer...
- 3840-&Imprimer les relations...
- 4-&Imprimer...
- 2939-Propriétés de la &base
- 3967-Propri&étés de la page
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 831-&Nom de fichier récent
- 5950-Déconne&xion
- 752-&Quitter

Nous aurons donc besoin des boutons 4 et 752.

Nous allons maintenant créer notre menu fichier avec le code suivant

VBA

```
Public Sub CreerMenuFichier()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Ce code va nous permettre de créer un menu fichier avec 2 boutons
    '================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim cmbPop As Office.CommandBarPopup
    Dim btn As Office.CommandBarButton

    ' ===== suppression de la barre existante =====
```

```VBA
    On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== création de la barre =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarTop, True)
    ' création du menu fichier
    Set cmbPop = cmb.Controls.Add(msoControlPopup)
    With cmbPop
        .Caption = "&Fichier"
    End With
    ' ajout des deux boutons
    Set btn = cmbPop.Controls.Add(msoControlButton, 4)
    Set btn = cmbPop.Controls.Add(msoControlButton, 752)

    ' ===== affichage du menu =====
    cmb.Visible = True
End Sub
```

Nous venons donc de créer 2 boutons avec toutes leurs propriétés héritées.

V-A-8 - Comment affecter un menu à une application, un formulaire ou un état

V-A-8-a - Une application

Il existe deux possibilités :

Votre menu a déjà été généré (vous avez lancé la routine dans l'éditeur VB):
Il suffit de la sélectionner dans les options de démarrage (Menu Outils/Démarrage).

Votre menu n'a pas été généré :
Il faut que le code qui génère votre menu soit dans une fonction (Function) et non dans une routine (Sub).

Il suffit de créer une macro que vous nommerez **AutoExec**, et sur la première ligne vous appellerez l'instruction **ExecuterCode**, et vous n'aurez plus qu'à sélectionner la fonction.

V-A-8-b - Les formulaires et les états

Si votre menu a déjà été généré:
Il suffit de le sélectionner dans les propriétés de votre formulaire (ou état) sur la ligne **Barre de Menu**

Si celui-ci n'a pas été généré :
Il suffit de lancer la routine qui crée le menu sur l'évènement **Load** du formulaire ou **Open** de l'état.

V-A-9 - Conclusion sur les menus

La création d'un menu n'est pas sorcier, il faut penser à rester homogène par rapport aux applications existantes.

- Ne mettez pas d'images dans le menu, uniquement dans les sous-menus.
- N'allez pas trop loin dans la profondeur des sous-menus, pour éviter un labyrinthe à l'utilisateur à la recherche d'un menu.
- Regrouper vos menus par thème.

V-B - Créer une barre d'outils

La création d'une barre d'outils est en tout point similaire à la création d'un menu, c'est pourquoi nous n'allons pas développer ce chapitre.

La différence réside dans le paramètre **MenuBar** de la méthode **Add**. Il suffit de mettre celui-ci à **False**.

De même pour affecter une barre d'outils à un formulaire ou un état, il suffit de mettre le nom de la barre d'outils à la propriété **Barre d'outils** de ceux-ci (dans la mesure ou ceux-ci ont été généré auparavent).

V-C - Créer un menu contextuel

La création d'un menu contextuel n'est pas différente, sauf que celui-ci n'apparaît que lorsque l'on a besoin de lui par un clic droit sur la souris.

V-C-1 - Création d'un menu contextuel simple

Le code suivant va créer un petit menu contextuel.

VBA

```
Public Sub ContextuelSimple()
    ' ================================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Création d'un menu contrextuel simple
    '================================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton

    ' ===== suppression de la barre existante =====
    On Error Resume Next
    Application.CommandBars("Mnu_Context").Delete

    ' ===== génération du menu contextuel ======
    Set cmb = Application.CommandBars.Add("Mnu_Context", msoBarPopup)
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "Le premier"
        .Style = msoButtonCaption
        .OnAction = "=MsgBox('Vous avez cliqué sur le bouton 1')"
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "Le deuxième"
        .Style = msoButtonCaption
        .OnAction = "=MsgBox('Vous avez cliqué sur le bouton 2')"
    End With

    ' ===== faire apparaitre le menu contextuel =====

    cmb.ShowPopup

End Sub
```

Petit commentaire :

C'est dans l'ajout de la barre de commandes que nous déclarons que nous avons affaire à un menu contextuel avec le paramètre de **Position** de la méthode **Add**.

VBA
```
Set cmb = Application.CommandBars.Add("Mnu_Context", msoBarPopup)
```

Nous utilisons la méthode **ShowPup** pour afficher ce menu, et non la propriété **visible**.

VBA
```
cmb.ShowPopup
```

V-C-2 - Ajouter un sous-menu à un menu contextuel

Le code suivant va ajouter un sous-menu au menu contextuel.

VBA
```
Public Sub ContextuelSimpleAvecSousMenu()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Création d'un menu contrextuel simple avec sous menu
    '================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton
    Dim cmbPop As Office.CommandBarPopup

    ' ===== suppression de la barre existante =====
    On Error Resume Next
    Application.CommandBars("Mnu_Context").Delete

    ' ===== génération du menu contextuel ======
    Set cmb = Application.CommandBars.Add("Mnu_Context", msoBarPopup)
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "Le premier"
        .Style = msoButtonCaption
        .OnAction = "=MsgBox('Vous avez cliqué sur le bouton 1')"
    End With
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Caption = "Le deuxième"
        .Style = msoButtonCaption
        .OnAction = "=MsgBox('Vous avez cliqué sur le bouton 2')"
    End With

    ' ===== ajout du sous menu avec deux boutons
    Set cmbPop = cmb.Controls.Add(msoControlPopup)
    cmbPop.Caption = "Sous Menu"
    Set btn = cmbPop.Controls.Add(msoControlButton)
    With btn
        .Caption = "Le bouton 1 du sous-menu"
        .Style = msoButtonCaption
        .OnAction = "=MsgBox('Vous avez cliqué sur le bouton 1 du sous menu')"
    End With
    Set btn = cmbPop.Controls.Add(msoControlButton)
    With btn
        .Caption = "Le bouton 2 du sous-menu"
        .Style = msoButtonCaption
        .OnAction = "=MsgBox('Vous avez cliqué sur le bouton 2 du sous menu')"
    End With


    ' ===== faire apparaitre le menu contextuel =====
```

```VBA
    cmb.ShowPopup

End Sub
```

Comme vous le voyez, l'ajout d'un sous-menu à un menu contextuel est similaire à la création d'un sous-menu des autres barres de commandes.

V-C-3 - Comment affecter un menu contextuel à un formulaire ou à un état

Contrairement aux autres barres de commandes, on utilise un menu contextuel par clic sur le bouton droit de la souris (lorsque celle-ci est en configuration droitier).

V-C-3-a - Le menu contextuel existe déjà.

Il suffit de mettre le nom du menu contextuel dans la propriété **Barre de menu Contextuel** de votre formulaire (ou état). Il faut aussi que la propriété **Menu contextuel** soit à **Oui**.

Ou par le code :

```VBA
Private Sub Form_Load()
    With Me
        .ShortcutMenu = True
        .ShortcutMenuBar = "Mnu_Context"
    End With
End Sub
```

V-C-3-b - Vous créer dynamiquement votre menu contextuel

```VBA
Private Sub Form_Load()
    Call ContextuelSimpleAvecSousMenu
    With Me
        .ShortcutMenu = True
        .ShortcutMenuBar = "Mnu_Context"
    End With
End Sub
```

*Vous pouvez avoir dans un formulaire des menus contextuels différents sur vos contrôles, il suffit de paramétrer la propriété **ShortcutMenuBar** pour chaque contrôle. Mais il est impératif que la propriété **ShortcutMenu** du formulaire soit à **True**.*

VI - BONUS

Dans cette partie je vais vous montrer quelques utilisations possibles des barres de commandes ou des trucs que j'ai trouvé lors de la rédaction de cet article.

VI-A - Créer les lettres d'un répertoire

On peut faire beaucoup de choses avec Access, entre autres, on peut gérer un répertoire de contacts, personnel, etc...

Dans beaucoup d'applications, vous verrez que pour sélectionner une liste de personnes on clique sur la lettre qui représente l'initiale du nom.

Ce que je vous propose c'est de vous montrer comment avec quelques lignes on gère cela.

Nous allons donc créer une barre d'outil avec les lettres de l'alphabet, hors dans la longue série d'images qui composent les icônes des barres d'outils, nous avons ces lettres. Celles-ci sont numérotées de 80 à 105.

Le code suivant va créer une barre de commandes avec toutes les lettres de l'alphabet, et sur l'action du bouton, va lancer une fonction que vous pouvez utiliser en ouvrant un formulaire avec un filtre sur la lettre.

La création de la barre

VBA

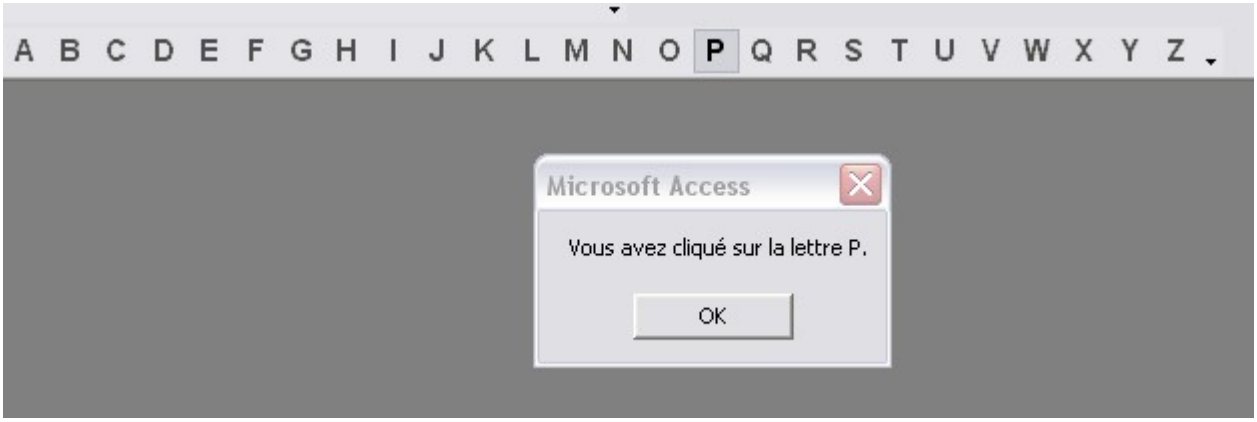
```
Public Sub CreationAgenda()
    ' ============================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Cette routine va permettre de créer une barre d'outil représentant
    '                 les lettres d'un agenda
    '============================================================================================

    ' ===== déclaration =====
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton
    Dim i As Integer
    Dim j As Integer ' pour 80 à 105

    On Error Resume Next
    Application.CommandBars("mnu_Agenda").Delete

    ' ===== réalisation de la barre d'outil =====
    Set cmb = Application.CommandBars.Add("mnu_Agenda", msoBarTop)
    j = 80
    For i = 65 To 90
        Set btn = cmb.Controls.Add(msoControlButton)
        With btn
            .Caption = Chr(i)
            .FaceId = j
            .OnAction = "=FiltreFormulaire(" & i & ")"
        End With
        j = j + 1
    Next i

    ' ===== affichage de la barre =====
    cmb.Visible = True

End Sub
```

La fonction

VBA

```
Public Function FiltreFormulaire(i As Long)
   MsgBox "Vous avez cliqué sur la lettre " & Chr(i) & "."
End Function
```

Le résultat



VI-B - Affecter les icônes des barres de commandes à un bouton de commande

Il est possible de récupérer les icônes des boutons des barres de commandes pour les mettre sur des boutons de commande et de profiter des fonctions de ceux-ci.

Cependant, nous n'allons pas prendre les boutons de commandes standard d'Access, mais le bouton de commande **MsForms 2.0**.

Le code suivant va permettre d'affecter l'icône Ouvrir à un bouton de commande **MSForms 2.0 CommandButton**.

VBA
```
Dim picImage As IPictureDisp
Set picImage = Application.CommandBars("DATABASE").Controls(2).Picture
Me.cmdOuvrir.Picture = picImage
```

- On déclare d'abord une variable IPictureDisp qui servira à stocker l'image.
- On affecte à cette variable l'image du deuxième bouton de la barre de commandes DATABASE.
- On affecte ensuite le résultat au bouton de commande.

Le bouton de commande MSForms 2.0 se trouve dans la liste des ActiveX et est fourni avec votre version d'Access.
Il n'est malheureusement pas possible de redimensionner l'image.

VI-C - Utiliser les fonctions des barres de commandes

Il est possible d'utiliser les fonctions des barres de commandes, c'est ce que nous allons voir dans cet exemple.

Nous allons en quelques lignes créer un correcteur d'orthographe.

Dans la base en téléchargement vous trouverez un formulaire qui se nomme **frm_ControleOrthographe**.

Construisons le décor

- Un bouton de commande **MS Forms 2.0 CommandButton** que nous nommerons **cmdOrthographe**.
- Une zone de texte que nous nommerons **txtSaisie** et dont la propriété **Effet Touche Entrée** est à **Nouvelle lgn dans chp** (ce qui nous permettra de saisir du texte).

Nous allons affecter à ce bouton l'icône de la correction orthographique.

VBA
```
Dim picImage As IPictureDisp
    Set picImage = Application.CommandBars("DATABASE").Controls(7).Picture
    With Me.cmdOrthographe
        .Picture = picImage
        .BackColor = RGB(240, 210, 255)
    End With
```

Sur l'évènement Clic de celui-ci, nous allons lancer l'exécution de l'action du bouton de la barre de commandes.

VBA
```
Private Sub cmdOrthographe_Click()
    ' on met le focus sur la zone de texte
    Me.txtSaisie.SetFocus
    ' on lance la correction
    Application.CommandBars("DATABASE").Controls(7).Execute
End Sub
```

Testez le formulaire, et vous avez une zone de texte avec une correction orthographique.

Malheureusement on ne peut pas faire ce que l'on veut, car il faut que le bouton soit dans son contexte pour pouvoir utiliser son action. Par exemple vous ne pouvez pas lancer l'action d'un bouton qui appartient à la barre de commandes d'un formulaire en mode création, car il n'aura aucun effet.

VI-D - Récupérer l'icône d'un bouton de barre de commandes dans un fichier

Il est possible de récupérer l'image d'un bouton dans un fichier Gif.

Le code suivant permet de récupérer l'image du correcteur d'orthographe.

VBA
```
Public Sub RecupIcone()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Cette procédure va vous permettre de récupérer l'icône d'un bouton
    '================================================================================

    ' ===== déclaration ====
    Dim picImage As IPictureDisp


    ' ===== récupération de l'icône =====
    Set picImage = Application.CommandBars("DATABASE").Controls(7).Picture
    SavePicture picImage, "C:\Documents and Settings\JOCHMANS PHILIPPE\Bureau\test.gif"

End Sub
```

Les images récupérées sont au format 16 x 16.

VI-E - Ajouter une autre image à un bouton de barre de commandes

Il est aussi possible d'ajouter une image de votre cru à un bouton de commande.

Le code suivant vous permet donc d'ajouter une image (146.gif) à un bouton de commande d'une barre d'outils.

VBA

```
Public Sub AffecterIcone()
    ' ================================================================================
    ' Auteur        : Starec - Philippe JOCHMANS - http://starec.developpez.com
    ' Description   : Cette procédure va affecter une icône à un bouton de commande
    '================================================================================

    ' ===== déclaration =====
    Dim picImage As IPictureDisp
    Dim cmb As Office.CommandBar
    Dim btn As Office.CommandBarButton

     ' ===== suppression de la barre existante =====
    On Error Resume Next
    Application.CommandBars("MaBarre").Delete

    ' ===== récupération de l'image =====
    Set picImage = LoadPicture("C:\Documents and Settings\JOCHMANS PHILIPPE\Bureau\146.gif")

    ' ===== création de la barre =====
    Set cmb = Application.CommandBars.Add("MaBarre", msoBarFloating)
    Set btn = cmb.Controls.Add(msoControlButton)
    With btn
        .Picture = picImage
    End With

    ' ===== affichage =====
    cmb.Visible = True
End Sub
```

VII - TELECHARGEMENT

Vous trouverez ci-dessous une base Access au format 2000.

Base Access au format 2000

Je n'ai pas utilisé de LateBinding pour réaliser cette base, il se peut que vous ayiez à référencer manuellement la bibliothèque Office.

VIII - REMERCIEMENTS

Remerciements à **Jeannot45** et **Lou Pitchoun** pour leur correction orthographique et remarques avisées.

Je remercie également tous les membres de la rédaction, du forum Office et tous les contributeurs de ce merveilleux site.