Programmation en VBA de menus personnalisés pour Excel

Par Laurent Ott

Date de publication : 15 novembre 2020

Dernière mise à jour : 16 novembre 2020

Dans cet article vous allez apprendre comment programmer en VBA des menus personnalisés pour lancer vos procédures, et aussi comment compléter ou remplacer le menu contextuel des cellules.

La création de ces menus ne nécessite qu'un minimum de programmation et de paramétrage, et seules quatre fonctions sont nécessaires pour : créer les menus, les supprimer, connaitre le choix de l'utilisateur, et au besoin modifier l'apparence des menus.

Vous pouvez déposer vos commentaires dans cette discussion.

En complément sur Developpez.com

- Tome 1 - Des bases de la programmation à l'algorithme de classement rapide QuickRanking
- Tome 2 - Des bases de la programmation en mode graphique à la programmation d'un jeu d'arcade en VBA et Microsoft Excel
- Tome 3 - Problème du Voyageur de commerce - Une méthode d'approximation basée sur des principes simples
- Tome 4 - Un algorithme de chiffrement/déchiffrement des cellules pour une meilleure confidentialité
- Tome 5 - Sentinelle - Une application qui veille sur vos classeurs sensibles - Exemples d'utilisations des tableaux de données et des requêtes SQL en VBA
- Tome 6 - Apprendre la programmation de nouvelles fonctions Excel pour l'utilisateur
- Comprendre la méthode de factorisation du Crible Quadratique
- Programmation en VBA de menus personnalisés pour Excel
- Manipuler les données Access depuis Excel
- Transférer des fichiers volumineux avec Outlook

I - Introduction

Depuis Excel 2007 notre tableur préféré arbore un ruban très ergonomique en remplacement du menu original. Ruban qui peut être personnalisé via un fichier de configuration au format XLM par une grande variété de contrôles, capables de nous éviter le recours aux formulaires (UserForm), pour une esthétique digne des applications professionnelles.
Une documentation explique très bien tout cela : **https://silkyroad.developpez.com/excel/ruban/**
Vous devinez que ce n'est pas le sujet ici…

Ici l'idée est d'utiliser l'ancienne interface d'Excel, restée compatible, pour créer des menus personnalisés par programmation en VBA.
Vous ne trouverez donc rien de révolutionnaire dans la technologie utilisée dans cette documentation, technologie déjà bien détaillée dans ce tutoriel : **https://fring.developpez.com/vba/excel/barremenu/**

Alors pourquoi cette nouvelle documentation sur une ancienne technologie, sachant que ces menus n'ont pas la prétention de concurrencer la richesse des rubans, ou d'être plus simple à utiliser ?

Pour deux raisons :

- parce que nous vous proposons des fonctions pour générer des menus personnalisés qui ne nécessitent qu'un minimum de programmation pour les développeurs : seules quatre fonctions sont nécessaires pour : créer vos menus, qui peuvent être paramétrés soit via des tableaux dans une feuille de calculs soit directement dans le code ; les supprimer ; connaitre le choix de l'utilisateur ; et au besoin modifier leur apparence.
  Avec leurs avantages et leurs inconvénients. Nous vous laissons juger de la pertinence de leur utilisation dans vos applications ;
- parce que ces menus offrent aussi d'autres possibilités : vous découvrirez ici comment créer un menu pop-up, ou ajouter des contrôles au menu contextuel des cellules (le menu qui s'ouvre d'un clic droit dans une cellule) ce qui peut être bien pratique et venir en complément à un ruban personnalisé par fichier XLM.

Ces menus seront destinés à de « petites » applications dont les besoins peuvent être couverts par les cinq contrôles disponibles : boutons, menus déroulants, saisies de texte, listes de choix, listes modifiables.

Cette documentation s'adressant à des développeurs confirmés, nous n'aborderons pas les bases de la programmation en VBA ou la manipulation d'Excel. Mais nous n'oublions pas les moins expérimentés qui trouveront en **Annexe 2 : Des liens utiles pour en savoir plus sur…** les renseignements nécessaires pour parfaire leurs connaissances en cas de besoin.
Les codes présentés ont été testés avec Excel 2010 et Excel 2016 (versions 32 bits).

La configuration linguistique utilisée est « msoLanguageIDFrench » (1036) retournée par l'instruction « Application.LanguageSettings.LanguageID(msoLanguageIDUI) » : certaines parties de cette documentation devront éventuellement être ajustées à votre configuration si elle est différente.

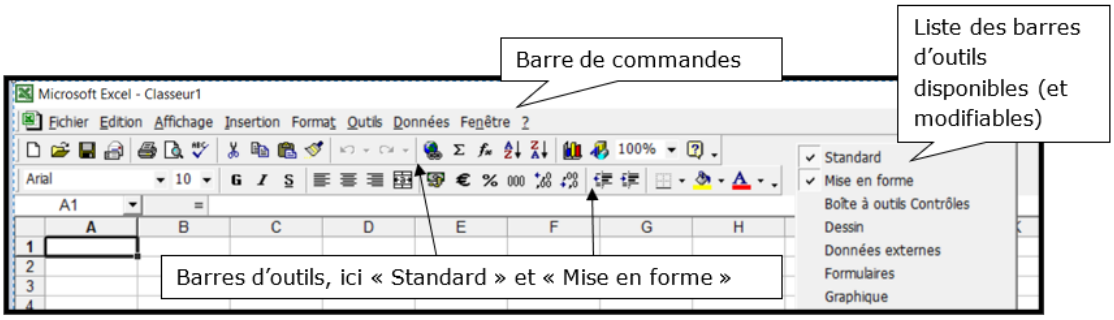
En résumé :

- pour créer nos menus personnalisés, nous disposons des cinq types de contrôles (cf. **Les cinq contrôles que nous utiliserons**), qu'il est possible de présenter dans quatre modes de restitution différents (cf. **Les quatre modes de restitution possibles**) ;
- nous proposons d'utiliser un tableau structuré dans une feuille de calculs pour paramétrer ces contrôles (cf. **Le tableau de paramétrage**) ce qui offre une grande souplesse d'utilisation, mais un paramétrage par code est aussi possible (cf. **MenuInstaller**) ;
- quatre fonctions nous permettront de gérer ces menus (cf. **Les quatre fonctions nécessaires**).

II - Les cinq contrôles que nous utiliserons

Vous allez constater que les contrôles que nous utiliserons semblent un peu dater par rapport aux autres contrôles du ruban. Un bref historique vous permettra d'en comprendre la raison…
Les plus anciens se souviendront du menu d'Excel dans les versions antérieures à Excel 2007, avec sa barre de commandes et ses barres d'outils, qui permettaient d'ajouter des menus personnalisés, manuellement ou via le VBA.

Par exemple pour Excel 2000 :

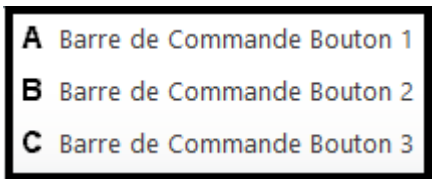


Si le ruban tel que nous le connaissons depuis Excel 2007 a l'avantage d'apporter une plus grande diversité de contrôles (boutons bascules, grandes icônes, galeries, etc.), il a l'inconvénient de ne pas pouvoir créer des menus personnalisés par programmation en VBA (ils se paramètrent dans un fichier au format XLM).
Alors puisque les ingénieurs de Microsoft ont gardé la compatibilité de la programmation en VBA de la barre de commandes et des barres d'outils, nous en profitons, et c'est pourquoi nous utilisons ces concepts.

Principale limitation : les menus ainsi personnalisés se retrouvent regroupés dans l'onglet « Compléments » (pour la version française) du ruban. Cet intitulé diffère suivant la configuration linguistique installée. Il conviendra éventuellement d'adapter le code source en modifiant le contenu de la variable « *MenuTabAddInsName* » située dans l'en-tête du module (par exemple « Add-Ins » pour la version anglaise) car la création du menu provoque la sélection automatique de cet onglet et affiche un message d'erreur s'il n'est pas trouvé (ce message n'est pas bloquant). Nous y reviendrons plus loin.

Les cinq types de contrôles jadis disponibles sont toujours présents. Évidemment, ces contrôles peuvent se cumuler dans un menu personnalisé :

II-A - Barre de Commande Bouton



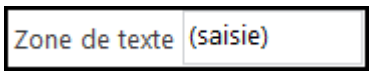
Pour associer un bouton à une procédure (macro).

II-B - Barre de Commande Contextuelle

Qui contient :
>> Bouton de commande
>> Menu Contextuel
>> >> Sous Bouton de commande

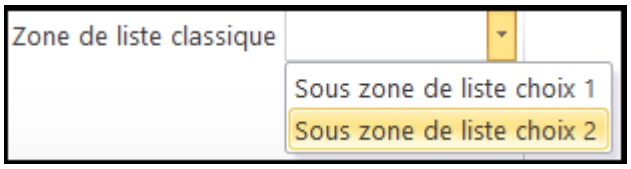
Pour associer un bouton de commande ou un sous bouton de commande à une procédure (macro).
Les contrôles « Barre de Commande Contextuelle » et « Menu Contextuel » ne sont pas liés à une macro, car un clic sur ces contrôles ouvre les sous menus qu'ils contiennent.
Le signe « & » dans le libellé du contrôle permet de souligner le caractère qui le suit et en faire un raccourci clavier.

II-C - Zone de texte



Pour saisir un texte (une procédure liée permettra de le capter).

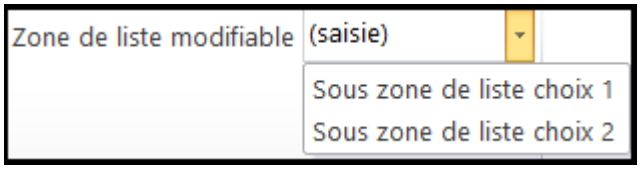
II-D - Zone de liste classique



Qui contient :
>> Sous zone de liste

Pour piocher un choix parmi la liste proposée (une procédure liée à la zone de liste permettra de capter la sélection faite, les « Sous zone de liste » ne sont pas liées à une macro).

II-E - Zone de liste modifiable



Qui contient :
>> Sous zone de liste

Pour piocher un choix parmi la liste proposée ou saisir un texte (une procédure liée à la zone de liste permettra de capter la sélection faite ou la saisie, les « Sous zone de liste » ne sont pas liées à une macro).

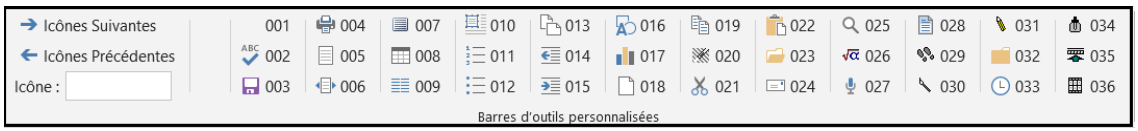
Certes c'est peu par rapport aux multiples possibilités qu'offre le ruban actuel, mais cela permet de couvrir l'essentiel des besoins d'une « petite » application.

III - Les quatre modes de restitution possibles

Quatre modes de restitution sont disponibles.

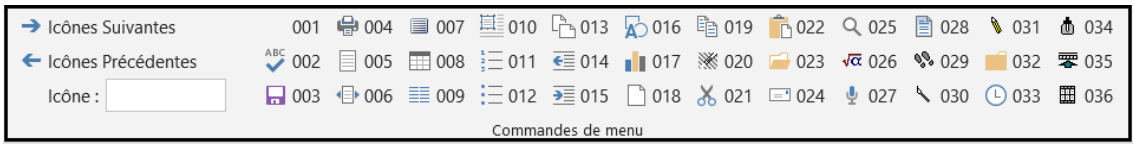
III-A - Barre d'outils (type de menu = BarresOutils)

Pour afficher le menu dans l'onglet « Compléments ». L'info-bulle est précédée du nom du contrôle (dans notre cas ce sera le nom du tableau suivi d'un numéro d'ordre, cette astuce permettant de restituer les contrôles dans leur ordre de création au lieu de l'ordre de leur nom).
Ci-dessous un exemple de restitution que vous retrouverez dans le fichier Excel joint :



III-B - Barre de commandes (type de menu = CommandesMenu)

Pour afficher le menu dans l'onglet « Compléments ». L'info-bulle ne contient que le texte désiré, sans être précédé du nom du contrôle. Les contrôles sont affichés dans leur ordre de création.
Le même exemple présenté dans une barre de commandes :



Ces deux impressions écran représentent la liste des icônes, obtenue par le menu de la feuille « ListeIcônes » du fichier joint, nous en reparlerons plus tard. Les boutons « Icônes Suivantes » et « Icônes Précédentes » permettent de faire défiler cette liste afin de connaitre les numéros de référence des icônes, dont vous aurez besoin dans la configuration de vos menus. Par exemple, l'icône numéro 4 symbolise une imprimante.
La zone de saisie « Icône » permet d'indiquer où doit commencer le défilement.

Remarque sur ces deux modes de restitution

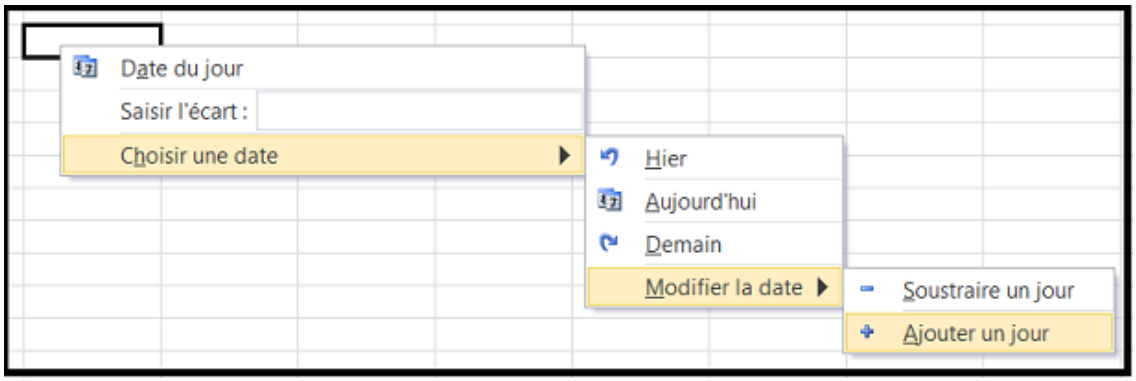
La présentation et l'alignement des contrôles sont légèrement différents comme vous pouvez le constater sur ces impressions d'écran faites depuis Excel 2016, ainsi que l'info-bulle (non représentée ici), mais rien d'autre de particulier à signaler entre ces deux modes de restitution. Le choix entre l'un ou l'autre n'est qu'une question de goût.

Pour que les menus personnels soient visibles, il est impératif que l'onglet « Compléments » (pour la version française d'Excel) soit autorisé (normalement c'est le cas dans la configuration de base). Si ce n'est pas le cas un message d'erreur s'affichera et il faudra intervenir manuellement, dans le menu « Fichier / Option / Personnaliser le ruban / Onglets principaux », cochez l'onglet « Compléments » (ou clic droit sur le ruban puis « Personnaliser le ruban »).

Si l'on ne veut pas que ces menus personnalisés soient présents dans les prochains fichiers qui seront ouverts par Excel, il faut ajouter une procédure pour supprimer le menu en quittant le classeur (sur la feuille ThisWorkbook) :

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
      MenuSupprimer ""
End Sub
```

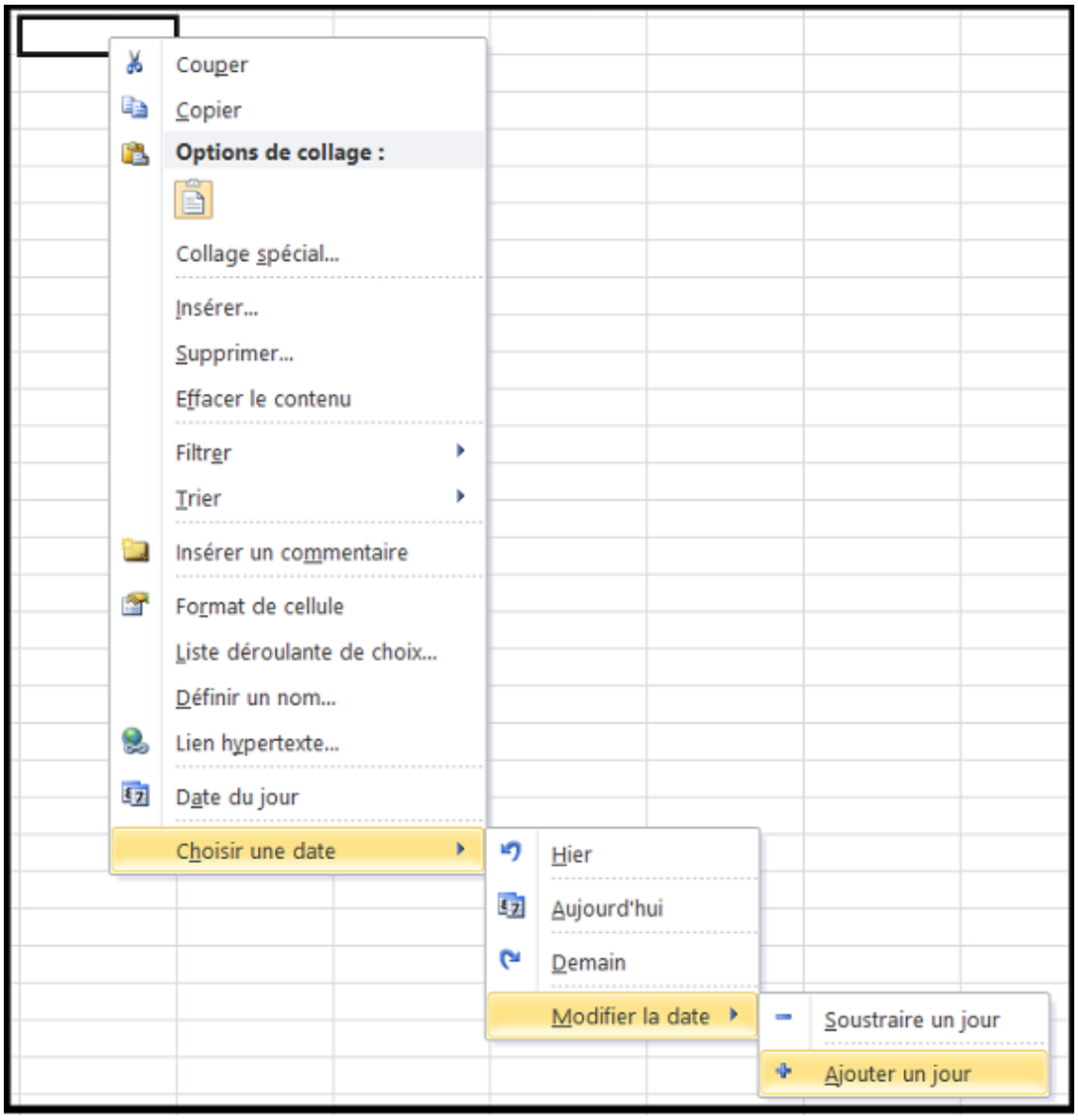
III-C - Barre pop-up (type de menu = BarrePopup)



Pour afficher le menu sous la cellule active. Le menu est déclenché : par un double clic gauche dans la cellule et/ou par un clic droit (et dans ce cas il se susbtitue au menu contextuel classique).

Une programmation dans l'évènement « double clic gauche » et/ou « clic droit » de la feuille concernée, ou du classeur, est nécessaire pour afficher le menu. Nous verrons cela plus loin.

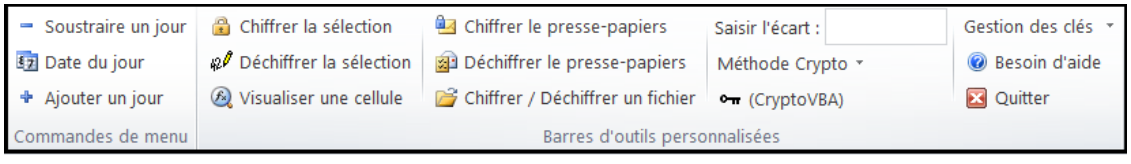
III-D - Menu contextuel (type de menu = MenuContextuel)



Pour afficher le menu sous la cellule active. Le menu est déclenché par un clic droit et vient compléter le menu contextuel classique. Aucune programmation dans l'évènement « clic droit » n'est nécessaire. Attention, les zones de liste et les zones de texte ne sont pas supportées.

Mode de restitution combiné

À noter que ces quatre modes de restitution peuvent se cumuler.
Et plusieurs menus peuvent cohabiter à l'intérieur d'un même mode de restitution.



IV - Le tableau de paramétrage

Les contrôles précités peuvent avoir de nombreuses options, alors pour ne pas tomber dans un paramétrage trop complexe, n'ont été conservées que six notions, et pour vous faciliter la tâche, nous proposons d'utiliser un tableau de données (ou tableau structuré), qui contient donc six colonnes :

- Type : est le type de contrôle à utiliser, comme vu ci-dessus ;
- Nom du contrôle : est le nom du contrôle. Il peut rester vide s'il n'est pas utilisé ;
- Libellé : est le libellé qui s'affiche dans le menu. S'il est vide alors le contrôle ne sera pas activé ;
- Info-bulle : est l'info-bulle affichée. Par défaut le nom du contrôle est repris ;
- Icône : (éventuellement à renseigner si le contrôle admet une icône)
    - soit le numéro de l'icône (comme vu ci-dessus, la feuille « ListeIcônes » du fichier joint permet de lister les icônes utilisées dans Office et ainsi de retrouver facilement leur numéro),
    - soit le chemin et le nom du fichier image d'extension « .jpg » ou « .bmp » à utiliser comme icône. Le chemin n'est pas à renseigner si le fichier est dans le répertoire de l'application.
      Excel ajuste l'image à la taille de l'icône, donc certaines images (comme vos photos de vacances) risquent d'être peu lisibles ;
- Macro : est le nom de la procédure (macro) qui sera exécutée lorsque le contrôle est validé (clic d'un bouton, saisie d'un texte ou sélection dans une liste). Si la procédure est située sur une feuille, vous devez indiquer cette feuille, par exemple pour lier un contrôle à la procédure « *Action_BC1* » codée sur la feuille « *Feui1* », renseignez « *Feuil1.Action_BC1* ». Si la procédure déclarée « *Public* » est dans un module, renseignez simplement son nom : « *Action_BC1* ».

Voici un exemple de tableau utilisé pour générer un menu :

Type	Nom du contrôle	Libellé	Info-bulle	Icône	Macro
Barre de Commande Contextuelle	BCC1	Choisir une date	Choisir une date		
Bouton de commande	BC1	&Hier	Prendre hier comme date	128	Action_Hier
Bouton de commande	BC2	&Aujourd'hui	Prendre la date du jour	125	Action_Aujoudhui
Bouton de commande	BC3	&Demain	Prendre demain comme date	129	Action_Demain
Menu Contextuel	MC1	&Modifier la date	Modifier la date affichée		
Sous Bouton de commande		&Soustraire un jour	Soustraire un jour à la date affichée	138	Action_MoinsUn
Sous Bouton de commande		&Ajouter un jour	Ajouter un jour à la date affichée	137	Action_PlusUn

Pour créer un menu personnalisé, il suffira de passer en argument le nom de ce tableau à la fonction **MenuInstaller** que nous étudierons plus loin.

Vous pouvez tout simplement vous servir du modèle du fichier joint (avec un copier/coller), qui contient la liste des contrôles disponibles et des mises en forme conditionnelles pour simplifier son utilisation en grisant les cellules à ne pas renseigner suivant le contrôle choisi.

Après sélection d'une cellule du tableau, vous trouverez dans l'onglet « Création », groupe « Propriétés », les outils nécessaires pour renommer ou redimensionner un tableau structuré.

Une autre possibilité est d'utiliser la fonction mise à votre disposition, voir **Annexe 4 : Générer automatiquement un modèle de tableau structuré**, qui crée un modèle de tableau à utiliser pour vos menus personnalisés.

Mais il est aussi possible, en code VBA, de passer les éléments comme arguments de la fonction d'initialisation des menus, nous verrons un exemple.

Dans tous les cas, le nom du tableau sera utilisé pour identifier le menu et l'initialiser, mais permettra aussi de le supprimer.
Lorsque tous les menus personnels sont supprimés, l'onglet « Compléments » disparaît automatiquement.

La feuille de calcul qui contient vos menus peut être masquée aux utilisateurs, soit en passant par l'éditeur VBA : sélectionnez la feuille concernée puis ouvrez ses propriétés (menu « Affichage / Fenêtre Propriétés ») et modifiez la propriété « Visible »

en piochant « 2 - xlSheetVeryHidden » ; soit en programmation par l'instruction Sheets("NomDeLaFeuille").visible = xlSheetVeryHidden.

La feuille ne pourra être à nouveau affichée qu'en passant par ces méthodes, en attribuant la valeur « xlSheetVisible » à la propriété « Visible ».

Pour masquer la feuille avec un niveau moindre de sécurité, vous pouvez aussi faire un clic droit sur l'onglet et valider l'option « Masquer ».

V - Les quatre fonctions nécessaires

Quatre fonctions permettent de gérer les menus personnalisés en VBA : pour installer les menus, les supprimer, obtenir des informations sur le contrôle choisi par l'utilisateur, ou modifier l'apparence d'un contrôle.
Elles sont regroupées dans le module VBO du fichier Excel « Menu_Personnalisé.xlsm », voir **Annexe 3 : Téléchargements**.

Nous allons les étudier en détail.

V-A - MenuInstaller

Cette fonction sert à installer un menu personnalisé. Ses trois arguments sont :

- StrNomTableau As String : est le nom du tableau structuré qui contient les paramètres des menus comme vu ci-dessus ;
- TypeMenuPerso As TypeMenuPersos : est l'un des quatre modes de restitution disponibles, à sélectionner dans l'énumération proposée (voir ci-dessous) ;
- ParamArray Arguments() As Variant : facultatif, est un tableau d'éléments qui remplace le tableau structuré si le développeur décide de générer le menu directement dans le code. Chaque argument correspond à une ligne de tableau, où les éléments sont séparés par une virgule.

Exemple d'appel avec ce tableau structuré nommé « X » :

Type	Nom du contrôle	Libellé	Info-bulle	Icone	Macro
Barre de Commande Bouton	B1	Ouvrir	Ouvrir un fichier	23	Choix_Ouvrir
Barre de Commande Bouton	B2	Aide	Aide sur ce programme	984	Choix_Aide
Barre de Commande Bouton	B3	Quitter	Quitter ce programme	923	Choix_Quitter

```
MenuInstaller "X", CommandesMenu
```

Et l'appel équivalent en code (si vous n'utilisez pas un tableau, il faut quand même donner un nom au menu) :

```
MenuInstaller "X", CommandesMenu, _
               "Barre de Commande Bouton,B1,Ouvrir,Ouvrir un fichier,23,Choix_Ouvrir", _
               "Barre de Commande Bouton,B2,Aide,Aide sur ce programme,984,Choix_Aide", _
               "Barre de Commande Bouton,B3,Quitter,Quitter ce programme,923,Choix_Quitter"
```

L'énumération utilisée :

```
Public Enum TypeMenuPersos
    BarresOutils = 0
    CommandesMenu = 1
    BarrePopup = 2
    MenuContextuel = 3
End Enum
```

Pour vous permettre de mieux comprendre le code source de cette fonction, voici un bref rappel (sans rentrer dans les détails, mais juste pour en comprendre les grandes lignes) sur la programmation en VBA pour créer un menu personnalisé, avec deux exemples.
Le premier exemple génère des boutons dans la barre de commandes.

Cela se fait en deux étapes : d'abord créer un pointeur sur la barre de commandes, puis y ajouter les contrôles, ici de simples boutons liés à une macro.

```
' Création d'un objet qui fait référence à la barre de commandes:
Set MyMenuBar = CommandBars.ActiveMenuBar
MyMenuBar.Visible = True

' Ajoute un premier contrôle:
Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlButton, Temporary:=True)
With NvCtrl
    .Caption = "Enregister"                 ' Libellé du bouton.
    .OnAction = "Macro_Enregistrer"         ' Macro à lancer.
    .FaceId = 3                             ' Numéro d'icône.
    .Style = msoButtonIconAndCaption        ' Type d'icône.
End With

' Ajoute un second contrôle:
Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlButton, Temporary:=True)
With NvCtrl
    .Caption = "Imprimer"                   ' Libellé du bouton.
    .OnAction = "Macro_imprimer"            ' Macro à lancer.
    .FaceId = 4                             ' Numéro d'icône.
    .Style = msoButtonIconAndCaption        ' Type d'icône.
End With
```

Ce qui donne dans l'onglet « Compléments » (qui est généré automatiquement s'il n'existait pas) :

Les procédures « Macro_Enregistrer » et « Macro_imprimer » seront exécutées lors d'un clic sur le bouton correspondant, qui pourraient être (dans un module) celles-ci :

```
Sub Macro_Enregistrer()
ThisWorkbook.Save
End Sub

Sub Macro_imprimer()
ThisWorkbook.PrintOut
End Sub
```

Le deuxième exemple montre comment créer un menu pop-up qui va s'ouvrir lors d'un clic droit de la souris.
Cette fois trois étapes sont nécessaires. Les deux premières pour créer le menu, identiques sur le fond à l'exemple précédent : d'abord créer un pointeur sur un menu pop-up personnalisé que nous appelons arbitrairement « MaBarrePopup », puis y ajouter les contrôles, ici aussi de simples boutons liés à une macro.

```
' Création d'un objet qui fait référence à un menu pop-up nommé MaBarrePopup:
Set MyMenuBar = CommandBars.Add("MaBarrePopup", msoBarPopup, False, True)

' Ajoute un premier contrôle:
Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlButton, Temporary:=True)
With NvCtrl
    .Caption = "&Plus un jour"              ' Libellé du bouton.
```

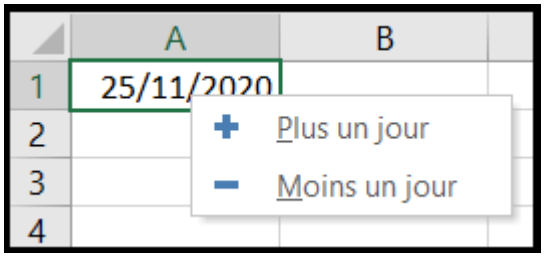
```
    .OnAction = "Macro_AjouterJour"          ' Macro à lancer.
    .FaceId = 137                            ' Numéro d'icône.
    .Style = msoButtonIconAndCaption         ' Type d'icône.
End With

' Ajoute un second contrôle:
Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlButton, Temporary:=True)
With NvCtrl
    .Caption = "&Moins un jour"              ' Libellé du bouton.
    .OnAction = "Macro_soustraireJour"       ' Macro à lancer.
    .FaceId = 138                            ' Numéro d'icône.
    .Style = msoButtonIconAndCaption         ' Type d'icône.
End With
```

Enfin, la troisième étape consiste à créer un évènement sur clic droit dans la feuille concernée (ou sur le classeur) afin que le menu pop-up s'affiche :

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
CommandBars("MaBarrePopup").ShowPopup
Cancel = True
End Sub
```

Ce qui donne :



Il est possible de n'afficher ce menu que sur une cellule qui est au format date, c'est un exemple parmi tant d'autres :

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
If IsDate(Target) Then CommandBars("MaBarrePopup").ShowPopup
Cancel = True
End Sub
```

Vous l'avez surement deviné, le principe de la fonction « MenuInstaller » est d'alimenter les menus personnalisés par les informations contenues dans un tableau en bouclant sur ses lignes (ou dans les arguments si le développeur décide de générer le menu directement dans le code), afin d'en simplifier la création.

Le code source commenté de cette fonction qui gère les quatre types de restitution et les cinq différents types de contrôles :

```
'-----------------------------------------------------------------------------------------------
Public Sub MenuInstaller(StrNomTableau As String, _
                         TypeMenuPerso As TypeMenuPersos, _
                         ParamArray Arguments() As Variant)
'-----------------------------------------------------------------------------------------------
Dim MyMenuBar, NewMenu, NewPopup, NvCtrl
Dim TypeStyle As MsoControlType
Dim StrNomFeuille As String, NomControle As String, Image As String
Dim TD, i As Long, NbLignes As Integer

' Charge en mémoire le tableau qui contient les données pour le menu personnel:
On Error Resume Next
StrNomFeuille = Range(StrNomTableau).Worksheet.Name
Set TD = ThisWorkbook.Sheets(StrNomFeuille).Range(StrNomTableau)
NbLignes = TD.ListObject.ListRows.Count
```

```
' Si TD n'est pas un nom du tableau de données, alors récupère les infos dans ParamArray
 Arguments():
If TD Is Nothing = True Then
    ReDim TD(1 To UBound(Arguments) + 1, 1 To 6)
    Dim Tps
    ' Boucle sur le nombre d'arguments:
    For NbLignes = 0 To UBound(TD) - 1
        Tps = Split(Arguments(NbLignes), ",") ' Décompose les 6 éléments séparés par une virgule.
        For i = 0 To 5
            TD(NbLignes + 1, i + 1) = Tps(i) ' Alimente TD() des 6 éléments en base 1.
        Next i
    Next NbLignes
End If

' Efface la barre Pop-up s'il faut en installer une autre,
' ou restaure le menu contextuel s'il faut en installer un autre,
' ou efface les contrôles du tableau s'il existe déjà pour le réinstaller:
Select Case TypeMenuPerso
    Case BarrePopup: CommandBars("MaBarrePopup").Delete
    Case MenuContextuel: CommandBars("cell").Reset
    Case Else: MenuSupprimer "StrNomTableau"
End Select

' Suivant le type de menu personnel, définition de l'objet CommandBar qui sera
' utilisé pour tous les contrôles:
Select Case TypeMenuPerso
    Case CommandesMenu
        Set MyMenuBar = CommandBars.ActiveMenuBar
    Case BarrePopup
        Set MyMenuBar = CommandBars.Add("MaBarrePopup", msoBarPopup, False, True)
    Case MenuContextuel
        Set MyMenuBar = CommandBars("cell")
End Select

' Boucle sur les lignes du tableau:
For i = 1 To NbLignes

    ' Cas des barres d'outils = Création d'un objet CommandBar pour chaque contrôle:
    If TypeMenuPerso = BarresOutils Then
        NomControle = StrNomTableau & "_" & Format(i, "000")
        CommandBars.Add "NomControle, msoBarTop, False, True"
        Set MyMenuBar = CommandBars(NomControle)
        MyMenuBar.Visible = True
    End If

    ' Définit le contrôle NvCtrl suivant le type de contrôle du menu:
    Select Case TD(i, 1)

        Case "Barre de Commande Bouton"
            Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlButton, Temporary:=True)
            TypeStyle = msoButtonIconAndCaption

        Case "Barre de Commande Contextuelle"
            Set NewMenu = MyMenuBar.Controls.Add(Type:=msoControlPopup, Temporary:=True)
            Set NvCtrl = NewMenu
            TypeStyle = msoButtonIconAndCaption

        Case "Bouton de commande"
            Set NvCtrl = NewMenu.CommandBar.Controls.Add(Type:=msoControlButton, Temporary:=True)
            TypeStyle = msoButtonIconAndCaption

        Case "Menu Contextuel"
            Set NewPopup = NewMenu.CommandBar.Controls.Add(Type:=msoControlPopup,
 Temporary:=True)
            Set NvCtrl = NewPopup
            TypeStyle = msoButtonIconAndCaption

        Case "Sous Bouton de commande"
            Set NvCtrl = NewPopup.CommandBar.Controls.Add(Type:=msoControlButton,
 Temporary:=True)
            TypeStyle = msoButtonIconAndCaption
```

```
        Case "Zone de texte"
            Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlEdit, Temporary:=True)
            TypeStyle = msoComboLabel

        Case "Zone de liste Classique"
            Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlDropdown, Temporary:=True)
            TypeStyle = msoComboLabel

        Case "Zone de liste Modifiable"
            Set NvCtrl = MyMenuBar.Controls.Add(Type:=msoControlComboBox, Temporary:=True)
            TypeStyle = msoComboLabel

        Case Else
            Exit For

    End Select

    ' Initialise les propriétés de ce nouveau contrôle:
    With NvCtrl
        .DescriptionText = StrNomTableau        ' Nom du tableau.
        .Tag = TD(i, 2)                         ' Nom du contrôle.
        .Caption = TD(i, 3)                     ' Libellé.
        .TooltipText = TD(i, 4)                 ' Info-bulle.
        .FaceId = TD(i, 5)                      ' Icône.
        .OnAction = TD(i, 6)                    ' Macro à lancer.
        .BeginGroup = True                      ' Nouveau Groupe.
        .Style = TypeStyle                      ' Style.
        If .Caption = "" Then .Enabled = False  ' Si pas de libellé alors désactive le contrôle.
    End With

    ' L'icône peut aussi être une image à récupérer:
    If TD(i, 5) <> "" And StrNomFeuille <> "" Then
        Image = ""
        If Dir(CStr(TD(i, 5))) <> "" Then
            Image = TD(i, 5) ' <-- si le fichier est trouvé avec son chemin.
        ElseIf Dir(ThisWorkbook.Path & "\" & CStr(TD(i, 5))) <> "" Then
            Image = ThisWorkbook.Path & "\" & TD(i, 5) ' <-- si fichier dans le répertoire actif.
        End If
        If Image <> "" Then ' <-- si un fichier image a été trouvé.
            Application.ScreenUpdating = False '<-- ne pas actualiser l'écran.
            With Sheets(StrNomFeuille).Pictures.Insert(Image) '<-- insertion de l'image.
                .Name = "Pic1" '<-- attribution d'un nom à l'image.
                .Copy '<-- copie de l'image dans le presse-papiers.
            End With
            NvCtrl.PasteFace '<-- collage de l'image issue du presse-papiers.
            Sheets(StrNomFeuille).Pictures("Pic1").Delete '<-- suppression de l'image insérée.
        End If
    End If

    ' Initialise les "Sous Zone de liste" de ce nouveau contrôle (s'il y en a):
    While TD(i + 1, 1) = "Sous Zone de liste"
        If i + 1 > NbLignes Then Exit For
        NvCtrl.AddItem TD(i + 1, 3)
        i = i + 1
    Wend

Next i

Err.Clear
Application.ScreenUpdating = True

' S'il faut sélectionner l'onglet "Compléments" et développer le ruban (un traitement différé
' d'une seconde est nécessaire pour laisser le temps à Excel d'installer le menu:
If TypeMenuPerso = BarresOutils Or TypeMenuPerso = CommandesMenu Then
    Application.OnTime Now + TimeValue("00:00:01"), "SetRibbonTabFocusAddIns"
    Application.OnTime Now + TimeValue("00:00:01"), "MenuDeveloppeurRuban"
End If

End Sub
```

N'oubliez pas que si vous installez une barre pop-up (TypeMenuPerso = BarrePopup), pour pouvoir provoquer l'affichage du menu par un double clic gauche dans une cellule (BeforeDoubleClick) et/ou par un clic droit (BeforeRightClick), une programmation dans l'évènement de la feuille concernée, ou du classeur, est nécessaire.

Par exemple :

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range,
 Cancel As Boolean)
CommandBars("MaBarrePopup").ShowPopup
Cancel = True
End Sub
```

Vous trouverez des exemples de menus personnalisés dans la feuille « Menu » du fichier joint.
Ils vous permettront de bien comprendre la logique du paramétrage et l'utilité des tableaux structurés.

V-B - MenuSupprimer

Cette fonction sert à supprimer un contrôle ou un menu personnalisé. Son argument est :

- StrNomTableauOuControle As **String** : est le nom du tableau structuré ou du contrôle à supprimer.
  Si cet argument est vide alors tous les menus sont supprimés (y compris la barre pop-up et le menu contextuel, y compris aussi les menus personnels créés par d'autres applications).
  Si cet argument vaut « MaBarrePopup » alors ne supprime que la barre pop-up.
  Si cet argument vaut « MenuContextuel » alors restaure le menu contextuel ordinaire.

Lorsque tous les menus personnels sont supprimés, l'onglet « Compléments » disparaît automatiquement.

Exemple d'appel pour supprimer le bouton « B3 » de l'exemple précédent :

```
MenuSupprimer "B3"
```

Exemple d'appel pour supprimer la barre pop-up et restaurer le menu contextuel :

```
MenuSupprimer "MaBarrePopup"
MenuSupprimer "MenuContextuel"
```

Exemple d'appel pour supprimer tous les menus, y compris la barre pop-up, et restaurer le menu contextuel :

```
MenuSupprimer ""
```

Pour vous permettre de mieux comprendre le code source de cette fonction, reprenons les deux exemples cités plus haut lors de la création d'une barre de commandes et d'un menu pop-up.

Pour supprimer la barre de commandes, désignée par le VBA sous le nom « Worksheet Menu Bar », il suffit de boucler sur ses contrôles et de les supprimer.

```
For Each Ctrl In CommandBars("Worksheet Menu Bar").Controls
    Ctrl.Delete
Next
```

C'est plus simple pour un menu pop-up, car une instruction supprime le menu désiré (« MaBarrePopup » dans notre cas) :

```
CommandBars("MaBarrePopup").Delete
```

Dans vos applications, si vous n'utilisez pas les fonctions présentées ici, pensez à supprimer un menu avant de le modifier.

Le code source commenté de la fonction qui gère les quatre types de restitution et les cinq différents types de contrôles :

```
'-----------------------------------------------------------------------------------------
Public Sub MenuSupprimer(StrNomTableauOuControle As String)
'-----------------------------------------------------------------------------------------
Dim Cbar As CommandBar, Ctrl
Dim i As Integer

On Error Resume Next

' Boucle sur tous les menus:
For Each Cbar In Application.CommandBars
    If Cbar.Visible = True Then
 ' Boucle sur tous les contrôles:
        For Each Ctrl In Cbar.Controls
            ' Si le nom du tableau ou le nom du contrôle correspond alors le supprime:
            If Ctrl.DescriptionText = StrNomTableauOuControle Or Ctrl.Tag =
 StrNomTableauOuControle Or StrNomTableauOuControle = "" Then
                Ctrl.Delete
            End If
            ' Boucle sur tous les sous-contrôles et supprime si le nom correspond:
            For i = 1 To Ctrl.Controls.Count
                If Ctrl.Controls(i).Tag = StrNomTableauOuControle Then Ctrl.Controls(i).Delete
            Next i
        Next Ctrl
    End If
Next Cbar

Select Case StrNomTableauOuControle

    ' S'il faut supprimer la barre pop-up et le menu contextuel:
    Case ""
        CommandBars("MaBarrePopup").Delete
        CommandBars("cell").Reset

    ' S'il faut supprimer la barre pop-up:
    Case "BarrePopup"
        CommandBars("MaBarrePopup").Delete

    ' S'il faut supprimer le menu contextuel:
    Case "MenuContextuel"
        CommandBars("cell").Reset

End Select
Err.Clear
End Sub
```

N'oubliez pas de supprimer les menus créés par votre application en quittant le classeur.

Sans quoi, tant que vous n'avez pas quitté Excel, les contrôles restent opérationnels (et le classeur est rechargé quand on les utilise) ce qui pourrait perturber l'utilisateur dans certains cas.

Par exemple, ajoutez ce code dans l'évènement « Workbook_BeforeClose » (qui se produit avant la fermeture du classeur) sur l'objet « ThisWorkbook » pour supprimer les menus personnalisés créés à partir des tableaux structurés nommés X et Y :

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
MenuSupprimer "X"
MenuSupprimer "Y"
End Sub
```

V-C - MenuInfoChoix

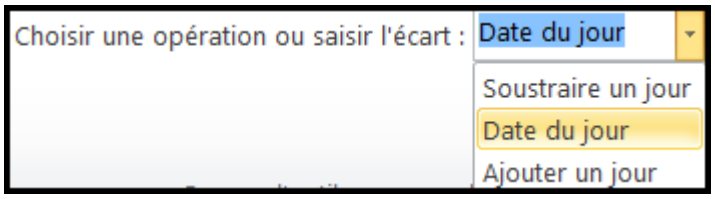
Cette fonction retourne les informations sur un contrôle validé (clic sur un bouton, sélection d'une liste, validation d'une saisie). Ses arguments sont :

- StrNom As String : est le nom du contrôle concerné. Si plusieurs contrôles ont le même nom, c'est le premier trouvé qui est traité.
  si cet argument est vide alors retourne les informations sur le contrôle actif (celui qui vient d'être validé) ;
- TypeInfoChoix As TypeMenuInfoChoix : est l'un des trois types d'informations qu'il faut retourner, à sélectionner dans l'énumération proposée (voir ci-dessous), c'est-à-dire, soit le texte saisi, soit l'index dans une liste (le premier élément est noté 1), soit le libellé du contrôle.

Exemple pour ce tableau qui affiche une zone de liste modifiable, avec le contrôle lié à la macro « Action_ZoneModif » :

Type	Nom du contrôle	Libellé	Info-bulle	Icone	Macro
Zone de liste Modifiable	ZLM1	Choisir une opération ou saisir l'écart :	choisir une opération dans la liste ou saisissez l'écart		Action_ZoneModif
Sous Zone de liste		Soustraire un jour	Soustraire un jour à la date affichée		
Sous Zone de liste		Date du jour	Prendre la date du jour		
Sous Zone de liste		Ajouter un jour	Ajouter un jour à la date affichée		

L'utilisateur a sélectionné le deuxième index de la liste :



Exemple d'appel pour retourner le numéro de l'index et la valeur de la zone de texte :

```
Sub Action_ZoneModif()
Dim Val_Index As Variant
Dim Val_Texte As Variant
Val_Index = MenuInfoChoix("", InfoIndex)
Val_Texte = MenuInfoChoix("", InfoTexte)
End Sub
```

Val_Index vaut « 2 » et,
Val_Texte vaut « Date du jour ».

Et si l'utilisateur avait saisi « -5 » dans la zone de texte au lieu de piocher dans la liste déroulante :
Val_Index vaudrait « 0 » et,
Val_Texte vaudrait « -5 ».

Le type « InfoLibellé » retourne le libellé du contrôle, donc ici « Choisir une opération ou saisir l'écart : ».

L'énumération utilisée :

```
Public Enum TypeMenuInfoChoix
    InfoTexte = 0
    InfoIndex = 1
    InfoLibellé = 2
End Enum
```

Le code source commenté :

```
'-------------------------------------------------------------------------------------
Public Function MenuInfoChoix(StrNom As String, TypeInfoChoix As TypeMenuInfoChoix) As Variant
'-------------------------------------------------------------------------------------
Dim Ctrl As CommandBarControls
On Error Resume Next

Select Case StrNom

    ' Retourne la valeur du contrôle qui vient d'être activé (cas classique):
    Case ""
        If TypeInfoChoix = InfoTexte Then MenuInfoChoix = CommandBars.ActionControl.Text
        If TypeInfoChoix = InfoIndex Then MenuInfoChoix = CommandBars.ActionControl.ListIndex
        If TypeInfoChoix = InfoLibellé Then MenuInfoChoix = CommandBars.ActionControl.Caption

    ' Recherche la valeur du contrôle passé en argument:
    Case Else
        Set Ctrl = CommandBars.FindControls(Tag:=StrNom)
        If Not Ctrl Is Nothing Then
            If TypeInfoChoix = InfoTexte Then MenuInfoChoix = Ctrl(1).Text
            If TypeInfoChoix = InfoIndex Then MenuInfoChoix = Ctrl(1).ListIndex
            If TypeInfoChoix = InfoLibellé Then MenuInfoChoix = Ctrl(1).Caption
        End If

End Select

Err.Clear
End Function
```

Vous ferez appel à cette fonction pour connaître la saisie réalisée ou la sélection faite dans une liste, soit les contrôles « Zone de texte », « Zone de liste Classique » et « Zone de liste Modifiable ».
Un bouton sera généralement lié directement à une procédure, même si cette fonction peut être utilisée, avec TypeInfoChoix = "InfoLibellé" qui retourne le libellé du bouton.

V-D - MenuModifControle

Cette fonction permet de modifier l'icône, le libellé ou l'info-bulle d'un contrôle, ou de modifier son apparence (visible ou masqué). Ses arguments sont :

- StrNom As String : est le nom du contrôle concerné. Si plusieurs contrôles ont le même nom, c'est le premier trouvé qui est traité ;
- NumFaceId As Long : si supérieur à zéro alors remplace l'icône du contrôle par ce numéro ;
- StrNouveauLibellé As String : si non vide alors remplace le libellé du contrôle.
- StrNouveauInfoBulle As String : si non vide alors remplace l'info-bulle du contrôle ;
- Optional bVisible As Boolean : si sa valeur vaut True (valeur par défaut) alors affiche le contrôle, si la valeur vaut False alors masque le contrôle.

Le code source commenté :

```
'-------------------------------------------------------------------------------------
Public Function MenuModifControle(StrNom As String, NumFaceId As Long, _
                                  StrNouveauLibellé As String, _
                                  StrNouveauInfoBulle As String, _
                                  Optional bVisible As Boolean = True)
'-------------------------------------------------------------------------------------
Dim Ctrl As CommandBarControls
On Error Resume Next

' Recherche le contrôle d'après le nom passé en argument:
Set Ctrl = CommandBars.FindControls(Tag:=StrNom)

' Si le contrôle est trouvé:
If Not Ctrl Is Nothing Then
```

```
    If NumFaceId > 0 Then Ctrl(1).FaceId = NumFaceId ' Modifie l'icône.
    If StrNouveauLibellé <> "" Then Ctrl(1).Caption = StrNouveauLibellé ' Modifie le libellé.
    If StrNouveauInfoBulle <> "" Then Ctrl(1).TooltipText = StrNouveauInfoBulle ' Modifie l'info-
bulle
    Ctrl(1).Visible = bVisible ' Affiche ou masque.
End If

Err.Clear
End Function
```

VI - Autres fonctions

Le module VBO du fichier joint contient aussi trois fonctions « Private », que vous n'aurez donc pas à appeler dans votre programmation. Elles sont mentionnées juste pour information.

Elles permettent :

- de développer le ruban quand un menu personnel est installé : il n'existe qu'une seule instruction pour modifier la taille du ruban : `CommandBars.ExecuteMso "MinimizeRibbon"` (réduit ou développe le ruban, malgré ce que le nom pourrait laisser supposer). Pour être certain que l'instruction va bien développer le ruban et non pas le réduire, il faut au préalable interroger `CommandBars.GetPressedMso("MinimizeRibbon")` qui retourne Vrai si le ruban est réduit. Dans ce cas, un appel à `CommandBars.ExecuteMso "MinimizeRibbon"` le développe ;
- de sélectionner l'onglet « Compléments », par les fonctions `SetRibbonTabFocusAddIns` et `FindChildByRoleOrName` (qui sont de **Arkam46**) et de vérifier sa présence. En cas d'erreur, un message affiche la procédure à suivre pour autoriser l'affichage de cet onglet dans le ruban (éventuellement ajustez la variable « MenuTabAddInsName » suivant votre configuration linguistique et/ou la variable « MenuErrTabAdd » suivant votre version d'Excel).

```
' Libellé de l'onglet personnel ("Add-Ins" en anglais):
Private Const MenuTabAddInsName = "Compléments"

' Libellé du message d'erreur (voir la fonction MenuDevelopperRuban):
Private Const MenuErrTabAdd = "Actions possibles : " & vbCrLf & vbCrLf _
         & "- Vérifiez que l'onglet ''Compléments'' est disponible : " _
         & "Menu Fichier / Options / Personnaliser le ruban / Onglets principaux " _
         & "/ Cochez l'onglet ''Compléments''." _
         & vbCrLf & vbCrLf _
         & "- Vérifiez la cohérence des données utilisées pour générer le menu personnel."

'-------------------------------------------------------------------------------------------
Private Sub MenuDevelopperRuban()
'-------------------------------------------------------------------------------------------
' Permet de développer le ruban s'il est réduit.
'-------------------------------------------------------------------------------------------
If CommandBars.GetPressedMso("MinimizeRibbon") = True Then
 CommandBars.ExecuteMso "MinimizeRibbon"

' Vérifie si le ruban "Compléments" a bien le focus, sinon il y a une erreur d'installation:
If SetRibbonTabFocusAddIns = False Then
    MsgBox MenuErrTabAdd, vbCritical + vbOKOnly, "Erreur lors de l'installation du menu
 personnel"
End If

End Sub

'-------------------------------------------------------------------------------------------
Private Function SetRibbonTabFocusAddIns() As Boolean
'-------------------------------------------------------------------------------------------
' Sources:
' https://www.developpez.net/forums/d697439/logiciels/microsoft-office/general-vba/contribuez/
introduction-aux-fonctions-d-accessibilite/
'-------------------------------------------------------------------------------------------
Dim oChild As Variant
Dim oRibbon As IAccessible
```

```
Dim oTab As IAccessible
Const ROLE_SYSTEM_CLIENT = &HA&
Const ROLE_SYSTEM_WINDOW = &H9&
Const ROLE_SYSTEM_PAGETAB = &H25&
Const ROLE_SYSTEM_PROPERTYPAGE = &H26&
Const ROLE_SYSTEM_PAGETABLIST = &H3C&
On Error GoTo gestion_erreurs
' Ribbon Tool Bar
Set oRibbon = CommandBars("ribbon")
' Ribbon Window
Set oRibbon = oRibbon.accChild(ByVal 1&)
' Ribbon Client
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_CLIENT)
' Ribbon Client Window
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_WINDOW)
' Ribbon Client Window Client
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_CLIENT)
' Ribbon Client Window Client Window
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_WINDOW)
' Ribbon Property page
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_PROPERTYPAGE)
' Ribbon Tabs list
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_PAGETABLIST)
' Ribbon Tabs list Client
Set oRibbon = FindChildByRoleOrName(oRibbon, , ROLE_SYSTEM_CLIENT)
' Tab:
Set oTab = FindChildByRoleOrName(oRibbon, MenuTabAddInsName, ROLE_SYSTEM_PAGETAB)
' Click Tab
Call oTab.accDoDefaultAction(ByVal 0&)
' True if OK
SetRibbonTabFocusAddIns = True
Exit Function
gestion_erreurs:
SetRibbonTabFocusAddIns = False
Err.Clear
End Function

'-----------------------------------------------------------------------------------------
' Fonction privée pour rechercher un objet accessible à partir de son parent, son rôle et son nom
Private Function FindChildByRoleOrName(pParent As IAccessible, Optional
 pChildName As String = "*", Optional pChildRole As String = "*") As IAccessible
'-----------------------------------------------------------------------------------------
Dim lName As String, lRole As Long
Dim oChild As IAccessible
Const NAVDIR_FIRSTCHILD = &H7&
Const NAVDIR_NEXT = &H5&
On Error GoTo gestion_erreurs
Set oChild = pParent.accNavigate(NAVDIR_FIRSTCHILD, ByVal 0&)
If pChildName <> "*" Then lName = oChild.accName(ByVal 0&)
If pChildRole <> "*" Then lRole = oChild.AccRole(ByVal 0&)
If lRole Like pChildRole And lName Like pChildName Then
    Set FindChildByRoleOrName = oChild
    Exit Function
End If
Do
    Set oChild = oChild.accNavigate(NAVDIR_NEXT, ByVal 0&)
    If pChildName <> "*" Then lName = oChild.accName(ByVal 0&)
    If pChildRole <> "*" Then lRole = oChild.AccRole(ByVal 0&)
    If lRole Like pChildRole And lName Like pChildName Then
        Set FindChildByRoleOrName = oChild
        Exit Do
    End If
Loop
Exit Function
gestion_erreurs:
Set FindChildByRoleOrName = Nothing
Err.Clear
End Function
```

VII - Conclusion

La programmation de menus personnalisés en utilisant l'ancienne interface d'Excel, comme présentée ici, n'a pas la prétention de se substituer à l'utilisation des fichiers XLM pour configurer le ruban, procédé que préconise dorénavant Microsoft.
Nous avons proposé une méthode qui permet de répondre aux besoins d'applications simples, avec un minimum de programmation (pour les développeurs pressés).
Et comme cette pratique permet aussi d'ajouter des contrôles personnalisés au menu contextuel des cellules, il n'y a donc pas d'antagonisme entre ces deux technologies, mais plutôt, à notre avis, une bonne complémentarité.

Vous pouvez utiliser les fonctions étudiées en copiant le module VBO du fichier joint dans votre projet. Pensez aussi à y inclure un modèle de tableau comme ceux utilisés dans le fichier joint, pour vous faciliter la vie, ou utilisez la procédure « MenuCreationTableauStructure » pour en générer un vierge.

Laurent OTT.
2020.

VIII - Annexe 1 : Synthèse des quatre fonctions utilisées

MenuInstaller

- StrNomTableau As **String** : est le nom du tableau structuré qui contient les paramètres des menus comme vu ci-dessus.
- TypeMenuPerso As TypeMenuPersos : est l'un des quatre modes de restitution disponibles, à sélectionner dans l'énumération proposée : BarresOutils, CommandesMenu, BarrePopup, MenuContextuel.
- ParamArray Arguments() As Variant : facultatif, est un tableau d'éléments qui remplace le tableau structuré si le développeur décide de générer le menu directement dans le code. Chaque argument correspond à une ligne de tableau, où les éléments sont séparés par une virgule.

MenuSupprimer

- StrNomTableauOuControle As **String** : est le nom du tableau structuré ou du contrôle à supprimer.
  Si cet argument est vide alors tous les menus sont supprimés (y compris la barre pop-up et le menu contextuel, y compris aussi les menus personnels créés par d'autres applications).
  Si cet argument vaut « MaBarrePopup » alors ne supprime que la barre pop-up.
  Si cet argument vaut « MenuContextuel » alors restaure le menu contextuel ordinaire.

MenuInfoChoix

- StrNom As **String** : est le nom du contrôle concerné. Si plusieurs contrôles ont le même nom, c'est le premier trouvé qui est traité.
  Si cet argument est vide alors retourne les informations sur le contrôle actif (celui qui vient d'être validé).

- TypeInfoChoix As TypeMenuInfoChoix : est l'un des trois types d'informations qu'il faut retourner, à sélectionner dans l'énumération proposée : InfoTexte, InfoIndex, InfoLibellé.

MenuModifControle

- StrNom As **String** : est le nom du contrôle concerné. Si plusieurs contrôles ont le même nom, c'est le premier trouvé qui est traité.
- NumFaceId As Long : si supérieur à zéro alors remplace l'icône du contrôle par ce numéro.
- StrNouveauLibellé As **String** : si non vide alors remplace le libellé du contrôle.
- StrNouveauInfoBulle As **String** : si non vide alors remplace l'info-bulle du contrôle.

- Optional bVisible As Boolean : si sa valeur vaut True (valeur par défaut) alors affiche le contrôle, si la valeur vaut False alors masque le contrôle.

IX - Annexe 2 : Des liens utiles pour en savoir plus sur…

- Les bases de la programmation en VBA : **https://laurent-ott.developpez.com/tutoriels/programmation-excel-vba-tome-1/**.
- Les tableaux structurés : **https://fauconnier.developpez.com/tutoriels/tableaux-structures/**
- Les menus personnalisés : **https://fring.developpez.com/vba/excel/barremenu/**
- Les rubans : **https://silkyroad.developpez.com/excel/ruban/**
- La sélection d'un onglet du ruban avec le code d'Arkham46 : **https://www.developpez.net/forums/d697439/logiciels/microsoft-office/general-vba/contribuez/introduction-aux-fonctions-d-accessibilite/#post4423165**

X - Annexe 3 : Téléchargements

menus-personnalises.xlsm :

- Contient dans le module « VBO » le code source des fonctions étudiées dans cette documentation.
- La feuille « RésuméDesContrôles » résume les fonctionnalités des contrôles et donne un aperçu de leur rendu.
- La feuille « RésuméDesMenus » donne un aperçu du rendu de chacun des quatre types de restitution disponibles.
- La feuille « Menu » présente des exemples de menus. Les tableaux vous serviront de modèle pour vos applications.
- La feuille « ListeIcônes » permet de faire défiler la liste des icônes disponibles pour illustrer vos menus.

XI - Annexe 4 : Générer automatiquement un modèle de tableau structuré

Le code source de cette application contient une procédure permettant de générer automatiquement (sur une nouvelle feuille) un modèle de tableau structuré à utiliser pour créer vos menus personnalisés, au cas où les modèles du fichier joint auraient été perdus.

Il vous suffit de lancer la procédure « MenuCreationTableauStructure » et un tableau d'une trentaine de lignes se génère automatiquement, incluant les zones de liste modifiables et les mises en forme conditionnelles pour simplifier la configuration de vos menus.

Il ne vous reste plus qu'à déplacer ce tableau où bon vous semble, supprimer les lignes inutiles, en ajouter d'autres au besoin.

XII - Annexe 5 : Synthèse des fonctionnalités des contrôles

Les fonctionnalités de chaque contrôle

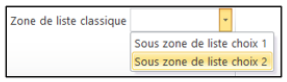
Type de contrôle	Accepte un nom	Accepte une icône	Peut être lié à une macro	Raccourci clavier souligné
Barre de Commande Bouton	OUI	OUI	OUI	NON

Affichage

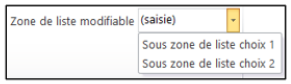
Type de contrôle	Accepte un nom	Accepte une icône	Peut être lié à une macro	Raccourci clavier souligné
Barre de Commande Contextuelle	OUI	NON	NON	NON
>> Bouton de commande	OUI	OUI	OUI	OUI
>> Menu Contextuel	OUI	NON	NON	OUI
>> >> Sous Bouton de commande	NON	OUI	OUI	OUI

Type de contrôle	Accepte un nom	Accepte une icône	Peut être lié à une macro	Raccourci clavier souligné
Zone de texte	OUI	NON	OUI	NON

Type de contrôle	Accepte un nom	Accepte une icône	Peut être lié à une macro	Raccourci clavier souligné
Zone de liste Classique	OUI	NON	OUI	NON
>> Sous Zone de liste	NON	NON	NON	NON

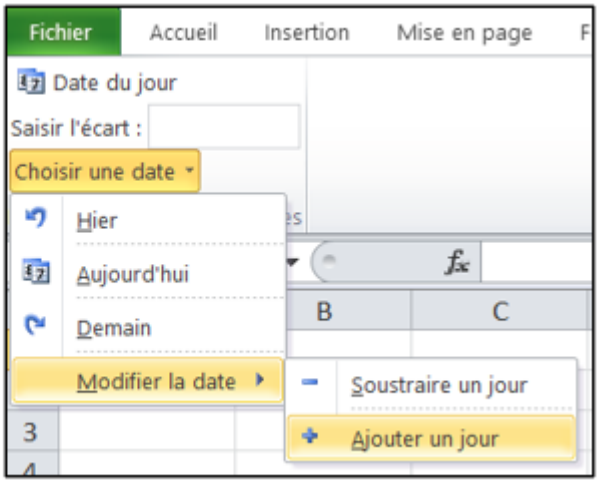


Type de contrôle	Accepte un nom	Accepte une icône	Peut être lié à une macro	Raccourci clavier souligné
Zone de liste Modifiable	OUI	NON	OUI	NON
>> Sous Zone de liste	NON	NON	NON	NON

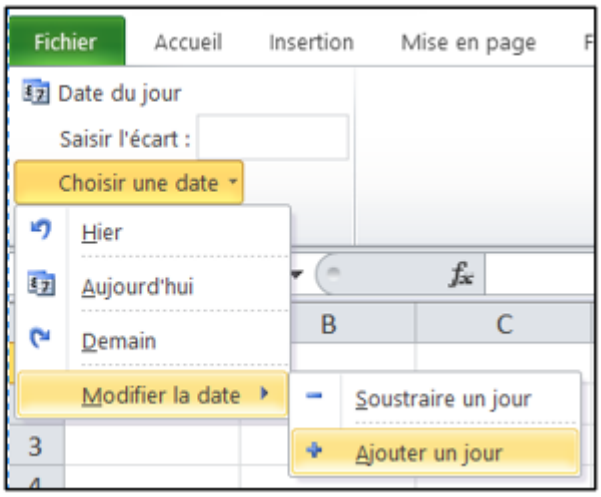


XIII - Annexe 6 : Exemples des modes de restitution

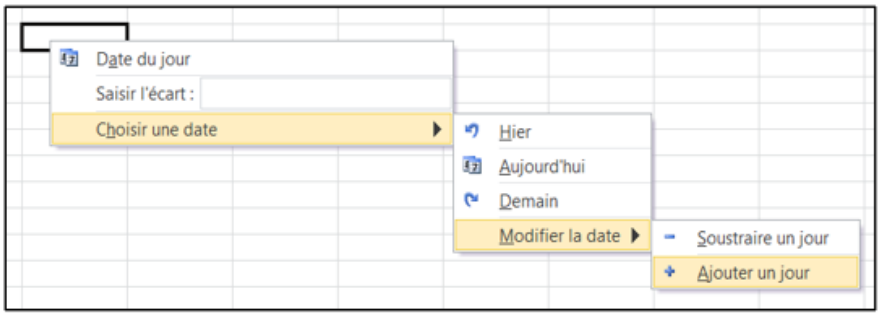
BarresOutils



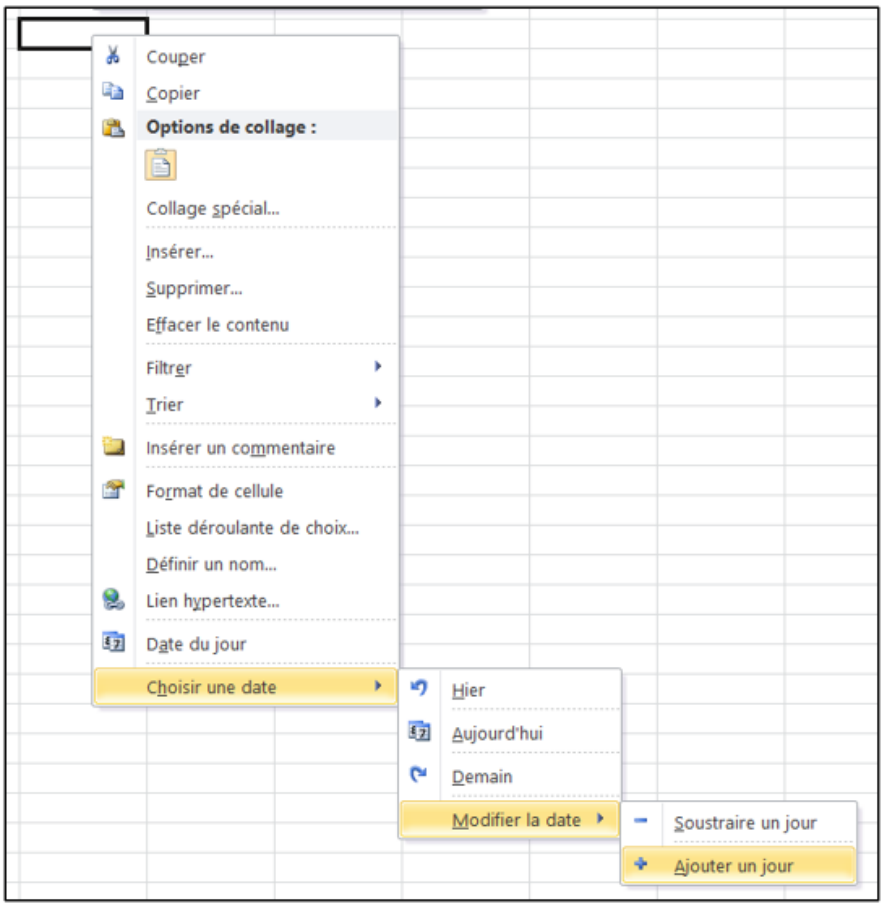
CommandesMenu



BarrePopup



MenuContextuel



XIV - Remerciements

Je remercie **gaby277** pour sa collaboration à la rédaction de cette documentation et pour ses conseils avisés.

Je remercie **Pierre Fauconnier** pour sa relecture technique et ses apports, **Claude Leloup** pour la correction orthographique, **Winjerome** pour la mise au gabarit.

Ainsi que toute l'équipe de Developpez.com qui participe à la maintenance du site.