Visual Basic pour Excel : compléments

Les plages de cellules

Nous avons exposé au chapitre précédent comment travailler avec des plages de cellules simples. Il est souvent nécessaire d'aller plus loin car la sélection de plages de cellules est souvent au cœur des programmes Visual Basic.

Les feuilles de calcul

Commençons par désigner la feuille de calcul dans laquelle nous voulons sélectionner des plages de cellules.

Il y a deux manières principales pour désigner une feuille de classeur : soit par son nom, soit par son numéro. Pour sélectionner une feuille on pourra donc écrire :

Sheets("Feuil1").Select

Ou :

Sheets(1). Select

Lorsque le classeur vient d'être créé, les deux écritures sont équivalentes car *Feuil1* correspond à la première feuille mais ce n'est plus le cas si l'on crée d'autres feuilles et qu'on les déplace.

Le numéro désigne la position de la feuille dans la liste des onglets, si on la déplace elle change de numéro. Ainsi, 1 est le numéro de la feuille la plus à gauche, 2 est le numéro de la feuille dont l'onglet est immédiatement à sa droite et ainsi de suite. À l'inverse une feuille ne change pas de nom quand on la déplace.

On peut déplacer une feuille avec Visual Basic en utilisant la méthode *Move*. Par exemple :

Sheets(4).Move before:=Sheets(1)

On aurait pu utiliser *after* au lieu de *before* pour déplacer la feuille après plutôt qu'avant. On peut utiliser la méthode *Copy* pour copier la feuille et non la déplacer.

Notons que nous avons utilisé ici l'opérateur ":=" et non l'opérateur "=". En effet, Visual Basic utilise l'opérateur = pour affecter une valeur

ou un objet à une variable et l'opérateur := pour affecter une valeur au paramètre d'une propriété ou d'une méthode.

On peut également changer le nom d'une feuille. Par exemple :

```
Sheets("Feuil1").Name = "Bonjour"
```

Lorsque dans un programme on fait plusieurs fois référence à la même feuille, il est plus pratique de l'affecter à une variable. Par exemple :

```
Set f = Sheets("Feuil1")
```

Lorsque plusieurs classeurs sont ouverts en même temps, il est préférable de préciser auquel on se réfère. Si l'on se réfère au classeur contenant le programme on peut écrire :

```
Set f = ThisWorkBook.Sheets("Feuil1")
```

On peut également se référer à un autre classeur ouvert, par exemple :

```
Set f = WorkBooks("Essai.xlsx").Sheets("Feuil1")
```

Les plages de cellules

On peut désigner une plage par ses références ou par son nom, par exemple Range("B2:C6") ou Range("Plage"). On peut également utiliser une variable texte désignant les références de la plage ou son nom. Par exemple :

```
Set f = ThisWorkbook.Sheets(1)
a = "A2:C8"
b = "Plage"
f.Range(a).Select
f.Range(b).Select
```

Pour utiliser la référence ou le nom d'un champ, il faut les mettre entre guillemets, pour utiliser une variable il ne faut pas de guillemets.

Les noms

On peut définir le nom d'une cellule ou d'une plage en utilisant la propriété Name. Par exemple :

```
Set f = ThisWorkbook.Sheets("Feuil1")
f.Range("A2").Name = "Essai"
```

Dans ce cas, le nom est défini au niveau du classeur. Pour définir un nom au niveau d'une feuille, il faut procéder autrement en utilisant la propriété RefersToR1C1, par exemple :

```
Set f = ThisWorkbook.Sheets("Feuil1")
f.Names.Add Name:="Essai", RefersToR1C1:="=R4C5:R7C8"
```

Ici, R fait référence au numéro de ligne et C au numéro de colonne. Notons qu'il faut un signe égal à l'intérieur des guillemets.

On aurait obtenu le même résultat en utilisant la propriété RefersTo de la manière suivante :

```
f.Names.Add Name:="Essai", RefersTo:="=$E$4:$H$7"
```

Ici, il faut utiliser la notation absolue avec le signe $.

On aurait pu utiliser la même procédure pour définir un nom au niveau d'une cellule en faisant référence au classeur et en spécifiant la feuille dans la définition de la plage. Par exemple :

```
ThisWorkbook.Names.Add Name:="Essai",
RefersToR1C1:="=Feuil1!R4C5:R7C8"
```

On peut modifier la cellule ou la plage à laquelle le nom fait référence en utilisant les propriétés RefersTo ou RefersToR1C1, par exemple :

```
f.Names("Essai").RefersTo = "=Feuil1!$C$4:$E$10"
```

Décaler ou redimensionner une plage

Pour sélectionner une plage décalée par rapport à une plage donnée on peut utiliser la méthode Offset. Par exemple :

```
f.Range("A2:C8").Offset(2, 4).Select
```
sélectionne "E4:G10" car elle a décalé la plage de 2 lignes vers le bas et de 4 colonnes à droite.

La méthode Resize permet de redimensionner une plage. Par exemple :

```
f.Range("A2:C8").Resize(20, 10).Select
```
sélectionne la plage "A2:J21" car elle a redimensionné la plage "A2:C8" à partir de son coin supérieur gauche en portant son nombre de lignes à 20 et son nombre de colonnes à 10.

Lignes et colonnes

Il est possible de sélectionner une ligne ou une colonne en utilisant les propriétés Rows et Columns. Par exemple Rows(4).Select sélectionne la troisième ligne et Columns(4).Select sélectionne la quatrième colonne.

Il est souvent utile de connaître le nombre de lignes et de colonnes d'une plage. On utilise pour cela la propriété Count. Par exemple, f.Range("A2:C8").Rows.Count renvoie 7, c'est-à-dire le nombre de lignes de la plage. f.Range("A2:C8").Columns.Count renvoie 3, c'est-à-dire le nombre de colonnes de la plage.

Attention : Row renvoie le numéro de la première ligne de la plage et Column renvoie le numéro de sa première colonne. Par exemple f.Range("A2:C8").Row renvoie 2 et f.Range("A2:C8").Column renvoie 1.

Pour sélectionner une plage rectangulaire autour d'une cellule, on utilise la méthode CurrentRegion, elle sélectionne une zone rectangulaire délimitée par une ligne et une colonne vides.

Travailler avec des plages

On peut faire des calculs avec les cellules d'une feuille, soit simplement, soit en les combinant avec les fonctions de visual basic. Par exemple :

```
f.Range("A1")=f.Range("A2")*34 + f.Range("B1")*66
f.Range("A1") = Mid(Range("B1"), 3, 4)
```

On peut aussi utiliser des fonctions de la feuille de calcul dans leur version anglaise en les précédant de Application. Par exemple

```
f.Range("A2") = Application.Average(Range("B2:E2"))
f.Range("A3") = Application.Sum(10.4, 22.6, 40)
```

On peut aussi faire directement des calculs sur des plages en utilisant des formules matricielles grâce à la propriété FormulaArray. Par exemple :

```
f.Range("G15:J20").FormulaArray = "=G4:G9*G11:J11/G13:J13"
```

Elle peut encore s'écrire :

```
f.Range("G15:J20").FormulaArray = "=R[-11]C:R[-6]C[3]*R[-4]C:R[-4]C[3]/R[-2]C:R[-2]C[3]"
```

Réduire les temps de traitement

Les opérations de lecture et d'écriture dans des cellules prennent beaucoup de temps en comparaison des opérations réalisées directement dans la mémoire. Lorsque l'on doit travailler sur de grandes plages de cellules, la durée de traitement des opérations peut être non négligeable et il peut être utile de la réduire.

Un premier moyen pour cela est de figer l'écran pendant les opérations de lecture et d'écriture dans les plages. Ce sera fait grâce à la propriété ScreenUpdating. Au début du programme, on figera l'écran en écrivant :

```
ScreenUpdating = False
```

A la fin du programme, il ne faut surtout pas oublier de réactiver l'écran en écrivant :

```
ScreenUpdating = True
```

Le deuxième moyen pour accélérer les programmes est de copier les plages dans des variables, de travailler sur ces variables et de les recopier ensuite dans des plages.

Par exemple, supposons que l'on veuille multiplier par 2 toutes les cellules de la plage "A1:BA50000". Une première méthode consiste à multiplier directement par 2 chaque cellule de la plage, par exemple avec le programme suivant :

```
Sub MultiplicationPlage()
Set f = ThisWorkbook.Sheets(1)
f.Range("A1:BA50000") = 100
For i = 1 To 50000
    For j = 1 To 53
        f.Cells(i, j) = 2 * f.Cells(i, j)
    Next j
Next i
End Sub
```

Ce programme est extrêmement long à s'exécuter et, si vous êtes pressé, je vous conseille de tester directement le programme suivant qui s'exécute très rapidement :

```
Sub MultiplicationPlage()
Dim a As Variant
Set f = ThisWorkbook.Sheets(1)
f.Range("A1:BA50000") = 100
a = f.Range("A1:BA50000")
For i = 1 To 50000
    For j = 1 To 53
        a(i, j) = 2 * a(i, j)
    Next j
Next i
f.Range("A1:BA50000") = a
End Sub
```

Variables et constantes

Les variables

Il est toujours préférable de déclarer les variables car cela permet une exécution plus rapide des programmes. Lorsqu'une variable n'est pas déclarée, Visual Basic lui attribue automatiquement le type Variant, c'est-à-dire qu'il est possible de lui attribuer n'importe quel type de valeur ou d'objet.

Il existe plusieurs manières de déclarer des variables. La plus habituelle est d'utiliser l'instruction Dim à l'intérieur d'une procédure, c'est-à-dire entre les instructions Sub et End Sub. Généralement, l'instruction Dim est placée directement après l'instruction Sub. Il y a cependant d'autres manières de déclarer une variable.

Visual Basic dispose de modules dans lesquels sont écrites les procédures, il existe également des procédures rattachées directement à des feuilles de calcul ou au classeur. Aussi, lorsque l'on veut définir une variable, on doit se demander quelle doit être sa portée, c'est-à-dire se demander si cette variable doit pouvoir être utilisée uniquement à l'intérieur d'une procédure spécifique, dans toutes les procédures d'un module ou dans toutes les procédures de l'ensemble des modules.

- Lorsqu'une variable ne doit être utilisée qu'au sein d'une procédure, elle doit être déclarée à l'intérieur de cette procédure.
- Pour qu'une variable puisse être utilisée par toutes les procédures d'un même module, elle doit être déclarée au tout début du module, avant la première procédure.

- Pour qu'une variable puisse être utilisée par toutes les procédures de l'ensemble des modules, elle doit être déclarée au tout début d'un module en utilisant l'instruction Public au lieu de Dim.

Les constantes

Il est souvent intéressant de travailler avec des constantes, c'est à-dire des variables dont la valeur ne changera pas tout au long de l'exécution du programme. C'est par exemple le cas lorsque l'on veut définir une codification qui sera reprise sans changement dans tout le programme.

Pour définir une constante, il faut utiliser l'instruction Const et fixer sa valeur, par exemple :

```
Const a=10.4
Const b as Integer=2
Const c as String="Bonjour"
```

Comme pour les variables, la portée des constantes dépend de leur mode de déclaration :

- une constante déclarée à l'intérieur d'une procédure n'est utilisable qu'à l'intérieur de cette procédure ;
- une constante déclarée au tout début d'un module est utilisable par toutes les procédures de ce module ;
- pour qu'une constante soit utilisable dans toutes les procédures de l'ensemble des modules, il faut la déclarer au tout début d'un module et faire précéder l'instruction Const de Public.

Il n'est pas possible de déclarer directement un tableau comme constante. Si on veut le faire, il faut utiliser une chaîne de caractères et la fonction Split qui décompose une chaîne en matrice à partir d'un séparateur. Le programme ci-dessous utilise le séparateur ; et transforme la chaîne des constantes en un tableau à une dimension :

```
Public Const tabl As String = "56;87;98"
---------------------------------------
Sub constantes()
Dim tableau(2)
Set f = ThisWorkbook.Sheets(1)
a = Split(tabl, ";")
For i = 0 To 2
    b = Split(a(i), ";")
    tableau(i) = Split(a(i), ";")
    f.Cells(i + 1, 10) = tableau(i)
Next i
End Sub
```

Si l'on veut définir un tableau de constantes à 2 dimensions, on peut utiliser une chaîne de caractères avec deux séparateurs différents, par exemple ; et > comme dans le programme ci-dessous :

```
Public Const tabl As String =
"1;11;12;13>2;21;22;23>3;31;32;33>4;41;42;43>5;51;52;53"
------------------------------------------------------------------------
-----
Sub constantes()
Dim tableau(4, 3)
Set f = ThisWorkbook.Sheets(3)
a = Split(tabl, ">")
For i = 0 To 4
    b = Split(a(i), ";")
    For j = 0 To 3
        tableau(i, j) = b(j)
        f.Cells(i + 1, j + 1) = tableau(i, j)
    Next j
Next i
End Sub
```

Les messages

Il est souvent utile de communiquer ou de demander des informations à l'utilisateur au cours de l'exécution d'un programme Visual Basic. Pour cela, Excel dispose de boîtes de dialogue qu'il est possible de définir à partir de Visual Basic. Nous ne présenterons ici que les plus courantes.

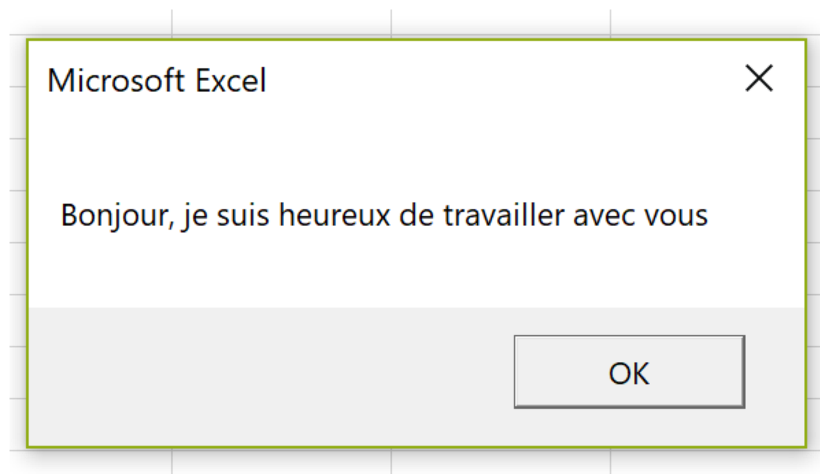
La fonction MsgBox pour afficher des messages

La fonction MsgBox permet soit d'afficher des messages simples, soit de demander une réponse à l'utilisateur.

Pour écrire un message simple, la fonction MsgBox doit être suivie du message placé entre guillemets, comme dans l'exemple suivant :

```
Sub Message()
MsgBox "Bonjour, je suis heureux de travailler avec vous"
End Sub
```

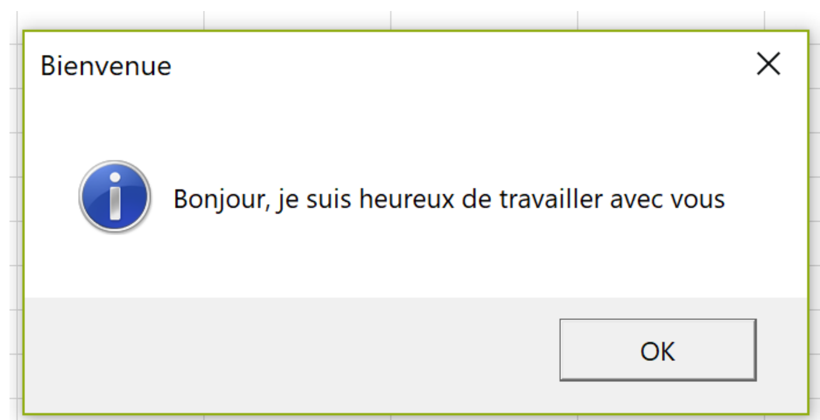
Ce programme fait apparaître le message suivant :



En cliquant sur OK, on fait disparaître le message et le programme poursuit son exécution.

On peut également rajouter un bouton et un titre. Par exemple, le programme suivant fait apparaître le message avec le titre *Bienvenue* et l'icône d'information :

```
Sub Message()
MsgBox "Bonjour, je suis heureux de travailler avec vous",
vbInformation, "Bienvenue"
End Sub
```



Si l'on veut afficher le titre mais pas l'icône, il faudra écrire :

```
MsgBox "Bonjour, je suis heureux de travailler avec vous", ,"Bienvenue"
```

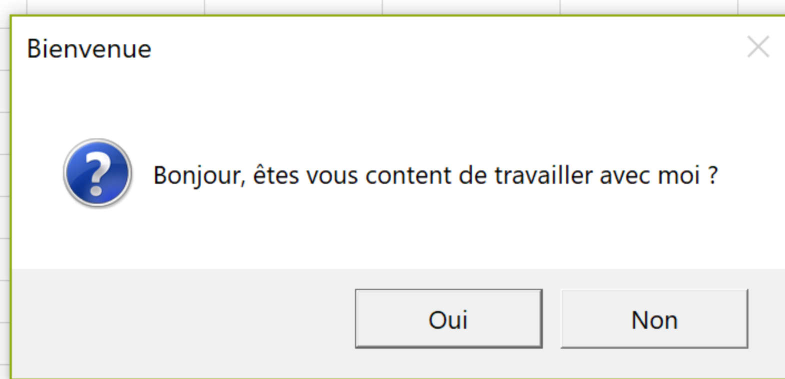
La fonction MsgBox pour poser des questions

Si l'on veut poser une question à l'utilisateur, il faut utiliser la fonction MsgBox en la faisant suivre de parenthèses comprenant la question, le type de bouton et le titre.

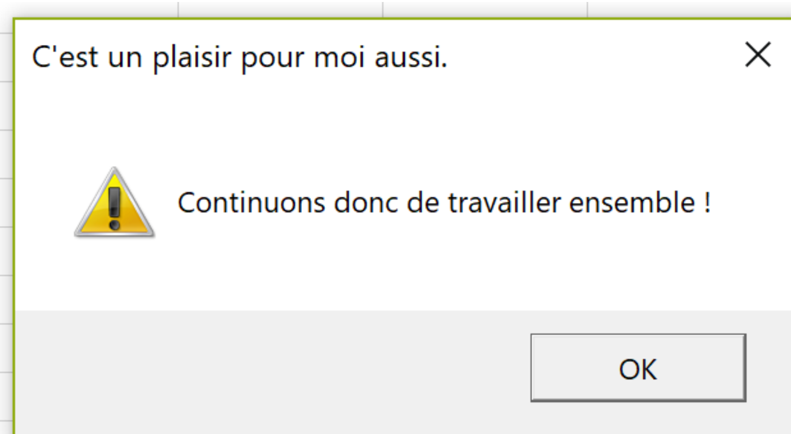
Prenons l'exemple du programme suivant :

```
Sub Message()
Dim Reponse As Integer
Reponse = MsgBox("Bonjour, êtes vous content de travailler avec moi
?", vbYesNo + vbQuestion, "Bienvenue")
If Reponse = vbYes Then MsgBox "Continuons donc de travailler
ensemble !", vbExclamation, "C'est un plaisir pour moi aussi."
If Reponse = vbNo Then MsgBox "Dans ces conditions, il vaut mieux
arrêter !", vbCritical, "J'en suis vraiment désolé !"
End Sub
```

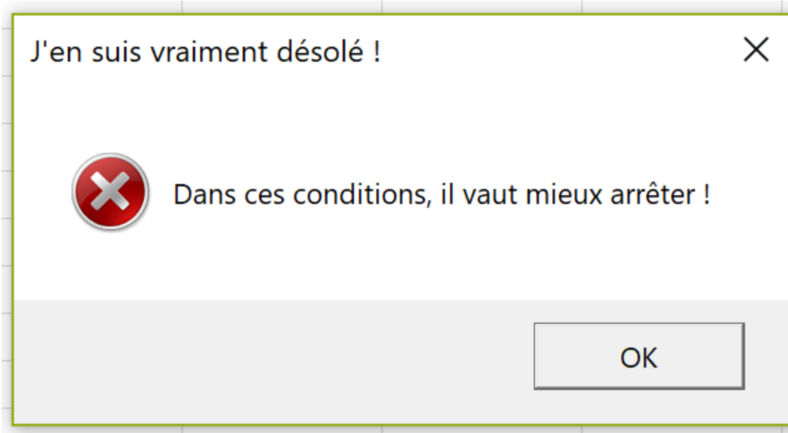
Ce programme fait apparaître le message suivant qui demande une réponse :



Si la réponse est Oui, il fait apparaître :



Si la réponse est non, il fait apparaître le message suivant :



Dans le programme, la première fonction MsgBox renvoie une valeur que nous plaçons dans la variable Reponse. Si cette valeur est vbYes nous affichons le deuxième message, si elle est vbNo c'est le troisième message qui est affiché.

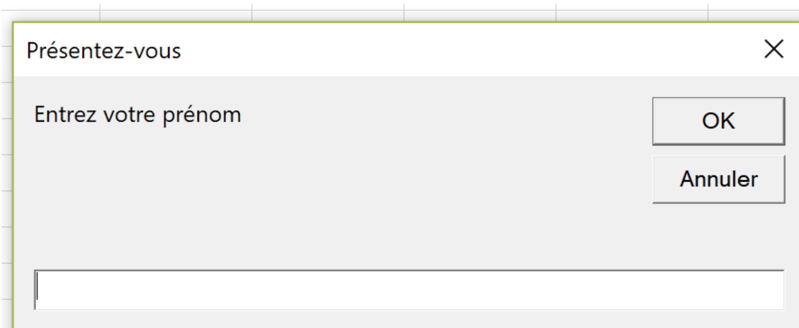
Remarquons que dans le premier message nous avons combiné les boutons Oui - Non avec l'icône interrogation en ajoutant vbYesNo et vbQuestion.

La fonction InputBox

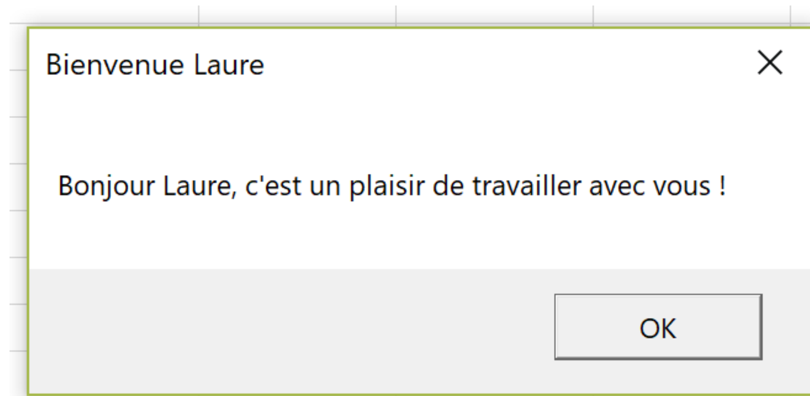
La fonction InputBox permet de communiquer un nombre ou un texte au programme. Par exemple, le programme ci-dessous demande d'introduire son prénom.

```
Sub Message()
Dim Prenom As String
Prenom = InputBox("Entrez votre prénom", "Présentez-vous")
MsgBox "Bonjour " & Prenom & ", c'est un plaisir de travailler avec vous
!", , "Bienvenue " & Prenom
End Sub
```

La boîte suivante apparaît :



Si le prénom entré est Laure, le message suivant apparaît :



Dans la fonction InputBox, le premier texte correspond au message et le second au titre.

Les contrôles ActiveX

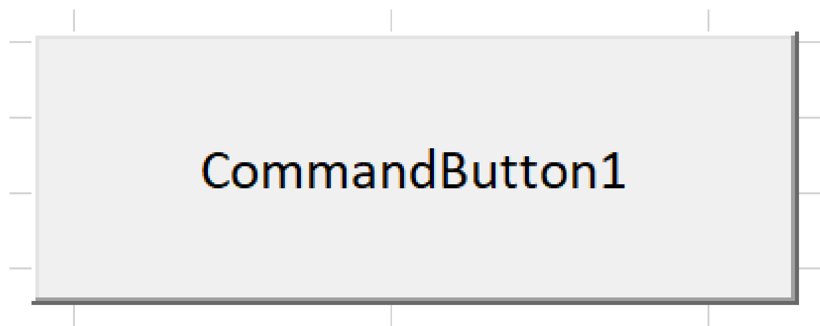
Pour dialoguer de manière plus approfondie avec l'utilisateur, Visual Basic utilise des contrôles. Il existe deux types de contrôles : les contrôles de formulaire et les contrôles ActiveX. Nous présenterons ici uniquement les contrôles ActiveX car ce sont les plus récents et les plus performants.

Pour utiliser un contrôle ActiveX, il faut aller dans le menu *Développeur* et sélectionner *Insérer*. Douze contrôles ActiveX sont alors proposés. Nous nous contenterons de présenter les plus courants.

Le bouton de commande

Le premier contrôle ActiveX proposé est le bouton de commande, c'est aussi le plus simple car il se contente d'exécuter un programme lorsqu'on clique dessus. Quand on le sélectionne, une petite croix apparaît sur la feuille de calcul, elle permet de dessiner le bouton de commande à l'endroit de son choix en maintenant enfoncé le bouton gauche de la souris.

Il se présente, par exemple, sous la forme suivante :



Il est ensuite possible de le modifier si le *Mode création* est activé dans le menu *Développeur*. En cliquant sur le bouton de commande avec le bouton droit de la souris, on fait apparaître différentes possibilités.

Si l'on sélectionne *Objet Bouton de Commande* puis *Edition*, on peut modifier son titre. Par exemple, on peut remplacer *CommandButton1* par *Bonjour*.

Si l'on sélectionne *Propriétés* on voit apparaître la liste des propriétés du bouton de commande. On voit que son nom est toujours *CommandButton1* et que le texte "Bonjour" correspond à *Caption*. Toutes les propriétés peuvent être changées directement dans la boîte de dialogue ou par un programme. Par exemple, le programme suivant remplace le texte "Bonjour" par "Bonjour Laure".

```
Sub Bouton()
ThisWorkbook.Sheets(1).CommandButton1.Caption = "Bonjour Laure"
End Sub
```

On peut également modifier plusieurs propriétés à la fois, par exemple avec le programme suivant :

```
Sub Bouton()
With ThisWorkbook.Sheets(1).CommandButton1
    .Caption = "Bonjour Laure"
    .BackColor = &HFFFF&
    .Font.Name = "Arial"
    .Font.Bold = True
    .Font.Size = 16
    .ForeColor = &HFF&
End With
End Sub
```

Le bouton suivant apparaît :

Mais un bouton de commande n'a d'intérêt que parce qu'il permet de lancer un programme. Pour créer le programme, il faut d'abord activer le *Mode Création* du menu *Développeur* puis cliquer avec le bouton droit de la souris sur le bouton de commande. Si l'on sélectionne *Visualiser le code*, le programme suivant apparaît :

```
Private Sub CommandButton1_Click()

End Sub
```

On peut alors écrire son programme, par exemple :

```
Private Sub CommandButton1_Click()
MsgBox "Bonjour Laure"
End Sub
```

Pour exécuter ce programme, il faut d'abord désélectionner le mode création puis cliquer sur le bouton de commande. Le message "Bonjour Laure" apparaît alors.

La zone de liste déroulante

Le deuxième contrôle ActiveX proposé par le menu Excel est la zone de liste déroulante. Elle permet de sélectionner dans une liste une valeur qui sera communiquée au programme. La sélection d'une valeur lancera également le programme associé à la zone de liste déroulante.

La zone de liste déroulante se crée de la même manière que le bouton de commande. Il faut ensuite lui associer au minimum une liste dans laquelle on pourra sélectionner un élément. On peut le faire par la boîte *Propriétés* ou par un programme. Par exemple, le programme suivant attribue la liste "A1:A4" à la zone de liste déroulante ComboBox1 :

```
Sub Liste()
Set f = ThisWorkbook.Sheets(1)
f.ComboBox1.ListFillRange = "A1:A4"
End Sub
```

On peut également créer une cellule liée qui affichera l'élément de la liste sélectionné. Il est également possible de faire apparaître le numéro dans la liste de l'élément sélectionné, le premier élément correspondant à zéro. Cela peut être fait avec le programme suivant :

```
Sub Liste()
Set f = ThisWorkbook.Sheets(1)
With f.ComboBox1
  ,   .ListFillRange = "A1:A4"
  ,   .LinkedCell = "D1"
End With
f.Range("D3") = f.ComboBox1.ListIndex
End Sub
```

Si l'on sélectionne *Sardine* dans la liste proposée, on obtient :

	A	B	C	D	E
1	Lapin			Sardine	
2	Sardine	Sardine ▾			
3	Pigeon			1	
4	Pomme				
5					

On peut sélectionner un élément de la liste par le programme grâce à la propriété ListIndex. Par exemple :

```
f.ComboBox1.ListIndex=2
```

sélectionne *Pigeon*.

Mais une zone de liste déroulante serait de peu d'intérêt si elle ne permettait pas de lancer un programme. Pour créer le programme, il faut d'abord activer le *Mode Création* du menu *Développeur* puis cliquer avec le bouton droit de la souris sur la zone de liste déroulante. Si l'on sélectionne *Visualiser le code*, le programme suivant apparaît :

```
Private Sub ComboBox1_Change()

End Sub
```

On peut alors écrire un programme, par exemple :

```
Private Sub ComboBox1_Change()
Dim Reponse As Integer
Set f = ThisWorkbook.Sheets(1)
Reponse = f.ComboBox1.ListIndex
If Reponse = 0 Then MsgBox "Le lapin est un mammifère"
If Reponse = 1 Then MsgBox "La sardine est un poisson"
If Reponse = 2 Then MsgBox "Le pigeon est un oiseau"
If Reponse = 3 Then MsgBox "La pomme est un fruit"
End Sub
```

Pour exécuter ce programme, il faut d'abord désélectionner le mode création puis sélectionner un élément dans la liste déroulante. Si l'on sélectionne le pigeon, le message "Le pigeon est un oiseau" apparaît alors.

Notons que si l'on modifie un élément de la liste déroulante, le programme est exécuté automatiquement. Par exemple, si l'on exécute le programme suivant qui modifie un élément de la liste, le programme associé à la liste déroulante s'exécute.

```
Sub Liste()
Set f = ThisWorkbook.Sheets(1)
f.Range("A4") = "Poire"
End Sub
```

Le message "Le pigeon est un oiseau" s'affiche alors.

Si l'on veut modifier les éléments de la liste déroulante sans relancer le programme associé, on peut supprimer la liste déroulante avant d'effectuer les modifications puis la rétablir ensuite. Par exemple, le programme suivant remplace "Pomme" par "Poire" sans relancer le programme associé à la liste déroulante.

```
Sub Liste()
Set f = ThisWorkbook.Sheets(1)
f.ComboBox1.ListFillRange = ""
f.Range("A4") = "Poire"
f.ComboBox1.ListFillRange = "A1:A4"
End Sub
```

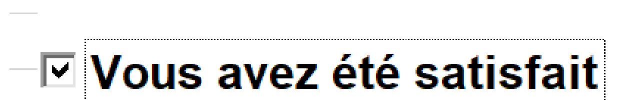
La case à cocher

La case à cocher est le troisième contrôle ActiveX proposé par le menu d'Excel. Elle renvoie la valeur *True* si elle est cochée et *False* sinon. Pour

la créer, il faut procéder comme pour les précédents contrôles. On peut la personnaliser par un programme. Par exemple :

```
Sub CaseAcocher()
With ThisWorkbook.Sheets(1).CheckBox1
    .Caption = "Vous avez été satisfait"
    .Font.Name = "Arial"
    .Font.Bold = True
    .Font.Size = 12
End With
End Sub
```

La case à cocher apparaîtra alors ainsi :



On peut également lui associer un programme, par exemple :

```
Private Sub CheckBox1_Click()
Set f = ThisWorkbook.Sheets(1)
If f.CheckBox1 Then MsgBox "J'en suis heureux !" Else MsgBox "J'en
suis désolé !"
End Sub
```

Si l'on coche la case, le message "J'en suis heureux !" apparaît, si on la décoche, le message "J'en suis désolé !" apparaît.

Les autres contrôles ActiveX peuvent être traités de manière analogue.

Protéger un classeur avec un mot de passe

Il est souvent utile de protéger un classeur avec un mot de passe pour éviter qu'il puisse être modifié de manière accidentelle. Cela peut se faire directement par le menu de la feuille de calcul mais cela peut aussi se faire depuis un programme Visual Basic grâce à la méthode *Protect*. Cette méthode peut appliquer aussi bien à un classeur qu'à une feuille de calcul. Par exemple :

```
ThisWorkbook.Sheets("Accueil").Protect Password:="ABCD"
```

Où *ABCD* est le mot de passe. On peut également interdire la sélection de cellules protégées de la manière suivante :

```
ThisWorkbook.Sheets("Accueil").EnableSelection = xlUnlockedCells
```

On peut rajouter des options à la méthode *Protect*, par exemple *AllowUsingPivotTables* qui permet d'utiliser les tableaux croisés dynamiques sans pouvoir toutefois les modifier. On l'écrit ainsi :

```
ThisWorkbook.Sheets("Accueil").Protect Password:="ABCD",
AllowUsingPivotTables:=True
```

On peut également protéger le classeur de la même manière. Par exemple :

```
ThisWorkbook.Protect Password:="ABCD", Structure:=True,
Windows:=False
```

Pour ôter la protection, il faut utiliser la méthode *Unprotect*, par exemple :

```
ThisWorkbook.Sheets("Accueil").Unprotect Password:="ABCD"
```

Si l'on veut créer un mot de passe particulièrement difficile à retrouver, on peut utiliser un mot de passe généré de manière aléatoire. Par exemple avec le programme :

```
a = Int(Rnd * 100000)
pw = "QGa!Ez" & a
ThisWorkbook.Sheets("Accueil").Protect Password:=pw
```

Bien entendu, dans ce cas il est indispensable de garder une copie non protégée pour pouvoir la modifier ultérieurement.

Protéger les programmes

Il est aussi possible de protéger les programmes Visual Basic. Pour cela, dans la feuille Visual Basic, il faut cliquer avec le bouton droit sur VBAProject. Il faut alors sélectionner *Propriétés de VBAProject*, dans le cadre qui apparaît il faut aller dans *Protection* puis entrer le mot de passe. La protection sera effective après avoir enregistré et fermé le classeur. Les programmes ne seront alors pas visibles lors de la prochaine ouverture.

Pour ôter la protection, il faut aller dans le menu *Développeur* puis sélectionner *Visualiser le code*. Après avoir entré le code, la feuille Visual Basic apparaît. On peut alors développer *VBAProject* et choisir le module auquel on veut accéder.

La gestion des erreurs

Lorsqu'il rencontre une erreur, par exemple une division par zéro, le programme Visual Basic s'arrête et affiche un message d'erreur. Cela permet généralement au programmeur de corriger une erreur involontaire mais, parfois, l'erreur provient de circonstances particulières et non véritablement d'une erreur de programmation.

Par exemple, on veut ouvrir un classeur Excel mais il n'existe plus ou a changé de nom. On peut, bien sûr, tester l'existence du classeur avant de tenter de l'ouvrir mais, lorsque les circonstances pouvant générer une erreur sont multiples, on peut préférer gérer toutes les sources d'erreur de la même manière. C'est possible grâce à l'instruction On Error.

L'instruction On Error se décline de trois manières :

- On Error GoTo, qui renvoie à une ligne désignée par une étiquette lorsqu'une erreur est rencontrée ;
- On Error Resume Next qui renvoie à la ligne suivant immédiatement la ligne ayant généré l'erreur ;
- On Error GoTo 0 qui désactive et réinitialise la gestion des erreurs.

Une étiquette est un nom placé au tout début d'une ligne et suivi de : (deux points), par exemple :

Erreur:

La gestion des erreurs n'est pas simple et pour mieux la comprendre nous partirons d'un exemple. Le programme suivant se propose de remplacer les nombres situés dans la plage A1:G1 par leur inverse, cette plage se présentant comme suit :

	A	B	C	D	E	F	G
1	0,2	0,1	1000	e	4		5

```
Sub EssaiErreur()
Dim a As Range
Set f = ThisWorkbook.Sheets("Feuil1")
For Each a In Range("A1:G1")
    a = 1 / a
Next a
End Sub
```

Quand on le lance, ce programme se bloque sur la cellule D1 et affiche le message "Incompatibilité de type". En effet, la cellule D1 contient la

lettre e et on ne peut donc pas calculer son inverse. Pour résoudre le problème, nous allons introduire dans le programme l'instruction On Error Resume Next :

```
Sub EssaiErreur()
Dim a As Range
Set f = ThisWorkbook.Sheets("Feuil1")
On Error Resume Next
For Each a In Range("A1:G1")
    a = 1 / a
Next a
End Sub
```

Ce programme fonctionne parfaitement, il ne calcule l'inverse que quand il le peut et ne se bloque pas.

Supposons maintenant que nous voulions introduire la valeur 1000 dans les cellules où l'inversion n'est pas possible. Il nous faut détecter ces cellules dans le programme et pour cela nous allons utiliser l'objet Err dont la propriété est un nombre qui caractérise le type d'erreur rencontré, ce nombre étant initialisé à zéro.

Cela veut dire que tant que le programme n'a pas rencontré d'erreur, Err a une valeur nulle et cette valeur ne change que quand une erreur est rencontrée. Après l'erreur, la valeur de Err ne revient pas à zéro automatiquement, pour le faire il faut utiliser la méthode Err.Clear.

Nous allons utiliser le programme suivant :

```
Sub EssaiErreur1()
Dim a As Range
Set f = ThisWorkbook.Sheets("Feuil1")
On Error Resume Next
For Each a In Range("A1:G1")
    a = 1 / a
    If Err <> 0 Then
      a = 1000
    End If
Next a
b = 1 / 0
End Sub
```

Nous avons inséré à la fin de ce programme une division par zéro pour tester le traitement des erreurs involontaires.

Si nous lançons le programme, nous constatons qu'il ne se bloque pas et qu'il affiche le résultat suivant :

	A	B	C	D	E	F	G
1	0,2	0,1	1000	1000	1000	1000	1000

Le programme a bien mis 1000 quand il a rencontré la lettre e mais il a continué à mettre 1000 même quand il n'y avait pas d'erreur. De plus, il ne nous a pas signalé notre erreur involontaire, c'est-à-dire la division par zéro.

En fait, notre programme n'a pas fonctionné comme voulu parce que Err ne revient pas automatiquement à zéro, si bien qu'il est resté positif après la première erreur. Pour que le programme fonctionne correctement, il faut donc réinitialiser Err après avoir traité l'erreur.

De plus, si l'on veut pouvoir détecter les erreurs involontaires dans la suite du programme, il nous faut désactiver la gestion des erreurs avant de poursuivre le programme. On aura donc :

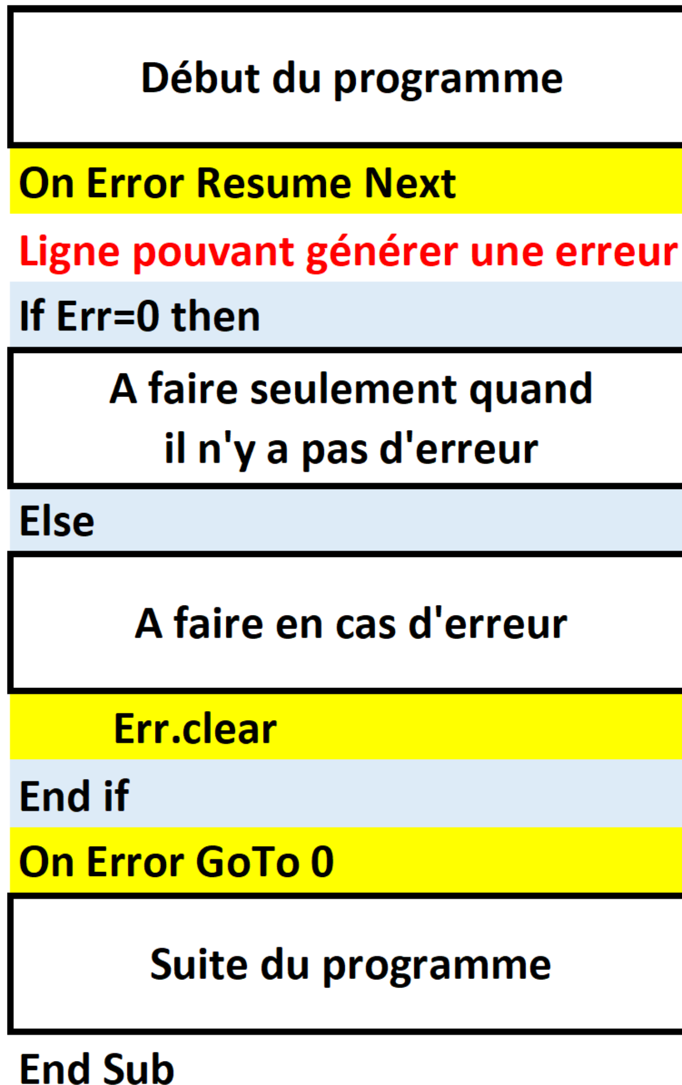
```
Sub EssaiErreur1()
Dim a As Range
Set f = ThisWorkbook.Sheets("Feuil1")
On Error Resume Next
For Each a In Range("A1:G1")
    a = 1 / a
    If Err <> 0 Then
        a = 1000
        Err.Clear
    End If
Next a
On Error GoTo 0
b = 1 / 0
End Sub
```

Ce programme fonctionne parfaitement car il fait bien l'inversion quand il le peut et se bloque pour nous indiquer la division par zéro involontaire.

D'une manière générale, lorsque nous utilisons l'instruction On Error Resume Next, nous avons deux possibilités :

- nous ignorons l'erreur et nous continuons, la ligne erronée n'est pas exécutée ;
- nous voulons agir différemment selon qu'il y a ou non une erreur ; dans ce cas, il faut indiquer au programme ce qu'il doit faire en cas d'erreur et ce qu'il doit faire quand il n'y a pas d'erreur.

Dans le premier cas, on se contentera de l'instruction On Error Resume Next, dans le second on pourra structurer le programme de la manière suivante :



Nous pouvons également utiliser l'instruction On Error GoTo pour gérer les erreurs.

Le programme précédent pourrait alors s'écrire :

```
Sub EssaiErreur4()
Dim a As Range
Set f = ThisWorkbook.Sheets("Feuil1")
On Error GoTo Erreur
For Each a In Range("A1:G1")
    a = 1 / a
Boucle:
Next a
On Error GoTo 0
Exit Sub
Erreur:
a = 1000
Resume Boucle
End Sub
```

Dans ce programme, nous avons utilisé des étiquettes et l'instruction Resume. Quand elle est suivie du nom d'une étiquette, cette instruction renvoie à la ligne de l'étiquette, sinon elle renvoie à la ligne qui a généré l'erreur. Dans ce dernier cas, attention aux boucles infinies si l'erreur n'est pas corrigée. L'instruction Resume réinitialise la gestion des erreurs, c'est-à-dire qu'elle remet Err à zéro.

Pour éviter que les lignes gérant les erreurs soient exécutées même quand il n'y a pas d'erreur, nous les avons placées à la fin du programme et nous les avons fait précéder de l'instruction Exit Sub qui provoque l'arrêt du programme.

D'une manière générale, la structure d'un programme utilisant l'instruction On Error GoTo pourrait être la suivante :

Sub Programme()

Début du programme

On Error GoTo Etiquette1

Ligne pouvant générer une erreur

A faire seulement quand il n'y a pas d'erreur

Etiquette2:

On Error GoTo 0

Suite du programme

Exit Sub

Etiquette1:

A faire en cas d'erreur

Resume Etiquette2

End Sub

Auteur : Francis Malherbe