Visual Basic for Applications

NOTIONS DE BASES

Par Frédéric GRIMAUD
Ecole des Mines de Saint-Etienne

Présentation de Visual Basic Editor

Visual Basic Editor (VBE) est l'environnement de développement intégré de VBA (Visual Basic for Application). On accède toujours à VBE à partir d'une application hôte en sélectionnant la séquence Outils/Macro/Visual Basic Editor (ou le raccourci Alt-F11). Dans cet environnement VBE, on retrouve :

- **L'explorateur de projet** : il permet de visualiser les différents projets et les différents éléments qui les composent (Objets, Modules, Modules de Classe, Feuille, …) sur lesquels nous reviendrons uttérieurement et également d'accéder à ces éléments et au code qui leur est attaché.
- **La fenêtre Propriétés** : elle permet de visualiser et de modifier l'ensemble des propriétés des objets constituant le projet
- **La fenêtre Code** : elle permet d'éditer le code associé aux éléments du projet
- **La fenêtre UserForm et la Boîte à Outils** : elle permet de concevoir les feuilles (Interfaces graphiques) à partir d'éléments de base (bouton, case à cocher, zone de liste modifiable, …) disponible dans la boîte à Outils.

Ces différents éléments ne sont pas obligatoirement présents à l'écran simultanément. Il est cependant conseillé de garder constamment à l'écran l'explorateur de projet, à partir duquel vous pourrez toujours activer les autres outils.

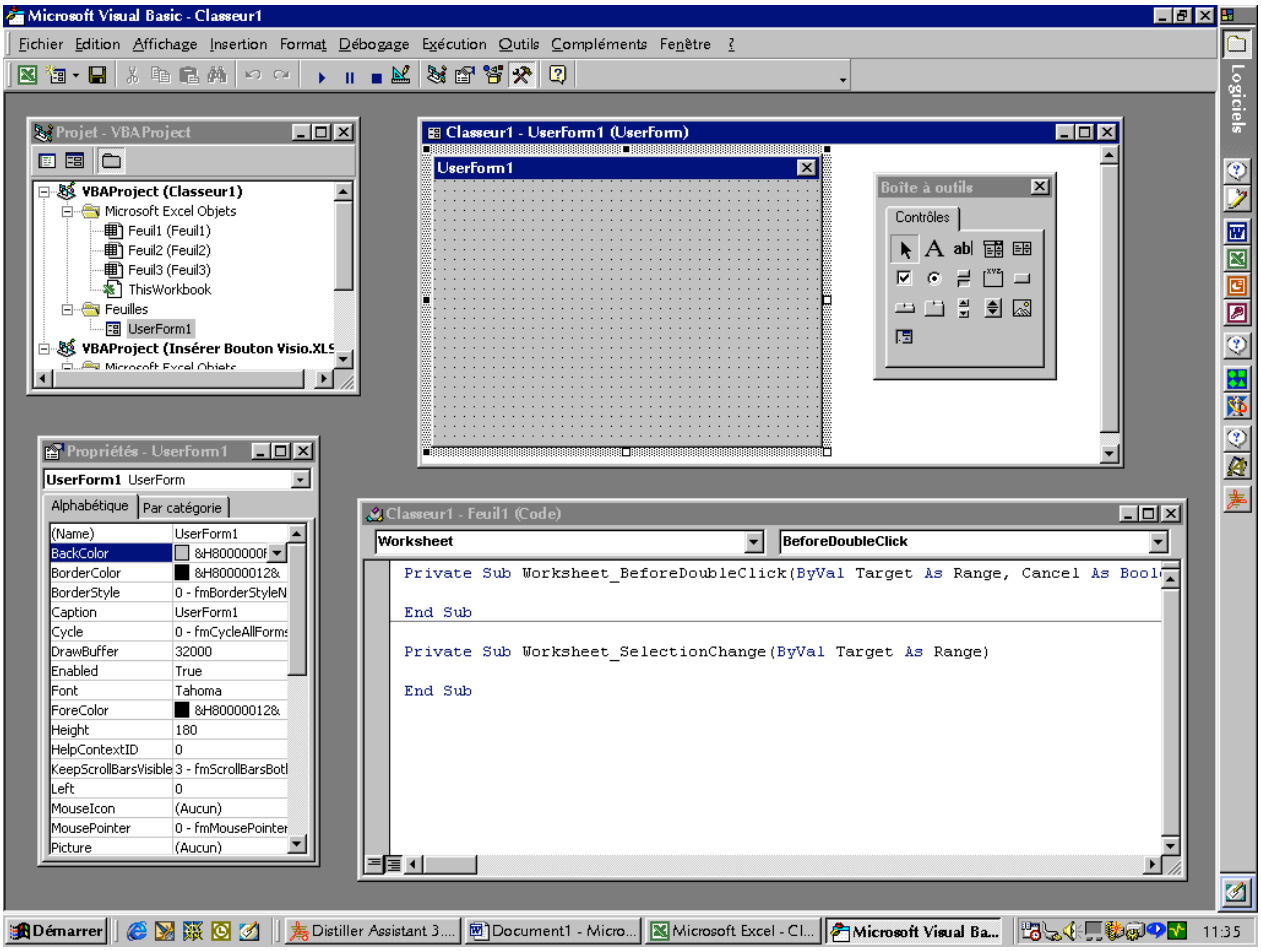


Figure 1 : L'environnement de développement VBE (Visual Basic Editor)

Introduction au langage de programmation VBA

1. Les différents types de Variables et Constantes

Pour créer une variable, vous devez la déclarer. Cette déclaration peut être explicite ou implicite (il est conseillé de rendre les déclaration de variables explicite en commençant la saisie du code d'un Module, d'un Module de Classe ou d'une Feuille par l'instruction **Option Explicit**. Le nom d'une variable ne doit pas commencer par une letttre, peut contenir jusqu'à 255 caractères sans y inclure d'espace ni de mot réservé VBA ni de . ! @ & $ #

- Les Chaînes de caractères

Pour déclarer une variable de type Chaîne de caractères, il faut utiliser la séquence « **Dim** NomVariable **as String** ». L'affectation d'une chaîne de caractère à une variable de type String se réalise en placant cette chaîne de caractère entre guillemets (‘’ ‘’). L'opérateur de concaténation de chaîne de caractère est le mot réservé **&**.

- Les Valeurs numériques

Pour déclarer une variable numérique, il faut utiliser la séquence « **Dim** *NomVariable* **as** *Type » ou Type* peut prendre les valeurs décrites dans le tableau suivant :

Type de données	Valeurs acceptées	Mémoire occupée
Byte	Nombre entier, compris ente 0 et 255	1 octet
Integer	Nombre entier, compris entre -32768 et 32767	2 octet
Long	Nombre entier, compris entre -2147483648 et 2147483647	4 octet
Single	Nombre réel compris entre -3,4 E+38 et 3,4 E +38	4 octet
Double	Nombre réel compris entre -1,8 E+308 et 1,8 E+308	8 octet

Notons que le séparateur décimal dans VBA est le point. Les opérateurs arithmétiques utilisables sur les variables numériques sont décrits dans le tableau suivant :

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
\	Division entière : seule la partie entière du résultat de la division est renvoyée
^	Elévation à la puissance

Les opérateurs relationnels utilisables sur les variables numériques sont décrits dans le tableau suivant :

Opérateur	Description
=	Egal à
>	Supérieur à
<	Inférieur à
<>	Différent de
>=	Supérieur ou égal à
<=	Inférieur ou égal à

Notons que VBA intègre de nombreuses fonctions permettant de manipuler les valeurs numériques, comme par exemple les fonction **Abs** et **Sqr** qui renvoient respectivement la valeur absolue et la racine carrée. La liste des fonctions VBA est consultable dans l'aide en ligne (Manuel de référence / Fonctions)

- Les valeurs booléennes

Pour déclarer une variable de type Booléenne, il faut utiliser la séquence « **Dim** NomVariable **as Boolean** ». Une variable de type Booléenne peut prendre la valeur true (1) ou False (0) et occupe 2 octets en mémoire.

Les opérateurs logiques utilisables sur les variables booléennes sont décrits dans le tableau suivant :

Opérateur	Description
Or	Ou
And	Et
Xor	Ou Exclusif
Not	Non

- Les Dates

Pour déclarer une variable de type Date, il faut utiliser la séquence « **Dim** NomVariable **as Date** ». Une variable de type Date peut prendre une valeur comprise ente le 01/01/100 et le 31/12/9999 et occupe 8 octets en mémoire sous forme d'un nombre réel. La partie entière de la variable de type Date représente le jour et la valeur 0 correspond au 30/12/1899.
Notons que VBA intègre de nombreuses fonctions permettant de manipuler les dates, par exemple les fonctions **Date**, **Time** et **Now** renvient respectivement la date courante, l'heure courante et la date courante suivie de l'heure courante.

- Les tableaux

Pour déclarer une variable de type tableau, il faut utiliser la séquence « **Dim** NomVariable**(***Nbr1, Nbr2, …***) as** *Type* » ou la séquence « **Dim** NomVariable**(***Début1* **to** *Fin1, Début2* **to** *Fin2, …* **) as** *Type* » où *Nbri* réprésente la valeur de la dimension i du tableau et où Début i et Fin i correspondent au indice de début et de fin pour la dimension i (alors égale à Fin i - Début i + 1).

- Les type de données personnalisés

Pour déclarer une variable de type Type de données personnalisés, il faut utiliser la séquence suivante dans la partie Déclaration d'un module :

Type NomType
Données1 **as** *Type 1*
Données2 **as** *Type 2*
…
End Type

On peut alors déclarer une variable de ce nouveau type avec la séquence classique **Dim** *NomVariable* **as** *NomType*

- Les constantes

Les constantes permettent d'attribuer un nom à une valeur fixe qui ne pourra donc pas évoluer pendant le déroulement du programme. Pour déclarer une constante, il faut utiliser la séquence « **Const** *NomConstante* **as** *Type* **=** *Valeur* ». Pour déclarer une constante accessible à toutes les procédures d'un module, il faut la déclarer dans la partie Déclaration d'un module.

2. La portée et la durée de vie des Variables et des Constantes

En plus de leur type et de leur valeur, les variables et les constantes sont caractérisées par leur portée. Elle désigne son accessibilité pour les procédures et les modules du projet. Les variables et les constantes peuvent être accessibles :

- Uniquement à l'intérieur d'une procédure : c'est le cas si la déclaration est effectuée à l'intérieur de la procédure
- à l'ensemble des procédures d'un module : c'est le cas si la déclaration est effectuée dans la partie Déclaration d'un Module, c'est-à-dire à l'extérieur de toute procédure, avec les séquences « **Dim** *NomVariable* **as** *Type* » ou « **Private** *NomVariable* **as** *Type* »
- ou encore à l'ensemble des modules du projet en cours : : c'est le cas si la déclaration est effectuée dans la partie Déclaration d'un Module, c'est-à-dire à l'extérieur de toute procédure, avec la séquence « **Public** *NomVariable* **as** *Type* »

La durée de vie d'une variable désigne le temps pendant lequel la variable conserve sa valeur. Une variable peut :

- conserver sa valeur tant qu'une procédure s'exécute et être réinitialisée lorsque la procédure est terminée : c'est la cas si l'on utilise la séquence classique « **Dim** *NomVariable* **as** *type* »
- ou conserver sa valeur entre les différents appels de la procédure : c'est le cas si l'on utilise la séquence « **Static** *NomVariable* **as** *Type* »

3. Les structures de contrôle

- La boucle While…Wend

La structure While…Wend permet de répéter une série d'instruction tant que la condition spécifiée est remplie. La syntaxe est :

While *Condition*
    Série d'instructions
Wend

- La boucle Do … Loop et ses 4 variantes

La structure **Do While … Loop** permet de répéter une série d'instruction tant que la condition spécifiée est remplie. La syntaxe est :

Do While *Condition*
    Série d'instructions
Loop

La structure **Do Until … Loop** permet de répéter une série d'instruction jusqu'à ce que la condition spécifiée soit remplie. La syntaxe est :

Do Until *Condition*
    Série d'instructions
Loop

La structure **Do … Loop While** permet d'exécuter une série d'instructions puis de la répéter tant que la condition spécifiée est remplie. La syntaxe est :

Do
    Série d'instructions
Loop While *Condition*

La structure **Do … Loop Until** permet d'exécuter une série d'instructions puis de la répéter jusqu'à ce que la condition spécifiée soit remplie. La syntaxe est :

Do Until
    Série d'instructions
Loop Until *Condition*

- La boucle For … Next

La structure **For** … **Next** permet de répéter une série d'instruction un nombre de fois déterminée dans le code, en utilisant un compteur. La syntaxe est :

For *compteur* **=** *x* **to** *y* **Step** *Pas*
    Série d'instructions
Next *compteur*

Cette séquence permet d'exécuter en boucle la série d'instruction spécifiée entre **For** et **Next** en incrémentant la variable *compteur* de la valeur de *Pas* à chaque itération.

4. Les strutures conditionnelles

- La structure If … Then … Else

La syntaxe est :

If *Condition 1* **Then**
    Série d'instructions 1
[**ElseIf** *Condition 2*
    Série d'instructions 2
ElseIf *Condition 3*
    Série d'instructions 3
…
Else
    Série d'instructions n]
EndIf

La partie entre crochet [] n'est pas obligatoire. Les conditions sont évaluées dans l'ordre d'apparition à l'intérieur de la structure.

- La structure Select Case

La structure Select case permet d'envisager différentes valeurs pour une même expression et de déterminer des instructions spécifiques pour chaque cas envisagé. La syntaxe est :

Select Case *Expression*
Case *Valeur 1*
Série d'instructions 1
Case *Valeur 2*
Série d'instructions 2
…
Case *Valeur n*
Série d'instructions n-1
[**Case Else**
Série d'instructions]
End Select

Lorsque la valeur renvoyé par *Expression* correspond à l'une des valeurs proposées derrière les **Case**, les instructions correspondants sont exécutés et la procédure se poursuit avec l'instruction qui suit le **End Select**. Sinon, les instructions attachées à **Case Else** sont exécutées.

5. La structure des programmes Visual Basic pour Application

La notion de Modules

Comme il est précisé dans le § précédent, on distingue dans un projet VBA les modules de Code, les modules de Classe et les Feuilles. Ces éléments sont appelés à interagir pour constituer le programme complet :

- Le code standard se trouve dans des modules de codes, stockés dans le dossier Modules.
- Le code décrivant les objets développés pour votre projet est stocké dans le dossier Modules de Classe
- Le code décrivant l'interface d'un programme et le code affecté aux différents événements associés à cette interface sont stockés dans des fichiers UserForms, eux même stockés dans le dossier Feuilles.

La notion de Procédure

A l'intérieur d'un même module, le code est structuré en procédures. Une procédure est une séquence d'instructions. On qualifie de procédure événementielle une procédure déclenchée par un événement (clic de souris, frappe d'une touche du clavier, ..) par opposition aux procédures standards indépendantes de tout événement de l'interface utilisateur :

- Dans les dossiers Feuilles, les procédures sont déterminées : il existe une procédure pour chaque événement pouvant affecter un élément de l'interface utilisateur. Il s'agit alors uniquement de créer le code à exécuter lors du déclenchement de cet événement.
- Dans les dossiers Modules ou Modules de Classe, c'est au concepteur du code de créer les différentes procédures. Il s'agit alors de créer la structure de la procédure (Procédure, Fonction, …) et de créer le code à exécuter lors de l'appel à cette dernière.

Construction d'une interface Utilisateur

Les feuilles sont les zones sur lesquelles nous allons placer des Contrôles (Case à cocher, zones de texte, …). Ces contrôles placés sur une feuille forment alors une interface graphique permettant une itéraction simple et intuitive entre l'utilisateur final et le programme.

Comme indiqué précédemment, les événements utilisateurs (clic de souris, …) sont automatiquement détectés par le programme sui exécute alors le code associé à cet événement. On parle de procédure événementielle.

Les phases de développement des feuilles sont :

1. Détermination des besoins : Quelles fonctions doit avoir cette interface ? De quels contrôles doit-elle être composée ? Comment seront-ils organisés sur la feuille ? A quels événements utilisateurs doivent-ils répondre ?
2. Création visuelle de la feuille : Pendant cette phase, il faut placer les contrôles voulus sur la feuille en mode création dans le VBE et les paramétrer en mettant à jour les valeurs dans la fenêtre Propriétés.
3. Ecriture du code attaché à cette feuille : Pendant cette phase, on détermine le comportement de la feuille face aux différents événements utilisateurs

1. Les composants de la boîte à outils

La boîte à outils contient les contrôles que vous pouvez placer sur votre feuille. Au même titre que la feuille elle-même , les contrôles sont des objets, avec des méthodes et propriétés. Pour pouvoir utiliser ces contrôles, vous devez uniquement connaître ces méthodes et propriétés.

- Le contrôle Label

Le contrôle Label permet de placé un intitulé informatif sur une feuille

- Le contrôle textBox

Le contrôle textBox permet de placer une zone de texte sur la feuille, dans laquelle l'utilisateur pourra saisir des informations

- Le contrôle ComboBox

Le contrôle ComboBox permet de placer une zone de liste modifiable sur la feuille, permettant à l'utilisateur de saisir une valeur manuellement ou de la sélectionner dans une liste. La saisie manuelle peut être désactivée.

- Le contrôle Frame

Le contrôle Frame permet de placer un cadre présentant un intitulé. Ce cadre pourra à son tour recevoir des contrôles

- Le contrôle ListBox

Le contrôle ListBox permet de placer une zone de liste non modifiables, dans laquelle l'utilisateur pourra saisir une ou plusieurs valeurs.

- Le contrôle CheckBox

Le contrôle CheckBox permet de placer une case à cocher qui peut être alors activée ou désactivée par l'utilisateur

- Le contrôle ToggleButton

Idem Contrôle CheckBox avec une présentation graphique différente.

- Le contrôle OptionButton

Le contrôle OptionButton permet de proposer à l'utilisateur un choix parmi plusieurs options. Si plusieurs contrôles OptionButton sont associés (en les mettant dans un contrôle Frame par exemple ou avec la propriété GroupName), un seul choix est possible parmi toutes les options proposées.

- Le contrôle CommandButton

Le contrôle CommandButton permet de proposer un bouton de commande à l'utilisateur.

- Le contrôle TabStrip

Le contrôle TabStrip permet de mettre en place un ensemble de page géré par onglet

- Le contrôle ScrollBar

Le contrôle ScrollBar permet de mettre en place une barre de défilement .

- Le contrôle SpinButton

Le contrôle SpinButton permet de mettre en place un bouton « Toupie » composé de 2 flèches

2. Quelques propriétés de ces contrôles

- La propriété Name

Elle correspond au nom de l'objet. Ce nom doit être utilisé pour faire référence à l'objet dans le code. Un même nom ne peut être utilisé pour plusieurs objets d'une même feuille.

- La propriété Caption

Elle correspond au texte descriptif du contrôle. Cela correspond par exemple au texte qui s'affiche à l'intérieur d'un bouton, ou en haut à gauche d'une feuille par exemple.

- La propriété Value

Elle correspond à la valeur du contrôle. Cela peut varier en fonction du contrôle choisi, par exemple la propriété value d'un contrôle CheckBox correspond à un booléen qui est à Vrai si la case est cochée, à Faux sinon. La liste des valeurs prises la propriété Value en fonction des types de contrôles est donnée dans le tableau suivant :

Contrôle	Valeur acceptée par la propriété Value
CheckBox, OptionButton, ToggleButton	Valeur de type Boolean indiquant l'état du contrôle : True s'il est activé, et False sinon
TextBox	Valeur de type String représentant le texte dans la zone d'édition du contrôle
ComboBox ListBox	Valeur représentant l'élément selectionné dans la liste des éléments du contrôle
ScrollBar SpinButton	Valeur de type Integer comprise entre les valeurs des propriétés Min et Max du contrôle

- La propriété Visible

Elle correspond à la visibilité d'un contrôle par l'utilisateur ou non (True = Visible, False = Invisible)

- La propriété Enabled

Elle correspond à l'accessibilité (écrire un texte dans une TextBox, cocher une case, …) d'un contrôle par l'utilisateur ou non (True = Accessible, False = Inaccessible). L

- La propriété Locked

Elle correspond à l'autorisation ou non d'une modification d'un contrôle (écrire un texte dans une TextBox, cocher une case, …) par l'utilisateur ou non (True = Modifiable, False = Non Modifiable).

3. Création des procédures événementielles

Les contrôles placés sur une feuille sont réceptifs aux événements utilisateur qui les affectent. Nous allons apprendre à créer des procédure dites événementielles, qui se déclencheront lorsqu l'événement correspondant sera repéré.
Une procédure événementielle doit être décrite directement dans la fenêtre de code de la Feuille, en sélectionnant une des méthodes proposées

- Les événements

Les événements sont nombreux et varient selon les contrôles. Pour accéder à la liste des événements gérés par un type de contrôle, sélectionnez le contrôle en question sur une feuille et tapez sur la touche F1 pour ouvrir la rubrique d'aide associée. Sélectionnez alors Evéenements pour les afficher. Nous avons sélectionné les plus couramment utilisé et les définitions brièvement :

- Evénement **AfterUpdate** : Détecté lorsque la valeur du contrôle est modifié suite à un changement de focus (un autre contrôle est sélectionné).
- Evénement **Change** : Détecté lors de chaque modification de la valeur d'un contrôle.
- Evénement **Click** : Détecté lorsque l'utilisateur clique sur un contrôle ou que l'équivalent clavier est effectué.
- Evénement **DblClick** : Détecté lorsque l'utilisateur double-clique sur un contrôle
- Evénement **KeyPress** : Détecté lorsqu'une touche du clavier est enfoncé

- Acces à une procédure événementielle et saisie le code

Pour accéder à une procédure événementielle d'un contrôle, double-cliquez sur le contrôle, et sélectionnez dans la liste déroulante en haut à droite de la fenêtre qui s'est ouverte l'événement qui vous voulez maîtriser. Saisissez alors le code correspondant dans le masque la procédure généré automatiquement comme par exemple

```
Private Sub CommandButton1_Click()

End Sub
```

pour l'événement Click du bouton ayant CommandButton1 comme Propriété Name dans mon projet.

Le code est composé d'instructions telles que présentés dans le chapitre 2 et d'appels à des procédures et fonctions décrites dans les Modules et/ou à des objets décrits dans les Modules de Classes

Construction et Utilisation d'un Module

Un module est composé de définitions de Constantes, de Variables (locales au module ou non) et de procédures créées par l'utilisateur. Nous nous intéressons dans ce chapitre à construire des procédures.

1. Les procédures Sub

Une procédure Sub est une série d'instructions exécutant une tâche déterminée au sein du projet, sans renvoyer de valeur. La syntaxe est :

[**Private** | **Public**] [**Static**] **Sub** *NomDeLaProcédure*([**Arguments**])
Déclaration de variables locales
Série d'instructions
End Sub

Private ou **Public** (par défaut) indique s'il s'agit d'une procédure privée ou publique :

- Une procédure publique peut être invoquée par n'importe quelle procédure du projet, y compris celles stockées dans d'autres modules.
- Une procédure privée ne peut être invoquée qu'à partir d'une procédure stockée dans le même module.

L'option **Static** indique que les variables locales de la procédure Sub conservent leurs valeurs entre les différents appels de la procédure.

La sortie d'une procédure avant la fin de son exécution peut être activée en plaçant le mot-clé **Exit Sub** dans la série d'instruction

2. Les procédures Function

Une procédure Function est une série d'instructions exécutant une tâche déterminée au sein du projet, et renvoyant une valeur, qui pourra être exploité par d'autres procédures. La syntaxe est :

[**Private** | **Public**] [**Static**] **Function** *NomDeLaProcédure*([**Arguments**])
Déclaration de variables locales
Série d'instructions
…
NomDeLaProcédure = *Expression*
…
End Function

NomDeLaProcédure = *Expression* permet d'affecter une valeur à la fonction. C'est cette valeur qui sera renvoyée lorsque l'exécution de la fonction sera terminée.

Les mot-sclés **Private** ou **Public** (par défaut) indique s'il s'agit d'une procédure privée ou publique :

- Une procédure publique peut être invoquée par n'importe quelle procédure du projet, y compris celles stockées dans d'autres modules.
- Une procédure privée ne peut être invoquée qu'à partir d'une procédure stockée dans le même module.

L'option **Static** indique que les variables locales à la procédure Function conservent leurs valeurs entre les différents appels de la procédure.

La sortie d'une procédure Function avant la fin de son exécution peut être activée en plaçant le mot-clé **Exit Function** dans la série d'instruction

3. Les arguments des procédures Sub ou Function

Arguments représente les arguments passés à la procédure **Sub** ou **Function** par la procédure appelante. La virgule sert de séparateur entre les différentes valeurs transmises.
Chacun des arguments de *Arguments* répond à la syntaxe suivante :

[**Optional**] [**ByVal** | **ByRef**] *NomVariable* [**As** *Type*] [= *ValeurParDéfaut*]

Les signification des mots-clé sont donnés par le tableau ci-dessous :

Elément	Description
Optional	ce mot-clé indique que les arguments transmis sont facultatifs (Les arguments facultatifs doivent être placé en fin de liste des arguments
ByVal	L'argument est passé par Valeur. C'est la valeur de la variable et non son adresse qui est transmise à la procédure appelée. Autrement dit, la valeur de la variable passée est exploitée par la procédure appelée mais ne peut être modifiée.
ByRef	L'argument est passé par Référence. C'est l'adresse de la variable et non sa valeur qui est transmise à la procédure appelée. Autrement dit, la procédure peut modifier la valeur de cette variable. (Passage de paramètre par défaut).
As	Le type de donnée de l'argument passé est spécifié à la procédure
= ValeurParDéfaut	ce mot-clé indique une valeur par défaut pour l'argument si **Optional** a été utilisé pour cet argument.

4. Utilisation d'un module

Appeler une procédure consiste à demander à une procédure de s'exécuter à partir d'une autre procédure. La procédure appelée est exécutée et la procédure appelante reprend la main.

- L'appel d'une procédure Sub

Pour appeler une procédure de type Sub, il faut utiliser la syntaxe « **Call** NomProcédure »
Evidemment, pour pouvoir appeler une procédure Sub, il faut que cette procédure soit visible pour la procédure appelante.

- L'appel d'une procédure Function

Pour appeler une procédure de type Function, il suffit de placer le nom de la la procédure dans une expression à l'emplacement ou une valeur est attendue.
Evidemment, pour pouvoir appeler une procédure Function, il faut que cette procédure soit visible pour la procédure appelante.

- Le passage des arguments

Pour passer les arguments à une procédure, il faut que les arguments coincident en terme de Type de Donnée et soit déclarés dans le même ordre que lors de la déclaration de la procédure dans le module.

Traitements intra-application et inter-applications

1. Le modèle Objet d'Excel

Voir documentation en ligne de Microsoft Office - Rubrique « Objets Microsoft Excel » pour connaître le modèle objet de Excel.

- Accéder au modèle objet d'Excel dans un module définit dans VBA Excel

1. Il existe un pointeur appelé ThisWorkBook qui permet d'atteindre les objets composant le fichier .xls associé au projet courant.
2. A partir de cette référence, on accède par exemple à la valeur de la case A1 de la feuille Feuil1 par la séquence
   ThisWorkBook.WorkSheet("Feuil1").Range("A1").Value

Ensuite, il faut maîtriser le modèle objet pour travailler utilement. Par exemple, l'objet Application.WorksheetFunction permet d'accéder à l'ensemble des fonctions prédéfinies dans Excel.

- **Accéder au modèle objet d'Excel dans un module définit dans une autre application (VBA Compliant) qu'Excel**

Une variable objet peut être affectée à un projet d'une application autre que l'application hôte du projet (pour exploiter par exemple des données issues d'une feuille de calcul Excel dans un programme VBA Word.
Pour qu'un projet puisse accéder à la bibliothèque d'objets d'une autre application, celle-ci doit être référencé dans le projet.

0. Ne pas oubliez de référencer la bibliothèque d'objets Excel dans le projet en cours
   Pour référencer une bibliothèque d'objets, il faut choisir les options Outils/Références, puis choisir la bibliothèque à référencer en la cochant comme montré dans la figure 2.

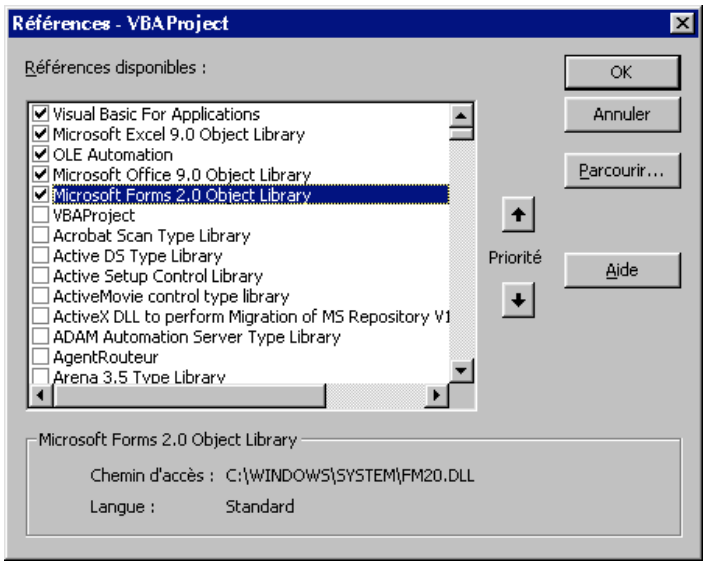


Figure 2 : Les Références du projet

1. Déclarez les variables objet qui feront référence aux objets de l'autre application en tant que types spécifiques. Veillez à qualifier chaque type à l'aide du nom de l'application dont l'objet est issu. Par exemple, l'instruction suivante déclare une variable pointant un classeur Microsoft Excel :
   ```
   Dim wbXL As Excel.Workbook
   ```
2. Utilisez la fonction CreateObject avec l'identificateur de programmation OLE de l'objet sur lequel vous voulez travailler dans l'autre application, comme indiqué dans l'exemple suivant. Pour voir la session de l'autre application, affectez la valeur **True** à la propriété **Visible**.
   ```
   Dim appXL As Excel.Application
   ```

```
Set appXL = CreateObject("Excel.Application")
appXL.Visible = True
```

3. Appliquez les propriétés et les méthodes à l'objet contenu dans la variable
4. Lorsque vous avez terminé de travailler dans l'autre application, utilisez la méthode **Quit** pour la fermer, comme indiqué dans l'exemple suivant.

```
appXL.Quit
```

2. Le modèle objet Word

Le principe est le même que pour Excel, en remplacant Excel par Word dans les déclarations de objets concernés (Word.Application par exemple).

Ensuite, il faut maîtriser le modèle objet pour travailler utilement …

3. Le modèle objet Direct Access Object DAO 3.6

Les DAO (objets d'accès aux données) vous permettent d'utiliser un langage de programmation pour extraire et manipuler des données dans des bases de données locales ou distantes basées sur le moteur de base de données Microsoft Jet. Ils vous permettent également de gérer des bases de données (notamment Acces puisque cet outil est basé du Microsoft Jet) , ainsi que les objets et la structure de ces dernières.

- Accéder au modèle objet d'une base de données supportant les DAO dans un module VBA

0. Ne pas oubliez de référencer la bibliothèque d'objets DAO 3.5 (Direct Access Object) dans le projet en cours
1. Appliquez les propriétés et les méthodes à l'objet contenu dans la variable
   Dans l'exemple ci-dessous on accède au fichier MaBase.mdb, et on consulte les enregistrements de la table MaTable. Les champs de la table MaTable sont MonChamp1 et MonChamp2. Lorsque la valeur de MonChamp1 est 1 alors on met la chaine de caractère ''E'' dans le champ MonChamp2.

```
Public Mydb As Database                          'Pointeur vers une base de données
Public MyContenu As Recordset                    ' Pointeur vers une table d'une base de données

Set Mydb = OpenDatabase("MaBase")                ' Pointer vers la base de donnée MaBase durépertoire courant
Set MyContenu = Mydb.OpenRecordset("MaTable")    ' Pointer  la table MaTable
MyContenu.MoveFirst                              ' Se positionner sur le premier enregistrement de la table
Do Until MyContenu.EOF                           ' Tant qu'il y a des enregistrement, pour l'enr courant
If MyContenu![Champ1] = 1 Then
    MyContenu.Edit                               ' On accède en écriture au Champ2 de l'enregistrement courant
    MyContenu![Champ2] = "E"
    MyContenu.Update                             ' On modifie l'enregistrement courant
End If
MyContenu.MoveNext                               ' On passé à l'enregistrement suivant si il existe
Loop
```

4. Le modèle objet Acces

Le principe est le même que pour Excel, en remplacant Excel par Access dans les déclarations de objets concernés (Acces.Application par exemple).

Ensuite, il faut maîtriser le modèle objet pour travailler utilement …

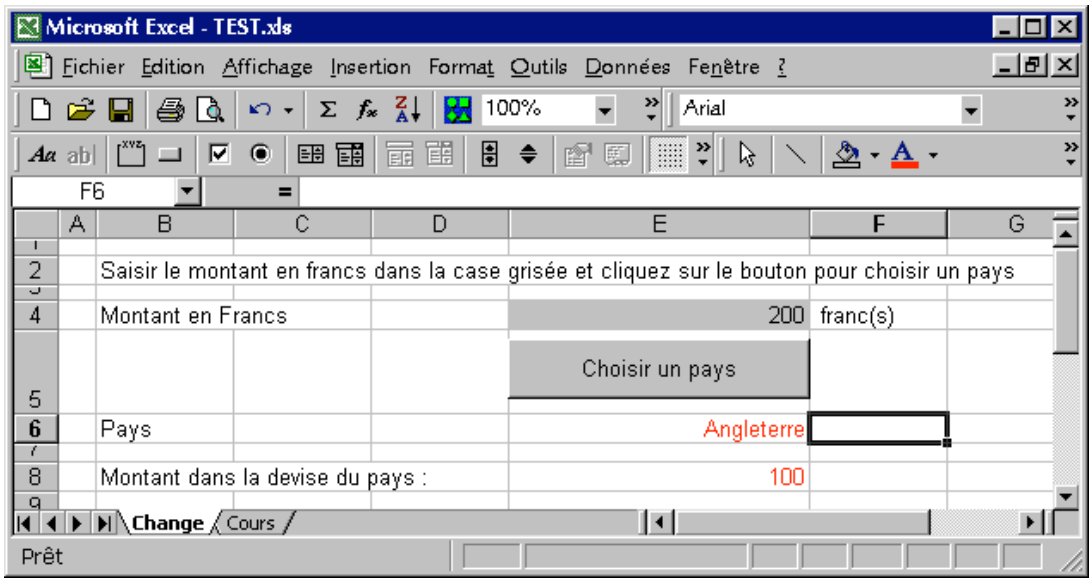
Etude de cas

Nous souhaitons mettre en place un outils EXCEL dans lequel un utilisateur non-expert en informatique doit etre capable de faire des calculs de conversion entre différentes monnaies. Nous allons détaillé dans la suite de cet exposé, la construction d'un tel outil basé sur les concepts définis dans la cours VBA.

1. Création du fichier EXCEL

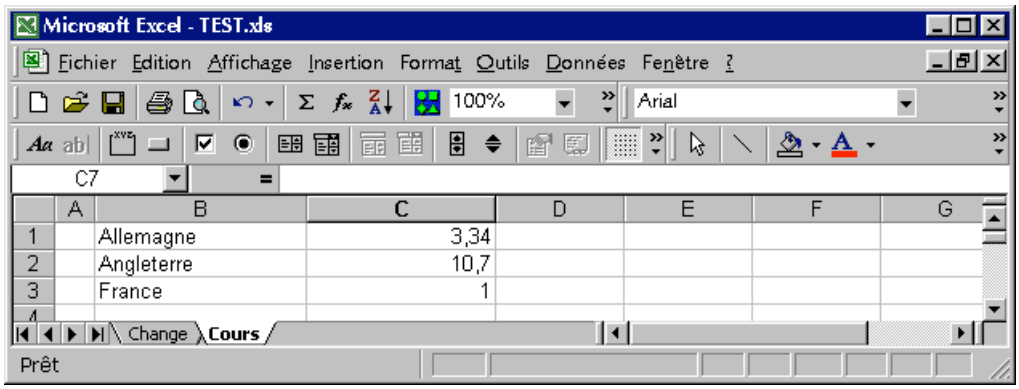
- Création d'une feuille nommée « Change »

Dans la case B4, saisissez « Montant en Francs », dans la case B6 saisir « Pays », et dans la case B8 saisissez « Montant en devise du pays sélectionné ». Grisez la case E4 pour indiquer à l'utilisateur l'endroit où il doit saisir la valeur en francs. Insérez un bouton que vous placerez et nommerez comme indiqué sur la figure ci-dessous.



- Création d'une feuille nommée « Cours »

Dans la case B1, saisissez Allemagne, et dans la case C1, saisir la valeur de 1 DM en francs. Dans la case B2, saisissez Angleterre, et dans la case C2, saisir la valeur de 1 £ en francs. Répétez l'opération précédente pour vous constituer un jeu d'essai suffisamment représentatif.



Enregistrer votre fichier EXCEL dans un répertoire TEST sous le nom TEST.XLS

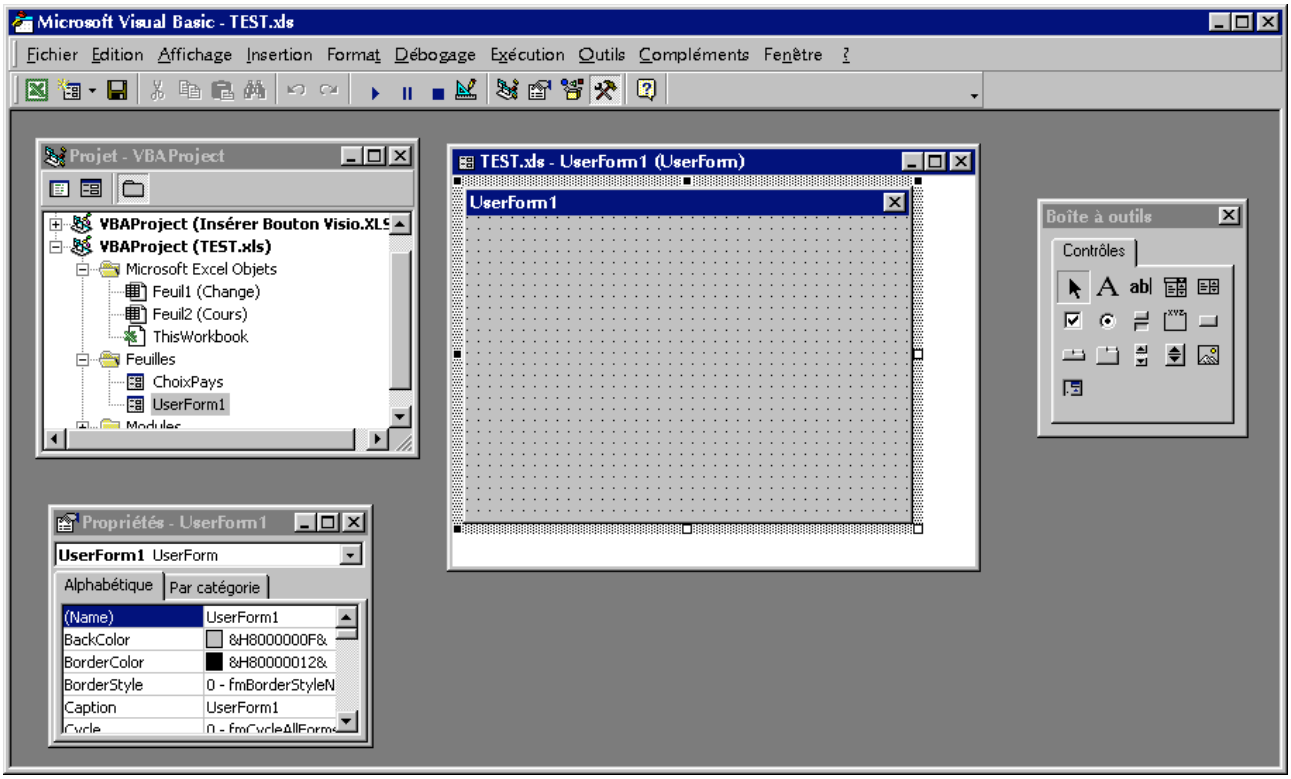
2. Construction de l'interface graphique de choix du pays

- Ouverture de VBE

Ouvrir VBE avec les menus Outils/Macros/Visual Basic Editor

- Ouverture d'une fenêtre graphique et placement des objets d'interface

Sélectionnez le menu Insertion/UserForm pour insérer une nouvelle UserForm dans notre projet. Rappelons que UserForm correspond à une fenêtre de notre interface graphique. Une nouvelle fenêtre apparaît, avec une fenetre grisée vide de tout objet. C'est dans cette fenêtre que nous allons définir notre interface graphique.

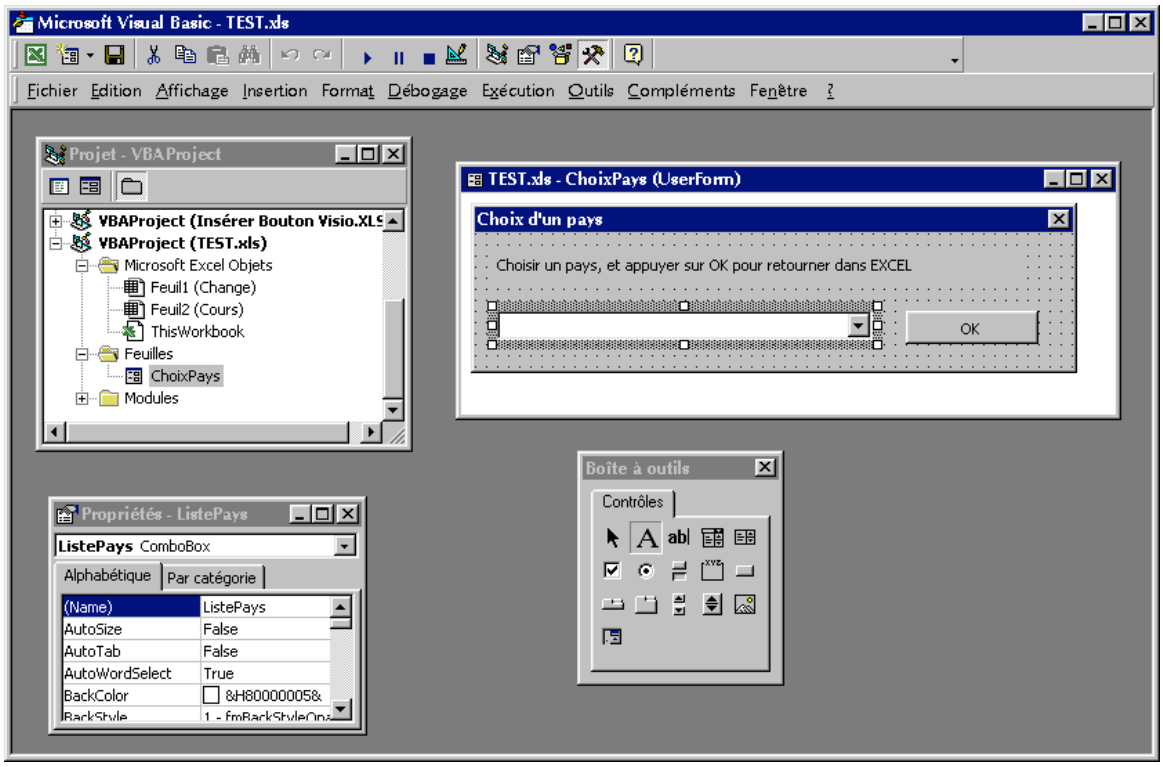


Nommons notre Userform avec un nom plus significatif. A la place de UserForm1, sur la ligne (Name) de la fenêtre Propriétés, saisissez ChoixPays. Votre UserForm s'appelle donc maintenant ChoixPays.

Plaçons un objet Bouton dans notre UserForm ChoixPays . Cliquez dans la boite à outils sur le contrôle Bouton, et faites un Drag and Drop sur la fenêtre ChoixPays. Viens se placer un bouton s'appelant CommandButton1. De la même manière que pour le nom de la Userform1, nous allons changer la valeur de (Name) pour le CommandButton1, en remplaçant ce terme dans le fenêtre Propriétés par Btn_OK. Pour faire apparaître OK comme message écrit à l'intérieur du bouton, sélectionnez la propriété Caption de l'objet Btn_OK, et saisissez OK comme valeur de cette propriété.

Remarquez que lorsque vous sélectionnez un objet de votre environnement VBA, les propriétés correspondantes à cet objet apparaissent dans la fenêtre propriétés.

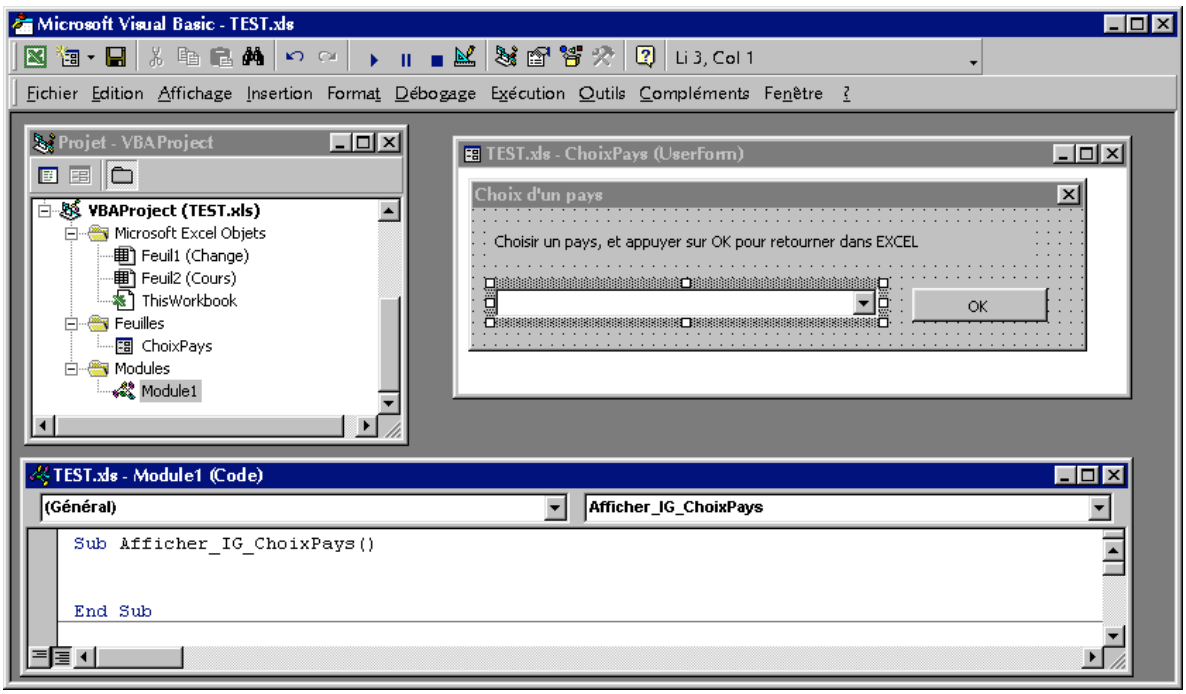
Placez maintenant un objet Zone de liste modifiable. Appelez cet objet ListePays. Saisissez dans la propriétés BoundColumn la valeur 0, indiquant que le résultat renvoyé par la propriété Value, sera l'index du pays choisi dans l'ordre d'insertion dans la liste. Placez ensuite un objet Intitulé, et saisissez dans la propriété Caption, le texte « Choisir un Pays, et cliquez sur OK pour revenir à Excel ». Ajustez les tailles de ces objets et de la fenêtre pour obtenir une configuration équivalent à la figure 4.



3. Construction de la Macro d'activation de cette fenêtre graphique dans Excel

Il nous faut maintenant activer l'affichage de cette interface graphique ChoixPays lorsque l'utilisateur clique sur le bouton « Choisir un Pays » de la feuille de donnée » Change ». Pour y parvenir, nous allons créer une procédure Sub sans paramètre que nous allons stocker dans un Module pour qu'elle soit accessible depuis n'importe quel endroit de notre projet.

Insérons un nouveau module, en sélectionnant les menus Insertion/Module de l'environnement VBA. Une nouvelle fenêtre apparaît. C'est dans cette fenêtre que nous allons créer notre procédure Afficher_IG_ChoixPays. Saisissez le texte Public Sub Afficher_IG_ChoixPays dans cette nouvelle fenetre et appuyez sur la touche Entrée de votre clavier. Automatiquement, apparaît la structure vide de votre nouvelle procédure, comme le montre la figure 5



Il ne vous reste plus qu'à insérer le code de votre procédure entre le Sub et le End Sub. Saisissez le code suivant :

```
Dim i As Integer
ChoixPays.ListePays.Clear
i = 1
Do While ThisWorkbook.Worksheets("Cours").Range("B" & i) <> ""
    ChoixPays.ListePays.AddItem ThisWorkbook.Worksheets("Cours").Range("B" & i)
    i = i + 1
Loop
ChoixPays.Show
```

A quoi correspond ce code ?

L'instruction « Dim i As Integer » permet de déclarer une variable de type entier qui ne sera accessible que dans cette procédure. L'instruction « ChoixPays.ListePays.Clear » permet d'activer la méthode Clear de l'objet Zone de liste modifiable ListePays qui se trouve dans le userform ChoixPays de notre projet. Cette méthode vide le contenu de la zone de liste modifiable. Nous trouvons ensuite une boucle qui scrute les valeurs des cases Bi (avec i initialisé à 1) et qui s'arrêtera lorsque la valeur trouvée sera égale à « vide ». Pour chaque ligne non vide trouvée, on insère grace à la méthode AddItem de l'objet ListePays la valeur de la case Bi. De cette manière, on charge la zone de liste avec les valeurs stockées dans la feuille Cours. Une fois ce chargement effectué, l'instruction « ChoixPays.Show » permet d'activer la méthode Show de l'objet ChoixPays qui fait apparaître à l'écran la fenetre ChoixPays.

La procédure est écrite, il nous reste à la faire exécuter lorsque l'utilisateur clique que le bouton « Choisir un pays » dans la fenêtre « Change ». Il faut revenir dans l'environnement Excel, et cliquez avec le bouton droite de votre souris sur l'objet Bouton. Sélectionnez alors l'option « Affecter une macro ». Apparaît alors une fenetre dans laquelle vous devez retrouver le nom de la procédure que l'on vient d'écrire dans VBA, à savoir Afficher_IG_ChoisPays. Sélectionnez cette procédure et cliquez sur OK.

4. Construction d'une procédure de calcul du change

Il faut maintenant créer la procédure qui va calculer la valeur de la somme en francs dans la devise choisie. Comme précédemment, nous allons créer une nouvelle procédure, qui sera cette fois ci une procédure Function,, qui permet de renvoyer un résultat. Saisissez le code

```
Public Function Calcule(ByVal Montant As Double, ByVal taux As Double) As Double
       Calcule = Montant / taux
End Function
```

Rappelons que ByVal permet de passer un paramètre par valeur. Cette fonction renvoie donc la valeur de Montant dans la devise correspondant au taux de change Taux.

5. Construction de la procédure événementielle « Click sur le bouton Btn_OK »

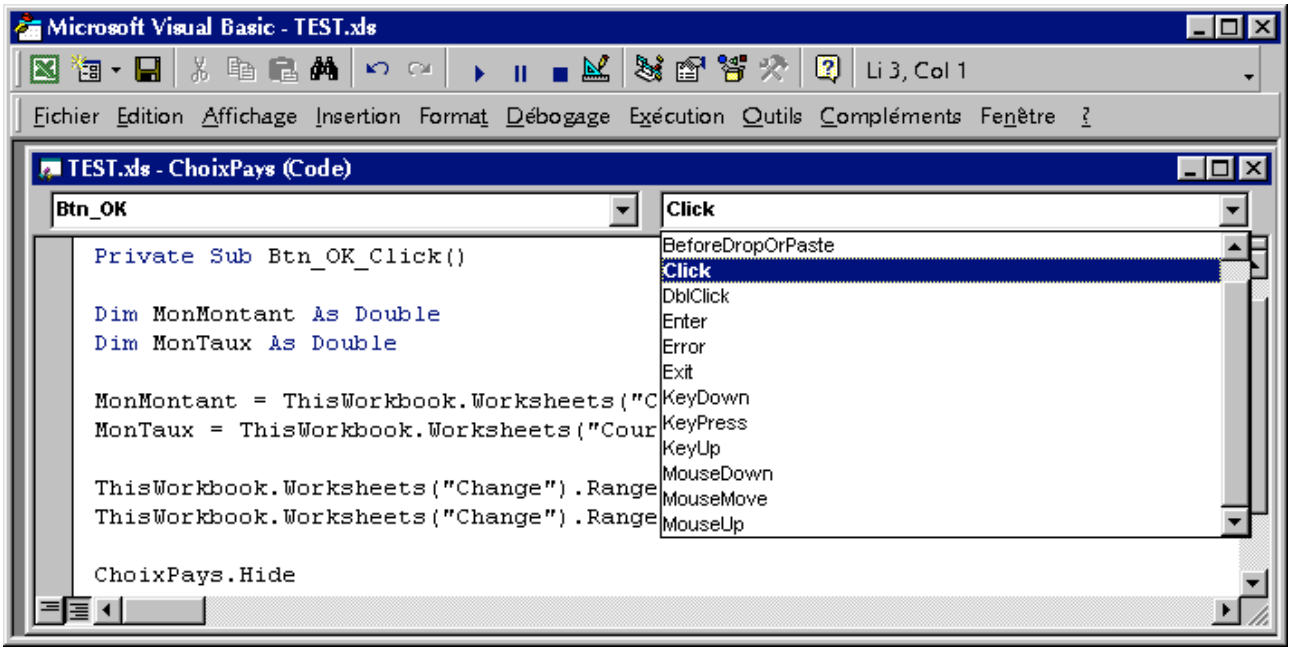
Nous devons maintenant écrire le code correspondant au déclenchement de l'événement Click du bouton Btn_OK. Dans ce code, il nous faudra récupérer les valeurs de la somme à changer, du pays, activer la fonction Calcule et renvoyer le résultat de cette fonction dans les cases Excel appropriées.
Pour trouver la procédure Click du bouton Btn_OK, cliquez avec le bouton droit de la souris sur l'objet Btn_OK et sélectionnez l'option Code. Une fenetre apparaît. Sélectionnez alors la méthode Click et apparaitra automatiquement le code :

```
Private Sub Btn_OK_Click()
End Sub
```

Placez dans cette procédure le code suivant :

```
Dim MonMontant As Double
Dim MonTaux As Double

MonMontant = ThisWorkbook.Worksheets("Change").Range("E4").Value
MonTaux = ThisWorkbook.Worksheets("Cours").Range("C" & (ChoixPays.ListePays.Value +
1)).Value

ThisWorkbook.Worksheets("Change").Range("E6").Value =
ThisWorkbook.Worksheets("Cours").Range("B" & (ChoixPays.ListePays.Value + 1)).Value
ThisWorkbook.Worksheets("Change").Range("E8").Value = Calcule(MonMontant, MonTaux)

ChoixPays.Hide
```

Comme mentionné dans la figure suivante :



A quoi correspond ce code ?

Les instructions « Dim MonMontant As Double » et « Dim MonTaux As Double » permettent de déclarer des variables de type réelles codées sur 8 octets. Ces variables sont internes à la procédure. L'instruction MonMontant = ThisWorkbook.Worksheets("Change").Range("E4").Value permet de stocker dans la variable MonMontant le contenu de la case E4 de la feuille « Change » du projet Excel. L'instruction ChoixPays.ListePays.Value permet de récupérer le rang du pays sélectionné dans la liste des pays (premier pays, valeur 0, deuxième pays, valeur 1, …).

L'instruction ChoixPays.Hide permet de cacher la fenêtre ChoixPays.

Notez l'appel à la fonction Calcule, créée précédemment.

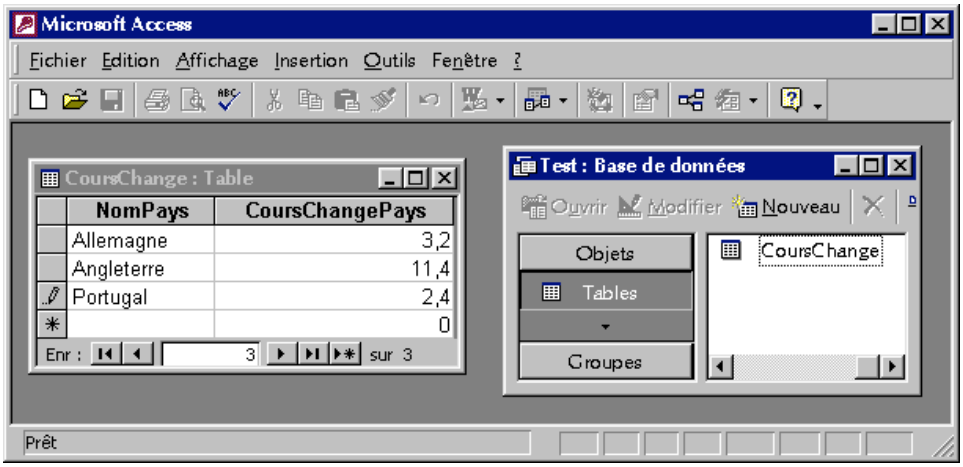
Le code que nous venons d'écrire ne permet pas de tester toutes les erreurs. Si vous saisissez par exemple TOTO dans la case E4, le programme ne s'exécutera pas correctement. Il faudrait prendre en compte tous ces cas de figures pour obtenir un résultat professionnel …

6. Mise à jour des valeurs de change à partir d'Access

Nous souhaitons dans cette partie, récupérer dans Excel, les valeurs de change stockée dans une base de données.

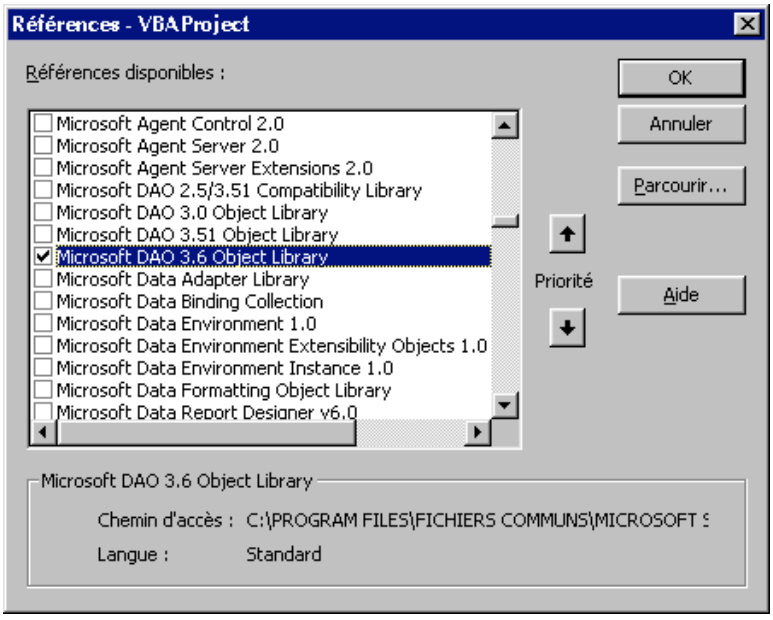
- Construction de la base de données

Nous supposerons que nous avons à notre disposition, dans le répertoire TEST, une base de donnée Access, dans laquelle nous avons une table CoursChange. Cette table est composée des champs NomPays, CoursChangePays. La figure ci-dessous nous donne une vue de cette base de donnée sous Access.



- Ajout de la référence DAO 3.6 dans le projet Excel

Pour ajouter cette référence, il faut sélectionner les menus Outils/Références et sélectionnez Microsoft DAO 3.6 Object Library comme indiqué sur la figure ci-dessous :



Après avoir cliqué sur OK, nous avons maintenant accès dans notre environnement Excel, aux objets de manipulation de bases de données basées sur le moteur Microsoft Jet. Access en faisant partie, nous allons pouvoir accéder au contenu d'une base Access.

Dans le module Module1 que nous avons créé précédemment, nous allons rajouter une nouvelle procédure intitulée Charger_Cours sans paramètre, donc directement appelable depuis un bouton d'une feuille Excel.

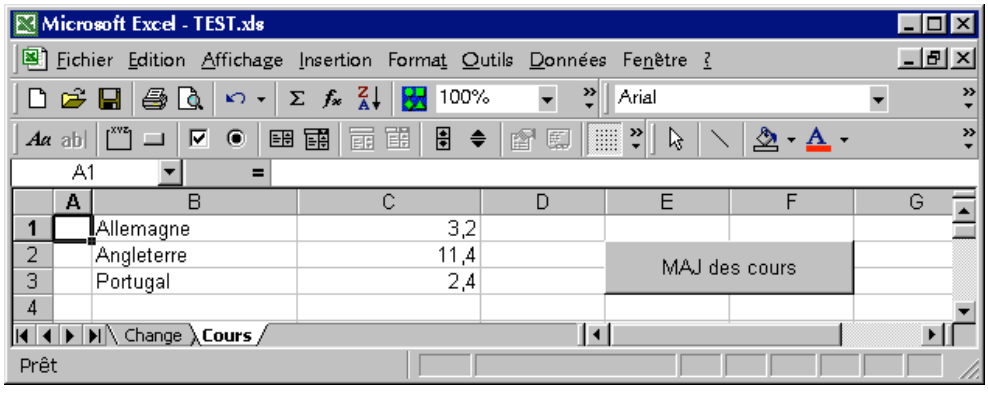
Saisissez le code suivant dans cette procédure :

```
Public Sub Charge_Cours()

Dim Mydb As Database
Dim MyContenu As Recordset
Dim i As Integer

Set Mydb = OpenDatabase("Test")
Set MyContenu = Mydb.OpenRecordset("CoursChange")
i = 1
MyContenu.MoveFirst
Do Until MyContenu.EOF
  ThisWorkbook.Worksheets("Cours").Range("B" & i).Value = MyContenu![NomPays]
  ThisWorkbook.Worksheets("Cours").Range("C" & i).Value = MyContenu![CoursChangePays]
  i = i + 1
  MyContenu.MoveNext
Loop
Mycontenu.close

End Sub
```

A quoi correspond ce code ?

Les variables « Mydb » et « MyContenu » pointe respectivement vers la base de données « Test.mdb » stockée dans le même répertoire que le projet Excel et la table « CoursChange » de cette base. Les méthodes « MoveFirst », « MoveNext » permettent de se déplacer dans les enregistrements de la table, l'instruction « MyContenu ![NomPays] » permet de récupérer la valeur du champs NomPays pour l'enregistrement courant.

Il nous faut maintenant rajouter dans la feuille Excel « Cours » un bouton que l'on associera à la procédure Charge_Cours et que l'on nommera « MAJ des Cours »



Testez différentes modifications dans la base de données, et cliquez sur le bouton pour constater la mise à jour des valeurs sous Excel.

Bon courage !!