Guide de formation avec cas pratiques

Excel 2013 Programmation VBA

Daniel-Jean David

Tsoft EDITEUR

EYROLLES

Table des matières

Création d'un Programme

1

Enregistrement d'une macro

Écriture des instructions VBA : l'Éditeur VBA

Règles fondamentales de présentation

Projets, différentes sortes de modules

Options de projets

Les différentes sortes d'instructions

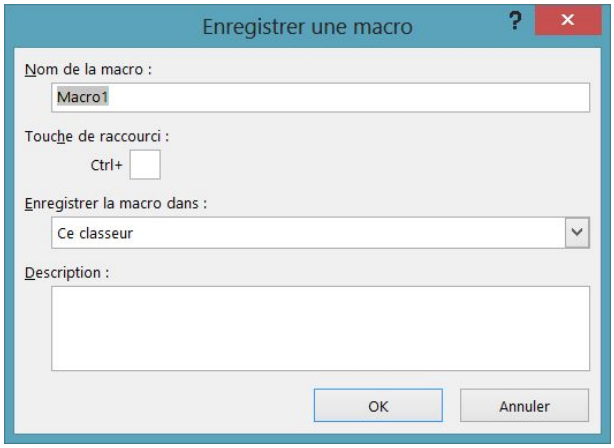
Les menus de l'Éditeur VBA

ENREGISTREMENT D'UNE MACRO

ENREGISTRER UNE SUITE D'OPÉRATIONS EXCEL

Nous allons voir qu'on peut mémoriser une suite d'opérations Excel pour pouvoir répéter cette suite ultérieurement sans avoir à refaire les commandes.

- Dans feuille de classeur Excel, faites 🗀*AFFICHAGE – [Macros] – Macros – Enregistrer une macro :*

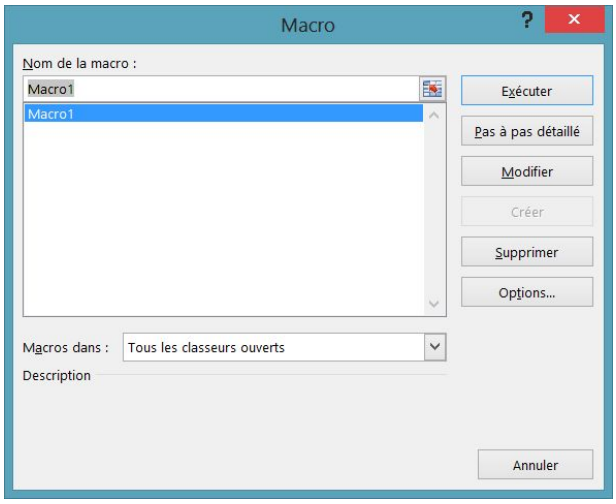


- Vous avez la possibilité de changer le nom de la macro, de la sauvegarder dans d'autres classeurs (le plus souvent, on la sauvegarde dans le classeur en cours) ou de donner une description plus complète de la macro en cours de définition. L'option probablement la plus utile est d'associer une touche de raccourci. Cliquez sur OK pour valider.
- Faites les opérations Excel que vous souhaitez enregistrer….
- Faites 🗀*AFFICHAGE – [Macros] – Macros – Arrêter l'enregistrement*.

Avant l'enregistrement, vous avez la possibilité de demander 🗀*AFFICHAGE – [Macros] – Macros – Enregistrer une macro – Utiliser les références relatives*, ce qui permet de décider que la rédaction de la macro traitera les coordonnées de cellules en relatif (c'est en absolu en l'absence de cette commande).

DÉCLENCHER UNE NOUVELLE EXÉCUTION

- Revenu sur la feuille Excel, modifiez éventuellement certaines données.
- Faites 🗀*AFFICHAGE – [Macros] – Macros – Afficher les macros*, le dialogue suivant s'affiche :



Ce dialogue permet de choisir une macro dans la liste. Cette liste est formée de toutes les procédures connues de Visual Basic soit dans tous les classeurs ouverts, soit dans le classeur spécifié grâce à la liste déroulante <Macros dans> en bas de la BDi.

- Après avoir sélectionné la macro, cliquez sur le bouton Exécuter, vous pouvez constater que vos opérations sont répétées

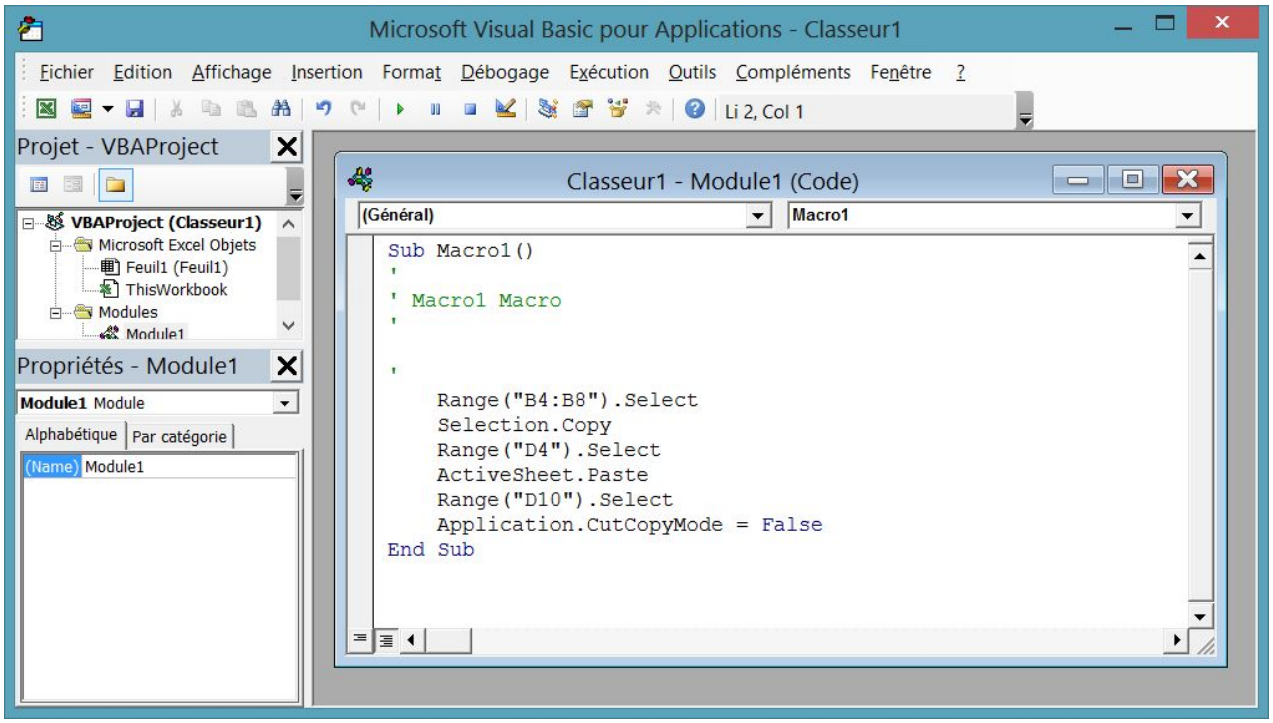
EXAMINER LA MACRO PRODUITE

Il faut pouvoir examiner ce qu'Excel a mémorisé en fonction des actions enregistrées. Cet examen est en particulier nécessaire si l'exécution de la macro ne produit pas les résultats voulus : c'est probablement qu'une action parasite a été enregistrée et il faudra enlever ce qui la représente dans l'enregistrement

Une autre raison d'examiner la macro telle qu'elle est enregistrée est de pouvoir la modifier. Des modifications mineures qu'on peut vouloir faire viennent du processus même de l'enregistrement : supposons que, voulant sélectionner la cellule A3, vous sélectionniez d'abord, suite à une hésitation, la cellule A4 ; bien entendu, vous allez rectifier et cliquer sur A3. Mais Excel aura enregistré deux opérations de sélection et il sera conseillé de supprimer la sélection de A4. Donc une première raison de modification est d'élaguer la macro des opérations inutiles.

Un autre motif de modification, beaucoup plus important, est de changer le comportement de la macro pour le rendre plus ergonomique, ou pour traiter d'autres aspects de l'application.

- Dans la boîte de dialogue 🗀*AFFICHAGE – [Macros] – Macros – Afficher les macros*, cliquez sur Modifier : la fenêtre de l'Éditeur VBA apparaît.



L'ONGLET DÉVELOPPEUR

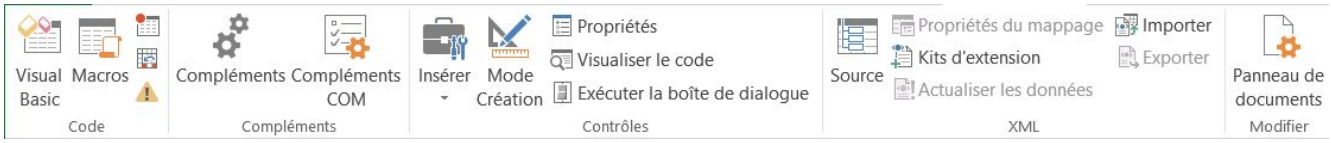
Nous voyons maintenant une autre manière d'appeler l'éditeur VBA. Une option permet d'ajouter un onglet appelé Développeur. Il est, de toutes façons, indispensable pour toute utilisation régulière de VBA.

ENREGISTREMENT D'UNE MACRO

Afficher l'onglet développeur

- Cliquez sur Fichier.
- Cliquez sur Options puis *Personnaliser le ruban*.
- Cochez ☑ Développeur dans la liste *Onglets principaux* et OK .

L'onglet Développeur se rajoute au ruban. Voici son contenu :



La commande 🗀*DÉVELOPPEUR – [Code] – Macros* fait apparaître la boîte de dialogue liste des macros. La commande 🗀*DÉVELOPPEUR – [Code] – Visual Basic* appelle l'éditeur VBA. Vous retiendrez rapidement son raccourci Alt+F11, best-seller auprès des programmeurs VBA.

On passe de la fenêtre de l'éditeur VBA à la fenêtre classeur et inversement par clics sur leurs boutons dans la barre en bas de l'écran ou à coups de Alt+F11.

À part ses barres de menus et d'outils, la fenêtre de l'éditeur VBA comprend deux volets. Celui de gauche se partage de haut en bas en Explorateur de projets et Fenêtre de propriétés ; le volet de droite est occupé par une ou plusieurs fenêtres de code.

- Si vous n'avez pas l'affichage correspondant à la figure, le plus probable est que vous n'ayez pas la fenêtre de code, mais que vous ayez le volet de gauche. Dans l'Explorateur de projets, vous devez avoir au moins une tête d'arborescence *VBAProject(nom de votre classeur)*. Pour VBA, un classeur et l'ensemble de ses macros forme un « projet ». L'arborescence de votre projet doit se terminer par une rubrique Modules.
- Si celle-ci n'est pas développée, cliquez sur son signe + : Module1 doit apparaître
- Double-cliquez sur le mot *Module1* : la fenêtre de code doit apparaître.
- Si vous n'avez pas le volet de gauche, appelez le menu *Affichage* et cliquez les rubriques *Explorateur de projets* et *Fenêtre Propriétés*, puis éventuellement arrangez leurs tailles et positions.

Avantages et inconvénients de la construction de macros par enregistrement

On peut créer une macro sans enregistrer des actions Excel, en écrivant le texte du programme souhaité directement dans une fenêtre module sous l'Éditeur VBA.

Un avantage de l'enregistrement d'une séquence de commandes est que, la macro étant générée par Excel, elle ne peut contenir aucune faute de frappe. Du côté des inconvénients, nous noterons un certain manque de souplesse : la macro ne peut que faire exactement ce qu'on a enregistré, sans paramétrage possible.

Autre inconvénient, plus grave et qui justifie que l'on puisse saisir des programmes directement au clavier : par enregistrement, on ne peut que générer un programme à logique linéaire où toutes les actions se suivent en séquence ; on ne peut pas créer un programme où, en fonction de premiers résultats, on effectue telle action ou bien telle autre : lors de l'enregistrement, on suivra une seule des voies possibles et elle seule sera enregistrée.

A fortiori, lorsqu'une sous-étape du traitement doit être répétée plusieurs fois, l'enregistrement ne mémorise qu'un passage. Ces possibilités appelées « alternatives » et « boucles » sont offertes par des instructions de VBA mais qui doivent être fournies directement. Ces instructions s'appellent instructions de structuration.

Mais un grand avantage de l'enregistrement, qui est à nos yeux le plus important, est que cette méthode est une extraordinaire machine à apprendre VBA, ou plutôt les objets Excel et leur manipulation : dès qu'on sait accomplir une action par les commandes Excel, on saura comment cela s'écrit en VBA, ou plutôt quels objets manipuler et comment. Il suffit de se mettre en mode enregistrement, d'effectuer les commandes Excel voulues, arrêter l'enregistrement puis examiner ce que le système a généré. Par exemple, pour voir comment on imprime, il suffit de commander une impression en mode enregistrement. Bien sûr, on pourrait trouver la réponse dans l'aide en ligne, mais la méthode de l'enregistrement épargne une longue recherche.

Sauvegarde d'un classeur contenant des macros

Bien entendu, votre classeur devra être sauvegardé. Dans la version Office 2013, les classeurs qui ne contiennent pas de macros ont l'extension .xlsx, tandis que ceux qui contiennent des macros ont l'extension .xlsm.

Pour la première sauvegarde du classeur, il faut revenir à la fenêtre Excel et :

- 🗀*FICHIER – Enregistrer sous – Classeur Excel prenant en charge les macros*.
- Fournir disque, répertoire et nom du fichier.

Pour les sauvegardes suivantes, la commande *Fichier – Enregistrer* de la fenêtre de l'éditeur VBA convient.

ÉCRITURE DES INSTRUCTIONS VBA : L'ÉDITEUR VBA

CRÉER UN MODULE

Depuis un classeur Excel, on arrive à l'écran VBA par la commande 🗀*DÉVELOPPEUR – [Code] – Visual Basic* ou Alt+F11. On a vu dans la section précédente comment assurer que la fenêtre de projets soit présente. Elle a au moins une arborescence *VBA Project (nom de votre classeur)* et celle-ci a au moins une rubrique *Microsoft Excel Objects*.

- Si le programme que vous souhaitez écrire doit gérer la réponse à des événements concernant une feuille de classeur ou le classeur, les modules correspondants apparaissent dans l'arborescence sous *Microsoft Excel Objects*. Double-cliquez sur la feuille voulue ou le classeur : la fenêtre de module apparaît.
- Dans les autres cas :
  - Sélectionnez le projet (clic sur sa ligne dans la fenêtre *Projets*), puis
  - *Insertion – Module* pour un module normal. Les autres choix sont *Module de classe* et *User Form* (Boîte de dialogue et module gestion des objets contenus). Ces cas sont traités dans d'autres chapitres, donc plaçons-nous ici dans le cas du module normal.
  - Une fois le module créé, la rubrique *Modules* apparaît dans l'arborescence. Pour écrire le programme, développez la rubrique, puis double-cliquez sur le nom du module voulu.
  - Il faut maintenant créer une procédure. Le menu *Insertion* a une rubrique *Procédure*, mais il suffit d'écrire `Sub <nom voulu>` dans le module.

SUPPRIMER UN MODULE

On peut avoir à supprimer un module, notamment parce que, si on enregistre plusieurs macros, VBA peut décider de les mettre dans des modules différents (par exemple Module2, etc.) alors qu'il est préférable de tout regrouper dans Module 1.

- Après avoir déplacé les procédures des autres modules dans Module 1, sélectionnez chaque module à supprimer par clic sur son nom sous la rubrique Modules.
- *Fichier – Supprimer Module 2* (le nom du module sélectionné apparaît dans le menu *Fichier*).
- Une BDi apparaît, proposant d'exporter le module. Cliquez sur Non.

EXPORTER/IMPORTER UN MODULE

Exporter :

Si dans la BDi précédente, vous cliquez sur Oui, vous exportez le module, c'est-à-dire que vous créez un fichier d'extension .bas qui contiendra le texte des procédures du module. Un tel fichier peut aussi se construire par :

- Mettez le curseur texte dans la fenêtre du module voulu.
- *Fichier – Exporter un fichier*.
- La BDi qui apparaît vous permet de choisir disque, répertoire et nom de fichier.

Importer :

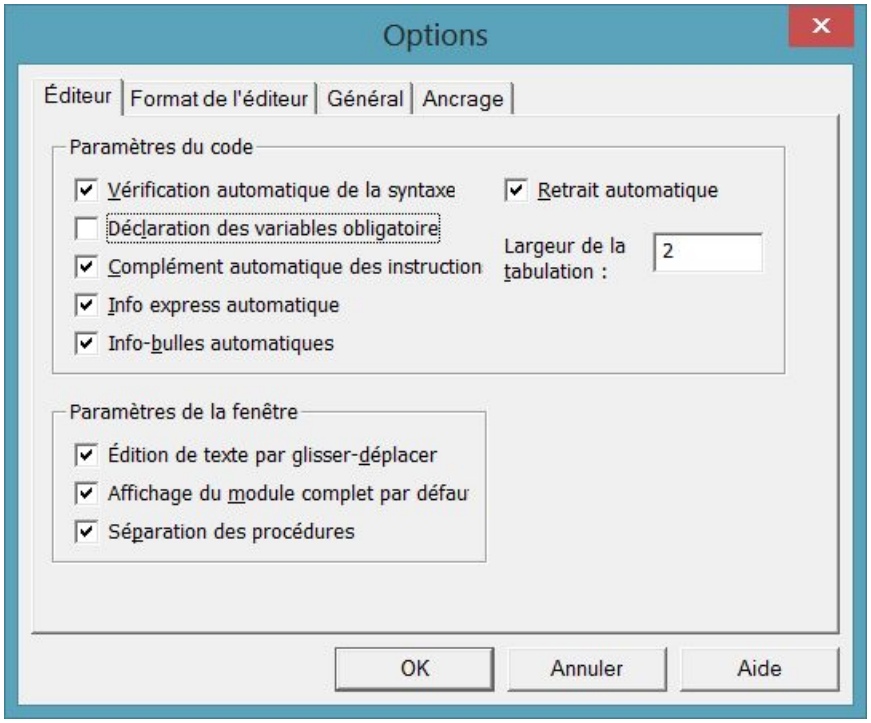
L'opération inverse est l'importation qui permet d'ajouter un fichier à un projet :

- Sélectionnez le projet concerné (par clic sur sa ligne dans la fenêtre de projets), puis faites *Fichier – Importer un fichier*.
- Dans la BDi, choisissez disque, répertoire et nom de fichier. Les extensions possibles sont .bas (module normal), .cls (module de classe) et .frm (BDi construite par l'utilisateur et le module de code associé).

Cette technique permet de développer des éléments, procédures ou BDi servant pour plusieurs projets.

OPTIONS RÉGLANT LE FONCTIONNEMENT DE L'ÉDITEUR

Dans l'écran VBA, faites *Outils – Options*. Le fonctionnement de l'éditeur obéit aux onglets *Éditeur* et *Format de l'éditeur*. L'onglet *Éditeur* règle le comportement vis-à-vis du contenu du programme notamment les aides à l'écriture procurées par l'éditeur :



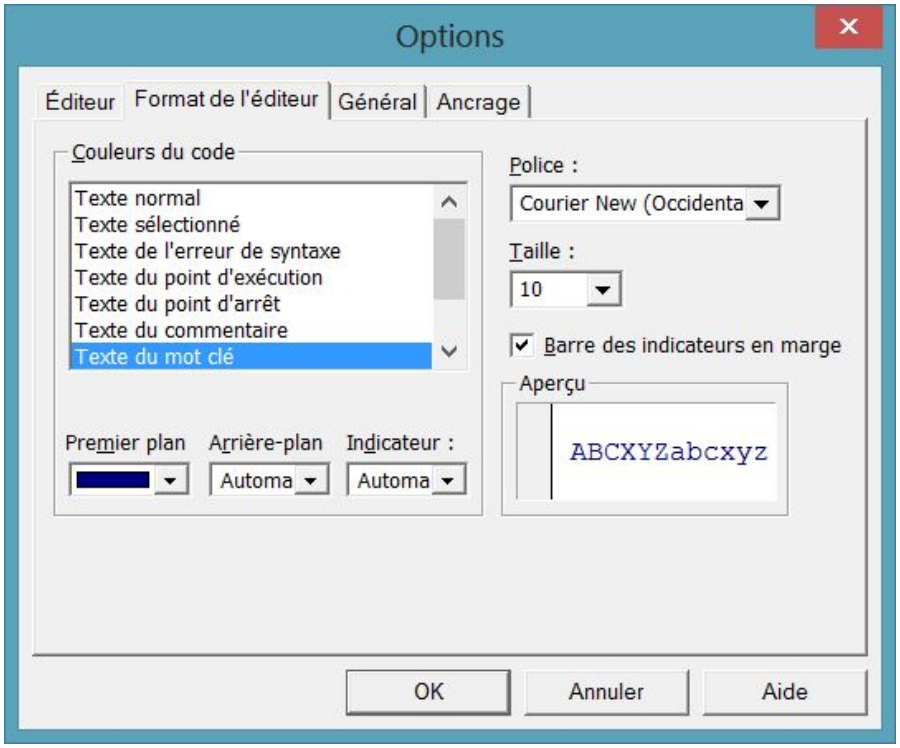
Les choix de la figure nous semblent les plus raisonnables.

- ☑ *Vérification automatique de la syntaxe* parle d'elle-même
- ☐ *Déclaration de variables obligatoire* si la case est cochée installe automatiquement Option Explicit en tête de tous les modules. Si la case n'est pas cochée, vous devez taper la directive partout où il le faut.
- ☑ *Complément automatique des instructions* présente les informations qui sont le complément logique de l'instruction au point où on est arrivé.
- ☑ *Info express automatique* affiche des informations au sujet des fonctions et de leurs paramètres au fur et à mesure de la saisie
- ☑ *Info-bulles automatiques* : en mode Arrêt, affiche la valeur de la variable sur laquelle le curseur est placé.
- ☑ *Retrait automatique* : si une ligne de code est mise en retrait, toutes les lignes suivantes sont automatiquement alignées par rapport à celle-ci. Pensez en même temps à choisir l'amplitude des retraits successifs (ci-dessus 2, au lieu de la valeur par défaut 4).

Les options *Paramètres de la fenêtre* sont moins cruciales.

- ☑ *Édition de texte par glisser-déplacer* permet de faire glisser des éléments au sein du code et de la fenêtre Code vers les fenêtres Exécution ou Espions.
- ☑ *Affichage du module complet par défaut* fait afficher toutes les procédures dans la fenêtre Code ; on peut, par moments, décider d'afficher les procédures une par une.
- ☑ *Séparation des procédures* permet d'afficher ou de masquer les barres séparatrices situées à la fin de chaque procédure dans la fenêtre Code. L'intérêt de cette option est diminué par le fait que ces séparations n'apparaissent pas à l'impression du listing ; une solution est d'insérer devant chaque procédure une ligne de commentaire remplie de tirets : '--------…

L'onglet *Format* de l'éditeur fixe les couleurs des différents éléments du code. C'est lui qui décide par défaut mots-clés en bleu, commentaires en vert, erreurs en rouge.



- ☑ *Barre des indicateurs en marge* affiche ou masque la barre des indicateurs en marge, indicateurs utiles pour le dépannage.
- Ayant choisi un des éléments dans la liste, vous déterminez la police, taille et couleur de façon classique ; en principe, on utilise une police de type Courrier parce qu'elle donne la même largeur à tous les caractères, mais rien ne vous y oblige.
- Les éléments possibles sont : Texte normal, Texte sélectionné, Texte de l'erreur de syntaxe, Texte du point d'exécution, Texte du point d'arrêt, Texte du commentaire, Texte du mot clé, Texte de l'identificateur, Texte du signet, Texte de retour de l'appel.

RÈGLES FONDAMENTALES DE PRÉSENTATION

UNE INSTRUCTION PAR LIGNE

La règle fondamentale est d'écrire une instruction par ligne. Lorsque vous tapez sur la touche ↵, VBA suppose qu'on passe à la prochaine instruction. Cette règle admet deux exceptions qui n'interviennent que très rarement.

- On peut mettre plusieurs instructions sur une ligne à condition de les séparer par le caractère deux-points (:).

```
x = 3 : y = 5
```

Cette pratique est tout à fait déconseillée ; elle ne se justifie que pour deux instructions courtes formant en quelque sorte un bloc logique dans lequel il n'y aura en principe pas de risque d'avoir à insérer d'autres instructions.

- Une instruction peut déborder sur la (les) ligne(s) suivante(s). La présentation devient :

```
xxxxxxxxxxxxxxxxxx(1re partie)xxxxxxxxxxxxxxxxxx   _
        yyyyyyy(2e partie)yyyyyyyyyyyyyyy
```

Les lignes sauf la dernière doivent se terminer par la séquence <espace><signe souligné>. Bien entendu, la coupure doit être placée judicieusement : là où l'instruction aurait naturellement un espace. On ne doit pas couper un mot-clé propre au langage, ni un nom de variable.

Cas particulier : on ne doit pas couper une chaîne de caractères entre guillemets (comme `"Bonjour"`). La solution est la suivante : on remplace la longue chaîne par une concaténation de deux parties ("partie 1" + "partie 2") et on coupera comme suit :

```
 ……."partie 1" +  _
"partie 2"
```

MAJUSCULES ET MINUSCULES

Sauf à l'intérieur d'une chaîne de caractères citée entre ", les majuscules et minuscules ne comptent pas en VBA. En fait, les mots-clés et les noms d'objets et de propriétés prédéfinis comportent des majuscules et minuscules et vous pouvez définir des noms de variables avec des majuscules où vous le souhaitez. Mais vous pouvez taper ces éléments en ne respectant pas les majuscules définies (mais il faut que les lettres soient les mêmes) : l'éditeur VBA rétablira automatiquement les majuscules de la définition ; pour les noms de variables, on se basera sur la 1re apparition de la variable (en principe sa déclaration).

Il en résulte un conseil très important : définissez des noms avec un certain nombre de majuscules bien placées et tapez tout en minuscules : si VBA ne rétablit pas de majuscules dans un nom, c'est qu'il y a une faute d'orthographe.

Un autre élément qui peut vous permettre de déceler une faute d'orthographe, mais seulement dans un mot-clé, est que si un mot n'est pas reconnu comme mot-clé, VBA ne l'affichera pas en bleu. Bien sûr, vous devez être vigilants sur ces points : plus tôt une faute est reconnue, moins il y a de temps perdu.

Pour les chaînes de caractères entre ", il s'agit de citations qui apparaîtront telles quelles, par exemple un message à afficher, le nom d'un client , etc. Il faut donc taper exactement les majuscules voulues.

COMMENTAIRES, LIGNES VIDES

Un commentaire est une portion de texte figurant dans le programme et n'ayant aucun effet sur celui-ci. La seule chose que VBA fait avec un commentaire, c'est de le mémoriser et de l'afficher dans le listing du programme. Les commentaires servent à donner des explications sur le programme, les choix de méthodes de traitement, les astuces utilisées, etc.

Ceci est utile pour modifier le programme, car, pour cela, il faut le comprendre ; c'est utile même pour le premier auteur du programme car lorsqu'on reprend un programme plusieurs mois après l'avoir écrit, on a oublié beaucoup de choses. Il est donc conseillé d'incorporer beaucoup de commentaires à un programme dès qu'il est un peu complexe.

VBA admet des commentaires en fin de ligne ou sur ligne entière.

En fin de ligne, le commentaire commence par une apostrophe. Ex. :

```
Remise = Montant * 0.1    ' On calcule une remise de 10%
```

Sur ligne entière, le commentaire commence par une apostrophe ou le mot-clé `Rem`. On utilise plutôt l'apostrophe. Si le commentaire occupe plusieurs lignes, chaque ligne doit avoir son apostrophe.

Les lignes vides sont autorisées en VBA ; elles peuvent servir à aérer le texte. Nous conseillons de mettre une apostrophe en tête pour montrer que le fait que la ligne soit vide est voulu par le programmeur.

LES ESPACES

Les espaces sont assez libres en VBA, mais pas totalement. Là où il peut et doit y avoir un espace, vous pouvez en mettre plusieurs, ou mettre une tabulation.

On ne doit en aucun cas incorporer d'espaces à l'intérieur d'un mot-clé, d'un nom d'objet prédéfini, d'un nombre ou d'un nom de variable : ces mots ne seraient pas reconnus.

Au contraire, pour former des mots, ces éléments doivent être entourés d'espaces, ou d'autres caractères séparateurs comme la virgule.

Les opérateurs doivent être entourés d'espaces, mais vous n'êtes pas obligés de les taper, l'éditeur VBA les fournira sauf pour &. Si vous tapez `a=b+c` vous obtiendrez `a = b + c`.

LES RETRAITS OU INDENTATIONS

Les instructions faisant partie d'une même séquence doivent normalement commencer au même niveau d'écartement par rapport à la marge. Lors de l'emploi d'instructions de structuration, les séquences qui en dépendent doivent être en retrait par rapport aux mots-clés de structuration. En cas de structures imbriquées, les retraits doivent s'ajouter. Exemple fictif :

```
x = 3
For I = 2 To 10
    a = 0.05 * I
    If b < x Then
         x = x - a
    Else
         b = b – a
    End If
Next I
```

En cas de nombreuses imbrications, le retrait peut être un peu grand : bornez-vous à 2 caractères à chaque niveau. Bien sûr, ces retraits ne sont pas demandés par le langage, ils n'ont que le but de faciliter la compréhension en faisant ressortir la structure du programme (ou plutôt, la structure souhaitée, car, dans son interprétation, VBA ne tient compte que des mots-clés, pas des indentations : mais justement un désaccord entre les mots-clés et les indentations peut vous aider à dépister une erreur).

Il est donc essentiel, bien que non obligatoire que vous respectiez les indentations que nous suggèrerons pour les instructions.

AIDE À LA RECHERCHE D'ERREURS

Nous avons vu plus haut que VBA introduisait de lui-même les majuscules voulues dans les mots-clés et les noms de variables, d'où notre conseil de tout taper en minuscules : s'il n'y a pas de transformation, c'est qu'il y a probablement une faute de frappe.

Pour les mots-clés, on a une aide supplémentaire : VBA met les mots-clés en bleu (en fait, la couleur choisie par option) ; si un mot n'est pas transformé, c'est qu'il n'est pas reconnu, donc qu'il y a une faute.

Une autre aide automatique est que, en cas d'erreur de syntaxe, VBA affiche aussitôt un message d'erreur et met l'instruction en rouge. Bien sûr cela ne décèle que les erreurs de syntaxe, pas les erreurs de logique du programme.

AIDES À L'ÉCRITURE

L'éditeur VBA complète automatiquement certaines instructions :

Dès que vous avez tapé une instruction `Sub` ou `Function`, VBA fournit le `End Sub` ou le `End Function`.

Si vous tapez `endif` sans espace, VBA corrige : `End If`. Attention, il ne le fait que pour celle-là : pour `End Select` ou pour `Exit Sub` ou d'autres, il faut taper l'espace.

Dès que vous tapez un espace après l'appel d'une procédure, ou la parenthèse ouvrante à l'appel d'une fonction, VBA vous suggère la liste des arguments. Il le fait toujours pour un élément prédéfini ; pour une procédure ou fonction définie par vous, il faut qu'elle ait été définie avant.

Dès que vous tapez le `As` dans une déclaration, VBA fournit une liste déroulante des types possibles ; il suffit de double-cliquer sur celui que vous voulez pour l'introduire dans votre instruction. Vous avancez rapidement dans la liste en tapant la première lettre souhaitée. Un avantage supplémentaire est qu'un élément ainsi écrit par VBA ne risque pas d'avoir de faute d'orthographe.

De même, dès que vous tapez le point après une désignation d'objet, VBA affiche la liste déroulante des sous-objets, propriétés et méthodes qui en dépendent et vous choisissez comme précédemment. L'intérêt est que la liste suggérée est exhaustive et peut donc vous faire penser à un élément que vous aviez oublié. Attention, cela n'apparaît que si l'aide en ligne est installée et si le type d'objet est connu complètement à l'écriture, donc pas pour une variable objet qui aurait été déclarée d'un type plus général que l'objet désigné (ex. `As Object`).

PROJETS, DIFFÉRENTES SORTES DE MODULES

DÉFINITION

Un **projet** est l'ensemble de ce qui forme la solution d'un problème (nous ne voulons pas dire « application » car ce terme a un autre sens, à savoir l'objet Application, c'est-à-dire Excel lui-même), donc un classeur Excel avec ses feuilles de calcul, et tous les programmes écrits en VBA qui sont sauvegardés avec le classeur. Les programmes sont dans des modules ; le texte des programmes est affiché dans des fenêtres de code. Il peut y avoir un module associé à chaque feuille ou au classeur. Il peut y avoir un certain nombre de modules généraux. De plus, le projet peut contenir aussi des modules de classe et des boîtes de dialogue créées par le programmeur : chaque BDi a en principe un module de code associé.

Un programme peut ouvrir d'autres classeurs que celui qui le contient ; ces classeurs forment autant de projets, mais secondaires par rapport au projet maître.

LES FENÊTRES DU PROJET

L'écran VBA contient principalement la fenêtre de projet où apparaît le projet associé à chaque classeur ouvert. Chaque projet y apparaît sous forme d'une arborescence (développable ou repliable) montrant tous les éléments du projet. Sous la fenêtre de projet, peut apparaître une fenêtre Propriétés qui affiche les propriétés d'un élément choisi dans la fenêtre de projet ou d'un contrôle sélectionné dans une BDi en construction.

La plus grande partie de l'écran sera consacrée aux fenêtres de BDi en construction ou de code. Comme ces fenêtres sont en principe présentées en cascade, on choisit celle qui est en premier plan par clic dans le menu *Fenêtre*. On décide de l'affichage d'un tel élément par double-clic dans l'arborescence.

On peut faire apparaître d'autres fenêtres par clic dans le menu *Affichage*. C'est le cas des fenêtres de (l'Explorateur de) Projets, Propriétés, Explorateur d'objets, Exécution, Variables locales et Espions, ces trois dernières servant surtout au dépannage des programmes.

Le menu *Affichage* permet de basculer entre l'affichage d'un objet (comme une BDi) et la fenêtre de code correspondante (raccourci touche F7).

Le choix des fenêtres à afficher peut se faire aussi par des boutons de la barre d'outils Standard de l'écran VBA.

DIFFÉRENTES SORTES DE MODULES

À chacune des quatre rubriques de la hiérarchie dépendant du projet correspond une sorte de module. À *Microsoft Excel Objects* (les feuilles et le classeur) correspondent des modules où se trouveront les programmes de réponse aux événements de la feuille (ex. *Worksheet_Change*) ou du classeur (ex. *Workbook_Open*).

À *Feuilles* correspondent les BDi construites par le programmeur (UserForms). Chacune a un module associé qui contient les procédures de traitement des événements liés aux contrôles de la BDi (ex. *UserForm_Initialize*, *CommandButton1_Click,* etc.) ;

À *Modules* correspondent les différents modules « normaux » introduits. C'est dans ces modules (en principe, on les regroupe en un seul) que sont les procédures de calcul propres au problème.

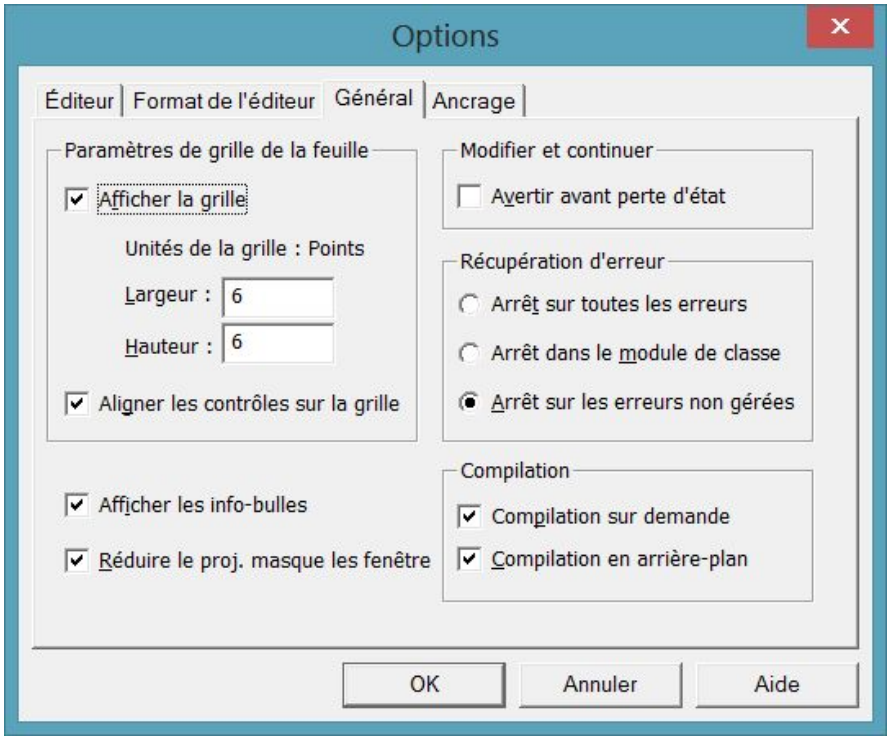
La dernière sorte de modules dépend de la rubrique *Modules de classe* ; les modules de classe permettent de définir des objets propres au programmeur. Ils sont beaucoup moins souvent utilisés car, vu la richesse des objets prédéfinis en Excel VBA, on en utilise rarement plus de 10 %, alors on a d'autant moins de raisons d'en créer d'autres !

Une dernière rubrique, *Références* peut être présente dans l'arborescence, mais elle n'introduit pas de modules.

OPTIONS DE PROJETS

LA COMMANDE OUTILS-OPTIONS

Cette commande concerne les projets par ses onglets *Général* et *Ancrage*. L'onglet *Ancrage* décide quelles fenêtres vont pouvoir être ancrées c'est-à-dire fixées en périphérie de l'écran. Ce n'est pas vital. L'onglet *Général* a plus à dire :

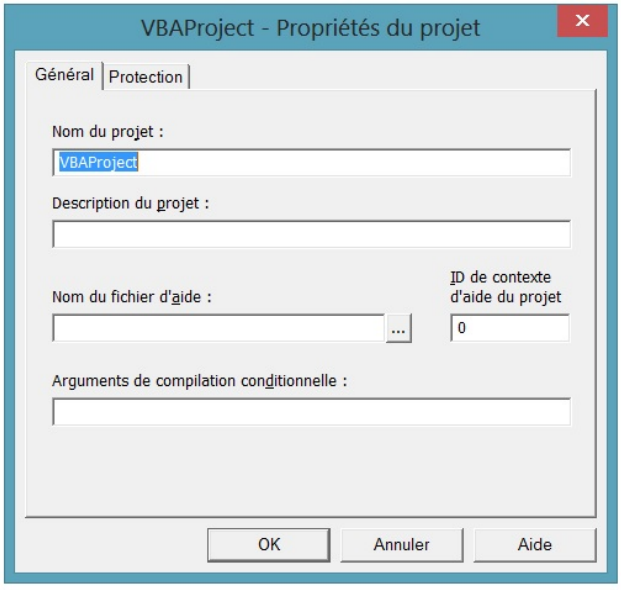


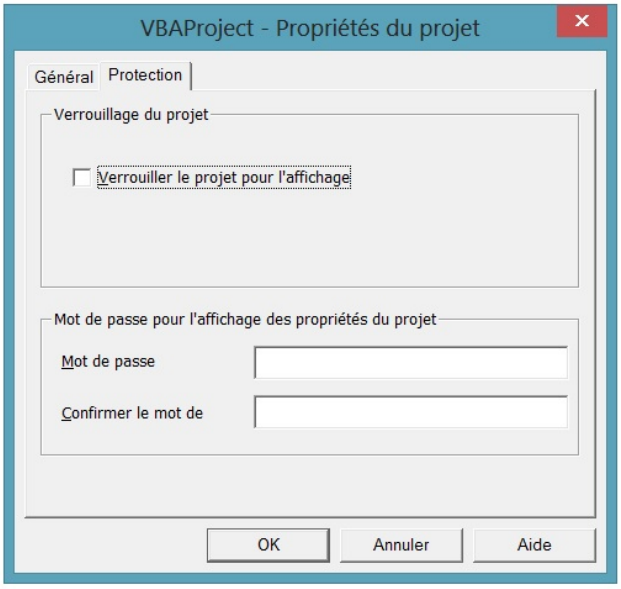
- Le cadre Paramètres de grille de la feuille gère le placement des contrôles sur une BDi construite par le programmeur, donc voir chapitre 6.
- ☑ *Afficher les info-bulles* affiche les infobulles des boutons de barre d'outils.
- ☑ *Réduire le proj. masque les fenêtres* définit si les fenêtres de projet, UserForm, d'objet ou de module sont fermées automatiquement lors de la réduction du projet dans l'Explorateur de projet.
- Le cadre Modifier et continuer.
  - ☑ *Avertir avant perte d'état* active l'affichage d'un message lorsque l'action demandée va entraîner la réinitialisation de toutes les variables de niveau module dans le projet en cours.
- Le cadre Récupération d'erreur définit la gestion des erreurs dans l'environnement de développement Visual Basic. L'option s'applique à toutes les occurrences de Visual Basic lancées ultérieurement.
  - ⊙ *Arrêt sur toutes les erreurs* : en cas d'erreur quelle qu'elle soit, le projet passe en mode Arrêt.
  - ⊙ *Arrêt dans les modules de classe* : en cas d'erreur non gérée survenue dans un module de classe, le projet passe en mode Arrêt à la ligne de code du module de classe où s'est produite l'erreur.
  - ⊙ *Arrêt sur les erreurs non gérées* : si un gestionnaire d'erreurs est actif, l'erreur est interceptée sans passage en mode Arrêt. Si aucun gestionnaire d'erreurs n'est actif, le projet passe en mode Arrêt. Ceci est l'option la plus conseillée.

- Compilation
  - ☑ *Compilation sur demande* définit si un projet est entièrement compilé avant d'être exécuté ou si le code est compilé en fonction des besoins, ce qui permet à l'application de démarrer plus rapidement, mais retarde l'apparition des messages d'erreur éventuels dans une partie de programme rarement utilisée.
  - ☑ *Compilation en arrière-plan* définit si les périodes d'inactivité sont mises à profit durant l'exécution pour terminer la compilation du projet en arrière-plan, ce qui permet un gain de temps. Possible seulement en mode compilation sur demande.

LA COMMANDE OUTILS-PROPRIÉTÉS DE <NOM DU PROJET>

Cette commande fait apparaître une BDi avec deux onglets :





- L'onglet *Général* permet de donner un nom plus spécifique que VBAProject, et surtout de fournir un petit texte descriptif. Les données concernant l'aide n'ont plus d'intérêt : la mode est maintenant de fournir une aide sous forme HTML. La compilation conditionnelle est sans réel intérêt.
- L'onglet *Protection* permet de protéger votre travail.
  - ☑ *Verrouiller le projet pour l'affichage* interdit toute modification de n'importe quel élément de votre projet. Il ne faut y faire appel que lorsque le projet est parfaitement au point !
  - La fourniture d'un mot de passe (il faut le donner deux fois, c'est classique) empêche de développer l'arborescence du projet dans la fenêtre Explorateur de projets si l'on ne donne pas le mot de passe. Donc un « indiscret » qui n'a pas le mot de passe n'a accès à aucune composante de votre projet.

LA COMMANDE OUTILS-RÉFÉRENCES

Permet de définir une référence à la bibliothèque d'objets d'une autre application pour y sélectionner des objets appartenant à cette application, afin de les utiliser dans votre code. C'est une façon d'enrichir votre projet.

LES DIFFÉRENTES SORTES D'INSTRUCTIONS

Les instructions VBA se répartissent en instructions exécutables ou ordres et instructions non exécutables ou déclarations.

INSTRUCTIONS EXÉCUTABLES

Ce sont les instructions qui font effectuer une action par l'ordinateur. Elles se répartissent en :

- **Instructions séquentielles**, telles que l'instruction qui sera exécutée après est l'instruction qui suit dans le texte.
  - La principale instruction de cette catégorie est *l'instruction d'affectation*, de la forme `[Set]<donnée>=<expression>`, où l'expression indique un calcul à faire. L'expression est calculée et le résultat est affecté à la donnée. En l'absence de `Set` (on devrait normalement mettre `Let`, mais il n'est jamais employé), l'expression conduit à une valeur et `<donnée>` est une variable ou une propriété d'objet ; elle reçoit la valeur calculée comme nouvelle valeur. Avec `Set`, l'expression a pour résultat un objet et `<donnée>` est une variable du type de cet objet : après l'instruction, cette variable permettra de désigner l'objet de façon abrégée. À part l'appel de procédures, cette instruction est la plus importante de tout le langage.
  - *Toute une série d'actions diverses*, notamment sur les fichiers (`Open`, `Close`, `Print#`...) ou sur certains objets (`Load`, `Unload`...) ou encore certaines opérations système (`Beep`, `Time`…). Ces instructions pourraient d'ailleurs aussi bien être considérées comme des appels à des procédures ou des méthodes prédéfinies.

- **Instructions de structuration**, ou de rupture de séquence, qui rompent la suite purement linéaire des instructions, aiguillant le traitement vers une séquence ou une autre selon des conditions, ou faisant répéter une séquence selon les besoins. Ces instructions construisent donc la structure du programme. La plus importante est :
  - *L'appel de procédure* : on déroute l'exécution vers un bloc d'instructions nommé qui remplit un rôle déterminé. La fin de l'exécution de la procédure se réduit à un retour dans la procédure appelante juste après l'instruction d'appel. Cela permet de subdiviser un programme complexe en plusieurs petites unités beaucoup plus faciles à maîtriser. La plupart du temps, l'instruction se réduit à citer le nom de la procédure à appeler.

Les autres instructions de structuration permettent d'implémenter les deux structures de la programmation structurée.

- *La structure alternative* où, en fonction de certaines conditions, on fera une séquence ou bien une autre. VBA offre pour cela deux instructions principales, `If` qui construit une alternative à deux branches et `Select Case` qui permet plusieurs branches.
- *La structure itérative* ou boucle, où on répète une séquence jusqu'à ce qu'une condition soit remplie (ou tant que la condition contraire prévaut). VBA offre pour cette structure les instructions `Do…Loop`..., `While…Wend` et, surtout, `For…Next` qui est la plus employée.

INSTRUCTIONS NON EXÉCUTABLES OU DÉCLARATIONS

Ces instructions ne déclenchent pas d'actions de l'ordinateur, mais donnent des précisions au système VBA sur la manière dont il doit traiter les instructions exécutables. La plus importante de ces instructions est la déclaration de variable qui :

- Annonce qu'on va utiliser une variable de tel ou tel nom.
- Indique le type (par exemple réel, ou entier, etc.) de la variable, c'est-à-dire des données qu'elle va contenir. Il est évident que les calculs ne s'effectuent pas de la même façon sur un nombre entier ou sur un réel. C'est en cela que les déclarations orientent le travail de VBA. **Elles sont donc aussi importantes que les instructions exécutables**.

Place des déclarations de variables

Normalement, il suffit qu'une déclaration de variable soit n'importe où avant la première utilisation de cette variable. En fait on recommande vivement de placer les déclarations de variables en tête de leur procédure. Par ailleurs, certaines déclarations de variables doivent être placées en tête de module, avant la première procédure du module.

Parmi les déclarations importantes, les couples `Sub … End Sub` et `Function … End Function` délimitent respectivement une procédure ou une fonction. `Sub` et `Function` ont en outre le rôle de déclarer des éventuels arguments. Les deux `End …` sont à la fois des déclarations – elles délimitent la fin de la procédure ou de la fonction – et des instructions exécutables : lorsque l'on arrive sur elles on termine la procédure ou la fonction et on retourne à l'appelant.

DIRECTIVES

Les directives sont des déclarations particulières qui jouent un rôle global au niveau du projet. Elles sont placées tout à fait en tête de module. Certaines peuvent être spécifiées sous forme d'options de projet auquel cas la directive est écrite automatiquement en tête de tous les modules.

`Option Explicit`

Exige que toute variable soit déclarée. Nous conseillons vivement cette option car si vous faites une faute de frappe dans un nom de variable, en l'absence de cette option, VBA « croira » que vous introduisez une nouvelle variable, alors qu'avec cette option, il y aura un message d'erreur vous permettant de la corriger aussitôt.

`Option Base <0 ou 1>`

Fixe à 0 ou à 1 la première valeur des indices de tableaux. La valeur par défaut est 0. Souvent les programmeurs utilisent les indices à partir de 1 sans spécifier `Option Base 1` : l'élément 0 est laissé vide. Cette pratique a un inconvénient : si par erreur un indice était calculé à 0, la directive assurerait un message d'erreur.

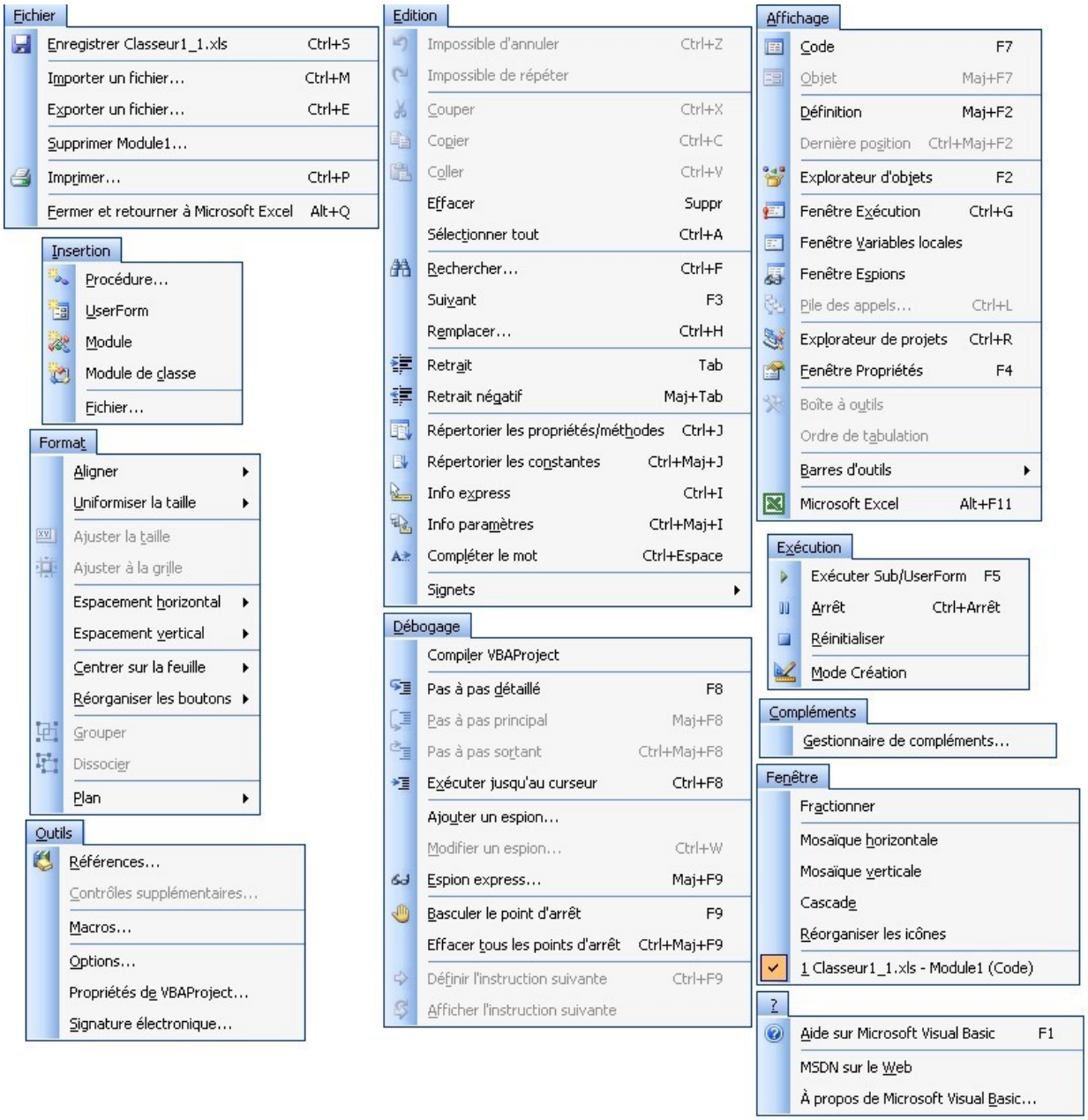
`Option Compare <choix>`

Fixe la façon dont les chaînes de caractères sont comparées. Avec `Text`, une majuscule et sa minuscule sont confondues alors qu'avec `Binary`, la comparaison est complète et les minuscules sont plus loin que les majuscules dans l'ordre alphabétique.

`Option Private Module`

Déclare le module entier comme privé, donc aucun de ses éléments, variables, procédures ou fonctions ne sera accessible depuis un autre module.

LES MENUS DE L'ÉDITEUR VBA



N.B. Certaines rubriques peuvent varier légèrement en fonction du contexte, selon qu'on est dans une procédure ou non et selon ce qu'on a fait précédemment ; ainsi *Edition – Impossible d'annuler* peut devenir *Edition – Annuler*, *Exécuter Sub...* peut devenir *Exécuter la macro,* etc.

Vie d'un programme

2

Différentes façons de lancer une procédure

Mise au point d'une macro

Utiliser l'aide

L'explorateur d'objets

Récupération des erreurs

DIFFÉRENTES FAÇONS DE LANCER UNE PROCÉDURE

PAR INSTRUCTION D'APPEL

Toute procédure peut être appelée depuis une autre procédure (ou fonction) par l'instruction d'appel de la forme :

```
[Call] <nom de la proc. appelée> [<arguments éventuels>]
```

Exemples :

```
Traitement          'il n'y a pas d'arguments
Calcul 5, 4         '2 arg. ; procédure supposée définie par :
                    'Sub Calcul (a as Integer, b as Integer)
```

Le mot-clé `Call` n'est presque jamais présent. Notez que la liste des arguments est entre parenthèses () dans la déclaration de la procédure, et sans parenthèse () dans l'appel. Les parenthèses () dans l'appel caractérisent une fonction ; si vous les mettez alors qu'il y a plusieurs arguments, il faut utiliser `Call`. Pour plus de détails sur ces points, voyez le chapitre *Procédures, fonctions, arguments*.

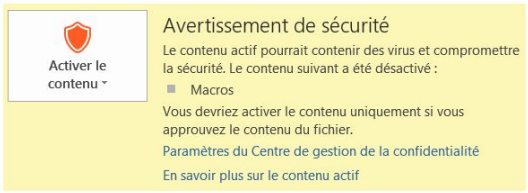
Cette manière de lancer une procédure est dite « méthode interne », mais elle pose question : comment lancer la procédure appelante. On voit qu'il faut des méthodes « externes ».

PAR MENUS STANDARDS

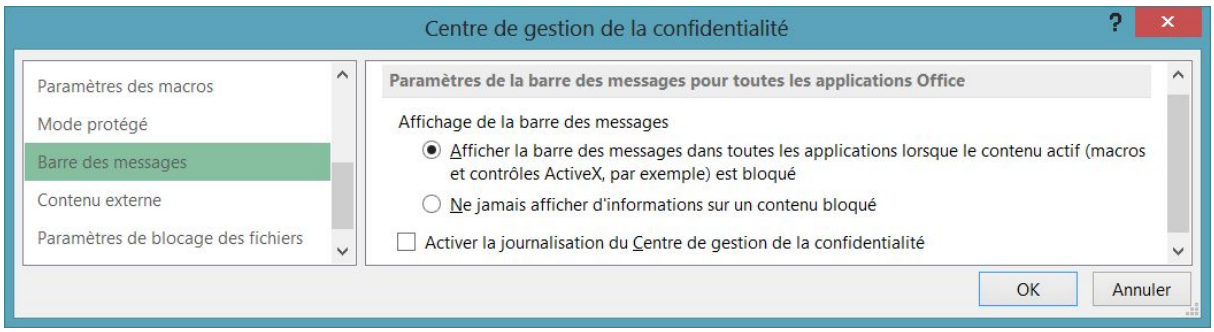
Depuis le classeur Excel

Lorsqu'on ouvre un classeur Excel qui contient des macros, le comportement varie en fonction du niveau de sécurité choisi dans les options. Avec le niveau le plus conseillé, vous avez une barre en haut de l'écran :

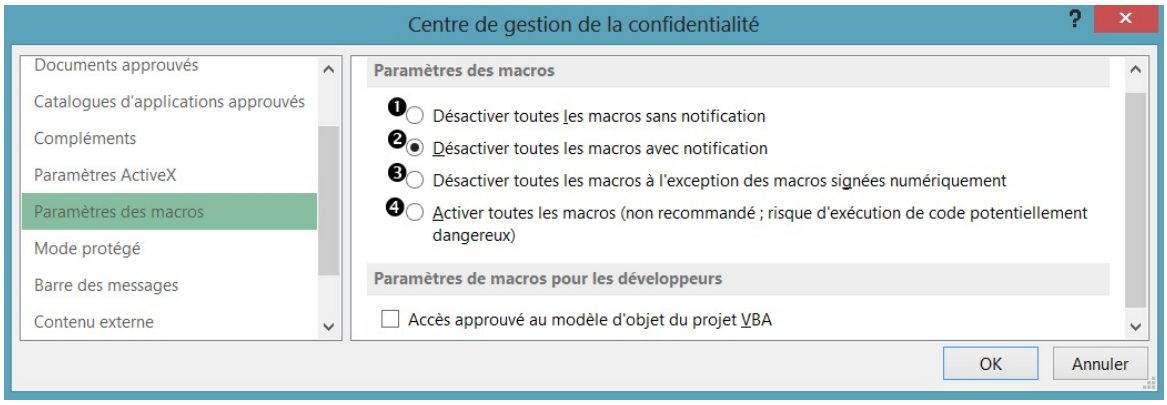
- Si vous actionnez Activer le contenu vous pourrez essayer les macros que vous aurez créées.
- Si vous actionnez *Les macros ont été désactivées*, la BDi de 🗀*FICHIER -Informations* s'affiche avec une partie d'avertissement :



- Actionnez *Paramètres du Centre de gestion de la confidentialité*. vous obtenez la même BDi que celle de 🗀*FICHIER – Options – Centre de gestion de la confidentialité* – Paramètres du Centre de gestion de la confidentialité. Son onglet *Barre des messages* doit être ainsi :



- Son onglet *Paramètres des macros* fixe le niveau de sécurité :

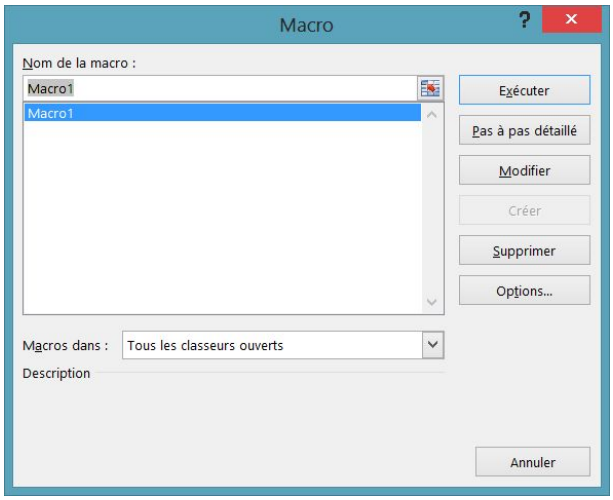


Le comportement que nous préconisons est obtenu avec l'option ❷. L'option ❶ est pour les utilisateurs trop prudents. Si vous n'utilisez que les macros de ce livre, vous pouvez choisir l'option ❸, ce qui vous évitera d'activer les macros à chaque ouverture de classeur. Ce livre ne vous apprendra pas à créer des macros à virus. Donc, le risque évoqué ici ne devrait pas trop nous effrayer. En revanche, avant tout essai de vos « œuvres », il est impératif que vous sauvegardiez le classeur, car il y a un risque réel de blocage de l'ordinateur suite à une erreur dans une macro VBA, même les exemples de ce livre : vous n'êtes pas à l'abri des fautes de frappe.

- Si vous utilisez aussi des classeurs « étrangers », choisissez l'option ❷, car ❹ n'affiche aucune notification.

Ensuite, on peut choisir la procédure à exécuter :

- Faites 🗀*DÉVELOPPEUR – [Code] – Macros* ; la BDi de choix de macro montre la liste de toutes les procédures dans les classeurs ouverts.
- Choisissez dans la liste déroulante *Macros dans* le domaine où chercher les macros, soit l'un des classeurs ouverts, soit tous.
- Cliquez sur la macro/procédure voulue et Exécuter.



Depuis l'éditeur VBA

- Étant dans l'écran de VBA, faites afficher la fenêtre de module voulue si elle ne l'est pas déjà.
- Dans cette fenêtre, placez le curseur texte n'importe où à l'intérieur de la procédure voulue (entre `Sub` et `End Sub`).
- *Exécution – Exécuter Sub/User Form* ou touche de raccourci F5.

Attention : cette méthode fait exécuter la procédure de la même façon que la précédente sauf qu'il peut n'y avoir aucune feuille ni cellule active, alors que depuis la fenêtre Excel, ces éléments étaient définis. Donc si l'écriture de la procédure fait des hypothèses sur ces données, le fonctionnement risque d'être incorrect. Le mieux serait de rendre la rédaction indépendante en ajoutant des instructions pour activer la feuille et la cellule voulues.

PAR ÉVÉNEMENTS

Tout événement (clic, déplacement ou autre) peut être associé à une procédure qui sera exécutée à la survenance de l'événement. Si on fournit une procédure elle sera exécutée à l'arrivée de l'événement avant (ou, si la procédure le spécifie, à la place de) l'action standard du système pour cet événement. Cette action système peut être rien, auquel cas, si vous ne fournissez pas de procédure, votre application sera insensible à cet événement.

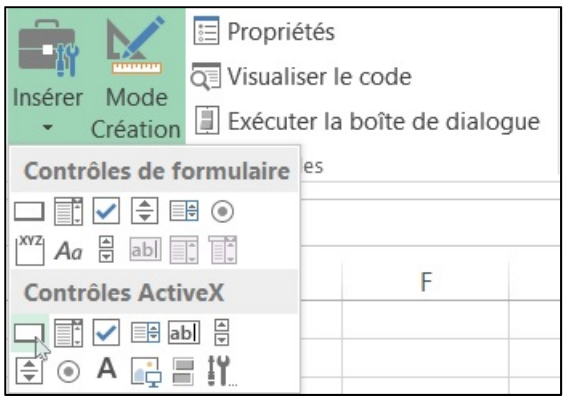
On distingue les *événements naturels* qui arrivent dans tout classeur (ex. changement de valeur dans une cellule, déplacement de la cellule active, activation d'une feuille, ouverture d'un classeur, passage d'un contrôle à un autre dans une BDi, validation d'une BDi…) et les *événements ad-hoc* qui sont introduits uniquement pour démarrer une certaine procédure par un simple clic, ce qui est beaucoup moins fastidieux que la méthode précédente.

Événements ad-hoc

On va créer un élément : bouton, forme géométrique, image, bouton de barre d'outils, nouveau menu ou nouvelle rubrique de menu et à l'événement clic sur cet élément on va associer la procédure que nous voulons lancer facilement. La personnalisation des barres d'outils et menus est discutée dans le chapitre *Commandes par boutons, barres d'outils ou menus*. Ici, nous ne regardons que le cas des boutons ou des dessins.

Pour implanter un contrôle bouton :

- Affichez la boîte à outils *Contrôles* par 🗀*DÉVELOPPEUR – [Contrôles] – Insérer* :



- Choisissez l'outil *Bouton de commande (Contrôle ActiveX)*.
- Le curseur souris prend la forme d'une croix ; délimitez le rectangle du bouton par glissement souris sur la feuille.
- Cliquez droit sur le bouton ; choisissez *Propriétés* dans le menu déroulant. Il apparaît une fenêtre de propriétés analogue à celle de VBA. Le plus indispensable est de changer la propriété *Caption* (libellé qui s'affiche sur le bouton) pour remplacer le libellé passe-partout *CommandButton1* par une mention spécifique du traitement (Ex. *Nouveau Client*…)

- Fermez la fenêtre de propriétés. Nouveau clic droit sur le bouton et *Visualiser le code*. On passe alors à la fenêtre VBA et dans un module intitulé du nom de la feuille de calcul où se trouve le bouton on trouve l'enveloppe d'une procédure *CommandButton1 _Click*. Il suffit d'y taper l'appel de la procédure à associer, c'est-à-dire son nom.
- Quittez le mode création par clic sur 🗀*DÉVELOPPEUR – [Contrôles] – Mode Création*.

Pour implanter un contrôle dessin :

- Faites 🗀*INSERTION – [Illustrations] – Formes*. Cliquez sur un rectangle (vous pouvez aussi choisir l'ellipse ou *Image*).
  - Pour rectangle ou ellipse, délimitez le rectangle conteneur par glissement souris sur la diagonale. Clic droit sur le contrôle, *Ajouter du texte* dans le menu déroulant, tapez le texte voulu. Puis clic droit vous permet d'agir sur la police (nous suggérons de choisir *Gras*) et *Format de la forme* ; regardez plus particulièrement *Zone de texte* qui permet de spécifier l'alignement (nous suggérons *Centré* pour Horizontal et Vertical) et *Remplissage* où nous suggérons de spécifier un remplissage gris clair. Vous pouvez agir aussi sur l'épaisseur de bordure. Le groupe 🗀*FORMAT – [Styles de forme]* offre beaucoup de possibilités.
  - Pour une image faites 🗀*INSERTION – [Illustrations]* et choisissez le fichier voulu dans la BDi qui apparaît. L'image vient en superposition sur la feuille : ajustez sa taille et faites la glisser à l'emplacement souhaité.
- Clic droit sur le contrôle et *Affecter une macro* dans le menu déroulant. Une BDi de choix de macro quasi identique à la figure du début de chapitre apparaît.
- Choisissez la procédure voulue et OK.
  Si au lieu de choisir une procédure existante dans la liste vous gardez le nom proposé d'emblée (exemple : *Rectangle1_QuandClic*), on passe dans l'éditeur VBA ce qui vous permet de taper le contenu de la procédure dans le Module 1.

Événements naturels

Ce sont les événements pour lesquels il n'y a pas besoin de créer un objet à cliquer. Ces événements peuvent se produire d'office. Si vous ne fournissez pas de procédure affectée à un tel événement, c'est l'action normale du système qui prévaut. Si vous fournissez une procédure, elle est exécutée avant l'action système et elle peut éventuellement l'inhiber.

Ces procédures doivent être placées dans la fenêtre de code du module associé au conteneur de l'objet concerné :

- pour un contrôle d'une BDi, c'est le module de code de la BDi.
- pour une cellule ou une zone de feuille de calcul, c'est le module associé à la feuille : vous ouvrez un tel module par double-clic sur *Feuil<n>* dans l'arborescence *Microsoft Excel Objects* ; pour un élément concernant le classeur entier, c'est *ThisWorkbook* dans la même arborescence. Ces fenêtres de code ont en haut deux listes déroulantes.

Pour définir une telle routine, choisissez l'objet dans la liste de gauche, puis la routine dans la liste de droite. Principaux objets et événements :

Conteneur	*Objet*	*Événements*
Feuille	Contrôle	`CommandButton<n>_Click` : clic sur le contrôle (ex. bouton)
"	Worksheet	`WorkSheet_SelectionChange` : on active une autre cellule
"	"	`Worksheet_Change` : on change le contenu de cellule

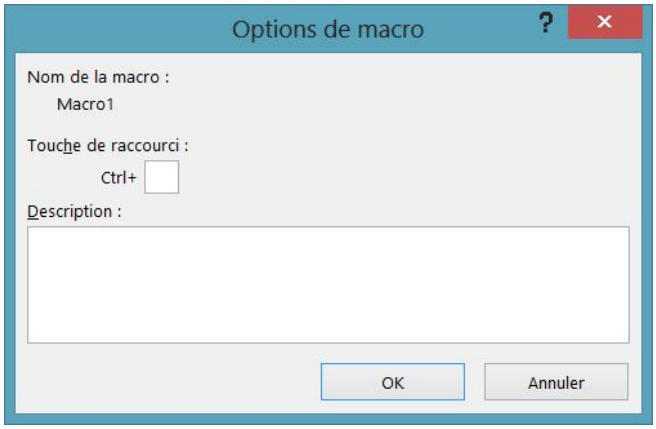
Conteneur	*Objet*	*Événements*
Classeur entier	Workbook	`Workbook_Activate` : activation du classeur
"	"	`Workbook_Open` : ouverture du classeur (*)
"	"	`Workbook_BeforeClose` : avant fermeture du classeur
BDi	Contrôle	`<Contrôle>_Click` : clic sur le contrôle
"	"	`<Contrôle>_Enter` : on arrive sur le contrôle
"	"	`<Contrôle>_Exit` : on quitte le contrôle
"	"	`<Contrôle>_Change` : on change la valeur du contrôle

`(*) Workbook_Open` permet d'implanter un traitement qui se fera dès qu'on ouvrira le classeur ; c'est le moyen d'assurer le **démarrage automatique** d'une application.

PAR RACCOURCI CLAVIER

Une autre solution semble très séduisante : on peut associer une combinaison Ctrl + Touche au déclenchement de l'exécution. On peut spécifier la touche dans la BDi d'enregistrement de la macro. Il faut y penser juste avant l'enregistrement. Si vous n'y avez pas pensé ou s'il s'agit d'une procédure entrée directement par l'Éditeur :

- *DÉVELOPPEUR – [Code] – Macros* : La BDi (de la page 27) apparaît. Attention, il faut demander cette commande depuis la fenêtre Excel, et non VBA ; depuis VBA, la commande *Outils – Macros* fait apparaître la même BDi, mais sans bouton Options.
- Choisissez la procédure voulue.
- Options : La BDi suivante apparaît (elle permet aussi de fournir une description).



L'inconvénient à notre avis rédhibitoire de ce dispositif est que si vous choisissez une combinaison qui a déjà une fonction, celle-ci disparaît et le système ne prévient absolument pas. Vous risquez ainsi de perdre irrémédiablement un raccourci extrêmement important.

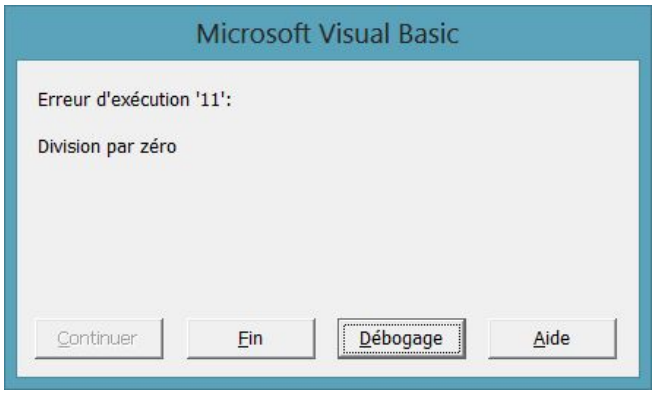
Une alternative plus intéressante est offerte par l'événement `OnKey` de l'objet `Application`. Il offre même plus de possibilités : on n'est pas limité aux combinaisons avec Ctrl et on peut rétablir l'ancienne fonction de la combinaison. Ceci est traité au chapitre 8.

Une fois écrite, la macro ne donne pas forcément du premier coup les résultats souhaités. Différents comportements sont possibles au moment où on demande l'exécution pour un premier essai (redonnons d'ailleurs ce conseil qu'on ne répétera jamais assez : sauvegardez le classeur avant de demander l'exécution) :

- le programme peut s'arrêter avant même d'avoir démarré en signalant une erreur de compilation (1) ;
- le programme s'arrête sur message d'erreur (2) ;
- le programme tourne indéfiniment (3) ;
- le programme s'achève, mais les résultats sont faux ; signalons que pour pouvoir déceler une telle erreur, il faut effectuer certains essais avec des données telles qu'on connaisse d'avance les résultats, ou qu'ils soient facilement calculables (4).

(1) Montre l'instruction en cause surlignée en jaune. Les erreurs de syntaxe concernées sont plus subtiles que celles qui sont décelées à l'écriture ; elles mettent souvent en jeu des incompatibilités entre plusieurs instructions alors qu'à l'écriture, l'analyse se limite à une instruction.
On peut faire apparaître ces erreurs en demandant *Débogage – Compiler VBAProject*.
L'avantage par rapport à l'exécution est que ceci détecte toutes les erreurs de syntaxe alors que l'exécution ne donne que celles des instructions par où on est passé.

(2) Fait apparaître une BDi comme :



et le programme se trouve arrêté. Si le bouton Débogage est présent (il est absent si l'écran VBA n'est pas activé), et si vous cliquez dessus, vous passez à l'affichage du module et l'instruction en cause est surlignée en jaune. Nous verrons plus loin ce qu'on peut faire.

(3) Est vraisemblablement dû à une portion de programme qui boucle. Le plus souvent, on arrive à reprendre le contrôle par la combinaison Ctrl + Pause. On est alors ramené au cas précédent : une des instructions de la boucle en cause est surlignée. On peut donc voir quelle est la boucle infinie et, de là, comprendre si la condition d'arrêt est mal exprimée ou si les données qui y interviennent sont mal calculées.

(4) Est le plus difficile à gérer puisque là, c'est la logique du programme qui est en cause. Les outils à mettre en œuvre sont les mêmes que pour les autres cas.

OUTILS DE MISE AU POINT

Les outils offerts par VBA pour aider à comprendre les erreurs sont, d'une part des moyens d'affichage (infobulles, fenêtre Variables locales, Pile des appels, Espions), d'autre part des moyens d'exécution (Pas à pas, Points d'arrêt, instruction Stop).

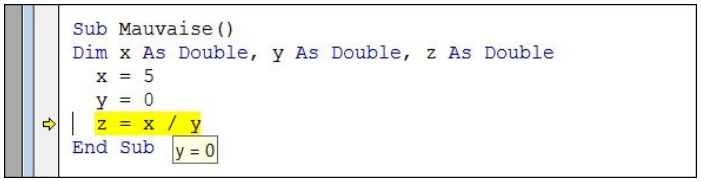
La fenêtre Exécution appartient aux deux catégories puisqu'on peut y afficher des données, mais aussi y taper des instructions. Ces moyens servent plus souvent en mode arrêt, mais certains peuvent être exploités pendant que le programme tourne et ce n'en est que mieux.

MOYENS D'AFFICHAGE

Infobulles

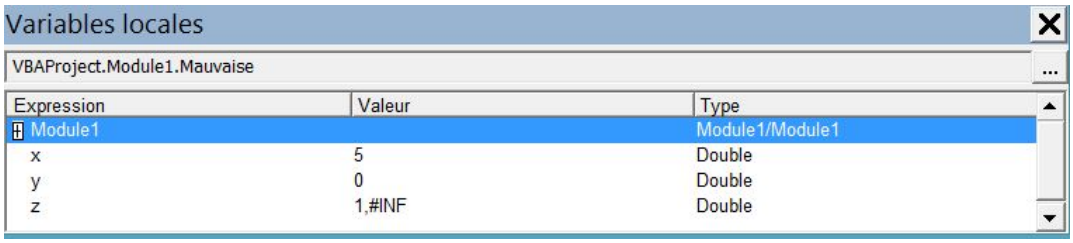
Lorsque le programme est arrêté sur erreur, si vous amenez le curseur souris sur une variable dans la procédure où on se trouve, il apparaît une info bulle qui donne la valeur.

L'exemple de code suivant qui sert à afficher le dialogue que vous voyez au bas de cette page ; si vous amenez le curseur souris sur `y`, vous obtenez une infobulle `y=0` :



Fenêtre variables locales

On l'obtient par *Affichage – Fenêtre Variables locales* dans l'écran VBA. Elle donne la valeur des variables :



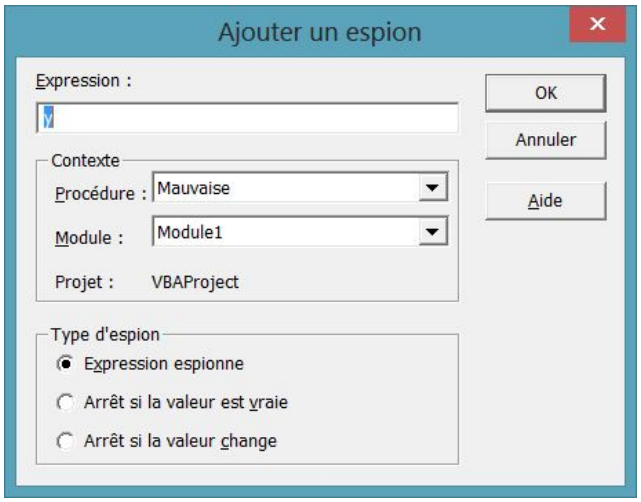
Un point très important est que vous pouvez modifier une valeur dans cette fenêtre : sélectionnez la valeur, modifiez-la puis cliquez ailleurs dans la fenêtre.

Pile des appels

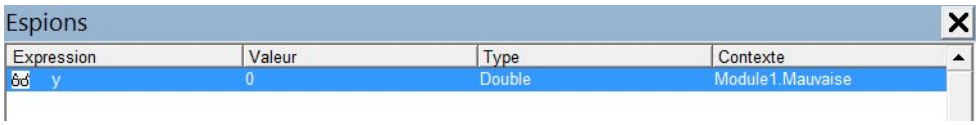
Un clic sur le bouton `...` à l'extrême droite de la ligne VBAProject…, ou *Affichage – Pile des appels* donne une fenêtre qui affiche la succession des appels de procédures. C'est utile dans les cas les plus complexes.

Espions

- Sélectionnez la variable *y*.
- *Débogage – Ajouter un espion*. *y* apparaît comme expression espionne :



Les choix les plus intéressants sont les boutons radio. Ils parlent d'eux-mêmes.



Espion express

Si vous avez oublié de définir un espion avant que le programme ne s'arrête sur erreur, il est encore temps :

- Sélectionnez l'expression voulue.
- Débogage – Espion express.

Un clic sur Ajouter ajoute l'expression comme espion.

MOYENS D'EXÉCUTION

Pas à pas

On peut demander l'exécution pas à pas, c'est-à-dire instruction par instruction. On l'obtient par *Outils – Macro – Macros*, puis Pas à pas détaillé depuis l'écran VBA (🗀 *DÉVELOPPEUR – [Code] – Macros* depuis Excel). Sinon, ayant le curseur souris dans la procédure voulue, demandez *Débogage – Pas à pas détaillé*.

Ceci est extrêmement fastidieux et ne doit être utilisé qu'en dernier ressort si on ne comprend pas la cause de l'erreur. Un peu moins fastidieux sont (*Débogage –*)*Pas à pas principal* (qui exécute les procédures appelées à vitesse normale) et *Pas à pas sortant* (qui fait sortir de la procédure en cours à vitesse normale). En mode pas à pas, on avance d'une instruction par F8.

Points d'arrêt

Il peut être préférable d'introduire quelques points d'arrêt, par exemple avant un passage qu'on voudra surveiller particulièrement. Pour cela :

- Amenez le curseur sur l'instruction voulue.
- *Débogage – Basculer le point d'arrêt* (Raccourci F9). Cette même commande permet d'ailleurs de supprimer le point d'arrêt. Un point d'arrêt apparaît sous forme d'un point bordeaux dans la marge grise.

Supprimer les points d'arrêt

Nous venons de voir comment en supprimer un. Pour supprimer tous les points d'arrêt, c'est *Débogage – Effacer tous les points d'arrêt* (Ctrl+Maj+F9).

Exécuter jusqu'au curseur

Une autre commande qui fait le même effet qu'un point d'arrêt (mais il ne peut y en avoir qu'un) est *Débogage – Exécuter jusqu'au curseur* Ctrl+F8. Il faut bien sûr avoir préalablement placé le curseur dans la fenêtre module sur l'instruction voulue.

Instruction Stop

Les points d'arrêt ne sont pas conservés lorsqu'on sauve le programme. On peut à la place insérer des instructions `Stop` qui font arrêter le programme de la même façon et permettent tout autant d'examiner les variables et les espions.

Que faire après un arrêt ?

Après avoir éventuellement modifié certaines données, on peut :

- continuer pas à pas à coups de F8.
- reprendre l'exécution là où on est ; cela se fait par *Exécution – Continuer* ou F5 ou ▶
- reprendre l'exécution à une autre instruction. Pour cela, il suffit de faire glisser à la souris la flèche jaune qui marque l'instruction où on en est dans la marge grise. Une autre manière est de cliquer sur l'instruction voulue puis *Débogage – Définir l'instruction suivante* ou Ctrl+F9.
- tout remettre à zéro, soit parce qu'on voudra ré-exécuter depuis le début, soit parce qu'on veut abandonner temporairement pour étudier le problème. Cela s'obtient par clic sur ■ ou *Exécution – Réinitialiser* ou clic sur Fin dans la BDi de la figure page 31. Cela peut aussi avoir lieu si vous modifiez le programme : une BDi vous prévient.

La fenêtre Exécution

En fait, la technique moins fastidieuse pour comprendre ce qui se passe dans un programme est de l'exécuter à vitesse normale, mais en insérant par endroits des ordres d'impression de données stratégiques. Pour cela, on peut utiliser `MsgBox`, mais cette instruction crée un arrêt ; exactement ce que nous voulons éviter. La solution est d'utiliser la fenêtre Exécution. Au lieu de `MsgBox <donnée>`, on utilise `Debug.Print <donnée>` et l'écriture se fera dans la fenêtre Exécution, sans causer d'arrêt. Les données à imprimer ainsi peuvent être des valeurs de variables, des textes du genre « On décèle l'événement … », ou « On arrive à la procédure … ».

Pour visualiser la fenêtre Exécution dans l'écran VBA, faire *Affichage – Fenêtre Exécution.* (Ctrl+G).

Le mode immédiat

Une particularité très intéressante de la fenêtre Exécution est que vous pouvez y taper des instructions VBA. Chaque instruction sera exécutée dès que vous taperez Entrée. C'est ce qu'on appelle le mode immédiat.

L'instruction la plus utilisée dans ce contexte est `Print` (abrégé : `?`) `<variable>`. Elle est intéressante car si l'on est en mode arrêt, les valeurs des variables avant l'arrêt sont connues, donc un `?<cette variable>` a autant d'efficacité que les espions et fenêtre Variables locales.

Par exemple, `?ActiveWorkbook.Name` donne le nom du classeur actif ; si ce n'est pas celui que vous avez prévu, vous avez bientôt compris pourquoi le programme ne fonctionne pas.

Vous pouvez aussi taper des instructions qui modifient des valeurs de variables ou des données dans les classeurs, et reprendre l'exécution avec les données modifiées à l'instruction que vous voulez.

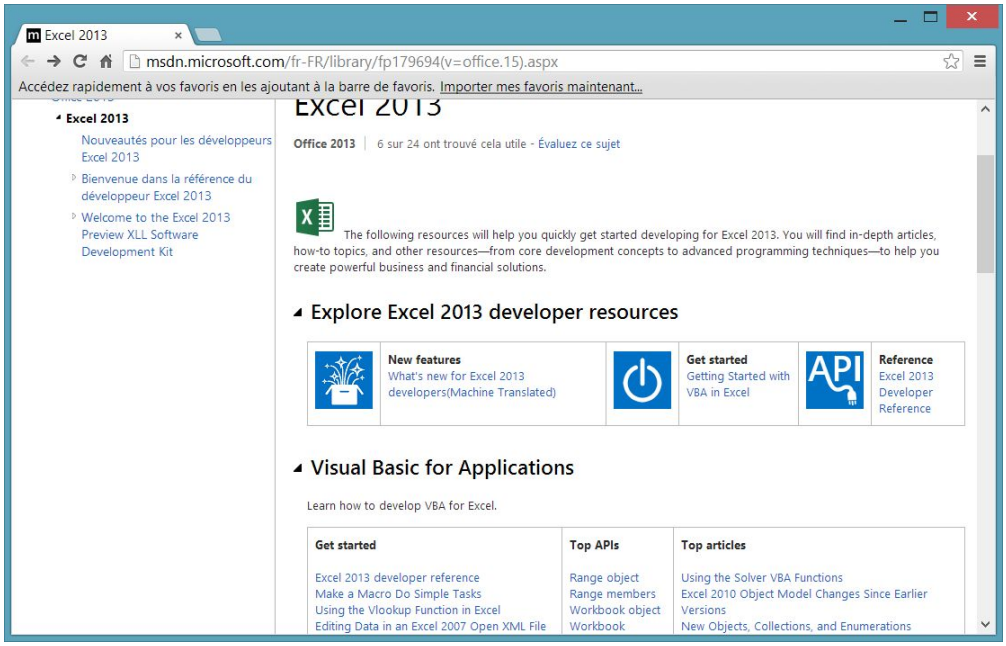
Une autre possibilité de la fenêtre Exécution est qu'elle permet d'essayer des instructions : vous tapez l'instruction à essayer dans la fenêtre Exécution et vous vérifiez les effets.

L'aide en ligne est un élément essentiel. Vous devez l'installer complètement. Si vous appartenez à une organisation où l'installation dépend du « Service Informatique », vous devez obtenir qu'il installe l'aide en ligne.

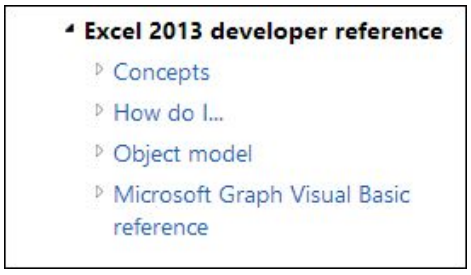
L'aide intervient déjà dans le fait de proposer automatiquement de compléter les instructions lors de leur écriture. De plus, si vous tapez F1 après un mot-clé ou alors qu'il est sélectionné, l'aide sur ce mot-clé apparaît. En outre, les BDi qui apparaissent lors d'un arrêt ont un bouton Aide qui amène à une page en rapport avec le problème.

Appel direct de l'aide

- Vous devez être dans l'écran VBA, sinon, c'est l'aide sur Excel que vous obtiendrez.
- ? – Aide sur Microsoft Visual Basic ou clic sur 



- Appelez ensuite *Excel 2013 Developer reference* qui propose les choix :



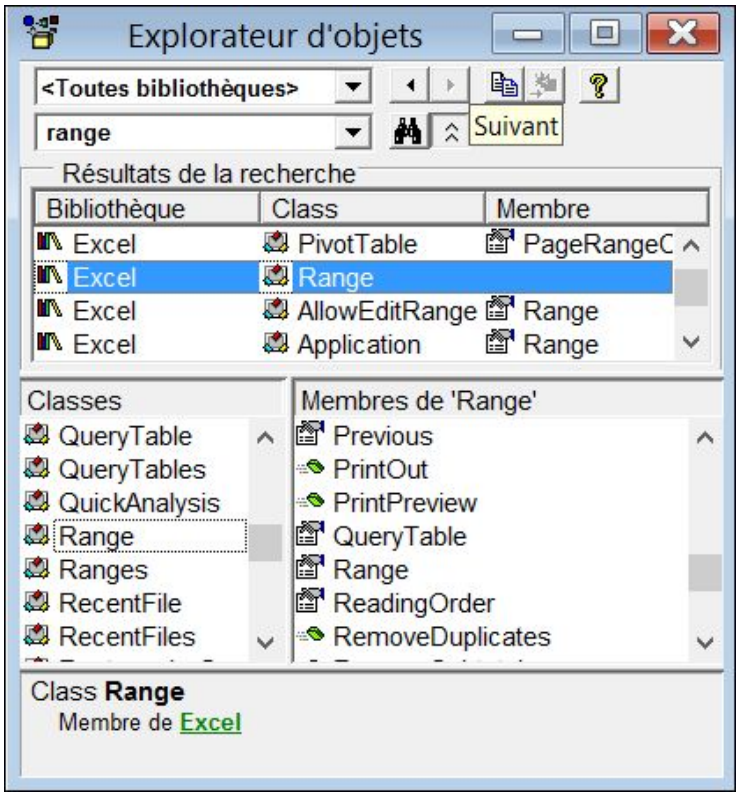
Nous n'insistons pas sur le mode d'emploi de la navigation qui est classique : on développe une arborescence en cliquant sur le livre fermé et on la résorbe en cliquant sur le livre ouvert. Sinon, c'est un hypertexte classique.

La zone d'entrée *Rechercher* sert à taper un mot et le système propose des rubriques ou demande de reformuler la question.

L'EXPLORATEUR D'OBJETS

L'Explorateur d'objets est une extraordinaire source de renseignements, d'autant que la programmation VBA est surtout dépendante des objets de l'application hôte (Excel dans notre cas).

- Dans l'écran VBA, faites *Affichage – Explorateur d'objets* (F2)



- Dans la première liste déroulante, choisir :
  - Soit <Toutes bibliothèques>
    - Tapez le mot cherché dans la 2e liste déroulante.
    - Choisissez ensuite une classe ou un membre sous *Résultats de la recherche*.
    - Vous pouvez alors choisir un membre dans la dernière liste. Le type d'un membre se reconnaît à l'icône devant son nom :

  - Soit une des bibliothèques, par exemple VBA.
    - Choisissez une classe dans la liste *Classes*, puis un membre.

Une fois qu'un élément apparaît tout en bas, en vous avez déjà une description sommaire et, si vous tapez F1, vous aurez un écran d'aide sur cet élément.

RÉCUPÉRATION DES ERREURS

Il est très mauvais d'avoir un programme qui s'arrête sur une erreur, surtout s'il s'agit d'un développement pour un client car les messages du système sont culpabilisants et laissent entendre qu'il y a une erreur de programmation. VBA permet au programme de garder le contrôle en cas d'erreur.

- Juste avant l'instruction (ou le groupe d'instructions) où l'erreur risque de se produire, implantez `On Error GoTo <étiquette>`. Après le groupe, implantez `On Error GoTo 0`.
- Après l'étiquette, on implante la routine de traitement de l'erreur. Elle contient sûrement des instructions `MsgBox` qui préviennent de l'erreur et sont moins rebutantes que le message normal du système.
- En principe, on sait quelle est l'erreur produite puisqu'on connaît les instructions qui suivent le `On Error`… Toutefois, on peut tester `Err.Number` pour vérifier que c'est l'erreur prévue. Par exemple 11 est le numéro correspondant à la division par 0. `Err.Description` est une chaîne décrivant l'erreur.
- La routine doit se terminer par une instruction `Resume` :
  - `Resume` (tout court) fait revenir à l'instruction qui a causé l'erreur. Il faut donc que le traitement ait résolu le problème, sinon, elle se reproduit.
  - `Resume Next` fait revenir à l'instruction qui suit celle qui a causé l'erreur. Donc le traitement remplace celle-ci, ou on y renonce.
  - `Resume <étiquette>` (rarement employé) fait sauter à l'étiquette indiquée.
  - N'oubliez pas d'implanter un `Exit Sub` juste avant l'étiquette du traitement d'erreur, sinon, on tombe inopinément sur ce traitement.

Exemple : On essaie d'ouvrir un classeur ; en cas d'impossibilité, on demande à l'utilisateur de fournir la bonne désignation du fichier. Le retour se fait sur l'instruction d'ouverture, puisque l'erreur est censée être corrigée.

```
Sub Ouvrir()
Dim FN As String
  FN = "C:\ClasseurA.xlsx"
  On Error GoTo TraitErr
  Workbooks.Open Filename:=FN
  On Error GoTo 0
…
…
  Exit Sub
TraitErr:
  FN = InputBox("Impossible d'ouvrir " + FN + _
    vbCr + "Entrez la bonne désignation")
  Resume
End Sub
```

Il y a une autre version page 90 et un autre exemple à la fin du chapitre 5 (page 86).

Gestion d'une association

13

Étape 1 – Fichier HTM

Étape 2 – Nouveau membre

Étape 3 – Modification/Suppression

Pour aller plus loin

ÉTAPE 1 – FICHIER HTM

1. LE PROBLÈME

Nous allons gérer l'association des Amis des Animaux, c'est-à-dire inscrire les nouveaux membres, modifier leurs données, en supprimer, etc. Comme utilisation, nous allons créer la page WEB qui affichera le tableau des membres. D'autres utilisations sont envisageables, comme comptabiliser les cotisations, etc., nous les laissons de côté.

Deux classeurs sont en jeu, conformément au principe de séparation programme-données introduit au chapitre 10 : la base de données, feuille *Membres* du classeur *AmisAnimaux.xlsx* et le classeur *programme*. Dans la première étape, nous produisons le fichier .htm à partir de la BD telle qu'elle est. Les étapes suivantes feront évoluer la base.

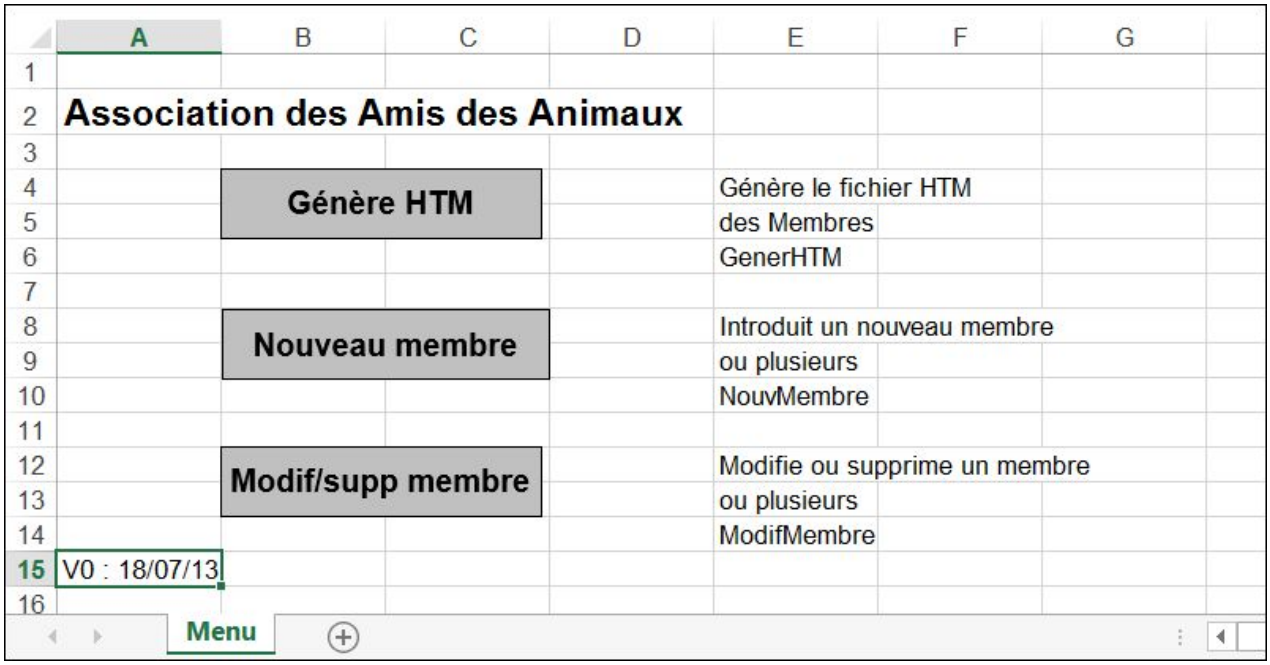
Voici les premières lignes de la BD :

	A	B	C	D	E	F	G	H	I
1		**Association des Amis des Animaux**							
2	**Nom**	**Prénom**	**Adresse 1**	**Adresse 2**	**CP**	**Ville**	**Tel**	**eMail**	**Cotis. à jour**
3	DUCK	Donald	Le Bois Sacré	1 rue du Débarquement	14000	Caen	02 20 10 05 02		
4	DUPONT	Georges	Ker Mag	20 Av Joffre	44500	La Baule	02 40 60 20 00	gdupont@mail.fr	
5	DURAND	Charles	12 rue de la Lune		75003	Paris	06 03 89 78 81	cdurand@yahoo.com	
6	FRICOTIN	Bibi	2 rue de Bellevue		75019	Paris			
7	GUIGNOL	Albert	13 Traboule de la Primatiale		69001	Lyon	04 78 25 00 00		
8	MOUSE	Mickey	Impasse du Fromage		38000	Grenoble		mmouse@noos.fr	
9									

Membres

La rubrique <Cotis. à jour> est prévue, mais elle ne sera pas gérée ici.

Le classeur programme *GestionAssoc0.xlsm* contient une seule feuille nommée *Menu*, qui propose un bouton par fonctionnalité comme suggéré au chapitre 10 :



On voit sur la figure que le classeur est nommé au départ *GestionAssoc0.xlsm* : le 0 est le numéro d'étape, il est rappelé en cellule A15 : n° de version et date. Après l'explication succincte à côté de chaque bouton, figure le nom de la procédure associée. Dans le classeur *GestionAssoc0.xlsm* , ces procédures sont toutes vides.

2. ROUTINE D'INITIALISATION

Nous commençons par l'introduction d'une routine d'initialisation `Init` *et* de quelques variables*:* `InitFait` (l'Init a été exécuté), `Repert` (le répertoire des fichiers), `Sep` (le séparateur \ ou :), `NomFichMemb` (le nom du fichier des membres), `Rdatex` (emplacement où on écrira la date d'exécution devant chaque bouton), `WkMemb` (classeur BD), `ShMemb` (feuille BD), `LdMemb` (ligne de début des membres : 3), `NbRub` (nombre de rubriques traitées : 8, car on ne s'occupe pas de la cotisation) et le tableau libre `NomsRub` (les noms de rubriques).

Les routines qui suivent doivent être saisies dans le module `Module 1`. Si vous avez oublié comment on accède à un module voir la partie Apprentissage pages 9 et 11 : ici, ayant ouvert le classeur *GestionAssoc0.xlsm*, appelez l'Éditeur VBA par Alt+F11 ; le classeur possède un module `Module 1`, donc vous n'avez pas à le créer (il faudrait utiliser *Insertion – Module*).

La routine `Init` initialise ces variables, note la date du jour à côté du bouton appelé et ouvre le fichier des membres ; pour cela, on récupère la fonction `Ouvert` (partie Apprentissage : page 146) afin de ne pas avoir de message d'erreur si le fichier des membres est déjà ouvert. Voici la routine `Init` et le début des autres procédures (remarquez les lignes de commentaires séparatrices) :

```
Public InitFait As Boolean, Rdatex As String, Sep As String
Public Repert As String, NomFichMemb As String, WkMemb As Workbook
Public ShMemb As Worksheet, LdMemb As Integer, NbRub As Integer
Public NomsRub()

'-------------------------------------------------------------Init
Sub Init()
  InitFait = True
  Range(Rdatex).FormulaLocal = Date
  ThisWorkbook.Save
  Repert = ThisWorkbook.Path
  Sep = Application.PathSeparator
  NomFichMemb = "AmisAnimaux.xlsx"
  If Not Ouvert(NomFichMemb) Then _
    Workbooks.Open Filename:=Repert + Sep + NomFichMemb
  Set WkMemb = Workbooks(NomFichMemb)
  WkMemb.Activate
  Set ShMemb = WkMemb.Sheets("Membres")
  LdMemb = 3
  NbRub = 8
  NomsRub = Array("Nom", "Prénom", "Adresse 1", "Adresse 2", "CP", _
     "Ville", "Tel", "eMail")
End Sub

'-----------------------------------------------------------GenerHTM
Sub GenerHTM()
  Rdatex = "A4"
  If Not InitFait Then Init

End Sub
```

```
'-------------------------------------------------------NouvMembre
Sub NouvMembre()
  Rdatex = "A8"
  If Not InitFait Then Init

End Sub

'-------------------------------------------------------ModifMembre
Sub ModifMembre()
  Rdatex = "A12"
  If Not InitFait Then Init

End Sub
```

N'oubliez pas de recopier la fonction `Ouvert` dans le module. Nous avons mis les variables en `Public` car il y aura plusieurs modules dans les étapes suivantes.

Nous sommes prêts maintenant à passer à la construction proprement dite de notre page Web.

3. CONSTRUCTION DU FICHIER .HTM

La structure est très simple : début de la page Web, tableau des membres, fin de la page. Le tableau a lui-même un début et une fin entourant une double structure répétitive pour les lignes (les membres) et les colonnes (les rubriques). Voici le texte HTML représentant le tableau à afficher :

<html><head>	début de la page
<title>Les Amis des Animaux</title>	|
</head><body>	|
Membres de l'Association	|
Les Amis des Animaux	|
<table border width=95%>	début du tableau
<tr><td>nom de rubrique </td></tr>	|ligne d'en-tête
<tr>	|chaque ligne
<td>rubrique</td>	|chaque rubrique
<td>rubrique</td>	|
</tr>	|fin de la ligne
</table>	fin du tableau
</body></html>	fin de la page

Membres de l'Association

Les Amis des Animaux

Nom	Prénom	Adresse 1	Adresse 2	CP	Ville	Tel	eMail
DUCK	Donald	Le Bois Sacré	1 rue du Débarquement	14000	Caen	02 20 10 05 02	
DUPONT	Georges	Ker Mag	20 Av Joffre	44500	La Baule	02 40 60 20 00	Courriel
DURAND	Charles	12 rue de la Lune		75003	Paris	06 03 89 78 81	Courriel
FRICOTIN	Bibi	2 rue de Bellevue		75019	Paris		
GUIGNOL	Albert	13 Traboule de la Primatiale		69001	Lyon	04 78 25 00 00	
MOUSE	Mickey	Impasse du Fromage		38000	Grenoble		Courriel

ÉTAPE 1 – FICHIER HTM

Dans le programme, chaque ligne d'écriture html se fait par un print # de la chaîne de caractères voulue ; on termine par `vbCr` et **;** pour avoir un parfait contrôle des lignes.
Si la rubrique est vide, on met `" "` (l'espace en HTML) pour assurer la continuité de la bordure.
Pour la rubrique eMail, la chaîne est : `"<a href=""mailto:adresse"">Courriel</a>"` : remarquez les doubles guillemets pour incorporer un guillemet.

On introduit les variables locales `Lig` (numéro de ligne), `Col` (numéro de rubrique/colonne) et `Rub` (le texte de la rubrique). On a ajouté en tête de module la directive `Option Base 1`. Le `<b></b>` entourant l'écriture de chaque nom de rubrique, le met en gras.

Voici le fichier *Membres.htm* obtenu à partir des données page 198 grâce à la procédure `GenerHTM` suivante, que vous implantez en tapant le texte en complément des instructions déjà présentes dans la procédure (quasi vide au départ).

```
'-------------------------------------------------------GenerHTM
Sub GenerHTM()
  Dim Lig As Integer, Col As Integer, Rub As String
  Rdatex = "A4"
  If Not InitFait Then Init
  Open Repert + Sep + "Membres.htm" For Output As #1
  Print #1, "<html><head>" + vbCr;  ' Début page
  Print #1, "<title>Les Amis des Animaux</title>" + vbCr;
  Print #1, "</head><body><center>" + vbCr;
  Print #1, "<h2>Membres de l'Association</h2>" + vbCr;
  Print #1, "<h2>Les Amis des Animaux</h2></center>" + vbCr;
  Print #1, "<table border width=95%>" + vbCr; ' Début tableau
  Print #1, "<tr>" + vbCr;
  For Col = 1 To NbRub                ' Noms de rubriques
    Print #1, "<td><b>" + NomsRub(Col) + "</b></td>" + vbCr;
  Next Col
  Print #1, "</tr>" + vbCr;
  For Lig = LdMemb To 1000            ' Les membres
    If IsEmpty(ShMemb.Cells(Lig, 1)) Then Exit For
    Print #1, "<tr>" + vbCr;
    For Col = 1 To NbRub - 1          'Les rubriques
      Rub = CStr(ShMemb.Cells(Lig, Col).Value)
      If Rub = "" Then Rub = " "
      Print #1, "<td>" + Rub + "</td>" + vbCr;
    Next Col
    Rub = CStr(ShMemb.Cells(Lig, NbRub).Value)
    If Rub = "" Then Rub = " " Else _
         Rub = "<a href=""mailto:" + Rub + """>Courriel</a>"
    Print #1, "<td>" + Rub + "</td>" + vbCr;
    Print #1, "</tr>" + vbCr;
  Next Lig
  Print #1, "</table>" + vbCr;
  Print #1, "</body></html>" + vbCr;
  Close 1
End Sub
```

ÉTAPE 1 – FICHIER HTM

Le nom du fichier Web produit est fixé à *Membres.htm* dans l'instruction `Open`. Une telle chose est en principe à éviter : on doit paramétrer au maximum. Dans notre exemple, on pourrait introduire une variable `NomFichWeb` obtenue par une InputBox :

```
NomFichWeb=InputBox("Nom du fichier Web à créer ? ",,"Membres.htm")
```

Nous vous laissons l'implantation complète à titre d'exercice complémentaire.

Après l'ouverture du fichier, une batterie de `Print #` écrit les lignes de début du fichier .htm tel qu'esquissé page 200. On implante la balise de début de tableau et la boucle `For Col`…. remplit la 1re ligne avec les noms de rubrique.

On reconnaît dans la boucle `For Lig =` … le test `If IsEmpty` … du parcours de la partie utile d'une feuille de calculs. Pour une ligne renfermant un membre de l'association, il y aura une ligne du tableau, donc on implante la balise `<tr>` de début de ligne. On a ensuite la boucle sur les rubriques. On termine par la balise `</tr>` de fin de ligne.

Les rubriques sont traitées en deux temps : les `NbRub-1` premières rubriques sont traitées dans la boucle `For Col`… Le contenu trouvé sur la feuille est converti en chaîne. S'il est vide, on inscrira ` ` qui figure un espace : si on ne le faisait pas, on aurait une case vide dans le tableau et la bordure aurait une discontinuité inesthétique (ceci est un problème HTML qui sort du sujet de ce livre). Bien sûr, chaque rubrique est annoncée par la balise `<td>…</td>`.

La dernière rubrique est traitée à part car il n'y a pas qu'à recopier l'adresse eMail, il faut construire le lien en insérant le contenu lu sur la feuille des membres entre les balises `<a>` et `</a>`.

Le programme se termine par les balises de fin de tableau et de fin d'HTML et, surtout, par la fermeture du fichier à ne pas oublier sous peine d'écriture incomplète.

Une fois la frappe finie, faites *Débogage – Compiler VBAProject*. Un certain nombre d'erreurs peuvent vous être signalées : comparez avec le listing ci-dessus et corrigez. Sauvegardez le classeur sous le nom *GestionAssoc1.xlsm*.
Attention, vous ne devez pas écraser le classeur de même nom téléchargé. Vous devez avoir conservé une copie des classeurs originaux téléchargés dans un autre dossier.

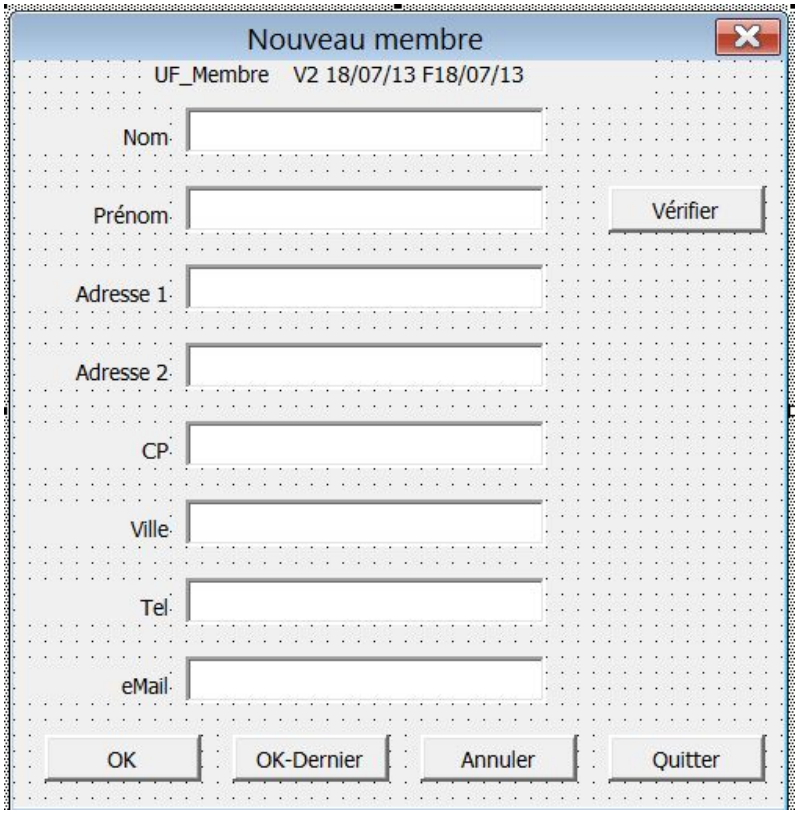
Il vous reste à tester l'exécution, ce qui s'obtient en cliquant sur le bouton Génère HTM de la feuille *Menu*. Si vous n'avez pas fait d'erreur, vous devriez obtenir un fichier *Membres.htm* et la visualisation par votre navigateur doit avoir l'aspect de la figure de la page 198.

ÉTAPE 2 – NOUVEAU MEMBRE

1. CRÉER UNE BDI

Nous passons à la gestion de la base de données, et, d'abord, à l'entrée d'un nouveau membre. Il nous faut donc une BDi pour entrer ses données.

- Faites *Insertion – UserForm*.
- Augmentez un peu la taille et renommez-la `UF_Membre`. Mettez la `Caption =` *Nouveau membre*.
- Créez un Label et une TextBox à côté ; sélectionnez les deux et faites *Copier*.
- Faites Coller 7 fois : vous avez 8 couples (on gère 8 rubriques).
- Sélectionnez les 8 labels et donnez la valeur 3 (droite) à `TextAlign`.
- Donnez aux labels les `Captions` respectives `Nom`, `Prénom` … (les noms de rubriques).
- Créez 5 boutons, les 4 premiers en bas, le 5e à côté du prénom. Donnez les `Name` (et `Caption`) respectifs `B_OK` (`OK`), `B_OKDern` (`OK-Dernier`), `B_Annul` (`Annuler`), `B_Quit` (`Quitter`) et `B-Ver` (`Vérifier`).
- Créez un Label en haut avec `Visible` = `False` et la `Caption = UF_Membre V2 date F date` : ce label apparaîtra au listing alors que le titre de la BDi n'apparaît pas. La BDi doit avoir l'aspect :



Le bouton Vérifier devra être cliqué après avoir entré nom et prénom : le système préviendra si nom et prénom identiques se trouvent déjà dans la base. Les boutons de validation ne seront activés qu'après cette vérification. La dualité OK, Annuler /OK Dernier, Quitter permet d'entrer une série de membres : pour le dernier, on valide par OK-Dernier.

Cette gestion utilise deux booléens `Satisf` et `Dernier` : `Satisf` est vrai si on a validé les données d'un membre, `Dernier` si c'est le dernier de la série. On a en plus une variable `Mode` qui distinguera le cas Nouveau membre du cas Modification, car, par économie, nous utiliserons la même BDi, à peine modifiée.

Ces variables sont publiques, ainsi que `Ligne` (numéro de ligne où s'insèrera le nouveau membre), et le tableau `DonMemb` des données du membre. `Col`, le numéro de rubrique est local aux procédures qui l'emploient.

En résumé, il s'ajoute en tête du module 1 les déclarations :

```
Public Mode As Integer, Satisf As Boolean, Dernier As Boolean
Public Ligne As Integer, DonMemb(8) As String
```

2. PROCÉDURES DE L'USERFORM

Rappelons (partie Apprentissage : page 95) que, pour ouvrir la fenêtre de code du module de l'UserForm, vous tapez F7 (en supposant active la fenêtre objet de l'UserForm). Sinon, vous pouvez toujours utiliser le menu *Fenêtre*. Ce module est essentiellement formé des procédures événements des contrôles de la BDi, mais il peut s'ajouter d'autres procédures si, comme ce sera le cas ici, une même opération est à effectuer à partir de plusieurs contrôles.

Pour implanter une procédure événement, vous pouvez taper `Sub <nomcontrôle>_<événement>`. Mais vous pouvez aussi sélectionner le contrôle dans la liste déroulante à gauche de la fenêtre de code, puis l'événement dans la liste déroulante à droite : `Sub` et `End Sub` sont alors implantées automatiquement sans risque de faute d'orthographe (et avec `Private` en prime). Il s'implante souvent inopinément des routines `_Click`, laissées vides : pensez à les supprimer.

Nous avons d'abord la routine `UserForm_Activate` où nous n'implantons que la branche `Mode=0` : on ne fait que fixer le titre de la BDi et la légende du bouton « Vérifier ».

Lorsqu'on a entré un nom et/ou un prénom, on désactive les boutons « OK », car on doit effectuer la vérification, d'où les deux routines `TextBox1_Exit` et `TextBox2_Exit`.

Les quatre routines des boutons de validation `B_Annul_Click`, `B_OK_Click`, `B_OKDern_Click` et `B_Quit_Click` sont très semblables : avant de fermer la BDi elles fixent en conséquence les booléens sur lesquels est basée la gestion : `Dernier` est mis à vrai pour les boutons qui terminent une série `B_OKDern` et `B_Quitter`. `Satisf` est mis à faux par les boutons d'annulation et à vrai par les boutons OK.

Les boutons OK appellent la procédure `CaptureDon` qui transfère les données des contrôles dans le tableau `DonMemb`. En effet, si on clique sur OK, c'est que les données entrées dans les contrôles sont correctes. Le tableau `DonMemb` sert à les mémoriser pour récupération dans Module 1 qui utilisera la procédure réciproque `EcritDon`. Cette routine ne correspondant pas à un événement, elle doit être tapée entièrement.

La routine `B_Ver_Click` est la plus délicate. Pour le moment, nous n'implantons que la branche `Mode=0`. On commence par exiger que le nom et le prénom aient été fournis, sinon, on ne pourrait rien vérifier. Ensuite, on parcourt toute la partie utile du classeur des membres et, si le nom et prénom de la BDi sont déjà présents, on positionne le booléen `Tr` à vrai.

Si `Tr` est faux, on active les boutons OK. Si `Tr` est vrai, on demande à l'utilisateur s'il veut tout de même entrer ce membre (deux membres peuvent avoir mêmes nom et prénom ; espérons qu'ils n'ont pas la même adresse !) et alors on active aussi les boutons OK. Si la réponse est non, l'utilisateur doit changer le nom et/ou le prénom ou bien annuler.

```
Sub CaptureDon()
  Dim Col As Integer
  For Col = 1 To NbRub
    DonMemb(Col) = Controls("TextBox" + CStr(Col)).Text
  Next Col
End Sub
```

```
Private Sub B_Annul_Click()
  Dernier = False
  Satisf = False
  Unload Me
End Sub

Private Sub B_OK_Click()
  CaptureDon
  Dernier = False
  Satisf = True
  Unload Me
End Sub

Private Sub B_OKDern_Click()
  CaptureDon
  Dernier = True
  Satisf = True
  Unload Me
End Sub

Private Sub B_Quit_Click()
  Dernier = True
  Satisf = False
  Unload Me
End Sub

Private Sub B_Ver_Click()
  Dim Tr As Boolean, Rep
  If Mode = 0 Then
    If (TextBox1.Text = "") Or (TextBox2.Text = "") Then
      MsgBox "Il faut au moins le nom et le prénom"
      Exit Sub
    End If
    Tr = False
    For Ligne = LdMemb To 1000
      If IsEmpty(ShMemb.Cells(Ligne, 1)) Then Exit For
      If (ShMemb.Cells(Ligne, 1).Value = TextBox1.Text) And _
       (ShMemb.Cells(Ligne, 2).Value = TextBox2.Text) Then _
         Tr = True: Exit For
    Next Ligne
    If Tr Then
      Rep = MsgBox("Ce nom et prénom sont déjà présents" + vbCr + _
        "voulez-vous tout de même entrer ce membre", vbYesNo + _
        vbExclamation)
      If Rep = vbYes Then B_OK.Enabled= True: B_OKDern.Enabled= True
    Else
      B_OK.Enabled = True: B_OKDern.Enabled = True
    End If
  Else
  ' laissé vide pour le moment
  End If
End Sub
```

```
Private Sub TextBox1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub

Private Sub TextBox2_Exit(ByVal Cancel As MSForms.ReturnBoolean)
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub

Private Sub UserForm_Activate()
  If Mode = 0 Then
    Me.Caption = "Nouveau membre"
    B_Ver.Caption = "Vérifier"
  Else
  ' laissé vide pour le moment
  End If
End Sub
```

3. PROCÉDURES DE MODULE 1

```
'-------------------------------------------------------EcritDon
Sub EcritDon()
  Dim Col As Integer
  For Col = 1 To NbRub
    ShMemb.Cells(Ligne, Col).Value = DonMemb(Col)
  Next Col
End Sub

'-------------------------------------------------------NouvMembre
Sub NouvMembre()
  Rdatex = "A8"
  If Not InitFait Then Init
  Dernier = False
  While Not Dernier
    Satisf = False
    Mode=0
    UF_Membre.Show
    If Satisf Then
      For Ligne = LdMemb To 1000
        If IsEmpty(ShMemb.Cells(Ligne, 1)) Then Exit For
        If DonMemb(1)+ DonMemb(2) < ShMemb.Cells(Ligne, 1).Value + _
            ShMemb.Cells(Ligne, 2).Value Then
          ShMemb.Activate
          ShMemb.Cells(Ligne, 1).EntireRow.Insert
          Exit For
        End If
      Next Ligne
      EcritDon
    End If
  Wend
  WkMemb.Save
End Sub
```

La structure de `NouvMemb` est en fait simple :

```
While Not Dernier          ' Tant qu'on n'a pas entré le dernier de la série
|    UF_Membre.Show        ' afficher la BDi
|     If Satisf Then       ' si on a obtenu une donnée correcte
|     |    For Ligne       ' chercher où l'insérer
|     |    |…
|     |    Next
|     |    EcritDon        ' insérer la donnée
|     End If
Wend
```

La boucle `For Ligne` suit exactement le schéma de routine que vous trouverez dans la partie Apprentissage, page 144. En effet, on conserve toujours l'ordre alphabétique des membres pour que la génération du fichier .htm qui doit respecter cet ordre soit facile.

Remarquez aussi la sauvegarde du classeur liste des membres. Il n'y en avait pas besoin pour l'étape 1, mais il la faut pour cette étape et la suivante.

La procédure `EcritDon` transfère dans la liste des membres à la ligne voulue par l'ordre alphabétique les données saisies par la BDi et transmises par le tableau `DonMemb`.

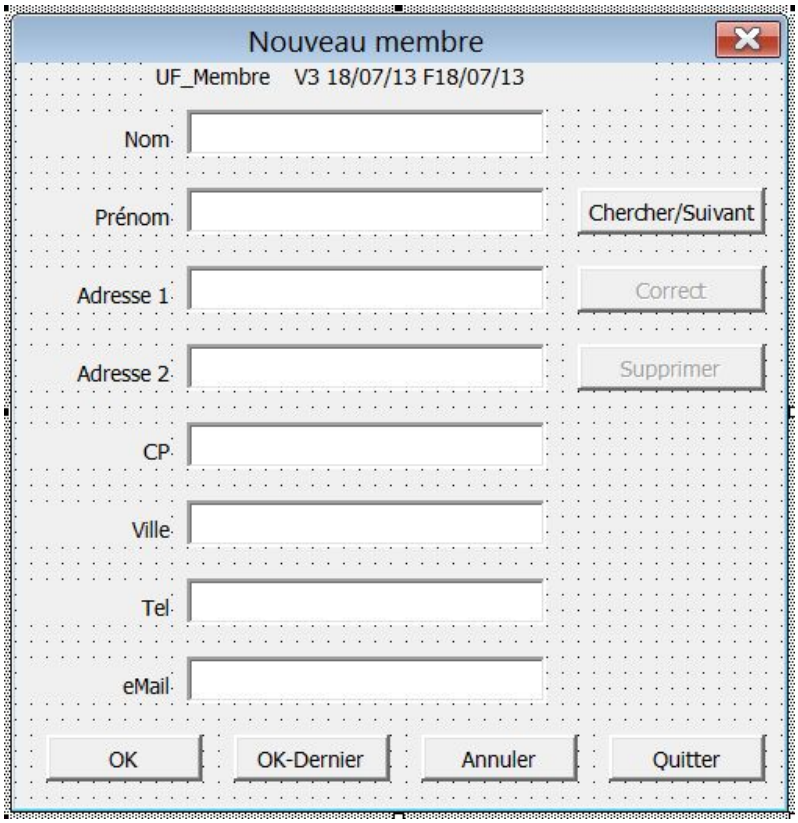
Sauvegardez le classeur sous le nom *GestionAssoc2.xlsm* avec toujours la même précaution d'avoir conservé une copie intacte des classeurs originaux téléchargés. Vous devez aussi avoir une copie à l'abri du classeur *Membres.xlsx* car les essais que vous devez effectuer maintenant vont l'altérer.

Pour tester le programme à l'étape 2, vous cliquez sur le bouton « Nouveau membre », vous entrez une série de nouveaux membres (clic sur OK après chaque et sur OK-Dernier après le dernier) : vous devez vérifier que les données des membres sont bien entrées et sont bien à leur place d'après l'ordre alphabétique.

ÉTAPE 3 – MODIFICATION/SUPPRESSION

1. CONSTRUIRE LA BDI

Pour la modification, le problème est de trouver l'enregistrement à modifier. On fait la recherche sur le nom : lorsqu'on a trouvé une concordance, on affiche l'ensemble des données de l'enregistrement et l'utilisateur doit cliquer sur Correct si c'est l'enregistrement cherché. Sinon, il doit cliquer sur Chercher/Suivant car il peut y avoir plusieurs membres de même nom. Le libellé "Chercher/Suivant" remplace le libellé "Vérifier" ; les deux boutons supplémentaires sont visibles et actifs seulement si `Mode=1`. Dans ce cas, on change aussi le titre de la BDi dans la routine `Activate`. Voici le nouvel aspect de la BDi :



2. PROCÉDURES DE L'USERFORM

Dans la fenêtre de code associée à la BDi, vous avez un certain nombre de routines à modifier, et il s'ajoute les routines de clic des deux boutons supplémentaires, donc, rappelons-le, choix du bouton dans la liste déroulante de gauche et choix de l'événement clic dans la liste de droite. Comme il s'agit d'événements clic, une autre manière d'obtenir l'ouverture de la routine serait de double-cliquer sur le bouton dans la BDi en cours de création/modification.

Le module de l'UserForm a une nouvelle variable, `Ldebut`, numéro de ligne où reprend la recherche si on a plusieurs enregistrements de même nom ; sa déclaration doit être tapée telle quelle.

Les routines des quatre boutons de validation et `CaptureDon` sont inchangées ainsi que les deux routines d'Exit des deux TextBox.

La procédure `UserForm_Activate` a maintenant aussi la branche pour `Mode=1` (sous le `Else`). Même la branche `Mode =0` est à modifier puisqu'il s'ajoute la gestion des activations et visibilité des boutons supplémentaires `B_Correct` et `B_Suppr`. Dans la branche `Mode=1` on initialise `Ldebut` à `LdMemb`, n° de 1re ligne utile dans la liste des membres.

La routine `B_Correct_Click` active le bouton Supprimer et les deux OK puisque le membre sur lequel on veut agir est maintenant trouvé.

`B_Suppr_Click` demande une confirmation et, si oui, effectue la suppression. La variable `Ligne` contient bien le numéro de ligne concernée. Remarquez que, après la suppression de l'enregistrement, on appelle `B_Annul_Click` : en effet, au retour dans le programme appelant, tout doit se passer comme si on avait annulé car la modification du classeur *Membres.xlsx* a été effectuée.

C'est la routine `B_Ver_Click` qui subit les plus importantes modifications. La branche `Mode=0` est inchangée, ce qui prouve la solidité de notre programmation. La branche `Else` commence par protester si le nom n'est pas fourni et on quitte la routine sans quitter la BDi pour permettre à l'utilisateur de le fournir.

Ensuite, démarre la boucle `For Ligne…` pour chercher un membre de ce nom. On interrompt la boucle soit lorsque le nom est trouvé, soit si le parcours de la partie utile de la feuille est terminé. Si le nom est trouvé, on appelle la routine `LitDon` pour afficher les données du membre : l'utilisateur pourra donc décider de cliquer sur Correct ou sur Chercher/Suivant . C'est en vue de cette possibilité de chercher plus loin que l'instruction `Ldebut=Ligne+1` fait que la prochaine recherche démarrera au membre suivant celui où on vient de s'arrêter. Si le nom n'est pas trouvé, un message en avertit l'utilisateur.

La routine `LitDon` écrit dans les contrôles de la BDi les données du membre dont le nom correspond au nom cherché. Si on clique sur Correct, ces valeurs pourront être modifiées et les valeurs modifiées validées puisque les boutons de validation auront été activés par `B_Correct_Click`.

```
Dim Ldebut As Integer
Sub CaptureDon()
  Dim Col As Integer
  For Col = 1 To NbRub
    DonMemb(Col) = Controls("TextBox" + CStr(Col)).Text
  Next Col
End Sub

Sub LitDon()
  Dim Col As Integer
  For Col = 2 To NbRub
    Controls("TextBox" + CStr(Col)).Text = ShMemb.Cells(Ligne, Col)
  Next Col
End Sub

Private Sub B_Annul_Click()
  Dernier = False
  Satisf = False
  Unload Me
End Sub
Private Sub B_Correct_Click()
  B_Suppr.Enabled = True
  B_OK.Enabled = True
  B_OKDern.Enabled = True
End Sub
```

```
Private Sub B_OK_Click()
  CaptureDon
  Dernier = False
  Satisf = True
  Unload Me
End Sub

Private Sub B_OKDern_Click()
  CaptureDon
  Dernier = True
  Satisf = True
  Unload Me
End Sub

Private Sub B_Quit_Click()
  Dernier = True
  Satisf = False
  Unload Me
End Sub

Private Sub B_Suppr_Click()
  Dim Rep
  Rep = MsgBox("Etes-vous sûr de vouloir supprimer ce membre ? ", _
    vbYesNo + vbQuestion)
  If Rep = vbYes Then
    ShMemb.Activate
    ShMemb.Cells(Ligne, 1).EntireRow.Delete
    B_Annul_Click
  End If
End Sub

Private Sub B_Ver_Click()
  Dim Tr As Boolean, Rep
  If Mode = 0 Then
    If (TextBox1.Text = "") And (TextBox2.Text = "") Then
      MsgBox "Il faut au moins le nom et le prénom"
      Exit Sub
    End If
    Tr = False
    For Ligne = LdMemb To 1000
      If IsEmpty(ShMemb.Cells(Ligne, 1)) Then Exit For
      If (ShMemb.Cells(Ligne, 1).Value = TextBox1.Text) And _
       (ShMemb.Cells(Ligne, 2).Value = TextBox2.Text) Then _
         Tr = True: Exit For
    Next Ligne
    If Tr Then
      Rep = MsgBox("Ce nom et prénom sont déjà présents" + vbCr + _
        "voulez-vous tout de même entrer ce membre", vbYesNo + _
        vbExclamation)
      If Rep = vbYes Then B_OK.Enabled= True: B_OKDern.Enabled= True
    Else
      B_OK.Enabled = True: B_OKDern.Enabled = True
    End If
```

```
  Else
    If (TextBox1.Text = "") Then
      MsgBox "Il faut fournir le nom"
      Exit Sub
    End If
    Tr = False
    For Ligne = Ldebut To 1000
      If IsEmpty(ShMemb.Cells(Ligne, 1)) Then Exit For
      If (ShMemb.Cells(Ligne, 1).Value = TextBox1.Text) Then _
         Tr = True: Exit For
    Next Ligne
    If Tr Then
      LitDon
      Ldebut = Ligne + 1
    Else
      MsgBox "Nom non trouvé"
    End If
  End If
End Sub

Private Sub TextBox1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub

Private Sub TextBox2_Exit(ByVal Cancel As MSForms.ReturnBoolean)
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub

Private Sub UserForm_Activate()
  If Mode = 0 Then
    Me.Caption = "Nouveau membre"
    B_Ver.Caption = "Vérifier"
    B_Correct.Enabled = False
    B_Correct.Visible = False
    B_Suppr.Enabled = False
    B_Suppr.Visible = False
  Else
    Me.Caption = "Modification/Suppression membre"
    B_Ver.Caption = "Chercher/Suivant"
    B_Correct.Enabled = True
    B_Correct.Visible = True
    B_Suppr.Enabled = False
    B_Suppr.Visible = True
    Ldebut = LdMemb
  End If
End Sub
```

3. PROCÉDURE MODIFMEMBRE

```
'-----------------------------------------------------ModifMembre
Sub ModifMembre()
  Rdatex = "A12"
  If Not InitFait Then Init
  Dernier = False
  While Not Dernier
    Satisf = False
    Mode = 1
    UF_Membre.Show
    If Satisf Then
      EcritDon
    End If
  Wend
  WkMemb.Save
End Sub
```

La structure est encore plus simple que `NouvMemb` car on n'a pas à chercher dans quelle ligne insérer les données, puisque le numéro de ligne concerné est trouvé lors de la recherche du nom et conservé dans la variable Ligne. La différence importante est la valeur donnée à la variable `Mode` : une erreur ou l'oubli de cette instruction empêcherait le fonctionnement correct.

Vous devez sauvegarder le classeur auquel nous sommes parvenus sous le nom *GestionAssoc3.xlsm* avec les mêmes précautions de conservation d'une copie intacte des originaux des classeurs téléchargés. Ensuite, vous cliquez sur le bouton et modifiez quelques données, puis vous examinez *Membres.xlsx* pour vérifier que les modifications sont bien entrées et sont à la bonne place.

POUR ALLER PLUS LOIN

Voici quelques directions de possibles améliorations :

Offrir un système d'Aide

Offrir un système d'aide. Il faut bien sûr créer les fichiers .htm voulus ; ce n'est pas le sujet de ce livre. Ensuite, il faut fournir au moins un bouton dans la feuille *Menu* du classeur programme et un bouton dans la BDi *UF_Membre*. Nous avons vu (partie Apprentissage : page 152) comment écrire les routines de clic de ces boutons.

Gérer des rubriques supplémentaires

C'est simple en s'inspirant des routines écrites pour les rubriques en place.

Ajouter des fonctionnalités

Par exemple faire un système de relance des adhérents en retard de cotisation ; il faudrait ajouter la rubrique date de dernière cotisation… C'est utile pour toutes les associations.

Dictionnaire de données

(Ceci plutôt à titre d'exercices de programmation) Gérer les placements des entrées de BDi par les Tags des contrôles et introduire dans les BD une feuille dictionnaire de données pour avoir une gestion capable de s'adapter à un déplacement des rubriques dans leur classeur.

Améliorer l'ergonomie

Si on clique sur Chercher/Suivant une fois de trop, le nom ne sera pas trouvé et il faudra reprendre la recherche au début. Il serait plus ergonomique d'implanter un bouton Précédent permettant des allers et retours.

Maintenant, quelques leçons à retenir de cette étude de cas (et de toutes) :

- Complémentarité de tous les éléments d'un projet : les instructions sont écrites en fonction de la structure des données dans les classeurs BD ; les procédures événements liés aux contrôles de BDi et les appels des BDi sont écrits en fonction les uns des autres et les transmissions de données doivent être prévues… Autre comportement de la BDi, autre façon de l'utiliser.
- Confirmation de l'intérêt du principe de séparation programme-données. On pourrait protéger les classeurs BD par mot de passe et faire qu'on ne puisse y accéder que par le programme (dont le texte devra être interdit de consultation, sinon, on pourrait lire les mots de passe : cela s'obtient par *Outils – Propriétés de projet*, onglet *Général* et ☑ *Verrouiller le projet pour l'affichage* (partie Apprentissage : page 20).
- Par ailleurs, un avantage du fait que nos BD soient des classeurs Excel est que l'on dispose d'un moyen indépendant de notre programme de les examiner et donc de vérifier ce que fait notre programme : ce sont les simples commandes d'Excel. Bien sûr, cet accès joue pendant la phase de mise au point, quand les classeurs ne sont pas encore protégés.