Programmation VBA

Pierre BONNET

La programmation VBA

↳ Historiquement, la programmation sous Excel avait comme fonction d'automatiser une succession d'actions faites dans la feuille à l'aide de la souris.

D'où la notion de Macro Excel

Le langage initial était spécifique (versions en Anglais et en Français)

↳ A partir d'Excel 97, abandon du langage spécifique pour un langage unique adoptant la syntaxe du Basic :

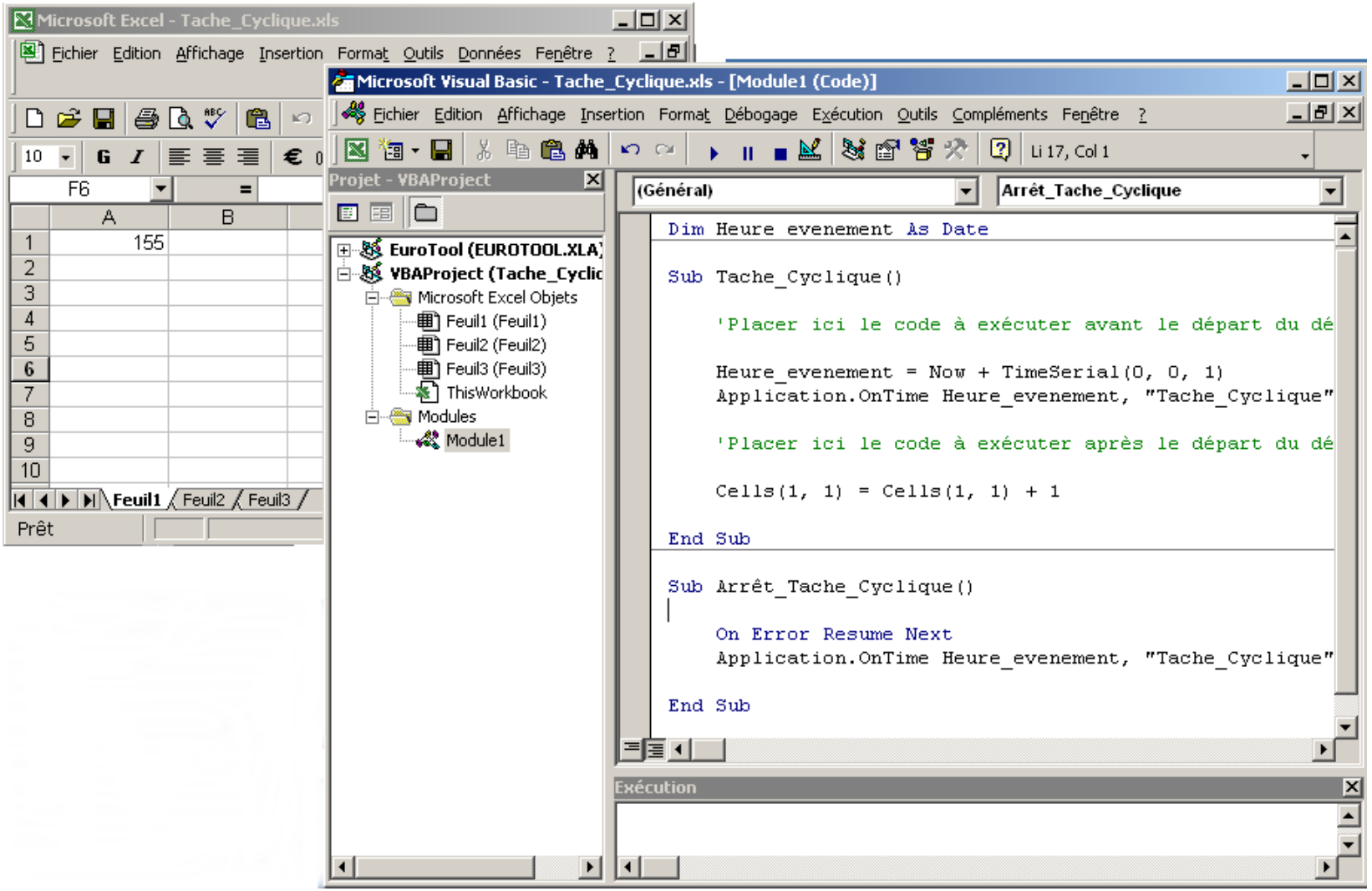
Visual Basic for Application VBA

C'est un enrichissement de VB par des fonctions spécifiques à chaque application (Excel, Word, Access....).

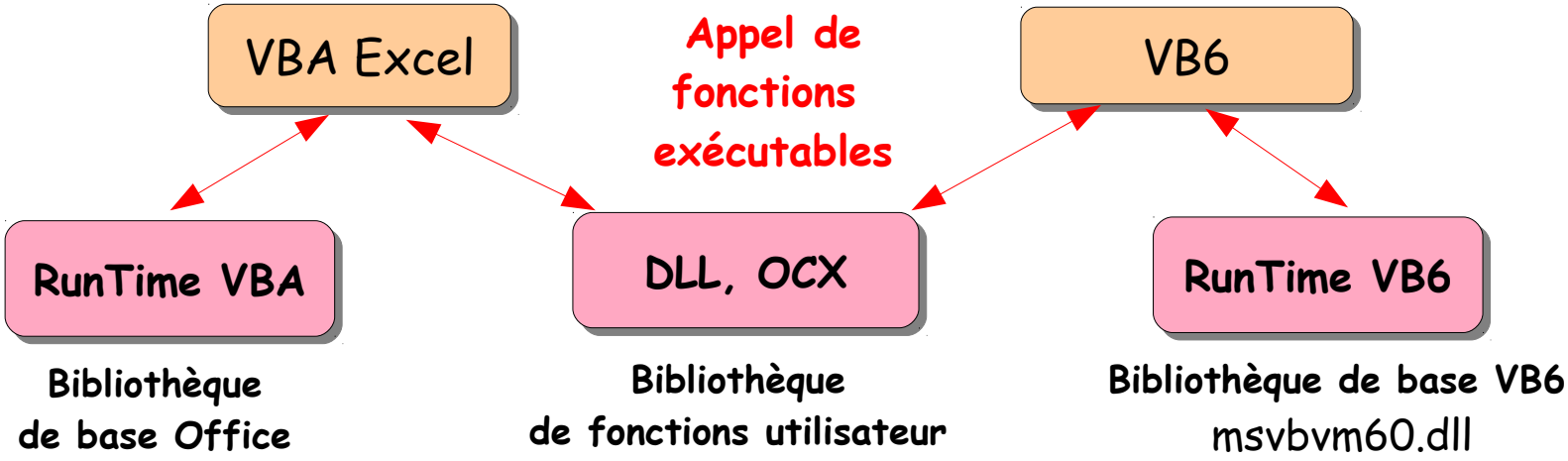
↳ Excel comprend tous les outils d'écriture et d'exécution de VB, y compris les possibilités d'extension avec des bibliothèques ou "contrôles" supplémentaires.

↳ La différence essentielle est que l'affichage se fait dans les feuilles d'un classeur

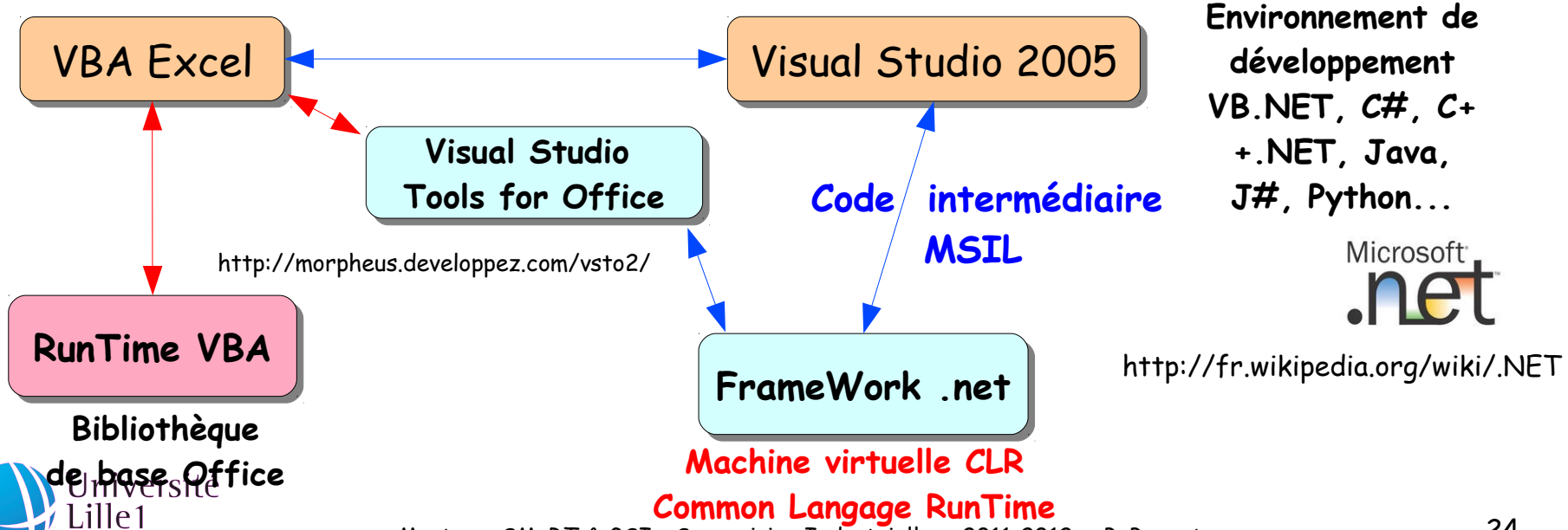
Accès à la programmation VBA → Alt F11



VBA/VB6



FrameWork .NET



Variables

Déclaration implicite

Par défaut, l'usage d'une variable dans une affectation tient lieu de déclaration (comme en Matlab)

Le type affecté est déterminé automatiquement par l'évaluation de la partie droite de l'évaluation

Danger de l'absence de déclaration : toute erreur de frappe du nom d'une variable crée une nouvelle variable!

Déclaration explicite

Utilisez la directive en début de programme `Option Explicit`

Toute déclaration a la structure : `Dim Nom_Variable as Type_Variable`

Portée de la déclaration

Une variable n'est visible que pour la feuille où elle est déclarée .

Pour une visibilité des autres feuilles/classeur, utiliser `Public Nom_Variable as Type_Variable`

Particularité : La notion de pointeur n'existe pas en VB/VBA d'où communication impossible avec des fonctions qui utilisent cette approche pour le dialogue

Variables

- **Boolean**

Prend les valeurs True et False. Par conversion depuis d'autres types numériques, la valeur 0 devient False et toutes les autres deviennent True. Dans le cas inverse, False devient 0 et True devient -1.

Conversion d'une donnée numérique en Boolean par la fonction Cbool

exemple : `A = CBool(10)  'A prend la valeur  True.`

- **Byte, Integer, Long**

Types de données entiers 8, 16 ou 32 bits .

Conversion d' une donnée numérique en Byte, utilisez la fonction CByte, idem pour les autres types .

Attention : l'adressage des lignes xls doit se faire en type long (valeur éventuellement supérieure à 32 767)

- **Single, Double**

Caractérise les nombres réels (ou flottants) simple et double précision

Variables

- **String** * Les chaînes de longueur fixe peuvent contenir de 1 à 2^16 caractères.

```
Dim ma_chaine as string
ma_chaine = "Bonjour"
MsgBox ma_chaine
```

Ou bien `Dim ma_chaine$`

Il existe en VBA toutes les fonctions de conversion nombre <--> chaîne

voir `str` et `val`

concaténation de chaîne : `ma_chaine = "Bon" + "jour"`

`ma_chaine = "Amplitude :" & val(signal)`

- **Tableaux** Tous les types peuvent être étendus en tableau

```
Dim Tableau1(10) as double
Tableau(3) = 3.14159

Dim MatriceXY(5,7) as Boolean      'dimension 60 au maximum
MatriceXY(2,3) = True
```

L'indice de début est implicitement 0 par défaut , 1 en utilisant la déclaration `Option Base 1` ou en spécifiant les indices :

```
Dim MonTableau(1 To 5, 10 To 20) As String
```

Recherche des indices min et max par les fonctions `LBound` et `Ubound`

```
NbElements = (Ubound(MatriceXY,1)-LBound(MatriceXY,1)+1)*(Ubound(MatriceXY,2)...
             ...  -LBound(MatriceXY,2)+1)
```

Tableau dynamique avec redimensionnement

```
Dim Karnaugh() as boolean
ReDim Karnaugh(20)
```

attention, les anciennes valeurs sont effacées (utiliser `ReDim Preserve`)
le type ne peut pas être modifié (sauf dans le cas d'un tableau de `variant`)

Le type Array permet de créer une liste de valeurs [de type tableau de variant]

```
MonTableau = Array("a", "b", "c")
```

Variables

- **Variant**

Variant est le type de données attribué à toutes les variables qui ne sont pas explicitement déclarées comme étant d'un autre type (à l'aide d'instructions telles que Dim, Private, Public ou Static). Le type de données Variant ne possède aucun caractère de déclaration de type. Variant est un type de données spécial pouvant contenir des données de toutes sortes, à l'exception des données de type String de longueur fixe.

La valeur Empty désigne une variable de type Variant qui n'a pas été initialisée (c'est-à-dire à laquelle aucune valeur initiale n'a été affectée). Un variant peut aussi contenir la valeur Null ou Error

Lorsqu'un variant contenant une chaîne est utilisé dans une opération requérant un type numérique, le variant est automatiquement converti en nombre.
Exemple:

```
Dim MyVar As Variant
MyVar = "20"
Cells(1, 1) = MyVar + 10
```

La valeur affichée dans la cellule est 30

Variables

- **Date (très important pour la gestion des données industrielles horodatées)**

Les dates sont comprises entre le 1er janvier 1900 et le 31 décembre 9999, les heures allant de 0:00:00 à 23:59:59.

Le type date est stocké dans un *double*. La partie entière correspond au nombre de jours écoulés depuis le 1er Janvier 1900; la partie fractionnaire correspond à la fraction de jour écoulée (1h = 1/24 de jour, 1mn = 1/(24x60) de jour , 1s =1/(24x60x60) de jour).

Exemple :

```
Dim ma_date As Date
Dim mon_heure, ma_dateheure As Double
ma_date = "16/09/2008"
mon_heure = 0.5  'une demi-journée de plus soit 12H
ma_dateheure = ma_date + mon_heure
Cells(3, 1) = ma_dateheure
```

donne comme résultat 16/09/08 12:00 PM (dépend du format d'affichage choisi)

- la conversion de type est implicite depuis le format chaîne.
- utilisez la fonction IsDate pour savoir si une valeur *variant* peut être convertie en date ou en heure.
- VBA propose des fonctions de conversion (DateSerial , TimeSerial)

Variables

Type utilisateur

Un type utilisateur définit une structure contenant les types de base.

Exemple:

Déclaration du type

```
Public Type FicheEtudiant
    Nom  As String*30          'variable type String  contenant le nom.
    DateNaissance As Date      'variable type Date contenant la date de naissance .
    Sexe As  Boolean           'variable type Boolean définissant le sexe
End Type                         (False=fém,True=masc.).
```

Usage du type :

```
Dim NewEtudiant  as  FicheEtudiant
NewEtudiant.Nom = "Dupont"
NewEtudiant.DateNaissance = "25/10/82"
NewEtudiant.Sexe = True
```

ou bien :

```
Dim NewEtudiant  as  FicheEtudiant
With NewEtudiant
        .Nom = "Dupont"
        .DateNaissance = "25/10/82"
        .Sexe = True
End With
```

Variables

- **Type objet**

Un objet [ou classe] est une variable structurée permettant de manipuler des concepts évolués. Les valeurs associées à un objet s'appellent des paramètres , les fonctions appliquées à un classe sont des méthodes

Pour pouvoir utiliser un objet, il faut commencer par le déclarer puis l'instancier :

```
Dim monObjet As MaClasse
Set monObjet = New MaClasse
monObjet.param1 = 10
monObjet.Afficher ...
```

Ou

```
Dim monObjet As New MaClasse
...
```

Lorsque l'objet n'est pas instancié mais seulement déclaré, il est alors équivalent à "Nothing" .

Destruction d'un objet `Set monObjet = Nothing`

Il existe de nombreux objets prédéfinis dans Excel : `WorkBook, Range`

Variables

- **Hiérarchie des principaux objets Excel**

Application
    Workbooks
        WorkBook
            Worksheets
                WorkSheet
                    Range, Cells

Exemple de référencement :

```
Excel.Workbooks(1).Sheets(1).Range("A1").Value = "salut"
```

L'objet Cell n'existe pas au sens strict en Excel

Référence : http://rdorat.free.fr/Enseignement/VBA/VBA/ObjExcel.html

Instructions de contrôle

```
If (condition) Then
    instruction 1
    instruction 2
    ...
Else
    instruction 1
    ...
End If
```

```
If (condition) Then
    instruction 1
    ...
Elseif (condition) Then
    instruction 1
    ...
End If
```

```
For (variable) = (début) To (fin)
    instruction 1
    ...
Next [nom de la variable d'incrément]
```

Expression des conditions

- opérateur de comparaison

```
a = b     a <= 10   a <> 3

ma_chaine = "Bonjour"
```

- opérateur logique

```
(a = b) AND (a >= 10)

( a = b)  OR  NOT( a <= 10)

( a <> b ) XOR (a = c)
```

Instructions de contrôle

- ```
  Do [while|until] (condition)
      instruction 1
      ...
  Loop
  ```
  ou
  ```
  While (condition)
      instruction 1
      ...
  Wend
  ```

- ```
  Do
      instruction 1
      ...
  Loop [while|until] (condition)
  ```

- ```
  Select Case (variable)
      case valeur1
         instruction 1
         ...
      case valeur2
         instruction 1
         ...
      case valeur3 To valeur4
         instruction 1
         ...
      case Is >= valeur5
         instruction 1
         ...
      case else
         instruction 1
         ...
  End Select
  ```
  ```

Macros

- Une macro est une procédure dont l'exécution est lancée depuis :
  - le menu Outils/Macro/Exécuter
  - par un raccourci clavier
  - par un bouton de la barre d'outils formulaire
  - une image
  - un bouton de la barre d'outils

↳ Le code VBA d'une macro peut être généré automatiquement par l'enregistreur de macro

Procédures

- Une procédure commence par `Sub` et termine par `End Sub`. Elle exécute des actions mais ne renvoie pas de valeurs .

Exemple sans argument: procédure de recopie la cellule active 1 ligne vers le bas et 0 colonne vers la droite, puis sélection de cette nouvelle cellule

```
Sub Copie_Decale()
      ActiveCell.Copy ActiveCell.Offset(1,0)
      ActiveCell.Offset(1,0).Select
End Sub
```

Exemple avec argument d'entrée: les références de la cellule à décaler sont passées en paramètre de type chaîne .

```
Sub Copie_Decale_bis(AdressePlage As String)
     Dim Ma_Plage As Range
     Set Ma_Plage = Range(AdressePlage)
     Ma_Plage.Copy Plage.Offset(1,0)
     Ma_Plage.Offset(1,0).Select
End Sub
```

Pour exécuter cette procédure, taper Copie_Decale_bis("C3") dans la fenêtre **Exécution**

Fonctions

- Une fonction accepte un ou plusieurs arguments en entrée et retourne un argument .

Elle commence par **Function** et se termine par **End Function**

Exemple: fonction retournant le code couleur de la cellule dont les coordonnées sont passées en paramètre de type *chaîne*. La fonction retourne une *chaîne* .

```
Public Function Code_Couleur(AdressePlage As String) As String
  Dim Ma_Plage As Range
  Dim Index as integer
  Set Ma_Plage = Range(AdressePlage)
  Index = Ma_Plage.Interior.ColorIndex
  Code_Couleur = "Couleur de la cellule " & AdressePlage & " : " &
                    str(Index)
End Function
```

Pour exécuter cette fonction, taper `range("B3") = Code_Couleur("A3")` dans la fenêtre **Exécution**. Le résultat est donné dans la cellule B3 .

Procédures/Fonctions

- Passage de paramètres

Par valeur (appel par défaut): C'est la valeur de la variable qui est passée dans la fonction. Tout calcul sur cette valeur est **local** à la fonction et n'affecte pas la valeur dans la structure appelante.

```
a = 5
increment(a)
MsgBox a                'la valeur affichée est 5
...
Sub increment (ByVal var1 as Integer)
    var1 =var1 + 1      ' var1 prend la valeur 5
End Sub
```

Par référence : le paramètre passé est l'adresse de la variable. Il en résulte que toute modification de la valeur dans la fonction affecte la valeur de la variable dans la structure appelante

```
a = 5
increment(a)
MsgBox a                'la valeur affichée est 6
...
Sub increment (ByRef var1 as Integer)
    var1 =var1 + 1      ' var1 pointe  sur a
End Sub
```

Exercice

- Faire le calcul de la variance d'un tableau de données Excel et l'afficher dans une boite de dialogue (voir suite transparents). L'interface de saisie est une feuille Excel

Le contenu du tableau est rangé à partir de la cellule A5 , le nb de lignes en B2 et le nombre de colonnes en B3.

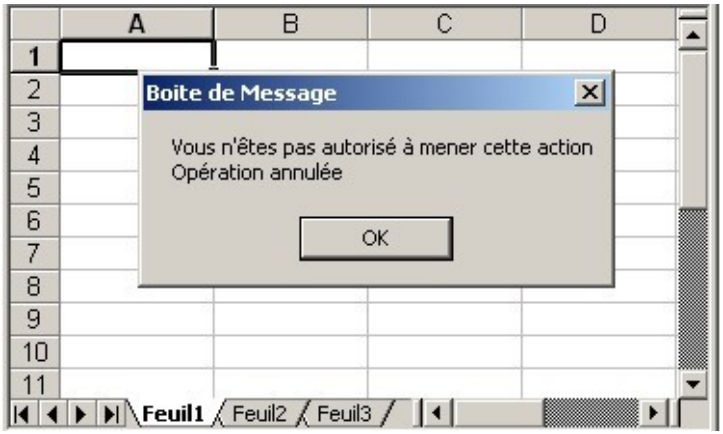
1) Faire un programme basique de calcul en VBA comprenant la récupération des valeurs, le calcul puis l'affichage

2) Isoler le calcul dans une ***fonction STD(mon_tableau)*** dont le paramètre d'entrée est le seul nom de variable du tableau (utiliser un tableau dynamique, rechercher les dimensions du tableau d'entrée et appliquer le calcul)

	A	B	C
1	Mon Tableau	A5	
2	Nb lignes	3	
3	Nb colonnes	2	
4			
5	10	12	
6	10	12	
7	10	12	
8			
9			

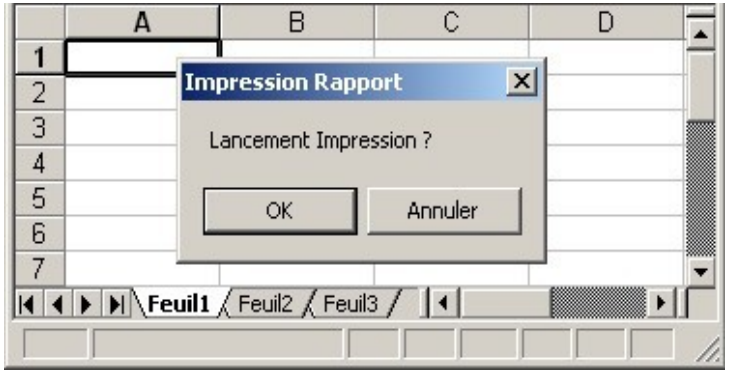
3) On souhaite que le tableau soit directement sélectionné à la souris. Utiliser les propriétés de l'objet Selection pour récupérer les données.

Autres Fonctionnalités



- **Envoyer un message d'avertissement**

```
Dim Mon Message as string
 MonMessage = "Vous n'êtes pas autorisé
   à mener cette action"+Chr$(13)+
   "Opération annulée"
MsgBox MonMessage
```



- **Envoyer un message avec réponse type choix**

```
Dim MonMessage, Titre As String
Dim Saisie As Integer

MonMessage = "Lancement Impression ?"
Titre = "Impression Rapport"

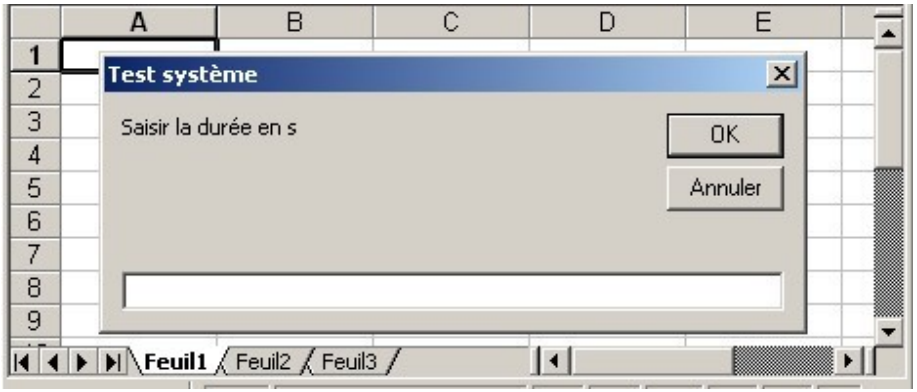
Saisie = MsgBox(MonMessage, vbOKCancel, Titre)

If Saisie = vbOK Then MsgBox"En cours"
```

Autres actions

- **Saisie d'une valeur**

```
Dim Saisie as Variant
    Saisie = InputBox("Saisir la durée en s","test système")
```

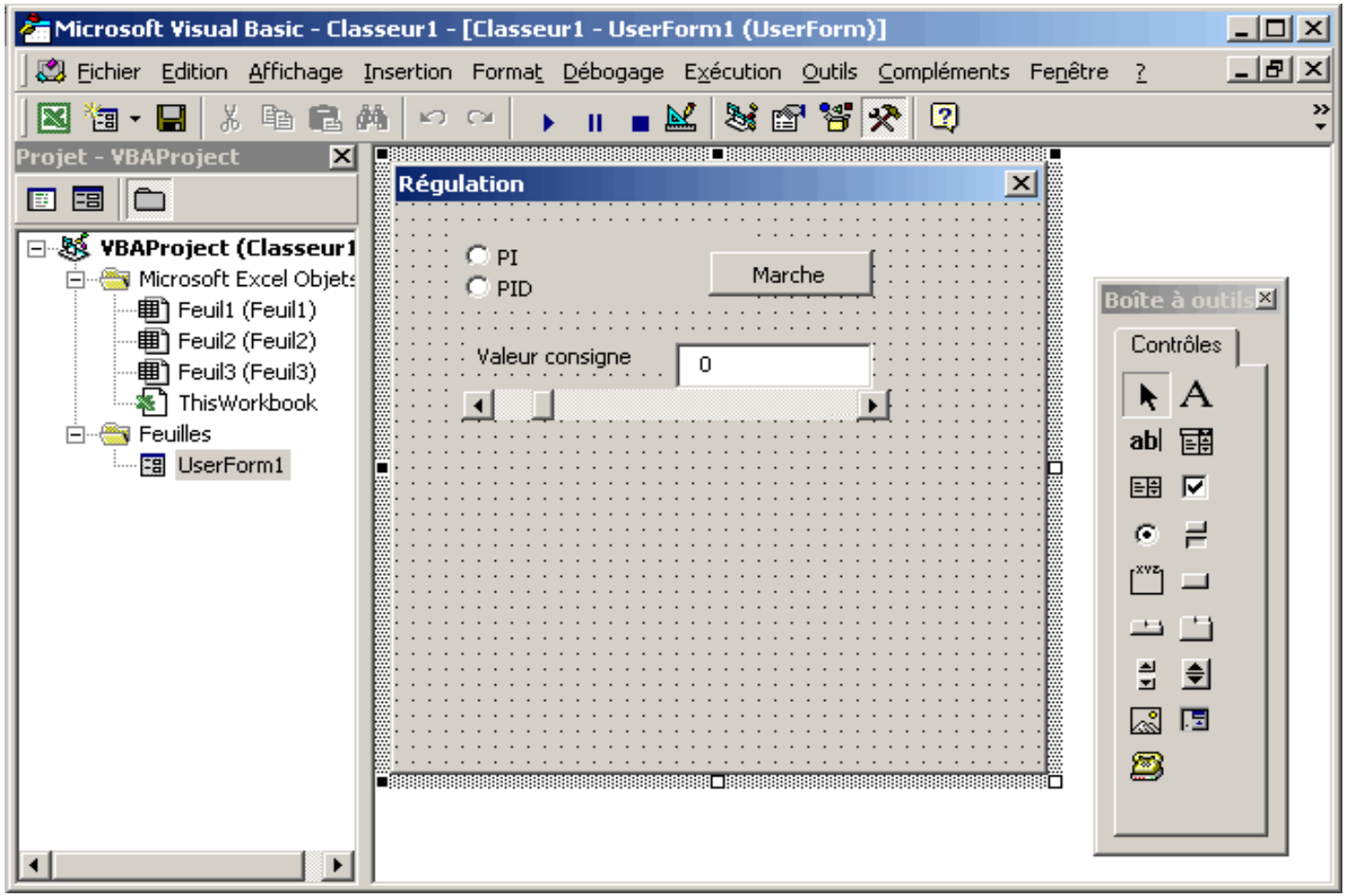


ATTENTION : Les fonctions MsgBox et InputBox sont blocantes jusqu'à réponse de l'opérateur.

Utilisation de UserForm

- permet de créer une fenêtre dans une application, indépendamment des feuilles Excel.

En VB pur , la fenêtre est l'élément de base de l'IHM



Utilisation de UserForm

- les contrôles usuels (curseur, case à cocher, fenêtre texte...) peuvent être insérés dans un UserForm
- Par programmation, toutes les propriétés des contrôles sont accessibles, ainsi que les cellules des feuilles Excel (la valeur d'un contrôle peut être lié directement à celle d'une cellule)
- par défaut, le Userform n'est pas affiché. Pour le faire passer en premier plan, il faut utiliser la méthode :

```
My_UserForm. Show
```

MASTERs SMaRT & GSI

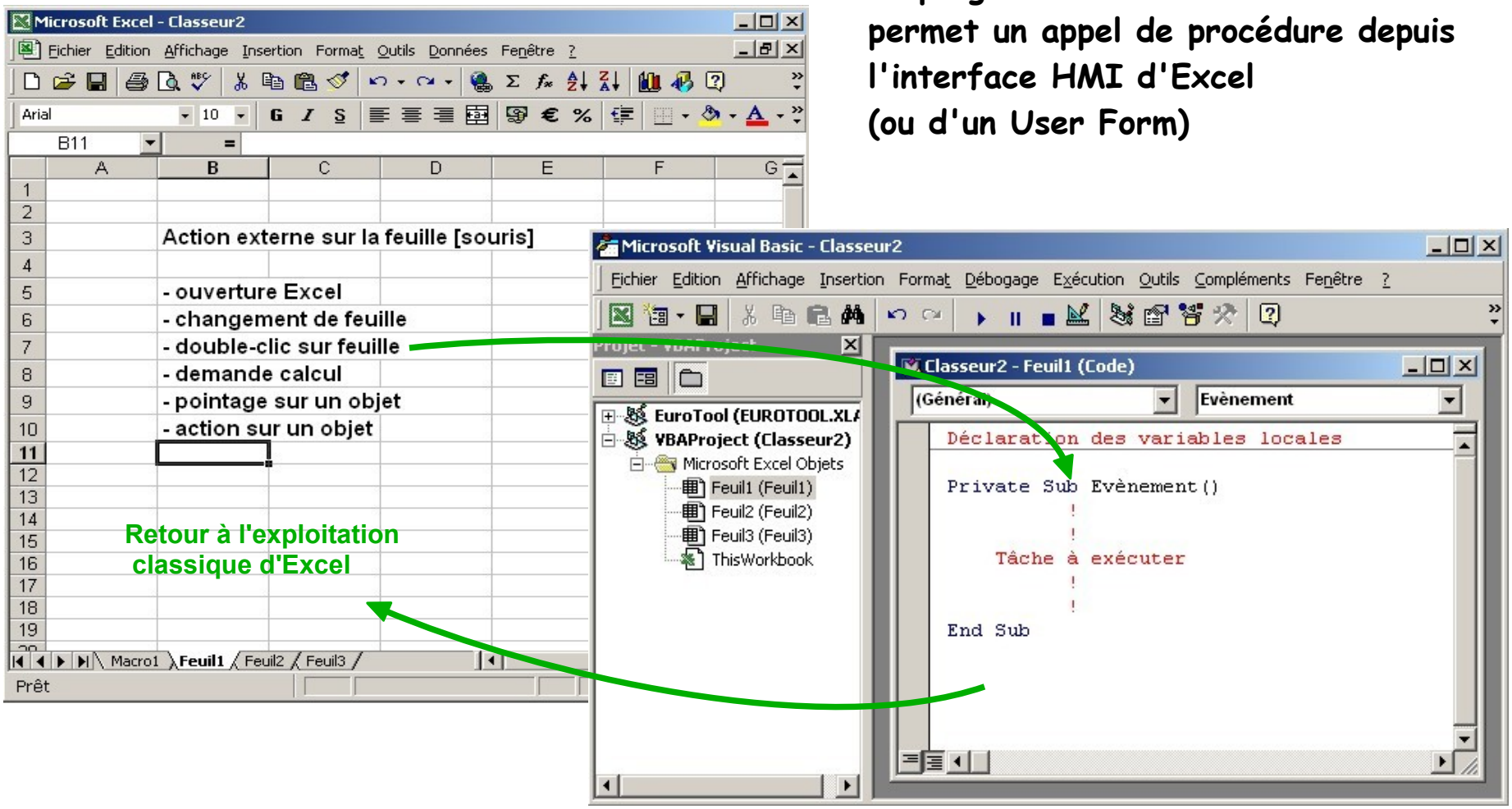
PROGRAMMATION EVENEMENTIELLE sur EXCEL

Pierre BONNET

Université Lille1 Sciences et Technologies

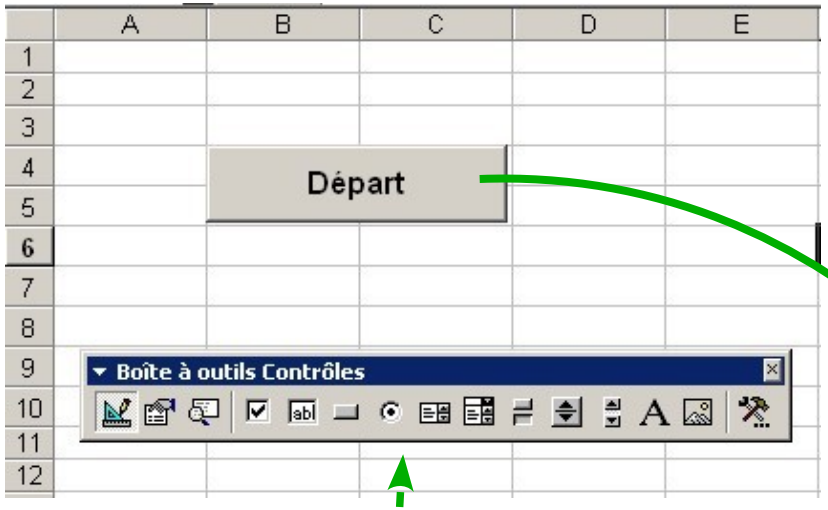
Programmation évènementielle

La programmation évènementielle permet un appel de procédure depuis l'interface HMI d'Excel (ou d'un User Form)



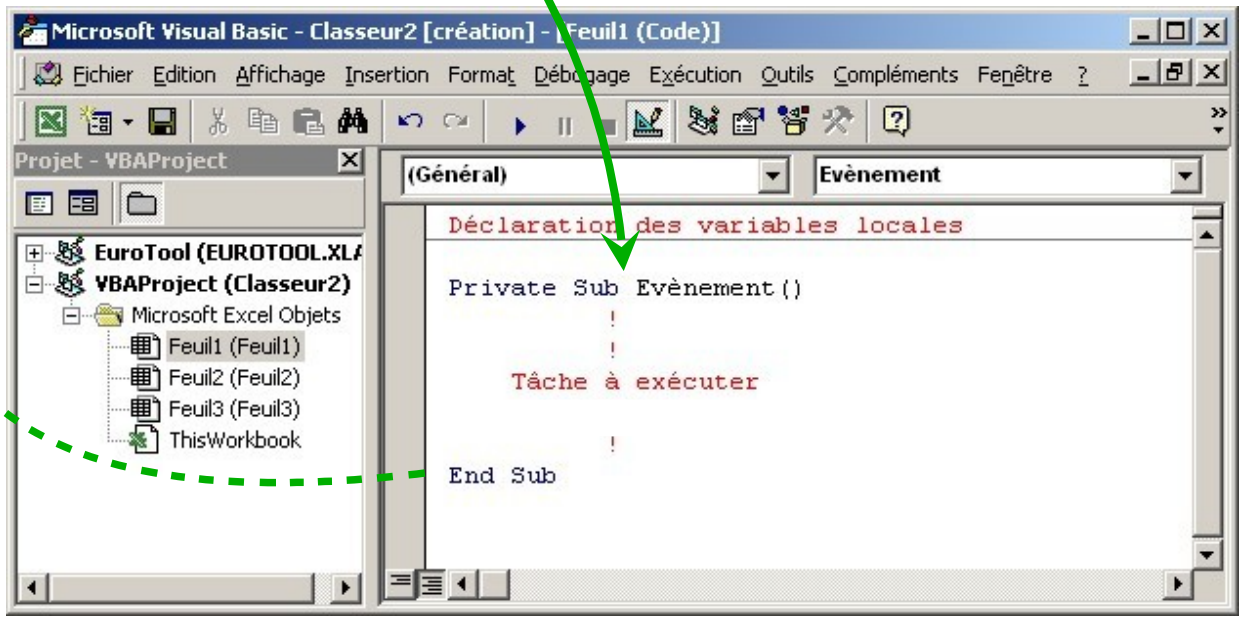
Attention: le nom de la procédure évènementielle est imposé par l'objet source en MsOffice

Programmation événementielle:



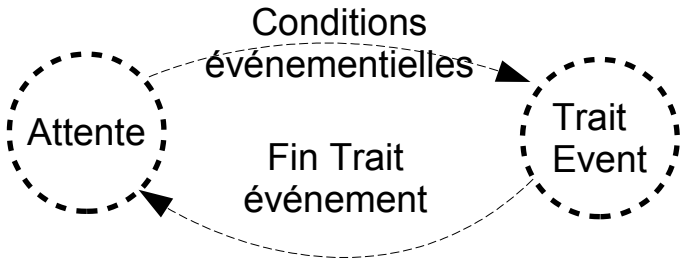
Evénement [front montant]

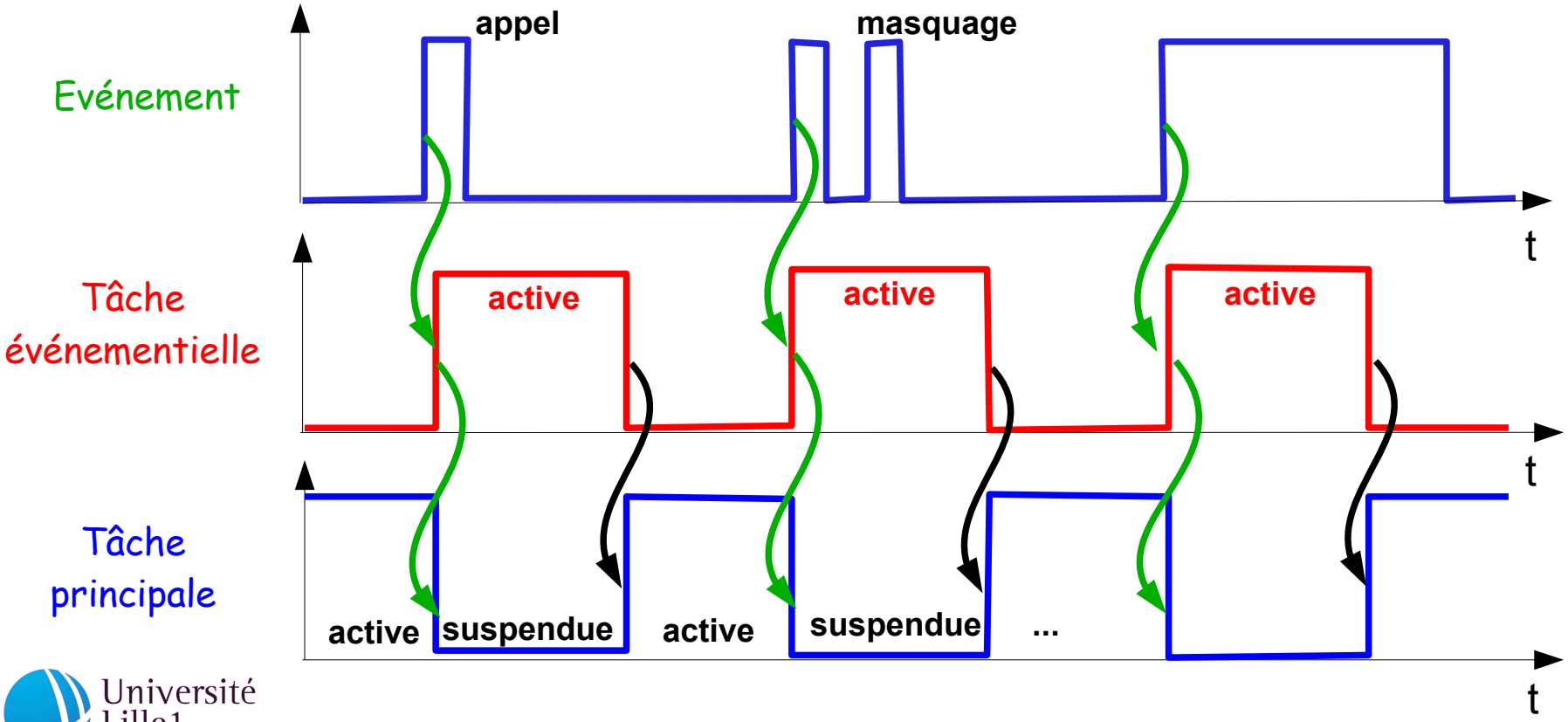
Retour de tâche événementielle



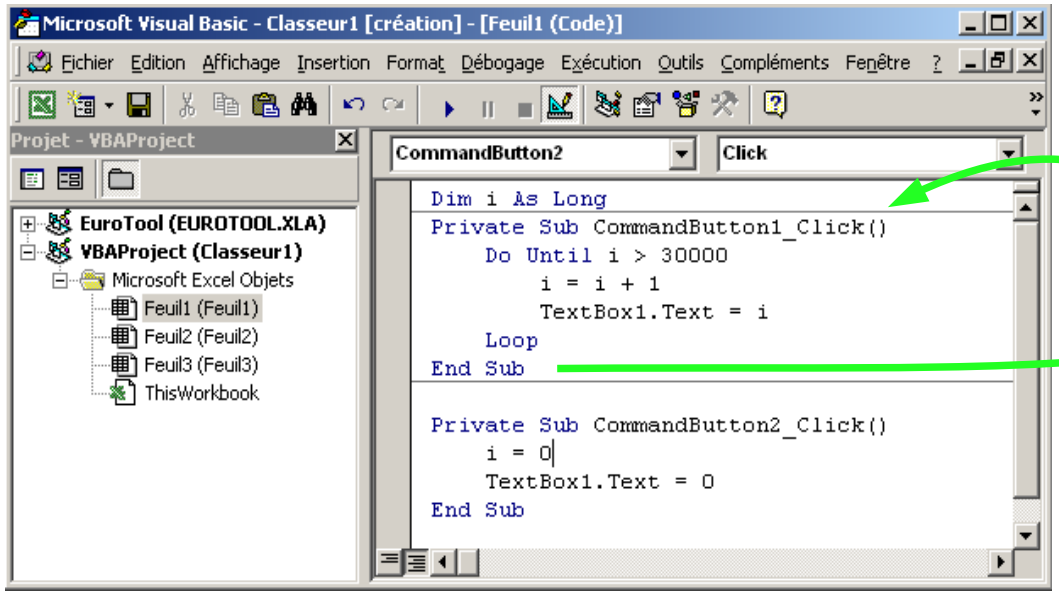
Programmation événementielle sur front:

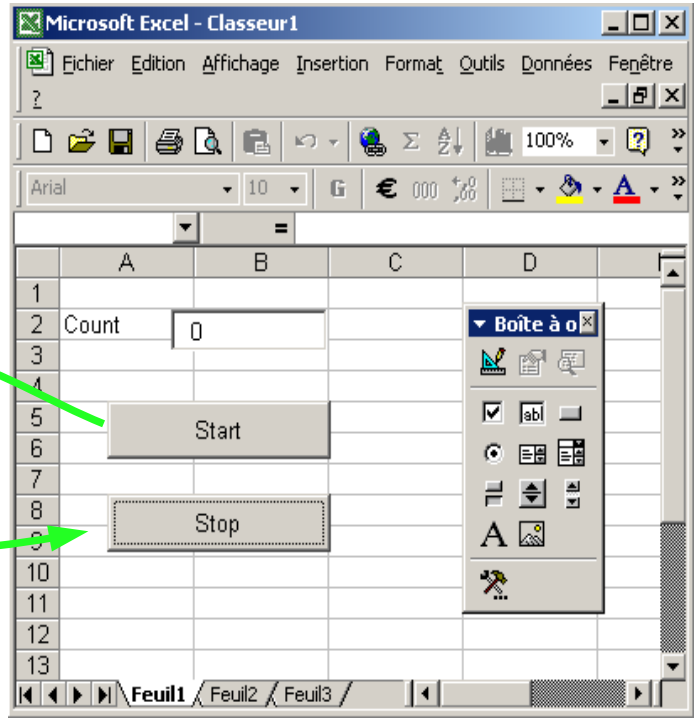
Diagramme de commande





Programmation événementielle: blocage temporaire par la tâche évènementielle

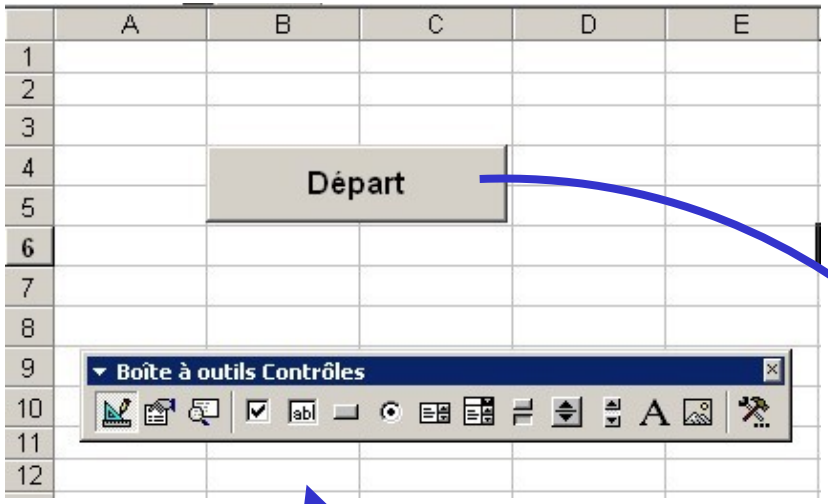




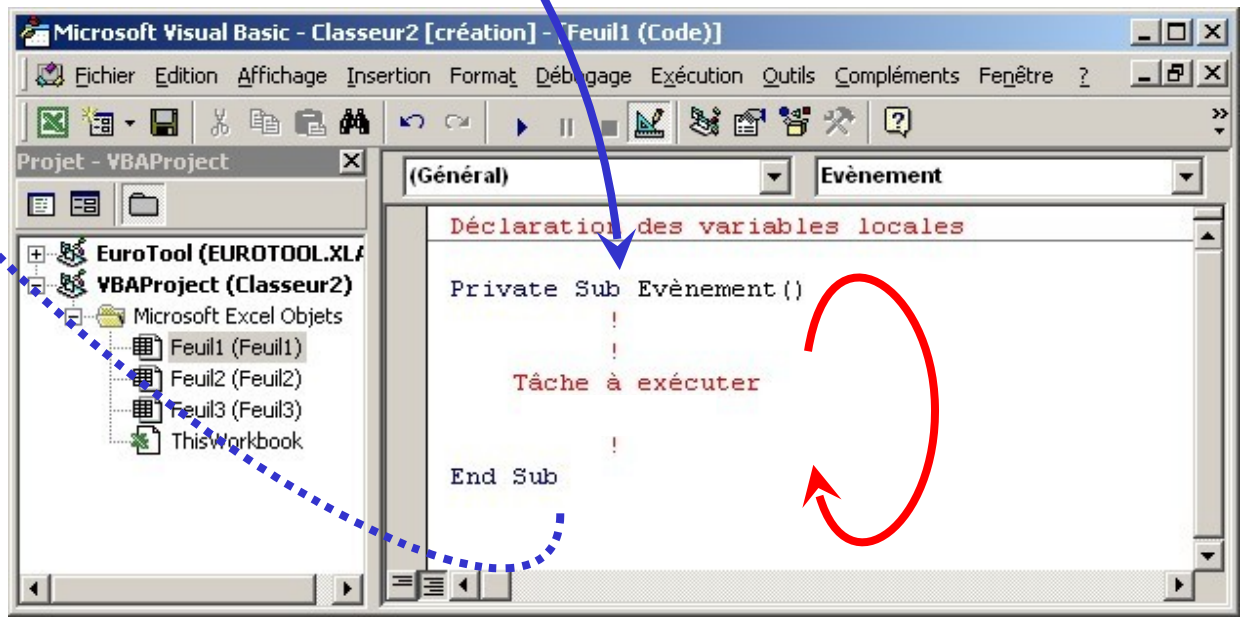
- Le retour à la fenêtre de présentation Excel se faite à la fin de la boucle. **Aucune activité apparente sur la fenêtre**: le compteur reste à la valeur 0; le bouton "Start" reste enfoncé; le bouton "Stop" reste enfoncé

- un **clic** sur "Stop" est **pris en compte** par le gestionnaire d'événements de l'interface graphique (Windows).et transmis au gestionnaire d'événements de l'application (Excel). Cet événement est **mis en pile** dans le gestionnaire de l'appli et n'est traité qu'après la fin de traitement de l'événement précédent.

Programmation événementielle:

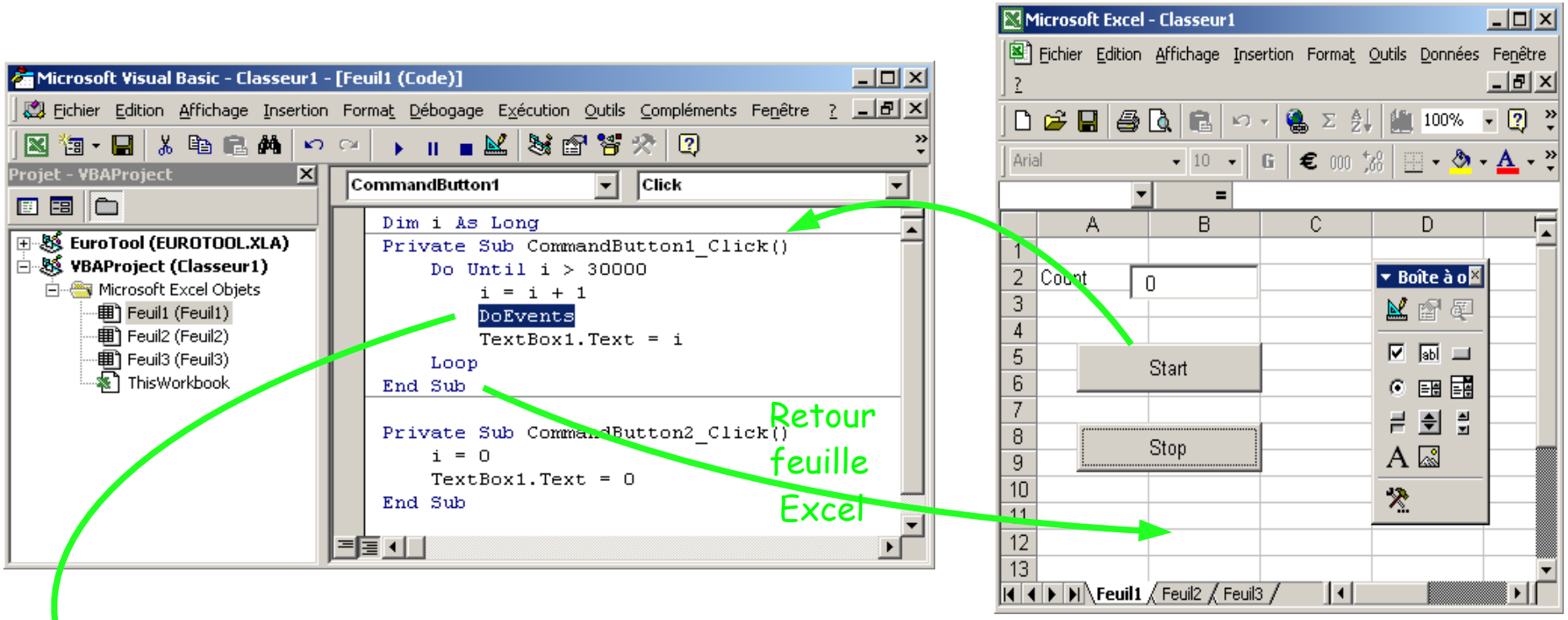


Les performances de l'application seront donc liées à la qualité de la programmation des tâches



Blocage définitif dans la tache sans fin

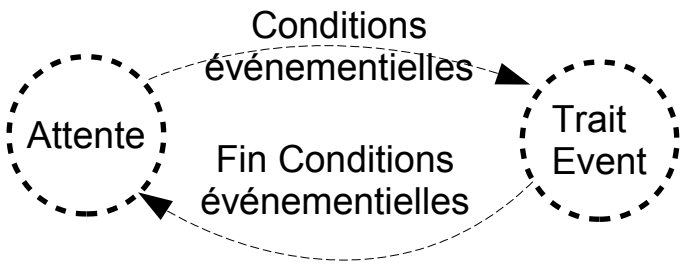
Programmation événementielle:



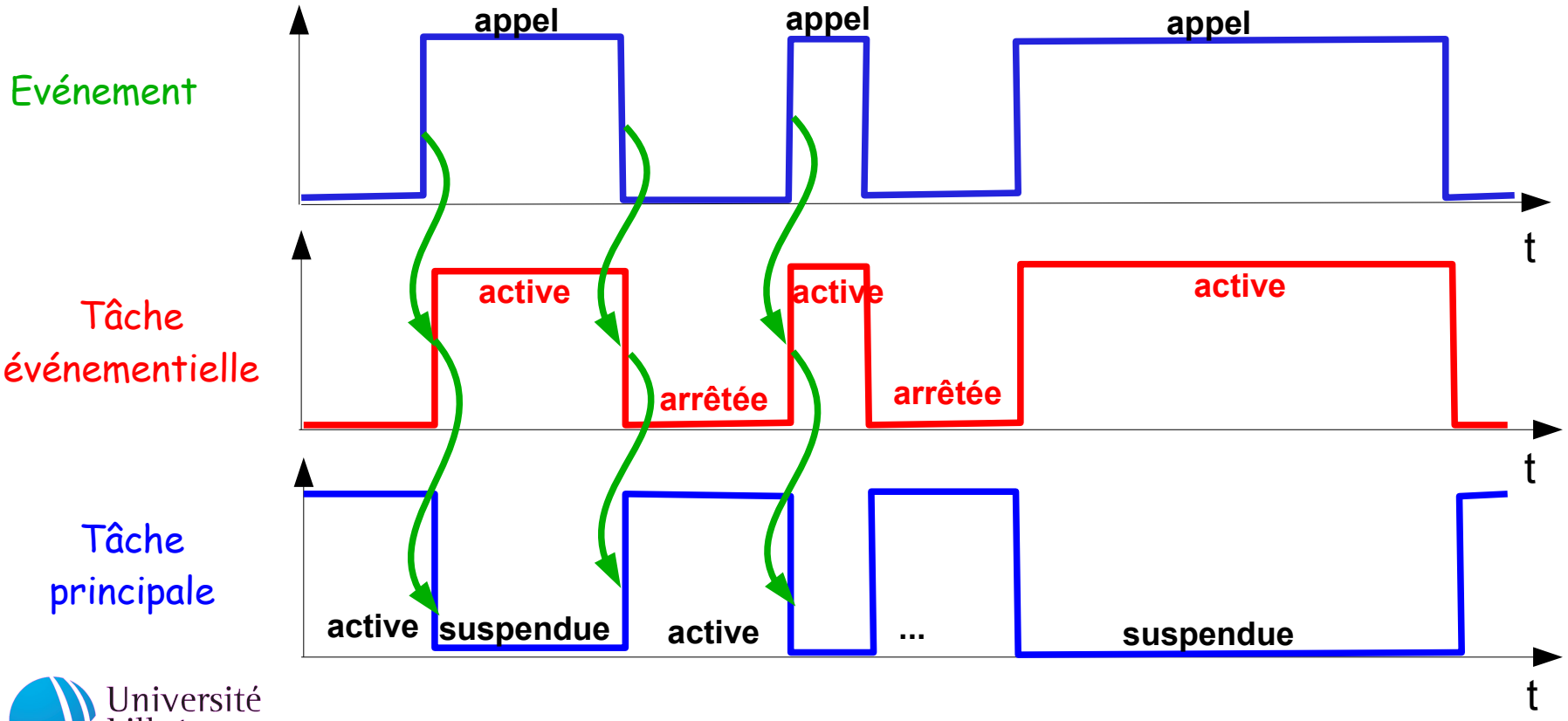
Gestionnaire d'événements

- **Appel forcé au gestionnaire d'événements: l'événement suivant dans la pile est traité**

Programmation événementielle sur niveau:

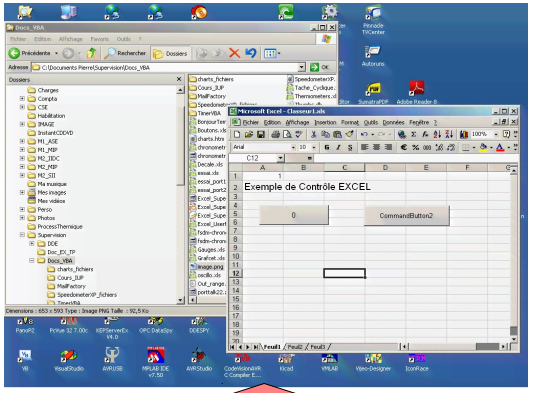


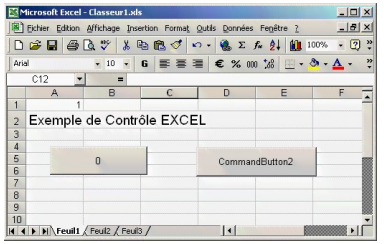
La durée d'exécution
de la tâche événementielle
est liée à la durée
d'activation de l'événement



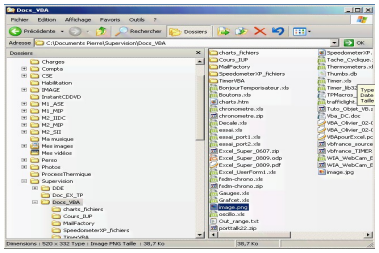
Programmation événementielle: principe

Chaque application possède son propre gestionnaire d'évènement

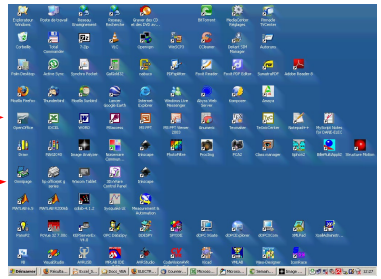




Excel, VB C++...



Explorer



Bureau

Gestionnaire de fenêtres graphiques

Evènements Messages

Système d'exploitation
ordonnanceur
Timers

Interruptions matérielles (cartes...)

Interruptions

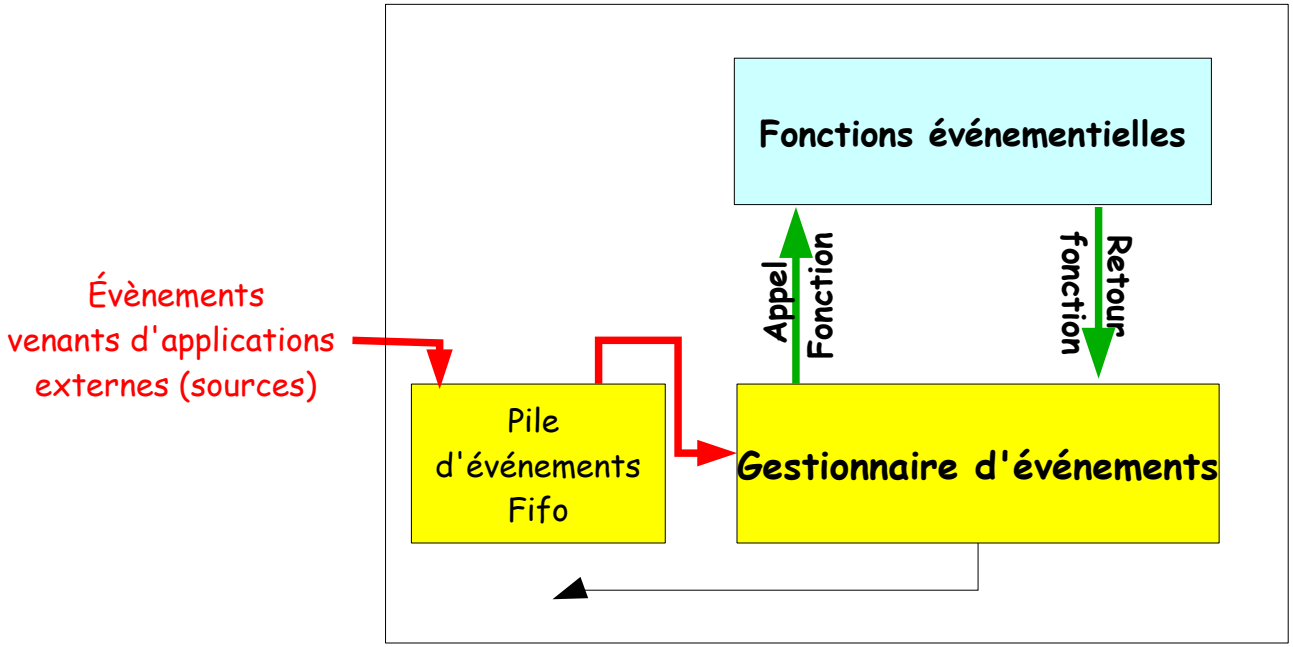
HMI

E/S

Horloge

Programmation événementielle: principe

- Structure d'une application



↳ les événements sont traités dans l'ordre d'apparition

↳ accès à la gestion de pile [écrasement volontaire]? Modification de l'ordre des événements dans la pile ? Appels multiples d'un même événement?

Événements Classeur

- **Ouverture classeur**

```
Private Sub Workbook_Open()
          Call MaProdedure
End Sub
```

- **Fermeture classeur**

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

- **Redimensionnement fenêtre classeur**

```
Private Sub Workbook_WindowResize(ByVal Wn As Window)
```

Événements Feuille

- **Activation/désactivation Feuille**

```
Sub WorkSheet_Activate()
Sub WorkSheet_Deactivate()
```

- **Changement de valeur sur feuille**

```
Sub WorkSheet_Change(ByVal Target As Range)
```

[`Target` représente la plage modifiée, elle peut contenir plusieurs cellules.]

Attention : si la procédure événementielle modifie elle-même des contenus de cellule, il y aura *récursivité* de l'événement. Dans une telle application, il est nécessaire de bloquer [masquer] les événements par `Application.EnableEvents = False` avant de modifier les contenus, puis de débloquer les événements par `Application.EnableEvents = True`

voir http://www.cpearson.com/excel/Events.aspx

- **Changement de sélection sur la feuille**

```
Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
```

[`Target` représente la nouvelle plage sélectionnée]

Evènements

- **Création d'une nouvelle procédure évènementielle**

Se placer dans le module "Classeur" (ThisWorkbook), insérer le code suivant :

```
Public WithEvents Mon_Appli As Application

Private Sub Workbook_Open()
    Set Mon_Appli = Application
End Sub
```

L'application "Mon_Appli" dispose maintenant de propriétés évènementielles associées à certains éléments du projet (classeur, feuilles...), tout comme une feuille, un classeur ou un module.

La liste des évènements disponibles (onglet en haut de droite) est consultable après avoir sélectionné l'application (onglet en haut à gauche)

Insertion d'autres contrôles ActiveX

- Les contrôles activeX sont des composants externes écrits conformément aux règles COM (Component Objet Model); ces composants possèdent des propriétés, exposent des méthodes et génèrent des évènements.

Pour être utilisable dans une application, ces composants doivent être enregistrés dans la base de registre (regserv32.exe).

Un composant ActiveX s'exécute dans l'espace mémoire de l'application principale appelée conteneur.

Un composant ActiveX comprend des propriétés, des méthodes et des événements.

Exemple : calendrier Calendar Microsoft

Contrôle du temps d'exécution d'une application

- **Lecture de l'heure et évaluation d'une durée**

L'heure courante est directement accessible sous Excel/VBA par la fonction `Now`

Le format est celui d'un flottant double précision avec :

- partie entière = nombre de jours depuis 1900
- partie fractionnaire = fraction de jour

(1h = 1/24 de jour, 1mn = 1/(24x60) de jour , 1s =1/(24x60x60) de jour)

⇨ Les calculs peuvent se faire directement avec les dates/heures par simple addition ou soustraction:

```
Heure_Depart = Now
        Calculs....
Heure_Fin = Now
Duree = Heure_Fin  - Heure_Depart
```

⇨ Il existe de nombreuses fonctions de manipulation des date/ heure qui facilitent l'expression des calculs ou des résultats

Contrôle du temps d'exécution d'une application

- **Gestion du temps par méthode d'attente**

```
Sub Attente()
    Dim Fin_Tempo [as Variant]
    Fin_Tempo = Now + TimeSerial(0, 0, Duree_attente_s )
    Application.Wait Fin_Tempo
    MsgBox "Attente terminée"
End Sub
```

⇨ La méthode Wait permet de faire intervenir la notion de temps dans l'exécution d'une fonction

⇨ L'attente est blocante pour la TOTALITE de l'application Excel

Contrôle du temps d'exécution d'une application

- **Méthode fondamentale pour une gestion multitâche non-blocante**

```
Heure_Execution = Now + TimeSerial(0, 0, Duree_attente )
Application.OnTime Heure_Execution , "Procédure_a_executer"
```

[*la procédure <macro> doit être écrite dans un "module" pour être accessible* au système d'exploitation]

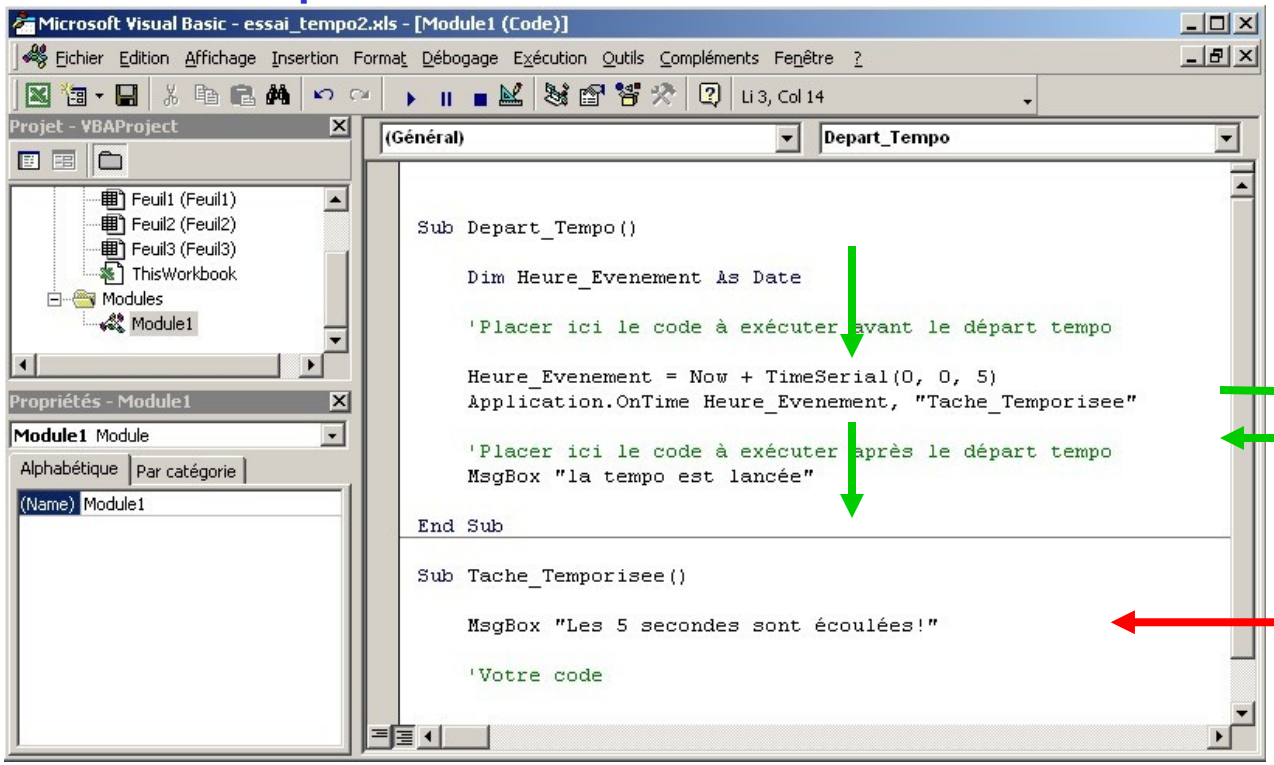
⇨ La méthode événementielle OnTime permet de disposer d'une gestion multitâche par appel de fonctions *depuis le système d'exploitation*.

⇨ La gestion de l'événement est externe à l'application Excel

⇨ La tâche s'exécute même en cas de fermeture du classeur (réouverture du classeur sous réserve qu'une instance d'Excel soit ouverte)

Contrôle du temps d'exécution d'une application

- **Principe de fonctionnement de la méthode OnTime**



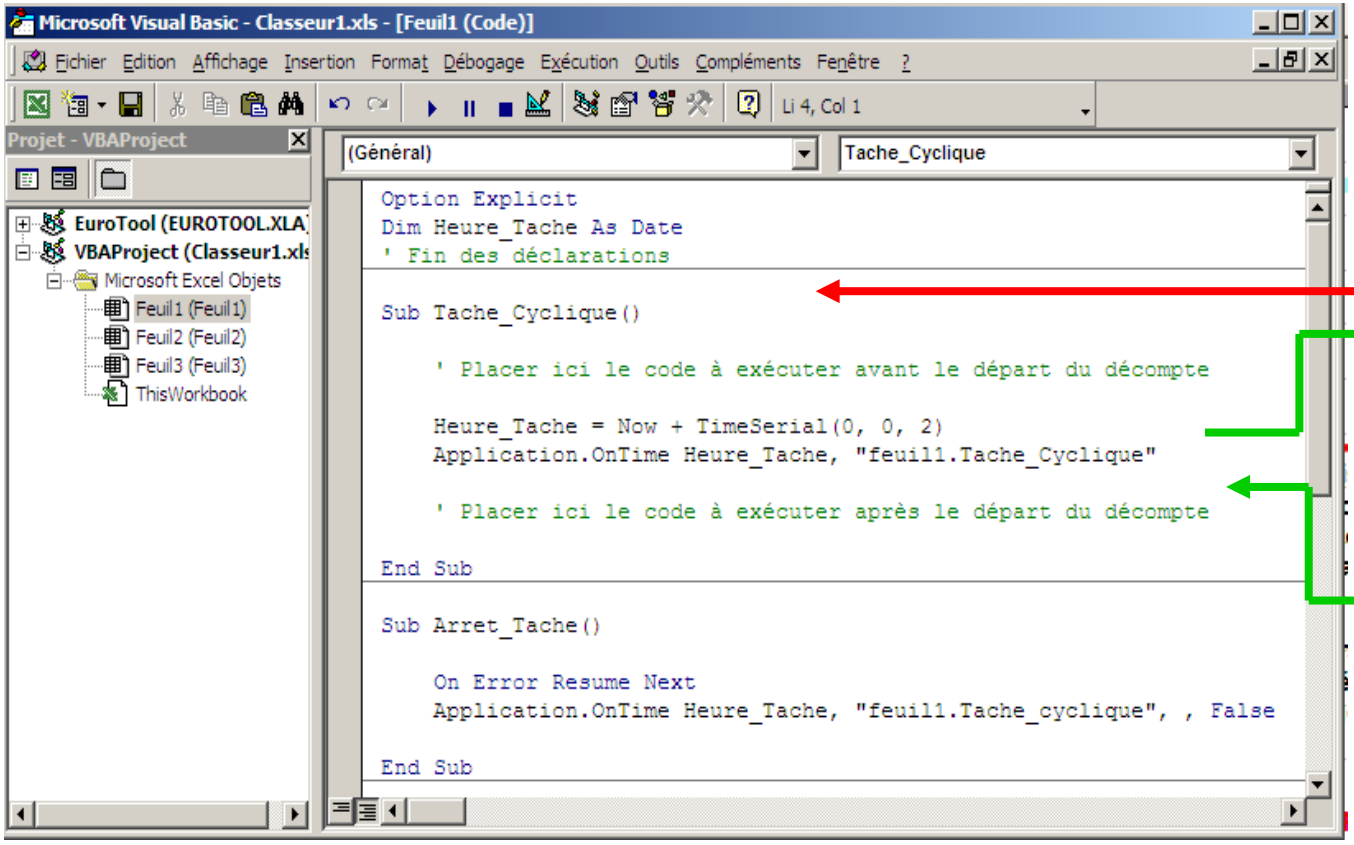
- **Enregistrement de la demande d'événement auprès du système d'exploitation [NT/XP]**
- **Poursuite de l'exécution de la tâche appelante**
- **Appel de la tâche temporisée par le système d'exploitation**

Attention: La tâche appelée doit se trouver dans un *module* (visibilité depuis Windows)

La méthode OnTime n'est pas bloquante car elle est gérée par le système d'exploitation

Contrôle du temps d'exécution d'une application

- **Tâche cyclique**



- **Enregistrement de la demande d'événement au système d' exploitation [NT/XP]**
- **Poursuite de l'exécution de la tâche appelante**
- **Rappel de la tâche cyclique par le système d'exploitation**

⇨ La temporisation est relancée sur elle-même et crée une *tâche cyclique* .

⇨ L'arrêt de la tâche cyclique ne peut se faire que par son enregistrement avec le paramètre de "Schedule" à False, ce qui la supprime de la liste des évènements.

Contrôle du temps d'exécution d'une application

- **Tâche cyclique**

Des erreurs sont souvent constatées dans le fonctionnement de l'exemple de la page précédente :

- l'activité s'exécute deux fois (voire plus) pour une périodicité donnée

- la périodicité de la Tâche cyclique apparaît comme fausse (période diminuée)

Explications :

⇨ après le lancement initial de la tâche cyclique, son fonctionnement n'est plus contrôlé par Excel mais par le système d'exploitation.

⇨ tout lancement *supplémentaire* de la tâche cyclique (par le menu "macro" ou par programmation) génère une *nouvelle instanciation* .

⇨ Les *instanciations* multiples fonctionnent en boucles imbriquées : les instants de déclenchements peuvent être identiques (activité évènementielle répétée plusieurs fois au même instant) ou décalés (activité relancée au cours d'une période de base)

Contrôle du temps d'exécution d'une application

- **Timer VB**

Sous VB6, l'utilisateur dispose de Timer ajustable de 1 ms à 65s . Sa programmation est très simple (non disponible sous VBA)

- **Timer Windows**

⇨ Le système d'exploitation Windows propose une fonction de type timer dans la librairie `user32.dll` . Elle permet de fixer une durée la durée (ou période) à 1 ms près.

Les éléments de programmation sont :

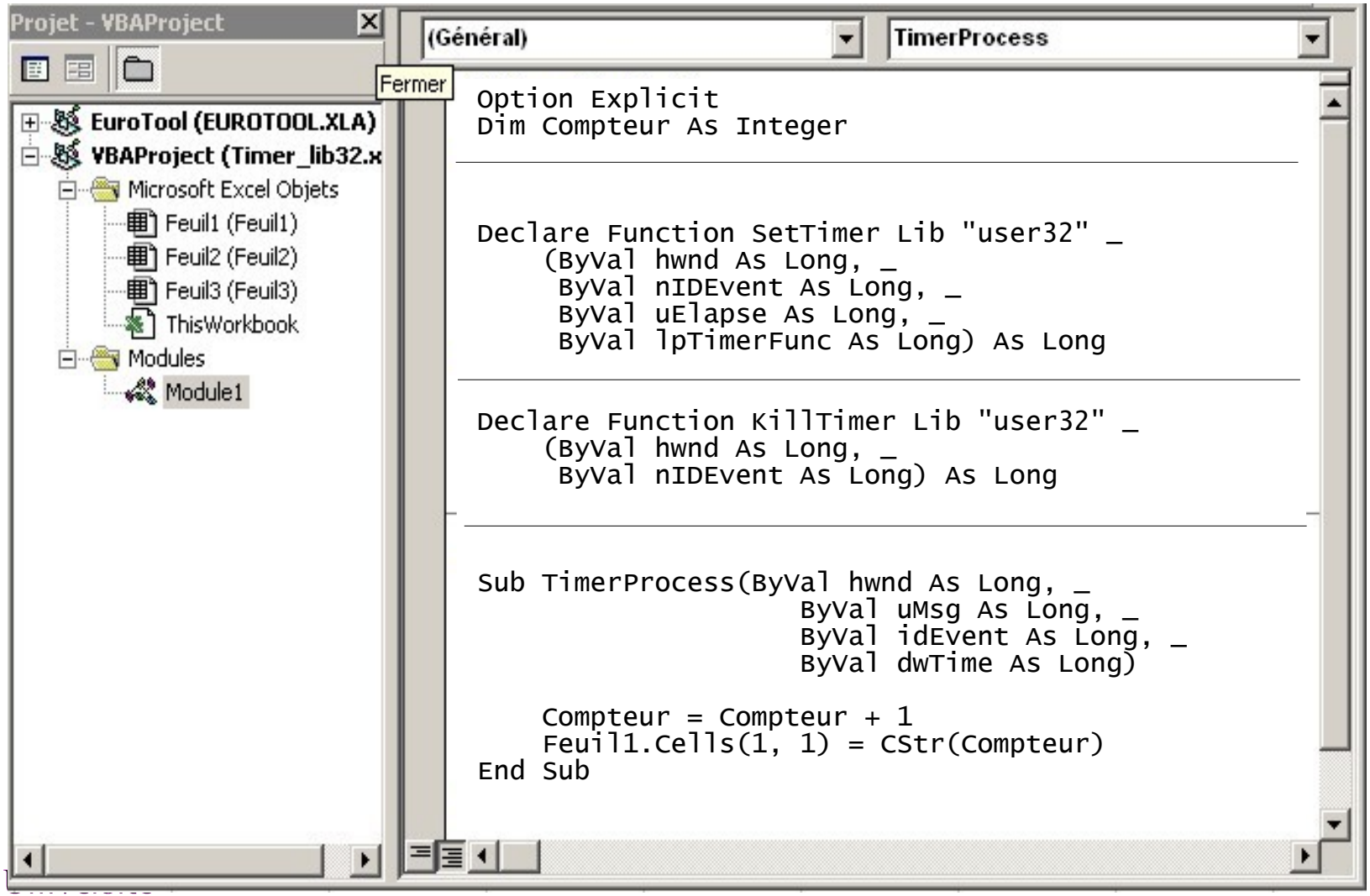
⇨ définition de la fonction évènementielle

⇨ lancement du `timer` avec passage du pointeur de la fonction évènementielle

⇨ arrêt du timer par la fonction `killtimer`

Contrôle du temps d'exécution d'une application

- **Timer système de la librairie `user32`**



Contrôle du temps d'exécution d'une application

- **Timer système de la librairie user32**