

Visual Basic Editor

par [SilkyRoad \(silkyroad.developpez.com\)](http://silkyroad.developpez.com)

Date de publication : 12/09/2006

Dernière mise à jour : 18/06/2007

Ce tutoriel montre comment manipuler l'éditeur de macros Excel ... par macro.

Vous y trouverez quelques méthodes pour gérer dynamiquement:

Les références et les macros complémentaires

Les modules

Les macros

Les objets (feuilles, UserForm, contrôles...)

Toutes les procédures de ce document ont été testées en utilisant Excel2002.

- I - Introduction
- II - Informations générales
 - II-A - Quelques manipulations de base
 - II-B - La propriété VBComponents
 - II-C - La propriété CodeModule
 - II-D - Automatiser la numérotation des lignes
- III - Les références et macros complémentaires
 - III-A - Les références
 - III-B - Les macros complémentaires
- IV - Gérer les modules et les procédures
 - IV-A - Créer
 - IV-B - Lire
 - IV-C - Modifier
 - IV-D - Supprimer
 - IV-E - Importer et Exporter
- V - Gérer les objets
 - V-A - Créer
 - V-B - Lire
 - V-C - Modifier
 - V-D - Supprimer
- VI - L'exécution des macros
- VII - Téléchargement

I - Introduction

L'éditeur de macros, aussi appelé Visual Basic Editor ou VBE, est l'environnement dans lequel vous pouvez modifier les macros que vous avez enregistré, écrire de nouvelles macros et de nouveaux programmes VBA.

L'éditeur VBE peut être modifié dynamiquement par macro. C'est l'objet de ce tutoriel. Vous pouvez de cette manière automatiser:

- * L'activation des références (Librairies).
- * L'importation et l'exportation des modules.
- * La création, modification ou suppression des macros dans vos classeurs.
- * La création des objets par macros et leur associer des procédures événementielles.
- * Le déclenchement de macros.

La majeure partie des procédures utilisées nécessite d'activer la référence [Microsoft Visual Basic for Applications Extensibility 5.3](#).

Dans l'éditeur de macros (Alt+F11):

Menu Outils

Références

Cochez la ligne "Microsoft Visual Basic for Applications Extensibility 5.3"

Cliquez sur "OK" pour valider.

II - Informations générales

II-A - Quelques manipulations de base

Pour commencer, voici quelques exemples simples pour manipuler l'éditeur de macros.

Récupérer la version VBE installée.

```
Vba
Sub afficherLaVersionVBE()
MsgBox Application.VBE.Version
End Sub
```

Ouvrir l'éditeur de macros (L'équivalent de Alt+F11).

```
Vba
Sub OuvreVBA()
Application.VBE.MainWindow.Visible = True
End Sub
```

Fermer l'éditeur de macros.

```
Vba
Sub FermeVBA()
Application.VBE.MainWindow.Visible = False
End Sub
```

Modifier le nom du projet (Le nom par défaut est "VBAProject").

```
Vba
Application.VBE.ActiveVbProject.Name = "MonProjet"
```

II-B - La propriété VBComponents

La propriété VBComponents renvoie la collection de composants contenue dans un projet.

- * ThisWorkbook.
- * L'ensemble des feuilles (CodeName).
- * L'ensemble des modules Standards et modules de Classe.

* Les UserForm.

Voici deux méthodes pour lister les composants du classeur actif.

Vba

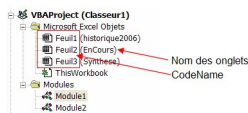
```
Sub boucleVBComponents_V01()  
    Dim i As Integer  
  
    For i = 1 To ActiveWorkbook.VBProject.VBComponents.Count  
        Debug.Print ActiveWorkbook.VBProject.VBComponents(i).Name  
    Next  
End Sub
```

Vba

```
Sub boucleVBComponents_V02()  
    Dim VBComp As VbComponent  
  
    For Each VBComp In ActiveWorkbook.VBProject.VBComponents  
        Debug.Print VBComp.Name  
    Next  
End Sub
```

Remarque:

Lorsque vous devez spécifier le nom d'une feuille dans la propriété **VBComponents**, il s'agit du **CodeName** et pas le nom de l'onglet.



II-C - La propriété CodeModule

La propriété **CodeModule** permet de modifier, ajouter, supprimer ou renvoyer des informations sur le contenu des procédures, pour chacun des composants.

Un exemple pour compter les lignes de macros dans chaque composant du classeur actif.

Vba

```
Dim VBComp As VbComponent  
Dim Mdl As CodeModule  
  
For Each VBComp In ActiveWorkbook.VBProject.VBComponents  
    Set Mdl = VBComp.CodeModule  
    Debug.Print VBComp.Name & " : " & Mdl.CountOfLines  
Next
```

II-D - Automatiser la numérotation des lignes

La fonction ERL permet de récupérer le numéro de ligne qui a provoqué une erreur à condition d'avoir préalablement numéroté les lignes dans l'éditeur de macros.

L'exemple suivant renvoie 8.

```
Vba

Option Explicit
Sub laProcedure()
4 Dim x As Integer

6 On Error GoTo errHandler

8 x = 5 / 0 'provoque une erreur (division par zéro)

10 Exit Sub
11 errHandler:
12 MsgBox "Une erreur est survenue , Ligne : " & Erl() & _
    vbCrLf & "Numéro d'erreur : " & Err.Number & vbCrLf & Err.Description
14 End Sub
```

Remarque:

- * Si la ligne `x = 5 / 0` n'est pas numérotée, la fonction ERL renvoie le numéro de la dernière ligne précédant l'erreur (6).
- * ERL renvoie 0 s'il n'y a pas d'erreur dans la procédure.
- * ERL renvoie 0 si aucune ligne n'est numérotée dans la macro.

Vous allez me dire, Comment faire pour numéroté rapidement les lignes d'une macro et ne pas être obligé de le faire manuellement?

Vous pouvez installer l'Add-In **MZ-Tools** qui fait cela très bien.

Sinon, juste pour le fun, la procédure suivante Numérote les lignes de la macro "laProcedure" dans le "Module1".

```
Vba

Option Explicit
Option Compare Text

Sub testAjout()
'laProcedure" est la macro dont vous souhaitez numéroté les lignes
NumerotationLignesProcedure "Module1", "laProcedure"
End Sub
```

Vba

```
Sub NumerotationLignesProcedure(nomModule As String, nomMacro As String)
'
'Cet exemple ne gère pas les erreurs s'il existe déjà une numérotation
'
Dim Debut As Integer, Lignes As Integer, x As Integer
Dim Texte As String, strVar As String

With ThisWorkbook.VBProject.VBComponents(nomModule).CodeModule
    Debut = .ProcStartLine(nomMacro, 0)
    Lignes = .ProcCountLines(nomMacro, 0)
End With

For x = Debut + 2 To Debut + Lignes - 1
    With ThisWorkbook.VBProject.VBComponents(nomModule).CodeModule
        Texte = .Lines(x, 1)
        strVar = Application.WorksheetFunction.Substitute(Texte, " ", "")
        strVar = Application.WorksheetFunction.Substitute(strVar, vbCrLf, "")
        strVar = Application.WorksheetFunction.Substitute(strVar, vbTab, "")

        'Adaptez les filtres en fonction de vos projets.
        'Remarque: les arguments PrivateFunction et PublicFunction sont volontairement accolés.
        '
        If strVar <> "" And _
        Left(strVar, 3) <> "Sub" And _
        Left(strVar, 10) <> "PrivateSub" And _
        Left(strVar, 9) <> "PublicSub" And _
        Left(strVar, 8) <> "Function" And _
        Left(strVar, 15) <> "PrivateFunction" And _
        Left(strVar, 14) <> "PublicFunction" And _
        Right(ThisWorkbook.VBProject.VBComponents(nomModule). _
            CodeModule.Lines(x - 1, 1), 1) <> "_" _
        Then .ReplaceLine x, x & " " & Texte
    End With
Next
End Sub
```

Et une procédure pour supprimer la numérotation:

Vba

```
Sub testSuppression()
    supprimeNumerotationLignes "Module1", "laProcedure"
End Sub

Sub supprimeNumerotationLignes(nomModule As String, nomMacro As String)
Dim Debut As Integer, Lignes As Integer, x As Integer
Dim Texte As String, strVar As String
Dim Valeur As Integer

With ThisWorkbook.VBProject.VBComponents(nomModule).CodeModule
    Debut = .ProcStartLine(nomMacro, 0)
    Lignes = .ProcCountLines(nomMacro, 0)
End With

For x = Debut + 2 To Debut + Lignes - 1
    With ThisWorkbook.VBProject.VBComponents(nomModule).CodeModule
        Texte = .Lines(x, 1)

        Valeur = Val(Texte)
        If Valeur <> 0 Then
            strVar = Mid(Texte, Len(CStr(Valeur)) + 2)
        Else
```

Vba

```
        strVar = Texte
    End If

    .ReplaceLine x, strVar
End With
Next
End Sub
```

III - Les références et macros complémentaires

III-A - Les références

Ce chapitre montre comment manipuler les références par macros.

Les références, aussi appelées bibliothèques ou librairies, offrent de nombreuses possibilités lorsqu'elles sont activées.

Quelques exemples d'utilisation parmi les nombreuses bibliothèques disponibles:

- * Le modèle **ADO (ActiveX Data Objects)** pour manipuler les bases de données.
- * La librairie **DSO** pour lire et modifier les propriétés des documents Office.
- * La librairie **Windows Image Acquisition** pour manipuler les images et gérer les WebCams.
- * ...etc...

Vous pouvez aussi utiliser les références pour piloter d'autres applications: par exemple **Word**, **Windows Media Player**, Outlook, Power Point...

Un exemple de procédure pour lister les références actives.

Vba

```
Dim Ref As Reference
For Each Ref In ThisWorkbook.VBProject.References
    Debug.Print Ref.Name 'Nom
    Debug.Print Ref.FullPath 'Chemin complet
    Debug.Print Ref.Description 'Description de la référence
    Debug.Print Ref.IsBroken 'Indique si la référence est manquante
    Debug.Print Ref.Major & "." & Ref.Minor 'Version
    Debug.Print "----"
Next Ref
```

Ajouter une référence dans votre projet.

Vba

```
'Ajoute la référence Outlook pour OfficeXP
Dim x As String
x = "C:\Program Files\Microsoft Office\Office10\MSOUTL.OLB"
ThisWorkbook.VBProject.References.AddFromFile x
```

Désactiver les références manquantes.

Vba

```
Dim Ref As Reference

'La procédure boucle sur la collection de références et supprime celles qui sont
'spécifiées manquantes.
For Each Ref In ThisWorkbook.VBProject.References
    If Ref.IsBroken = True Then _
        ThisWorkbook.VBProject.References.Remove Ref
Next Ref
```

III-B - Les macros complémentaires

Une macro complémentaire (.xla), aussi appelée AddIn, est un classeur Excel dont les feuilles sont masquées et que vous pouvez charger dès l'ouverture d'Excel. Les classeurs .xla sont principalement utilisés pour stocker les procédures et les fonctions VBA personnelles que vous utilisez régulièrement dans Excel.

Un exemple pour automatiser l'installation d'une macro complémentaire.

Vba

```
Sub installationMacroComplementaire()
    Dim oAddIn As AddIn

    'L'argument TRUE: de type Variant facultatif. Ignoré si le fichier complémentaire est
    'sur le disque dur. Affectez-lui la valeur True pour copier la macro complémentaire sur
    'votre disque dur si elle est sur support amovible (disquette ou CD-ROM).
    'Affectez-lui la valeur False pour la laisser sur le support amovible.
    'Si vous ne spécifiez pas cet argument, une boîte de dialogue s'affiche et vous
    'demande de choisir.
    Set oAddIn = Application.AddIns.Add("F:\leFichier.xla", True)
    oAddIn.Installed = True
End Sub
```

Lister les macros complémentaires.

Vba

```
Sub listeMacrosComplementaires()
    Dim i As Integer

    For i = 1 To Application.AddIns.Count
        Debug.Print Application.AddIns(i).Title
        Debug.Print Application.AddIns(i).Name
        Debug.Print Application.AddIns(i).FullName
        Debug.Print Application.AddIns(i).Installed
        Debug.Print "----"
    Next
End Sub
```

Vérifier si une macro complémentaire est installée.

Vba

Vba

```
Option Explicit
Option Compare Text

Sub controler_Si_MacroComplementaire_Installee()
    Dim X As AddIn
    Dim laMacro As String

    laMacro = "solver.xla"

    For Each X In Application.AddIns
        If laMacro = X.Name Then
            If X.Installed = True Then
                MsgBox "la macro complémentaire " & laMacro & " est installée."
            Else
                MsgBox "la macro complémentaire " & laMacro & " n'est pas installée."
            End If
            Exit Sub
        End If
    Next X

    MsgBox "la macro complémentaire " & laMacro & " n'a pas été trouvée."
End Sub
```

Imprimer la Feuil1 d'un classeur xla.

Vba

```
With Workbooks("Test.xla")
    .IsAddin = False
    .Worksheets("Feuil1").PrintOut Copies:=1, Collate:=True
    .IsAddin = True
End With
```

Activer l'utilitaire d'analyse (Funcres.xla) et l'utilitaire d'analyse-VBA (atpvbaen.xls)

Vba

```
Dim oAddIn As AddIn

Set oAddIn = Application.AddIns.Add _
    (Filename:=Application.LibraryPath & "\analyse\analys32.xll")
oAddIn.Installed = True
Application.RegisterXLL "Analys32.xll"

Workbooks.Open Application.LibraryPath & "\analyse\atpvbaen.xla"
AddIns("Utilitaire d'analyse - VBA").Installed = True
```

IV - Gérer les modules et les procédures

Ce chapitre propose des exemples pour manipuler les modules et les procédures contenus dans les classeurs.

IV-A - Créer

Créer un nouveau module, le renommer et y insérer une macro.

```
Vba

Sub creationModule()
    'Nécessite d'activer la référence
    'Visual basic For Application Extensibility 5.3
    '
    Dim Wb As Workbook
    Dim VbComp As VBComponent
    Dim X As Integer

    'Définit le classeur cible
    Set Wb = Workbooks("Classeur1.xls")

    'Ajoute un module standard dans le classeur
    Set VbComp = Wb.VBProject.VBComponents.Add(1)
    'Renomme le module
    VbComp.Name = "NouveauModule"

    'Ajoute une macro dans le module
    With VbComp.CodeModule
        X = .CountOfLines
        .InsertLines X + 1, "Sub laMacro()"
        .InsertLines X + 2, "Range(\"A1\").Value = \"Coucou\""
        .InsertLines X + 3, "End Sub"
    End With
End Sub
```

Utilisez `Wb.VBProject.VBComponents.Add(2)` pour créer un module de classe.

Utilisez `Wb.VBProject.VBComponents.Add(3)` pour créer un UserForm.

Créer une macro dynamiquement.

Cet exemple permet d'ajouter une procédure événementielle SelectionChange dans la Feuil1 du classeur actif.

```
Vba

Sub creationMacro()
    Dim X As Integer

    With ActiveWorkbook.VBProject.VBComponents("Feuil1").CodeModule
        X = .CountOfLines
        .InsertLines X + 1, "Private Sub Worksheet_SelectionChange(ByVal Target As Range)"

        'Chr(38) permet d'insérer le symbole & dans la procédure.
        .InsertLines X + 2, _
```

Vba

```
        MsgBox "La cellule sélectionnée: " & Chr(38) & " Target.Address, , "Message" "
    .InsertLines X + 3, "End Sub"
End With
End Sub
```

Un autre exemple complet:

Récupérer toutes les procédures d'un classeur endommagé.

IV-B - Lire

Lister toutes les procédures d'un classeur dans la feuille de calcul.

Vba

```
Sub listeMacros()
'Nécessite d'activer la référence
'Visual basic For Application Extensibility 5.3
'
Dim i As Integer, Ajout As Integer, x As Integer
Dim Msg As String
Dim VBCmp As VbComponent
Dim Wb As Workbook

'Indiquez le nom du classeur ouvert
'Set Wb = ThisWorkbook
Set Wb = Workbooks("Classeur1.xls")

Ajout = 1
For Each VBCmp In Wb.VBProject.VBComponents
    Msg = VBCmp.Name

    With Cells(Ajout, 1)
        .Interior.ColorIndex = 6
        .Value = Msg
    End With

    x = Wb.VBProject.VBComponents(Msg).CodeModule.CountOfLines
    For i = 1 To x
        Cells(Ajout + i, 1) = _
            Wb.VBProject.VBComponents(Msg).CodeModule.Lines(i, 1)
    Next

    Ajout = Ajout + x + 2
Next VBCmp
End Sub
```

Afficher le contenu d'une macro précise.

Vba

```
Sub Test()
'Nécessite d'activer la référence
'Visual basic For Application Extensibility 5.3
'
'Affiche la macro "testMacro" contenue dans la Module1 du Classeur1.xls
```

Vba

```
MsgBox recupereContenuMacro(Workbooks("Classeur1.xls"), "Module1", "testMacro")

'exemple pour récupérer la procédure événementielle "Workbook_Open" dans le module objet
ThisWorkbook:
MsgBox recupereContenuMacro(Workbooks("Classeur1.xls"), "ThisWorkbook", "Workbook_Open")
End Sub

Function recupereContenuMacro(Wb As Workbook, Mdl As String, NomMacro As String)
    Dim Debut As Integer, Fin As Integer, i As Integer
    Dim Resultat As String

    With Wb.VBProject.VBComponents(Mdl).CodeModule
        Debut = .ProcStartLine(NomMacro, vbext_pk_Proc)
        Fin = .ProcCountLines(NomMacro, vbext_pk_Proc) + Debut
        For i = Debut To Fin
            Resultat = Resultat & .Lines(i, 1) & vbCrLf
        Next
    End With

    recupereContenuMacro = Resultat
End Function
```

Vérifier l'existence d'un module dans le classeur.

Vba

```
Sub Test()
    'Nécessite d'activer la référence
    ' "Visual basic For Application Extensibility 5.3"
    '
    MsgBox VerifierExistenceModule(Workbooks("NomClasseur.xls"), "Module2")
End Sub

Function VerifierExistenceModule(Wb As Workbook, Mdl As String) As Boolean
    Dim VBComp As VBComponent

    On Error Resume Next
    Set VBComp = Wb.VBProject.VBComponents(Mdl)

    If VBComp Is Nothing Then
        VerifierExistenceModule = False
    Else
        VerifierExistenceModule = True
    End If
End Function
```

Vérifier l'existence d'une macro dans le classeur.

Vba

```
Sub Test()
    'Nécessite d'activer la référence
    ' "Visual basic For Application Extensibility 5.3"
    '
    MsgBox VerifierExistenceMacro(Workbooks("NomClasseur.xls"), "testMacro")

End Sub
```

Vba

```
Function VerifierExistenceMacro(Wb As Workbook, NomMacro As String) As Boolean
    Dim x As Integer
    Dim vbcomp As VBComponent

    For Each vbcomp In Wb.VBProject.VBComponents
        On Error Resume Next
        x = vbcomp.CodeModule.ProcStartLine(NomMacro, 0)

        If x > 0 Then Exit For
    Next vbcomp

    If x = 0 Then
        VerifierExistenceMacro = False
    Else
        VerifierExistenceMacro = True
    End If

End Function
```

IV-C - Modifier

Remplacer un mot dans toutes les procédures d'un classeur.

Cet exemple remplace la chaîne "Feuil1" par "Feuil3".

Vba

```
Sub RemplacementMotDansProcedure()
    'Nécessite d'activer la référence
    'Visual basic For Application Extensibility 5.3"
    '
    Dim Ancien As String, Nouveau As String, Cible As String
    Dim VBComp As VBComponent
    Dim i As Integer
    Dim Wb As Workbook

    Set Wb = Workbooks("NomClasseur.xls")

    Ancien = "Feuil1"
    Nouveau = "Feuil3"

    For Each VBComp In Wb.VBProject.VBComponents
        For i = 1 To VBComp.CodeModule.CountOfLines
            Cible = VBComp.CodeModule.Lines(i, 1)
            Cible = Replace(Cible, Ancien, Nouveau)
            VBComp.CodeModule.ReplaceLine i, Cible
        Next i
    Next VBComp

End Sub
```

Renommer un module.

Vba

```
Sub Test()
    Dim Wb As Workbook
```

Vba

```
Set Wb = Workbooks("NomClasseur.xls")
RenommeModule Wb, "Module1", "NouveauNom"
End Sub

Sub RenommeModule(Wb As Workbook, Ancien As String, Nouveau As String)
Dim VBComp As VbComponent

Set VBComp = Wb.VBProject.VBComponents(Ancien)
VBComp.Name = Nouveau
End Sub
```

Remplacer une macro dans tous les modèles Word .DOT d'un répertoire.

(Procédure VBA Word).

Vba

```
Sub remplacement_Macro_WordDot()
'http://www.developpez.net/forums/viewtopic.php?p=2011706#2011706
Dim Debut As Integer, Lignes As Integer, X As Integer
Dim Fichier As String, Direction As String
Dim Doc As Document

Application.ScreenUpdating = False

'boucle sur tous les fichiers .dot du repertoire
Direction = "C:\Documents and Settings\michel\dossier\general\excel"
Fichier = Dir(Direction & "\*.dot")
Do While Fichier <> ""

Set Doc = Documents.Open(Direction & "\" & Fichier)

'suppression de la macro nommée "essai" dans Module1
With Doc.VBProject.VBComponents("Module1").CodeModule
Debut = .ProcStartLine("essai", 0)
Lignes = .ProcCountLines("essai", 0)
.DeleteLines Debut, Lignes
End With

'ajoute la macro nommée "MaNouvelleMacro" dans Module1
With Doc.VBProject.VBComponents("Module1").CodeModule
X = .CountOfLines
.InsertLines X + 1, "Sub MaNouvelleMacro()"
.InsertLines X + 2, "MsgBox " & "Coucou", vbInformation "
.InsertLines X + 3, "End Sub"
End With

DoEvents
Doc.Close True
Set Doc = Nothing

Fichier = Dir
Loop
Application.ScreenUpdating = True
End Sub
```

IV-D - Supprimer

Supprimer le module nommé "Module2" dans le classeur contenant cette macro.

```
Vba
Sub supprimerUnModule()
  With ThisWorkbook.VBProject.VBComponents
    .Remove .Item("Module2")
  End With
End Sub
```

Supprimer tous les modules vides dans le classeur actif.

```
Vba
Sub supprimerTousModulesVides()
  Dim vbComp As VbComponent
  Dim i As Integer, j As Integer

  For Each vbComp In ActiveWorkbook.VBProject.VBComponents
    If vbComp.Type = 1 Then
      i = vbComp.CodeModule.CountOfDeclarationLines + 1
      j = vbComp.CodeModule.CountOfLines
      If j < i Then ActiveWorkbook.VBProject.VBComponents.Remove vbComp
    End If
  Next
End Sub
```

Sauvegarder le classeur contenant cette macro, puis supprimer la totalité des procédures.

La macro "SupprimeTout" est aussi détruite.

```
Vba
Sub SupprimeTout()
  'Enregistre le classeur et supprime la totalité des procédures
  Dim VbComp As VbComponent

  'Enregistre le classeur
  ThisWorkbook.SaveAs "C:\Sauvegarde.xls"

  For Each VbComp In ThisWorkbook.VBProject.VBComponents
    Select Case VbComp.Type
      Case 1 To 3
        ThisWorkbook.VBProject.VBComponents.Remove VbComp
      Case Else
        With VbComp.CodeModule
          .DeleteLines 1, .CountOfLines
        End With
    End Select
  Next VbComp

  'Sauvegarde les modifications
  ThisWorkbook.Save
End Sub
```

Supprimer une macro précise nommée "Macro1" dans le "Module3".

Vba

```
Sub test()  
    'Nécessite d'activer la référence  
    ' "Visual basic For Application Extensibility 5.3"  
    ,  
    Dim Wb As Workbook  
  
    Set Wb = Workbooks("Classeur1.xls")  
    SupprimerMacroPrecise Wb, "Module3", "Macro1"  
  
    'Un autre exemple pour supprimer un procédure événementielle  
    'SupprimerMacroPrecise Wb, "ThisWorkbook", "Workbook_Open"  
  
End Sub  
  
Sub SupprimerMacroPrecise(Wb As Workbook, Mdl As String, NomMacro As String)  
    Dim Debut As Integer, Lignes As Integer  
  
    With Wb.VBProject.VBComponents(Mdl).CodeModule  
        Debut = .ProcStartLine(NomMacro, 0)  
        Lignes = .ProcCountLines(NomMacro, 0)  
        .DeleteLines Debut, Lignes  
    End With  
End Sub
```

IV-E - Importer et Exporter

Excel possède une méthode d'export pour enregistrer les composants dans des fichiers séparés: au format .bas pour les modules et .frm pour les UserForm. Vous pouvez ensuite utiliser la méthode d'import pour réutiliser les composants dans d'autres classeurs.

L'exemple suivant exporte 3 modules et 1 UserForm du classeur contenant la macro. Ensuite la procédure boucle sur tous les classeurs d'un répertoire cible afin d'y importer ces modules et UserForm.

Vba

```
Sub Export_Import_Module_Et_Userform()  
    Dim Fichier As String, Repertoire As String  
    Dim Wb As Workbook  
    Dim i As Byte  
  
    Application.ScreenUpdating = False  
  
    '-----  
    'Export du UserForm1 et de 3 Modules nommés Module1, Module2 et Module3  
    'qui sont dans le classeur contenant cette macro.  
    ThisWorkbook.VBProject.VBComponents("UserForm1").Export "C:\copieUSF.frm"  
  
    For i = 1 To 3  
        ThisWorkbook.VBProject.VBComponents("Module" & i).Export "C:\copieModule" & i & ".bas"  
    Next i  
    '-----  
  
    'Adaptez le répertoire des classeurs à modifier  
    Repertoire = "C:\Documents and Settings\michel\dossier"  
    Fichier = Dir(Repertoire & "*.xls")
```

Vba

```
'Boucle sur les classeurs du répertoire cible
Do While Fichier <> ""
    Set Wb = Workbooks.Open(Repertoire & "\" & Fichier)

    '-----
    'La procédure ne gère pas les erreurs si le nom des modules existe déjà
    'dans les classeurs.
    With Wb.VBProject 'Transfert l'USF et les modules dans les classeurs
        .VBComponents.Import "C:\copieUSF.frm"

        For i = 1 To 3
            .VBComponents.Import "C:\copieModule" & i & ".bas"
        Next i
    End With

    Wb.Close True
    Fichier = Dir
Loop

Application.ScreenUpdating = True

MsgBox "Opération terminée."
End Sub
```

L'exemple suivant montre comment importer et remplacer les procédures du module **ThisWorkbook**.

Il est possible de sauvegarder la procédure d'un module **ThisWorkbook** en l'enregistrant sur le disque dur (extension de fichier **.cls**). Vous pouvez ensuite réimporter ce fichier dans un autre classeur, pour par exemple effectuer la mise d'une procédure. Lors de l'import, le fichier va être créé comme un module de classe et non en lieu et place du module ThisWorkbook.

La procédure ci dessous permet d'automatiser le chargement du fichier **.cls** et le remplacement des macros dans le module objet **ThisWorkbook**.

Vba

```
'Nécessite d'activer la référence "Microsoft Visual Basic for Applications Extensibility 5.3"
Dim Wb As Workbook
Dim oModule As CodeModule
Dim VbComp As VBComponent
Dim x As Integer
Dim Cible As String

Cible = "NomModule"
'Définit le classeur de destination (qui doit être préalablement ouvert).
Set Wb = Workbooks("NomClasseur.xls")

'Charge le module dans le classeur
Set VbComp = Wb.VBProject.VBComponents.Import("C:\ThisWorkbook.cls")
'Le renomme (pour le supprimer plus facilement ultérieurement
VbComp.Name = Cible
```

Vba

```
Set oModule = VbComp.CodeModule

'Transfère les données chargées dans ThisWorkbook.
'Attention les données existantes dans "ThisWorkbook" sont écrasées.
With Wb.VBProject.VBComponents("ThisWorkbook").CodeModule
    x = .CountOfLines
    .DeleteLines 1, x
    .InsertLines 1, oModule.Lines(1, oModule.CountOfLines)
End With

'Suppression du module précédemment chargé
With Wb.VBProject.VBComponents
    .Remove .Item(Cible)
End With
```

V - Gérer les objets

Les objets peuvent être les feuilles de calcul, les UserForm, les contrôles...

Ce chapitre montre comment manipuler les objets et leurs propriétés. Vous y trouverez aussi deux méthodes pour associer des procédures événementielles à ces objets:

- * En "dur": Ecriture des nouvelles macros directement dans l'éditeur VBE, comme cela a été vu dans le chapitre IV-A.
- * En utilisant un module de classe.

V-A - Créer

Il est parfois utile de créer des objets dynamiquement, par exemple pour afficher une série de contrôles dont vous ne connaissez pas à l'avance le nombre nécessaire.

Voici une solution pour ajouter une série de CheckBox dynamiquement dans un UserForm, et gérer les événements de ces nouveaux contrôles à partir d'un module de classe.

Vba

```
'-----  
'Procédure de création des CheckBox à placer dans l'UserForm.  
'L'UserForm doit préalablement contenir un bouton nommé CommandButton1  
  
Option Explicit  
  
Private Sub CommandButton1_Click()  
Dim Obj As Control  
Dim Cl As Class1  
Dim i As Integer  
  
Set Collect = New Collection  
  
For i = 1 To 3 'boucle pour la création des CheckBox  
Set Obj = Me.Controls.Add("forms.CheckBox.1")  
With Obj  
    .Name = "moncheckbox" & i  
    .Object.Caption = "le texte" & i  
    .Left = 140  
    .Top = 30 * i + 10  
    .Width = 50  
    .Height = 20  
End With  
  
    'ajout de l'objet dans la classe  
Set Cl = New Class1  
Set Cl.ChkBx = Obj  
Collect.Add Cl  
Next i  
  
End Sub  
'-----
```

Vba

Vba

```
'-----  
'A placer dans un module standard  
Option Explicit  
  
Public Collect As Collection  
'-----
```

Vba

```
'-----  
'A placer dans un module de classe nommé "Classe1"  
'  
Option Explicit  
  
Public WithEvents ChkBx As MSForms.CheckBox  
  
'Exemple pour gérer l'évènement clic sur les objets type CheckBox  
Private Sub ChkBx_Click()  
    'cet exemple affiche le nom et la valeur de l'objet cliqué  
    MsgBox ChkBx.Name & " : " & ChkBx.Value  
End Sub  
'-----
```

Cet exemple montre comment créer un UserForm et une ListBox par macro ainsi qu'une macro événementielle "Click" pour la ListBox.

Vba

```
Option Explicit  
Dim Usf As Object  
  
Sub lancementProcedure()  
    Dim X As Object  
    Dim i As Integer  
    Dim strList As String  
  
    strList = "ListBox1"  
    Set X = creationUserForm_Et_listBox_Dynamique(strList)  
  
    For i = 1 To 10  
        X.Controls(strList).AddItem "Donnee " & i  
    Next i  
  
    X.Show  
  
    ThisWorkbook.VBProject.VBComponents.Remove Usf  
    Set Usf = Nothing  
End Sub  
  
Function creationUserForm_Et_listBox_Dynamique(nomListe As String) As Object  
    Dim ObjListBox As Object  
    Dim j As Integer  
  
    Set Usf = ThisWorkbook.VBProject.VBComponents.Add(3)  
    With Usf  
        .Properties("Caption") = "Mon UserForm"  
        .Properties("Width") = 300  
        .Properties("Height") = 200  
    End With  
End Function
```

Vba

```

End With

Set ObjListBox = Usf.Designer.Controls.Add("Forms.ListBox.1")

With ObjListBox
    .Left = 20: .Top = 10: .Width = 90: .Height = 140
    .Name = nomListe
    .Object.ColumnCount = 1
    .Object.ColumnWidths = 70
End With

With Usf.CodeModule
    j = .CountOfLines
    .InsertLines j + 1, "Sub " & nomListe & "_Click()"
    .InsertLines j + 2, "If Not " & nomListe & ".ListIndex = -1 Then MsgBox " & nomListe
    .InsertLines j + 3, "End Sub"
End With

VBA.UserForms.Add (Usf.Name)
Set creationUserForm_Et_listBox_Dynamique = UserForms(UserForms.Count - 1)
End Function
    
```

Cet exemple permet de créer une nouvelle feuille de calcul et d'y ajouter un bouton.

La procédure associée au nouveau bouton supprime le contenu des cellules.

Vba

```

Dim Ws As Worksheet
Dim Obj As OLEObject
Dim laMacro As String
Dim x As Integer

Set Ws = Sheets.Add 'Ajoute une nouvelle feuille

'ajoute un CommandButton dans la nouvelle feuille
Set Obj = Ws.OLEObjects.Add("Forms.CommandButton.1")
With Obj
    .Name = "monBouton" 'renomme le bouton
    .Left = 50 'position horizontale par rapport au bord gauche de la feuille
    .Top = 50 'position verticale par rapport au bord haut de la feuille
    .Width = 150 'largeur
    .Height = 30 'hauteur
    .Object.Caption = "Supprimer données feuille"
End With

'Spécifie le contenu de la macro qui sera associée au bouton
laMacro = "Sub monBouton_Click()" & vbCrLf
laMacro = laMacro & "Cells.Clear" & vbCrLf
laMacro = laMacro & "End Sub"

'Ajoute la procédure dans la feuille
With ThisWorkbook.VBProject.VBComponents(ActiveSheet.Name).CodeModule
    x = .CountOfLines + 1
    .InsertLines x, laMacro
End With
    
```

Il existe une deuxième méthode pour associer la procédure événementielle au bouton.

Vba

```
Dim Ws As Worksheet
Dim Obj As OLEObject
Dim laMacro As String
Dim x As Integer

Set Ws = Sheets.Add 'Ajoute une nouvelle feuille

'ajoute un CommandButton dans la nouvelle feuille
Set Obj = Ws.OLEObjects.Add("Forms.CommandButton.1")
With Obj
    .Name = "monBouton" 'renomme le bouton
    .Left = 50 'position horizontale par rapport au bord gauche de la feuille
    .Top = 50 'position verticale par rapport au bord haut de la feuille
    .Width = 150 'largeur
    .Height = 30 'hauteur
    .Object.Caption = "Supprimer données feuille"
End With

'Ajoute la procédure dans la feuille
With ThisWorkbook.VBProject.VBComponents(ActiveSheet.Name).CodeModule
    .CreateEventProc "Click", "monBouton"
    x = .ProcStartLine("monBouton_Click", vbext_pk_Proc)
    .InsertLines x + 2, "Cells.Clear"
End With
```

V-B - Lire

Boucler sur la collection de contrôles dans la feuille de calcul "Feuil1".

Vba

```
Sub boucleControlesFeuille()
    Dim Obj As OLEObject

    For Each Obj In Feuil1.OLEObjects
        Debug.Print Obj.Name 'Nom
        Debug.Print TypeName(Obj.Object) 'Type de contrôle
    Next Obj
End Sub
```

Boucler sur la collection de contrôles dans un UserForm.

Vba

```
Dim Ctrl As Control

For Each Ctrl In Me.Controls
    Debug.Print Ctrl.Name 'Nom
    Debug.Print TypeName(Ctrl) 'Type de contrôle
Next Ctrl
```

Compter le nombre de UserForm dans un classeur.

Vba

Vba

```
Sub CompteurUserForm()  
    Dim Wb As Workbook  
    Dim VBComp As VBComponent  
    Dim Valeur As Byte  
  
    Set Wb = Workbooks("leClasseur.xls")  
  
    For Each VBComp In Wb.VBProject.VBComponents  
        If VBComp.Type = 3 Then Valeur = Valeur + 1  
    Next VBComp  
  
    MsgBox " Il y a " & Valeur & " UserForm dans le fichier " & Wb.Name  
End Sub
```

Lister les propriétés d'un UserForm.

Vba

```
Dim i As Integer  
Dim VBcomp As Object  
  
On Error Resume Next  
Set VBcomp = ThisWorkbook.VBProject.VBComponents("UserForm1")  
  
For i = 1 To VBcomp.Properties.Count  
    Debug.Print VBcomp.Properties(i).Name & " : " & VBcomp.Properties(i).Value  
Next i
```

V-C - Modifier

Cet exemple permet de modifier les propriétés d'un UserForm

Celui ci ne doit pas être en cours d'affichage.

Vba

```
'Change la couleur de fond du UserForm.  
ThisWorkbook.VBProject.VBComponents("UserForm1").Properties("backcolor") = RGB(200, 200, 200)
```

Modifier les propriétés d'un objet CommandButton contenu dans un UserForm.

Celui ci ne doit pas être en cours d'affichage.

Vba

```
'Change la couleur de fond du CommandButton  
ThisWorkbook.VBProject.VBComponents("UserForm1"). _  
    Designer.Controls("CommandButton1").BackColor = RGB(225, 0, 0)
```

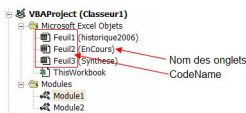
Changer le CodeName d'une Feuille.

Le CodeName correspond à la propriété Name de la feuille de calcul, à ne pas confondre avec le nom de l'onglet.

L'utilisation du CodeName est particulièrement pratique lors de la rédaction de vos procédures car Il permet:

- * l'affichage de la saisie semi automatique.
- * de faire référence à une feuille dans la macro, sans nécessité de modification lorsque le nom de l'onglet est changé.

Le CodeName et le nom de l'onglet sont identiques lors de l'ouverture d'un classeur vierge mais la modification du nom de l'un n'entraîne pas la modification de l'autre.



Un exemple pour modifier le CodeName de la 1ere feuille du classeur.

Remarque: Le codeName ne doit pas contenir d'espace.

Vba

```
Sub RenommerCodeName()
    Dim Ws As Worksheet

    'Spécifie la lere feuille dans le classeur contenant cette macro.
    Set Ws = ThisWorkbook.Sheets(1)

    'Attention: le nouveau nom ne doit pas contenir d'espace.
    ThisWorkbook.VBProject.VBComponents(Ws.CodeName).Name = "NouveauNom"
End Sub
```

V-D - Supprimer

Supprimer un UserForm par macro.

L>UserForm ne doit pas être en cours d'affichage.

Vba

```
Sub suppressionUSF()
    'Nécessite d'activer la référence
    'Microsoft Visual Basic for Applications Extensibility 5.3

    'L>UserForm ne doit pas être en cours d'affichage
    Dim VBComp As VBComponent

    Set VBComp = ThisWorkbook.VBProject.VBComponents("UserForm1")
```

Vba

```
ThisWorkbook.VBProject.VBComponents.Remove VBComp  
End Sub
```

Supprimer un CommandButton placé dans la feuille de calcul "Feuil1".

Vba

```
Dim Obj As OLEObject  
  
Set Obj = Sheets("Feuil1").OLEObjects("monBouton")  
Obj.Delete
```

Supprimer les contrôles qui ont été créés dynamiquement dans un UserForm.

(Voir l'exemple des CheckBox dans le 1er exemple du chapitre V-V-A)

Vba

```
Private Sub CommandButton2_Click()  
    Dim i As Integer  
  
    For i = Me.Controls.Count - 1 To 0 Step -1  
        'Attention la procédure est sensible à la casse.  
        If TypeName(Me.Controls(i)) = "CheckBox" Then _  
            Me.Controls.Remove Me.Controls(i).Name  
    Next i  
End Sub
```

VI - L'exécution des macros

Ce dernier chapitre propose quelques procédures pour déclencher des macros lorsque l'instruction `Call` ne peut pas être utilisée:

- * Lancer une macro dont le nom ou les arguments sont contenus dans des variables.
- * Déclencher les procédures événementielles des UserForm.
- * Déclencher les procédures contenues dans d'autres classeurs.

La méthode `Run` permet de déclencher des macros, et notamment événementielles.

Vba

```
'Déclenche l'évènement Open du module objet ThisWorkbook
Application.Run "ThisWorkbook.Workbook_Open"

'Déclenche l'évènement Click d'un CommandButton placé dans la Feuille
Application.Run "Feuille1.CommandButton1_Click"
```

Les anciennes macros XL4 peuvent aussi être utilisées pour lancer les macros.

Cet exemple exécute une macro dont le nom est contenu dans une variable String.

Vba

```
Dim sVar As String
'http://support.microsoft.com/kb/q108519/
sVar = "NomMacro"
Application.ExecuteExcel4Macro "RUN( "" & sVar & "" )"
```

Utilisez la fonction `CallByName` pour déclencher une procédure événementielle d'un contrôle contenu dans un UserForm.

Important: Le préfixe "Private" doit être préalablement supprimé.

Vba

```
CallByName UserForm1, "CommandButton1_Click", VbMethod
```

Utilisez la propriété Caller pour renvoyer le nom de l'objet appelant la macro.

Vba

```
MsgBox Application.Caller
```

Cette propriété est particulièrement intéressante lorsqu'une même macro est attachée à plusieurs objets (par exemple des formes automatiques) et que vous souhaitez retrouver la source du déclenchement.

Quelques exemples pour exécuter une macro contenue dans un autre classeur ouvert.

Il suffit d'utiliser la méthode **Run** en précisant le nom du classeur.

Vba

```
Application.Run "NomClasseur.xls!Module1.NomMacro"  
  
'déclenche une procédure événementielle ThisWorkbook  
'Application.Run "NomClasseur.xls!ThisWorkbook.Workbook_Open"  
  
'déclenche l'évènement Click d'un CommandButton placé dans la Feuille  
'Application.Run "NomClasseur.xls!Feuille1.CommandButton1_Click"
```

Vous pouvez aussi spécifier les arguments d'une procédure.

Vba

```
Application.Run "NomClasseur.xls!Module1.NomMacro", "Argument1", "Argument2"
```

Remarque:

Si le nom du classeur contient des espaces ou des caractères spéciaux, le nom doit être encadré par des quotes.

Vba

```
Application.Run "'Nom du Classeur.xls'!Module1.NomMacro"
```

VII - Téléchargement

