Cah. Techn. Inra, 2008, 65, 13-22

Des macros Excel pour exporter et importer des modules de code VBA

Jean-Baptiste Duclos[1]

Résumé : *Cet article présente deux macros VBA Excel pour sauvegarder facilement des macros sous Excel®. Il montre comment et pourquoi manipuler des macros sous Excel®*

Mots clés : macro VBA,Visual Basic Application, Microsoft Excel, importer, exporter, module de code, informatique, bureautique, tableur.

Introduction

Lors de la conception de macros sous Excel®, il est pratique de sauvegarder son programme dans un fichier texte ou de reprendre un code déjà créé. Nous allons présenter ici, la méthode manuelle sous Excel pour le faire ainsi que des macros permettant d'automatiser cette tâche. Cette méthode offre bien des facilités dès que l'on travaille sur plusieurs macros et plusieurs fichiers Excel à la fois. Elle facilite notamment la mise à jour du code des macros dans différents fichiers Excel qui contiendraient une même macro mais avec des versions différentes. Les paragraphes « *Petits rappel terminologique…* » et « *comment utiliser*… » de cet article seront utiles au débutant, bien que cette note soit plutôt destinée à celui qui connaît un minimum les Macros sous Excel® et la programmation.

1. Petits rappel terminologique, et de notions de bases

Dans ce texte, nous utilisons des termes techniques et des éléments dont la signification mérite d'être précisée afin de faciliter la lecture de l'exposé. Le terme code est synonyme de programmes, de macros, et le terme bibliothèque signifie collection de programmes réunie dans un même fichier ou répertoire. Le terme module désigne une feuille qui contient le programme de la macro. Les modules sont visibles dans l'éditeur de macro Excel, nommée Microsoft Visual Basic (**annexe 1**). Il y en a de plusieurs sortes (**encadré**), mais nous retiendrons le plus courant nommé le « module » sans plus de précision.

[1] UMR INRA-AgroParisTech Économie publique, UMR210 B.P. 01, 78850 Thiverval-Grignon
☎ 01 30 81 53 38 ✉ jean-baptiste.duclos@inrae.fr

http://jean-baptiste.duclos.club.fr/

Encadré

Les différents types de modules sous Excel

Le module désigne une feuille qui contient le programme d'une macro. Les types de modules sont nommés : module (normal), module lié aux objets Excel, module de classe. Ils ont différentes fonctionnalités :

Tout d'abord, parlons du plus courant, le **module normal**. Quand on parle d'un module sans préciser son type, il s'agit généralement de celui-ci. Personnellement, pour le désigner et éviter la confusion, j'utilise les termes module simple ou normal. Il permet de stocker les macros qui servent à tous les fichiers Excel et aux programmes d'autres applications. Pour créer un nouveau module, il faut actionner le menu « Insertion/Module » dans l'éditeur Microsoft Visual Basic.

Ensuite nous avons le **module lié aux objets Excel**. On le désigne sous le nom générique de module de feuille, car ses macros sont liées aux évènements qui se déroulent dans les feuilles d'Excel. Ainsi, à la sélection d'une cellule du classeur, Excel cherche à déclencher une macro du module de feuille correspondant à cette feuille de classeur. Si la macro est placée dans un autre module, elle ne se déclenchera pas. Il y a autant de modules liés aux feuilles que de feuille de classeur excel. De plus, il y a un module pour le classeur où l'on place, par exemple, les macros se déclenchant à l'ouverture ou à la fermeture du classeur… On ne peut ni les créer, ni les supprimer, ils sont présents dès la création d'un classeur. Mais bien sûr ils ne contiennent pas de macros au départ. Ces modules sont classés sous la branche Microsoft Excel Objets, de l'arbre des projets (menu « Affichage/Explorer Projet » dans l'éditeur Microsoft Visual Basic). Dans certains programmes, pour cause de compatibilité avec Excel 95, on peut trouver ces évènements programmés avec une ancienne méthode et peuvent être mis dans un module simple.

Enfin le **module de classe** sert à définir des méthodes et des propriétés, et permet de bénéficier des techniques issues de la programmation orientée objet. Il est plutôt réservé aux utilisateurs avancés. Pour créer un nouveau module de classe, il faut actionner le menu « Insertion/Module de classe » dans l'éditeur Microsoft Visual Basic.

2. Sauvegarder l'ensemble de ses macros Excel dans un fichier texte

La méthode manuelle pour sauvegarder des macros Excel est simple ; je la décris en **annexe 1**. Cependant, elle devient vite laborieuse quand on a plusieurs modules à sauvegarder, sans compter qu'elle peut se répéter autant de fois qu'il y a de fichiers à traiter… L'objectif de la macro (export_module) proposée en **annexe 2** est d'automatiser les sauvegardes de code, en l'effectuant d'emblée pour tous les modules et feuilles de code du classeur Excel. Des paramètres permettent de contrôler la procédure en définissant le fichier Excel à analyser, le répertoire de sauvegarde des macros, le type de module à sauvegarder, le nom de sauvegarde à utiliser, et si nécessaire d'exporter les feuilles de code vides ou inutiles. Pour plus de détails, consultez les commentaires du code fournis en **annexe 2**.

3. Restaurer des macros sous format texte dans un fichier Excel

Pour reprendre un code extérieur manuellement, la technique est très similaire : il suffit d'aller dans le menu « importer un fichier … » (annexe 1). De même, importer plusieurs modules de code pour plusieurs fichiers, peut être long. Et afin d'automatiser cette tâche, la macro « import_module » apporte les fonctionnalités complémentaires à « export_module ». Les arguments en paramètres sont identiques. Cette macro est utile à deux titres :
- en utilisation directe (**annexe 3**, lancement d'une macro Excel), elle permet d'aller chercher le code de tous les modules composant le fichier Excel et de les mettre à jour avec une nouvelle version du code disponible dans un autre fichier.
- une utilisation en macro (**annexe 2**, exemple de lancement: call…) permet automatiquement d'ajouter des bibliothèques de fonctions ou de macros à un fichier Excel.

4. Concrètement comment utiliser ces macros sans rien comprendre ou presque ?

Le programme « Gestion_Macro_V1,0.xls » facilite l'utilisation de ces macros (**annexe 4**). Grâce à celui-ci, vous disposez d'une boîte de dialogue conçue pour exploiter les fonctionnalités des macros d'importation et d'exportation présentées ci-dessus. Cette boîte de dialogue affiche deux boîtes à liste. Elles montrent du côté droit une liste avec les modules de macros à importer présents sur le disque dur et sur le côté gauche une liste avec les modules déjà présents dans l'application gestion macro v1.0.
Si vous n'utilisez pas ce programme, il faut créer un répertoire nommée « C:\Macro_VBA » qui stockera les macros VBA sous forme texte puis lancer la macro « import_module » ou « export_module » comme cela est expliqué dans l'**annexe 3**. Vous pourrez ensuite à loisir éditer les macros sous forme texte en les ouvrant à partir du répertoire « C:\Macro_VBA » et de l'explorateur Windows. Vous pourrez aussi apporter d'autres fonctionnalités à ce fichier Excel en y ajoutant les macros proposées en exemple (cf. §5). Vous pouvez télécharger les fichiers contenant les macros et les programmes sur mon site internet (cf. §6).

5. Exemple de macros apportant une fonctionnalité supplémentaire à Excel

En recopiant le programme fourni en **annexe 5** et en l'intégrant aux macros d'Excel, vous avez un exemple de fonctionnalité simple que l'on peut apporter à Excel. Cette macro permet de définir des actions à faire avec les touches de fonctions F6, F7, F8 ; ici :

- la touche F6 copie le contenu de la cellule Excel située juste au dessus de la cellule active ;

- la touche F7 permet de mettre en mémoire (dans le buffer du copier-coller) le contenu de la cellule active ;

- la touche F8 permet de restituer le contenu de la mise en mémoire par la touche F7 sur la cellule active.

La définition de ces touches facilite la saisie de données répétitives sur une feuille Excel. Pour activer l'action de ces touches, il faut lancer la macro def_touche_colle_copie. Pour la supprimer il faut lancer la def_touche_annule_colle_copie. Pour que cette fonctionnalité s'active toute seule à l'ouverture du classeur, il faut créer une procédure Auto_open, qui lancera la macro def_touche_colle_copie par l'instruction « call def_touche_colle_copie ». Vous pouvez aussi importer le module de code « Macro_Auto_Open_Colle_Copie.bas ». Si vous voulez avoir un fichier Excel prêt contenant ces fonctionnalités, télécharger le fichier

« Macro_Touche_F6-F7-F8.xls » (Tous les programmes et annexes cités ci dessus peuvent se télécharger sur mon site).

6. Aller plus loin

À partir du code sous forme de fichier texte, il est facile de comparer des versions de son code grâce à un logiciel de comparaison de texte (comparaison et fusion de document sous Word, mais aussi le logiciel très pratique Winmerge -gratuit et code source libre-).

- http://winmerge.org/index.php

De plus, on peut créer des bibliothèques de code réutilisables facilement, et reprendre du code Visual Basic (.Net ou Vb6 en adaptant parfois…).
Deux sites en français vous apporteront un complément d'information :

- http://www.vbfrance.com/, site de partage de code Visual Basic, mais aussi de bien d'autres langages.
- http://www.excel-downloads.com/ site dédié à Excel.

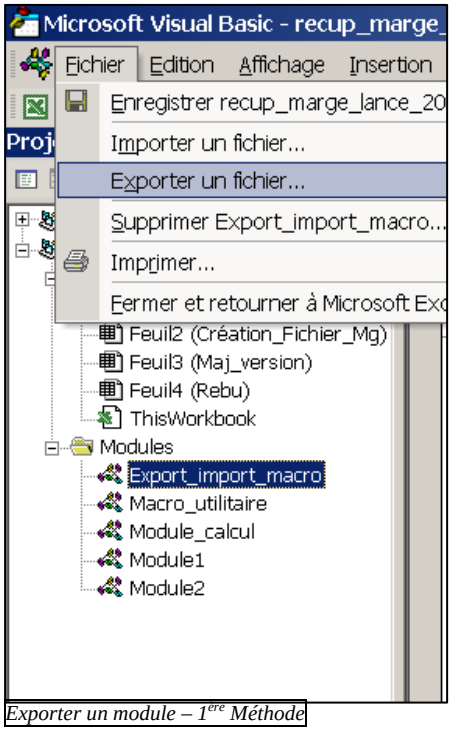
Les macros de sauvegarde et d'importation des macros de cet article peuvent être considérées comme des virus génériques sous Excel par les antivirus installés sur nos ordinateurs (surtout la partie import_module). Il n'en est rien évidement, mais il est certain qu'elles peuvent servir de base pour émettre un virus sous Excel à un auteur en herbe malveillant…
Vous pouvez télécharger ce code sur mon site personnel : http://jean-baptiste.duclos.club.fr

Conclusion

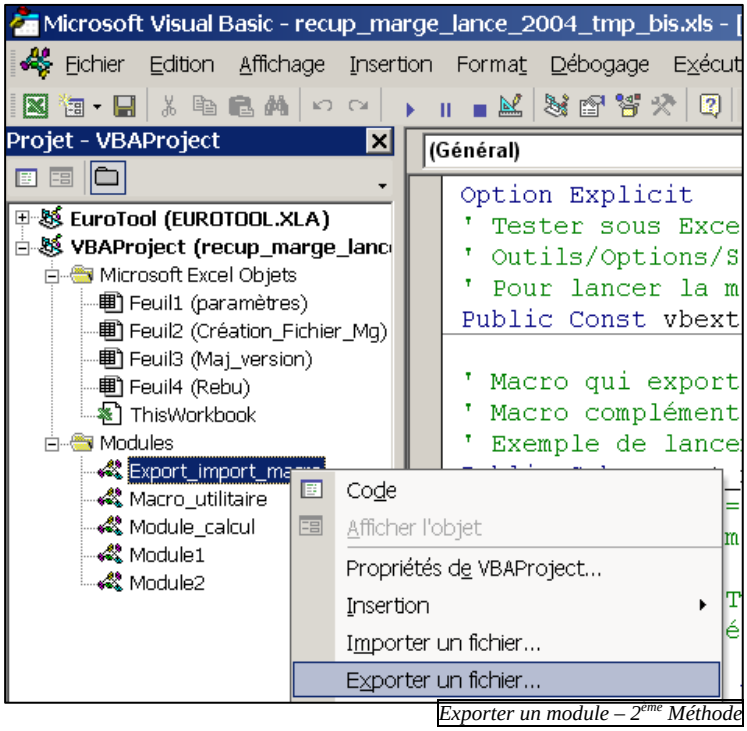
La sauvegarde et l'importation de code, nous permet d'ajouter et de conserver facilement tout un ensemble de programmes préconçus. De plus, une fois les programmes enregistrés sous format texte, il est facile de faire des comparaisons des versions de son code grâce à des logiciels dédiés. Elle permet aussi de faire des bibliothèques de code réutilisables facilement, et par conséquent, d'ajouter des fonctions et macros à Excel, de les partager.

Annexe 1

Méthode manuelle de sauvegarde des macros : Tout d'abord, ouvrir Microsoft Visual Basic en tapant Alt+F11, puis sélectionner le feuille de code à exporter, (ici Export_import_macro), et actionner le menu « Fichier\Exporter un Fichier … ».



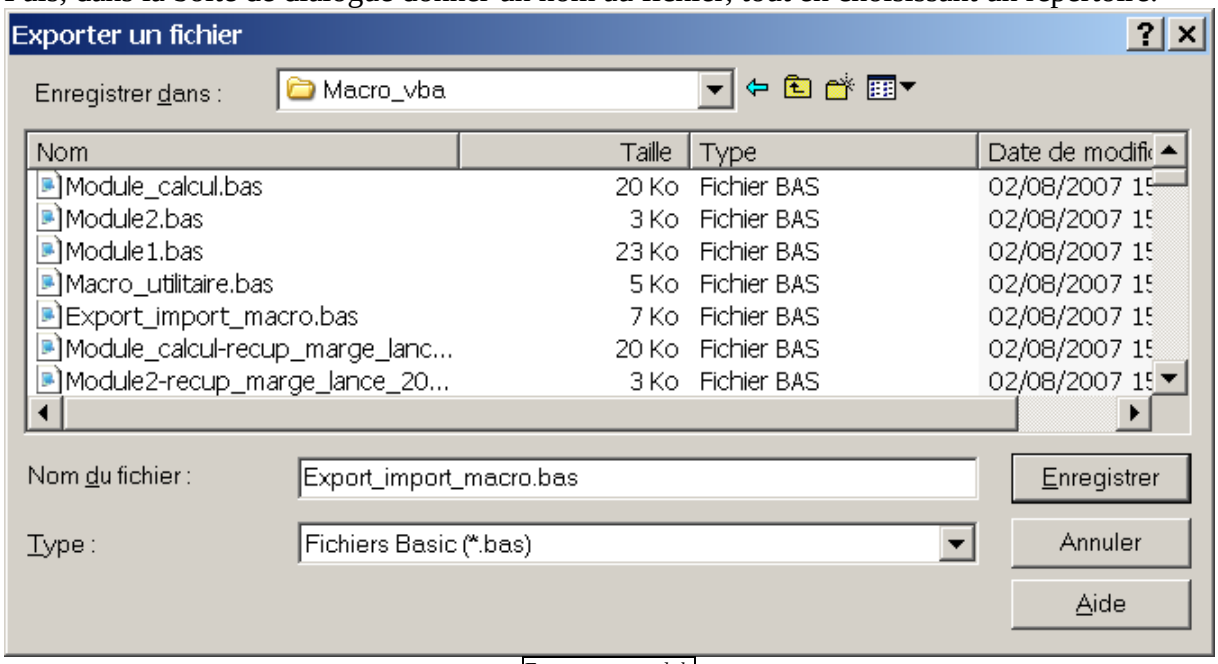
Exporter un module – 1ère Méthode



Exporter un module – 2ème Méthode

ou Clic droit sur le nom de la feuille de module du volet de l'explorateur de projet (menu : « Affichage\explorateur de projets CTRL+R»)

Puis, dans la boite de dialogue donner un nom au fichier, tout en choisissant un répertoire.



Exporter un module

Annexe 2

Code Visual Basic Application (VBA)

Macro export_module

```
Option Explicit
' Tester sous Excel 2002 (10.2614.2526), (VBA V10). Pour que ces macros marchent, il faut activer l'option suivante :
' Outils/Options/Sécurité/Sécurité des macros/Sources Fiables/"Faire confiance au projet Visual Basic"
' Pour lancer la macro, il faut désactiver sont antivirus, sinon l'antivirus prend ce programme pour un virus...
Public Const vbext_ct_StdModule As Integer = 1 'instruction utilisé pour compatibilité avec de vieille version Excel

' Macro qui exporte les codes des modules et feuilles dans un répertoire (y compris lui même, c'est à dire ce module)
' Macro complémentaire à import_module
' Exemple de lancement : Call export_module() ou Call export_module(Application.VBE.ActiveVBProject.Name,"C:\Macro_vba","Tous",1,false)
' (Remarque : si le code n'a qu'une ligne "option explicit", il est considéré comme vide)
Public Sub export_module(Optional nom_projet As Variant, Optional repert As Variant, Optional type_code As Variant, Optional type_nom As Variant, Optional vide As
Variant)
    ' nom_projet = nom du fichier dont on veut sauvegarder les macros (par défaut ce fichier)
    ' repert = Nom du répertoire où l'on veut sauvegarder ses macros (par défaut c:\macro_vba -il faut que ce répertoire existe, sinon çà plante )
    ' type_code = Type de code qu'on exporte : code module (=vbext_ct_StdModule), feuille, module de classe..., ou par défaut "Tous"
    ' type_nom = Type de nom qu'on donne aux fichiers exportés : type_nom=1->Nom du module,2->Nom du module+Nom du fichier, 3->1+2 (par défaut)
    ' vide = Booléen qui force la sauvegarde des modules sans code : True (sauvegarde tout mêmes les modules vides), False (Faux par défaut)
    Dim module As Variant
    Dim nom As String
    Dim nom_module As String, code_module As String
    Dim nb_module As Integer, i As Integer, nb_lg_code As Long

    'nom du projet à sauvegarder
    If IsMissing(nom_projet) Then nom_projet = Application.VBE.ActiveVBProject.Name
    'répertoire de sauvegarde des macros
    If IsMissing(repert) Then repert = "c:\macro_vba\"
    'type de module à exporter
    If IsMissing(type_code) Then type_code = "Tous"
    'type de nom à donner
    If IsMissing(type_nom) Then type_nom = 3
    'sauvegarder les modules vide ou pas
    If IsMissing(vide) Then vide = False

    nb_module = Application.VBE.VBProjects(nom_projet).VBComponents.Count

    For i = 1 To nb_module
      If Application.VBE.VBProjects(nom_projet).VBComponents(i).Type = type_code Or type_code = "Tous" Then
        Set module = Application.VBE.VBProjects(nom_projet).VBComponents(i)
        nom_module = module.Name
        nb_lg_code = module.codemodule.CountOfLines
        If nb_lg_code > 0 Then
          code_module = module.codemodule.Lines(1, Application.VBE.VBProjects(nom_projet).VBComponents(i).codemodule.CountOfLines)
          code_module = Replace(code_module, Chr(13) + Chr(10), "", 1, -1) ' Chr(13) + Chr(10) = vbCrLf
          code_module = Replace(code_module, "  ", " ", 1, -1)
        Else
          code_module = ""
        End If
        ' si le code ne contient que la ligne "option explicit" ou est vide on ne copie pas
        If Not (code_module = "Option Explicit") And nb_lg_code > 0 Or vide Then
          If type_nom = 1 Or type_nom = 3 Then
            ' sauvegarde avec nom du module uniquement (risque d'écrasement)
            nom = repert + nom_module + ".bas"
            Application.VBE.VBProjects(nom_projet).VBComponents(nom_module).Export (nom)
          End If
          If type_nom = 2 Or type_nom = 3 Then
            ' sauvegarde avec nom du module+nom du fichier (pas de risque d'écrasement)
            nom = repert + nom_module + "-" + Mid(ThisWorkbook.Name, 1, Len(ThisWorkbook.Name) - 4) + ".bas"
            Application.VBE.VBProjects(nom_projet).VBComponents(nom_module).Export (nom)
          End If
        End If
      End If
    Next i

End Sub
```

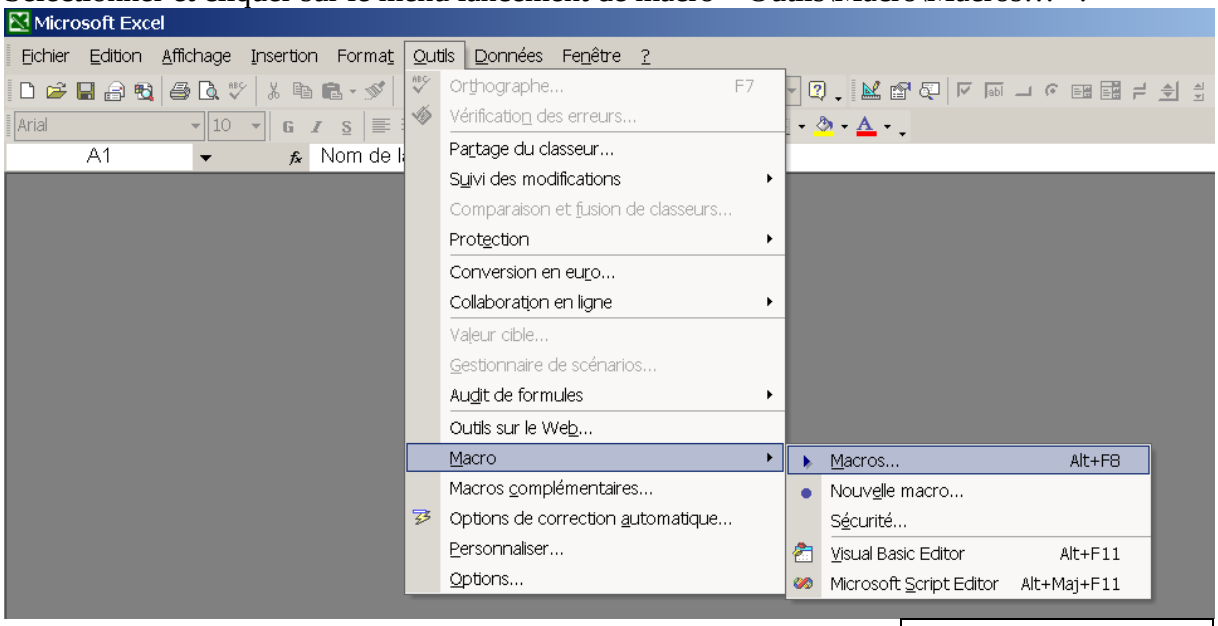
Macro import_module

```
' Macro qui importe les codes des modules et feuilles d'un répertoire (on a exporté auparavant)(non compris lui même, c'est à dire ce module)
' Dans l'optique de mettre à jour une nouvelle version de son code. (les modules existent déjà)
' Macro complémentaire à Exporte_Module
' Exemple de lancement : Call import_module() ou Call import_module(Application.VBE.ActiveVBProject.Name,"C:\Macro_vba","Tous",1,false)
' (Remarque : si le code n'a qu'une ligne "option explicit", il est considéré comme vide)
Public Sub import_module(Optional nom_projet As Variant, Optional repert As Variant, Optional type_code As Variant, Optional type_nom As Variant, Optional vide As
Variant)
    ' nom_projet = nom du fichier dont on veut sauvegarder les macros (par défaut ce fichier)
    ' repert = Nom du répertoire où l'on veut sauvegarder ses macros (par défaut c:\macro_vba -il faut que ce répertoire existe, sinon çà plante )
    ' type_code = Type de code qu'on exporte : code module (=vbext_ct_StdModule), feuille, module de classe..., ou par défaut "Tous"
    ' type_nom = Type de nom qu'on donne aux fichiers exportés : type_nom=1->Nom du module,2->Nom du module+Nom du fichier, 3->1+2 (par défaut)
    ' vide = Booléen qui force la sauvegarde des modules sans code : True (sauvegarde tout mêmes les modules vides), False (Faux par défaut)
    Dim module As Variant
    Dim nom As String
    Dim nom_module As String, code_module As String
    Dim nb_module As Integer, i As Integer, nb_lg_code As Long
    Dim module_import As Variant

    'nom du projet à sauvegarder
    If IsMissing(nom_projet) Then nom_projet = Application.VBE.ActiveVBProject.Name
    'répertoire de sauvegarde des macros
    If IsMissing(repert) Then repert = "c:\macro_vba\"
    'type de module à importer
    If IsMissing(type_code) Then type_code = "Tous"
    'type de nom donner au module
    If IsMissing(type_nom) Then type_nom = 3
    'sauvegarder les modules vide ou pas
    If IsMissing(vide) Then vide = False

    nb_module = Application.VBE.VBProjects(nom_projet).VBComponents.Count

    For i = nb_module To 1 Step -1
        If Application.VBE.VBProjects(nom_projet).VBComponents(i).Type = vbext_ct_StdModule Then
            Set module = Application.VBE.VBProjects(nom_projet).VBComponents(i)
            nom_module = module.Name
            nb_lg_code = module.codemodule.CountOfLines
            If nb_lg_code > 0 Then
                code_module = module.codemodule.Lines(1, Application.VBE.VBProjects(nom_projet).VBComponents(i).codemodule.CountOfLines)
                code_module = Replace(code_module, Chr(13) + Chr(10), "", 1, -1) ' Chr(13) + Chr(10) = vbCrLf
                code_module = Replace(code_module, "  ", " ", 1, -1)
            Else
                code_module = ""
            End If
            If type_nom = 1 Or type_nom = 3 Then
                ' sauvegarde avec nom du module uniquement (risque d'écrasement)
                nom = repert + nom_module + ".bas"
                Application.VBE.VBProjects(nom_projet).VBComponents(nom_module).Export (nom)
            End If
            If type_nom = 2 Then
                ' sauvegarde avec nom du module+nom du fichier (pas de risque d'écrasement)
                nom = repert + nom_module + "-" + Mid(ThisWorkbook.Name, 1, Len(ThisWorkbook.Name) - 4) + ".bas"
                Application.VBE.VBProjects(nom_projet).VBComponents(nom_module).Export (nom)
            End If
            'On n'importe pas ce module car il est en cours d'exécution...
            'Ucase à cause d'un bug sur le "i" de "Import" qui se transforme
            If Not (UCase(nom_module) = UCase("Export_Import_macro")) Then
                If Not (code_module = "Option Explicit") And nb_lg_code > 0 And vide Then
                    Set module_import = Application.VBE.VBProjects(nom_projet).VBComponents.Import(nom)
                    Application.VBE.VBProjects(nom_projet).VBComponents.Remove (module)
                    module_import.Name = nom_module
                End If
            End If
        End If
    Next i

End Sub
```

Annexe 3

Lancer une macro Excel

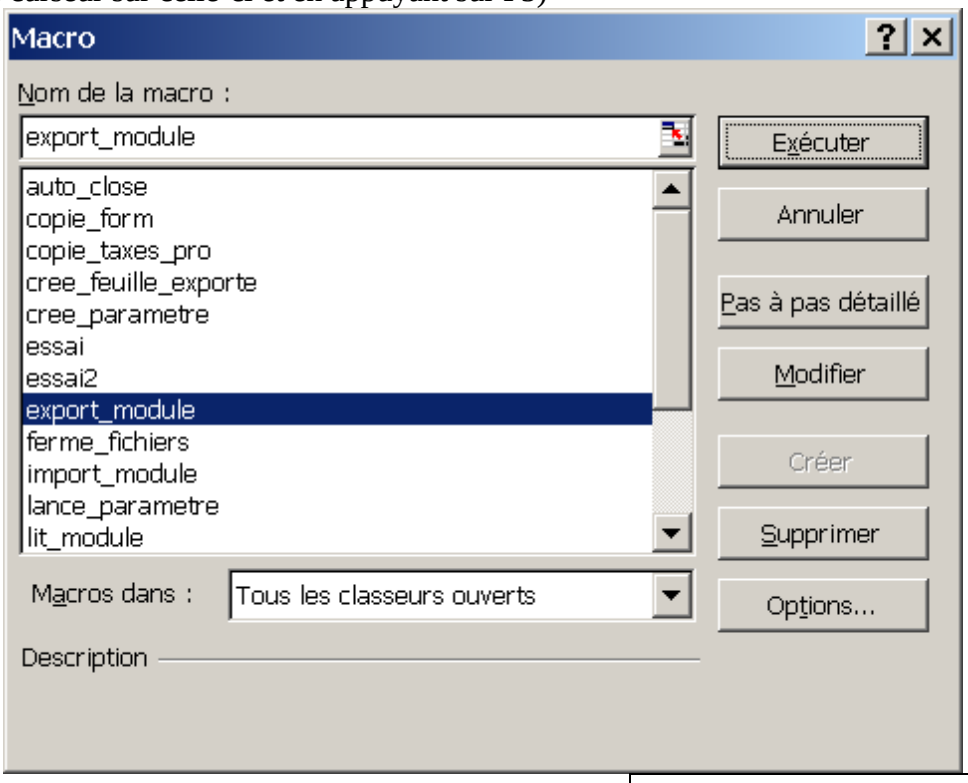
Sélectionner et cliquer sur le menu lancement de macro « Outils\Macro\Macros… ».



Lancement d'une macro

Sélectionner le nom de la macro, puis exécuter.

Autre méthode : ouvrir Microsoft Visual Basic, placer le curseur sur la macro à lancer puis appuyer sur F5… (Remarque : comme le montre la boîte de dialogue ci-dessus on peut ouvrir l'éditeur de macro à partir d'Excel en appuyant sur ALT+F11, puis actionner une macro en plaçant le curseur sur celle-ci et en appuyant sur F5)

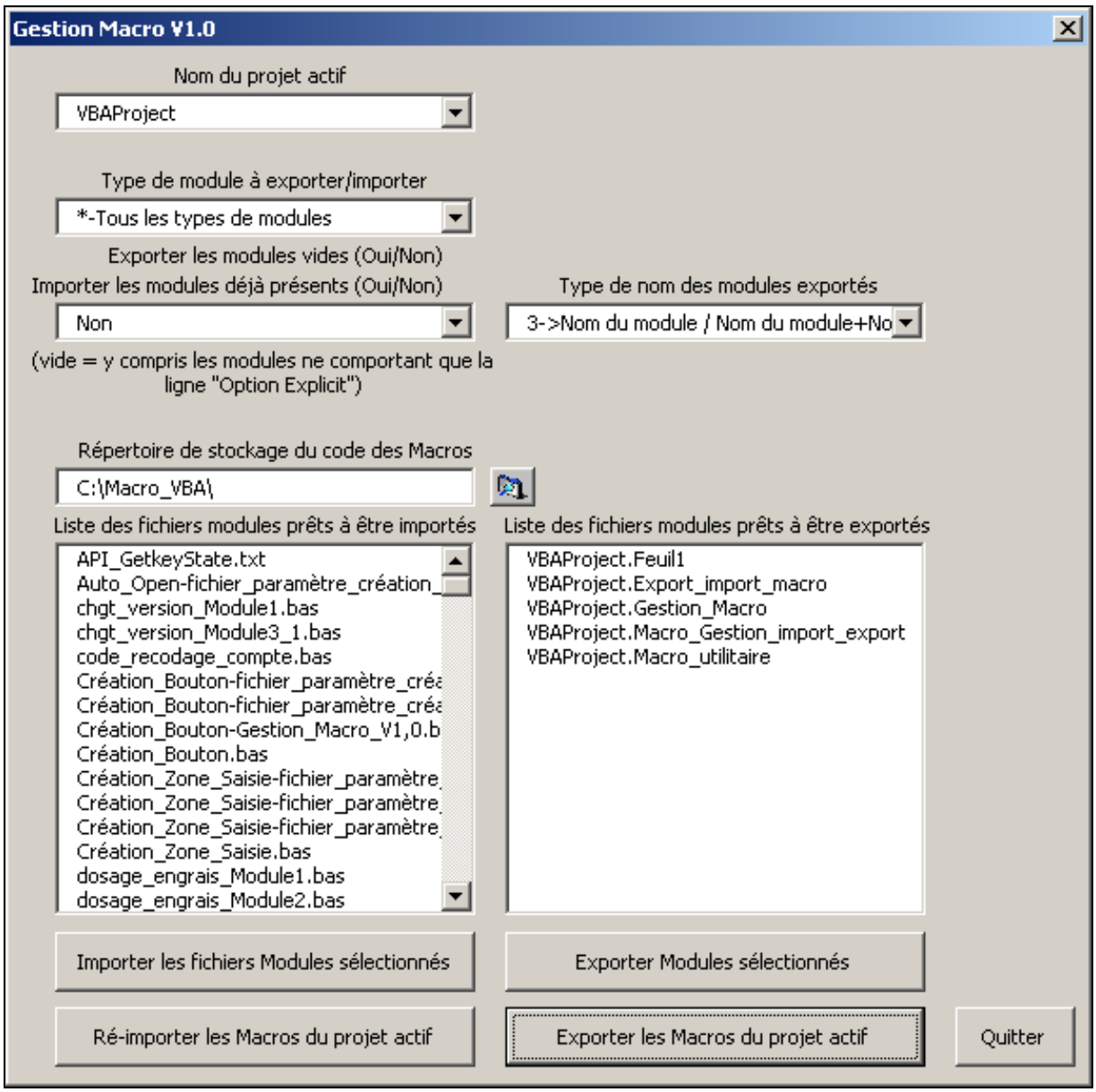


Lancement d'une macro sous Excel

Annexe 4

Application Gestion Macro V1.0 mettant en œuvre les macros de Gestion de macro

Dans la liste de droite nous avons l'ensemble des modules macros présents dans le répertoire « c:\macro_vba » du disque dur. Dans la liste de gauche nous avons l'ensemble des modules macros présents dans ce fichier « Gestion_Macro_V1,0.xls ». Vous retrouverez ce programme sur mon site internet (cf. §6).



Annexe 5

Présentation du code permettant d'ajouter des fonctionnalités sur les touches F6, F7, F8.

Ce code est à mettre dans un module normal. A partir de mon site internet, vous pouvez le télécharger le programme dénommé « Macro_deftouche_colle_copie.bas », et l'intégrer avec le menu « importer un fichier … » :

```
' Auteur : Jean-Baptiste Duclos
Option Explicit
Public Const NB_LIG_MAX As Long = 65536
Public memoire As Range
' F6, copie la cellule se situant au dessus dans la cellule courante. Si la cellule au dessus n'est pas visible, sinon colle le buffer
Public Sub copie_libellé()
    Dim cellule As Long
    Dim colonne As Long
    cellule = ActiveCell.Row
    colonne = ActiveCell.Column
    If cellule > 1 And cellule <= NB_LIG_MAX Then
        Do
            cellule = cellule - 1
            If cellule < 1 Then
                cellule = 1
                Exit Do
            End If
        Loop Until Cells(cellule, colonne).EntireRow.Hidden = False
        If Cells(cellule, colonne).EntireRow.Hidden = False Then
            Range(ActiveCell.Address) = Cells(cellule, colonne)
        Else
            Range(ActiveCell.Address) = memoire
        End If
    End If
End Sub
' F7, copie la cellule courante dans un buffer
Public Sub copie_libellé_buf()
    Set memoire = Range(ActiveCell.Address)
End Sub
' Auteur : Jean-Baptiste Duclos
' F8, colle la cellule courante dans un buffer
Public Sub colle_libellé_buf()
    Range(ActiveCell.Address) = memoire
End Sub
' Définition des comportements des touches F6,F7,F8 coller-copier
Public Sub def_touche_colle_copie()
    Application.OnKey "{F6}", "'" + ThisWorkbook.Name + "'!copie_libellé"
    Application.OnKey "{F7}", "'" + ThisWorkbook.Name + "'!copie_libellé_buf"
    Application.OnKey "{F8}", "'" + ThisWorkbook.Name + "'!colle_libellé_buf"
End Sub
' Définition des comportements des touches quand on sort d'excel, et qu'on annule F6,F7,F8
Public Sub def_touche_annule_colle_copie()
    Application.OnKey "{F6}"
    Application.OnKey "{F7}"
    Application.OnKey "{F8}"
End Sub
```

À ajouter pour permettre une insertion et une désactivation automatique des fonctionnalités

```
Public Const vbext_ct_StdModule As Integer = 1
' Auteur : Jean-Baptiste Duclos
' macro de lancement d'un programme à l'ouverture (définition de l'action des touches)
Public Sub auto_open()
    Call def_touche_appli
    MsgBox "les touches F6,F7,F8 sont définies", vbInformation, "avertissement"
End Sub
' macro de lancement d'un programme à la fermuture du fichier (annulation des touches définies)
Public Sub auto_close()
    Call def_annule_touche_appli
End Sub
```