La gestion des boucles dans Excel

par SilkyRoad (silkyroad.developpez.com)

Date de publication : 19/08/2007

Dernière mise à jour :

Cet article présente différents types de boucles utilisables dans Excel.

Testé avec Excel2002 et 2007.

I - Introduction

En programmation, une boucle, aussi appelée itération, permet d'effectuer une série d'actions de façon répétitive.

Il existe plusieurs solutions pour créer une boucle:

* **For Each Next**: Boucle sur chaque objet d'une collection.

* **For Next**: Répète une action le nombre de fois spécifié par un compteur.

* **Do Loop**: Itération pendant ou jusqu'à ce qu'une condition soit remplie.

* **While Wend**: Répète une action tant qu'une condition est vraie.

* **Boucle récursive**: Crée une procédure qui s'appelle elle-même pendant ou jusqu'à ce qu'une condition soit remplie.

II - For Each Next

Ce type de boucle peut être traduit par:

```
Pour chaque [élément] d'un [Ensemble]
 Série d'actions
[élément] suivant
```

Ici, [Ensemble] représente une collection.

L'architecture Excel est construite sous forme d'objets et de collections. C'est pour cette raison que le langage Visual Basic For Applications (VBA) est dit 'orienté objet'. Chaque collection possède des objets qui peuvent eux mêmes contenir d'autres collections:

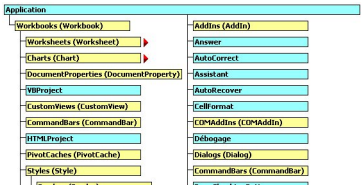
L'application possède une collection de classeurs.

Chaque classeur possède une collection de feuilles.

Chaque feuille peut contenir une collection de graphiques incorporés.

... etc ...

Dans ce schéma, issu de l'aide Excel 2002, L'objet Application représente le niveau le plus élevé.



Les éléments en bleu sont des objets.

Les éléments en jaune peuvent contenir des objets ou des collections.

Le principe de fonctionnement de l'instruction **For Each Next** va donc consister à boucler sur tous les objets d'une collection spécifique. Si la collection ne contient par d'objet ou quand tous les objets ont été parcourus, la boucle est fermée et l'exécution continue sur la ligne de code, juste après l'instruction **Next**.

Ce premier exemple boucle sur les classeurs ouverts dans l'application Excel:

Vba

```
Sub BoucleClasseurs()
    'Définit une variable qui va représenter un classeur à chaque itération.
    Dim Wb As Workbook

    'Boucle sur chaque classeur de l'application Excel
    For Each Wb In Application.Workbooks
        'Ecrit le nom de chaque classeur dans la fenêtre d'exécution Ctrl+G
        Debug.Print Wb.Name
    Next Wb
End Sub
```

Vous noterez que **Workbooks** représente la collection de tous les classeurs, et **Workbook** définit un objet de cette collection pour chaque itération. D'une manière générale en VBA, le nom des collections est différencié du nom d'objets par un s placé à la fin: Worksheets/Worksheet, ChartObjects/ChartObject ... etc ...

En reprenant le schéma vu précédemment, il est possible des créer des boucles imbriquées qui vont descendre jusqu'au niveau de détail le plus fin.

Ce nouveau code intervient sur la plage de cellules A1:A10, dans toutes les feuilles de tous les classeurs ouverts:

Vba

```
Sub BouclePlagesCellules()
    'Définit une variable qui va représenter un classeur
    Dim Wb As Workbook
    'Définit une variable qui va représenter une feuille de calcul
    Dim Ws As Worksheet
    'Définit une variable qui va représenter une cellule
    Dim Cell As Range

    'Boucle sur chaque classeur de l'application Excel
    For Each Wb In Application.Workbooks
        'Boucle sur chaque feuille de chaque classeur
        For Each Ws In Wb.Worksheets
            'Boucle sur chaque cellule de la plage A1:A10
            For Each Cell In Ws.Range("A1:A10")
                'Si la cellule contient la valeur 3, on multiplie la valeur par 2
                If Cell.Value = 3 Then Cell.Value = Cell.Value * 2
            Next Cell
        Next Ws
    Next Wb
```

Vba
```
    End Sub
```

Ce n'est pas une obligation pour le bon fonctionnement de la procédure, mais vous remarquez que chaque instruction **Next** est suivie du nom de la variable objet (Next Wb, Next Ws, Next Cell). Cette règle d'écriture améliore la relecture des codes volumineux et contenant de nombreuses boucles imbriquées. De même, utilisez l'indentation (décalage des portions de macro par l'insertion de tabulations) pour améliorer la lisibilité du code.

Pour gérer la sortie anticipée d'une boucle, utilisez l'instruction **Exit For**. Dans ce cas la procédure passe directement à la ligne de code qui suit l'instruction **Next**.

L'instruction **Exit For** peut être utilisée:

* Après la vérification d'une condition.

* **Lorsqu'une erreur se produit**.

Exemple:

Vba
```
Sub BoucleFeuilles()
    Dim Ws As Worksheet

    'Boucle sur les feuilles du classeur.
    For Each Ws In ThisWorkbook.Worksheets
        'On sort de la boucle si le nom de la feuille est "Feuil2".
        If Ws.Name = "Feuil2" Then Exit For

        MsgBox Ws.Name
    Next Ws
End Sub
```

III - For Next

Ce type de boucle peut être traduit par:

```
De [Compteur] = [Numéro de départ] à [Numéro d'arrivée]
 Série d'actions
[Compteur] suivant
```

L'instruction **For Next** permet de répéter des actions un nombre de fois prédéfini.

Vous devez spécifier une valeur de début [Numéro de départ] et une valeur de fin [Numéro d'arrivée]. La variable [compteur] va ensuite être incrémentée ou décrémentée à chaque itération.

Vba

```vba
Sub Test_V01()
    Dim x As Integer

    'La variable x va successivement prendre les valeurs 3 à 10
    For x = 3 To 10
        'Ecrit dans la plage de cellules A3:A10
        '( Cells(3, 1) correspond à la cellule A3,
        'Cells(4, 1) correspond à la cellule A4 ... etc... )
        Cells(x, 1) = "Ligne " & x
    Next x
End Sub
```

Par défaut, le compteur est incrémenté d'une unité positive à chaque itération.

Ajoutez l'argument **Step** afin de personnaliser le pas d'incrémentation. Cette valeur peut être positive ou négative. Une fois que toutes les actions spécifiées dans la boucle sont terminées, la valeur de l'argument **Step** est ajoutée à la variable [compteur].

La boucle s'exécute tant que la valeur [compteur] est inférieure ou égale à la valeur de fin [Numéro d'arrivée].

Vba

```vba
Sub Test_V02()
    Dim x As Integer

    'La variable x va successivement prendre les valeurs 3, 6 et 9
    For x = 3 To 10 Step 3
        'Ecrit dans les cellules A3, A6 et A9
        Cells(x, 1) = "Ligne " & x
    Next x
```

```vba
    End Sub
```

L'indication d'une valeur négative pour l'argument **Step**, permet de décrémenter le compteur de la valeur la plus élevée vers la valeur la plus basse. Vous devez spécifier une valeur de fin [Numéro d'arrivée] inférieure à la valeur de début [Numéro de départ].

La boucle s'exécute tant que la valeur [compteur] est supérieure ou égale à la valeur de fin [Numéro d'arrivée].

```vba
Sub Test_V03()
    Dim x As Integer

    'La variable x va successivement prendre les valeurs 10, 7 et 4
    For x = 10 To 3 Step -3
        'Ecrit dans les cellules A10, A7 et A4
        Cells(x, 1) = "Ligne " & x
    Next x
End Sub
```

Nota:

Lorsque vous utilisez l'argument **Step** pour décrémenter un compteur, le type de variable (x dans les exemples précédents) ne doit pas être **Byte**, sinon vous obtiendrez un message d'erreur 'Dépassement de capacité'.

Pour gérer la sortie anticipée d'une boucle, avant que le compteur n'atteigne la valeur de fin, utilisez l'instruction **Exit For**. Tout comme dans le type de boucle **For Each Next**, la procédure passe directement à la première ligne de code qui suit l'instruction **Next**.

L'instruction **Exit For** peut être utilisée:

* Après la vérification d'une condition.

* **Lorsqu'une erreur se produit**.

Exemple:

```vba
```

Vba
```
Sub Test_V04()
    Dim x As Integer

    'Boucle de 1 à 10
    For x = 1 To 10
        'On sort de la boucle si la cellule testée ne contient
        'pas une donnée numérique.
        If Not IsNumeric(Cells(x, 1)) Then Exit For
        'Multiplie le contneu de la cellule par 2
        Cells(x, 1) = Cells(x, 1) * 2
    Next x
End Sub
```

Vous pouvez bien entendu créer des boucles imbriquées et associer dans une même procédure les instructions **For Next** et **For Each Next**.

Vba
```
Sub TestBouclesImbriquees()
    Dim Ws As Worksheet
    Dim x As Integer, y As Integer

    'Boucle sur toutes les feuilles du classeur
    For Each Ws In ThisWorkbook.Worksheets
        'Crée une boucle de 1 à 10
        For x = 1 To 10
            'Crée une boucle de 1 à 7, avec un pas de 2.
            For y = 1 To 7 Step 2
                'Ecrit dans les colonnes A, C, E et G, de la ligne 1 à 10
                Ws.Cells(x, y) = "L" & x & "-C" & y
            Next y
        Next x

    Next Ws
End Sub
```

Bien que ce ne soit pas toujours conseillé, car cela complique la relecture du code, il est possible de modifier la valeur du compteur pendant les instructions de la boucle, entre chaque incrémentation.

Exemple pour extraire les valeurs numériques dans une chaîne de caractères:

Vba
```
Sub extraireValeursNumeriques_DansChaine()
    Dim x As Integer, Nb As Integer
    Dim Cible As String, Resultat As String
    Dim Nombre As Single

    Cible = "12,3azerty23,5 67"
    'il faut remplacer les virgules par des points
```

Vba

```
        'Pour que fonction Val puisse reconnaitre les décimales.
        Cible = Replace(Cible, ",", ".")
        'Remplacement des espaces par un caractère Alphabétique, pour
        'gérer deux nombres qui se suivent.
        Cible = Replace(Cible, " ", "$")

        'Boucle sur tous les caractères de la chaîne cible.
        '(La fonction Len renvoie le nombre de caractères contenu dans la variable "Cible").
        For x = 1 To Len(Cible)
            'Si le caractère est de type numérique
            If IsNumeric(Mid(Cible, x, 1)) Then
                'Extrait la donnée numérique de la chaîne
                Nombre = Val(Mid(Cible, x, Len(Cible) - x + 1))
                Nb = Nb + 1
                'Enregistre la valeur dans la variable de résultat
                Resultat = Resultat & Nombre & vbLf

                '--- Force l'incémentation de la variable compteur (x) du nombre de caractères
                'que contient la donnée numérique extraite:
                x = x + Len(Str(Nombre)) - 1
                ' ---
            End If
        Next

        MsgBox "Il y a " & Nb & " valeurs numériques dans la chaîne:" & vbLf & Resultat
    End Sub
```

IV - Do Loop

Les boucles **Do Loop**, associées aux mots clés **While** et **Until** permettent de répéter une ou plusieurs actions pendant ou jusqu'à ce qu'une condition soit remplie.

Les instructions peuvent être associées de différentes manières:

Do

Actions

Loop While condition

Qui peut être traduit par:

```
[Faire]
   Les actions à effectuer
[Recommencer] [Tant que] condition
```

Do While condition

Actions

Loop

Qui peut être traduit par:

```
[Faire] [Tant que] condition
   Les actions à effectuer
[Recommencer]
```

Do Until condition

Actions

Loop

Qui peut être traduit par:

```
[Faire] [Jusqu'à ce que] condition
   Les actions à effectuer
[Recommencer]
```

Do

Actions

Loop Until condition

Qui peut être traduit par:

```
[Faire]
   Les actions à effectuer
[Recommencer] [Jusqu'à ce que] condition
```

Vous pouvez utiliser les mots clés **While** et **Until** pour vérifier qu'une condition est remplie:

* Avant d'exécuter les actions contenues dans la boucle.

* Après chaque exécution de la boucle.

Do Loop peut exécuter des blocs d'instructions un nombre de fois indéfini. Il est donc important de s'assurer qu'une condition de sortie pourra être remplie et que la boucle ne tournera pas sans fin.

Dans cet exemple, la boîte de dialogue s'affiche tant que le mot de passe saisi est incorrect:

Vba

```
Sub SaisieMotDePasse()
    Dim Reponse As String

    'Affiche la boîte de dialogue tant que l'utilisateur n'a pas saisi
    '"mimi" (en minuscules).
    Do While Reponse <> "mimi"
        Reponse = InputBox("Saissez le mot de passe:", "Mot de passe")
    Loop

    MsgBox "OK, poursuite de la procédure."
End Sub
```

Cette autre procédure incrémente la variable "i" d'une unité à chaque itération, permettant ainsi de boucler sur les cellules de la colonne A jusqu'à trouver la chaîne "DVP".

Vba

```
Sub Boucle_V01()
    Dim i As Integer

    Do
        i = i + 1
    Loop While Cells(i, 1) <> "DVP"

    MsgBox "Trouvé ligne " & i
End Sub
```

Bien évidemment, si le mot "DVP" n'existe pas, la procédure ne va jamais s'arrêter. Pour y remédier vous pouvez ajouter l'instruction **Exit Do** qui permet d'anticiper la sortie de la boucle.

Voici une modification de la macro précédente afin de ne pas dépasser 1000 itérations, si le mot recherché n'est pas trouvé.

Vba

```
Sub Boucle_V02()
    Dim i As Integer

    Do
        i = i + 1
        'On sort après 1000 itérations.
        If i > 1000 Then Exit Do
    'On répète la boucle tant que le contenu de la cellule est différent
```

Vba
```
        'de la chaîne "DVP". (la procédure est sensible à la casse).
        Loop While Cells(i, 1) <> "DVP"

        'Affiche un message en fonction du résultat de la recherche.
        MsgBox IIf(Cells(i, 1) = "DVP", "Trouvé ligne " & i, "Pas trouvé")
    End Sub
```

Tout comme dans les autres méthodes, il est possible de créer des boucles imbriquées.

Voici une adaptation de l'exemple précédent (On sort de la procédure après 1000 itérations si le mot recherché n'est pas trouvé dans la colonne A):

Vba
```
    Sub Boucle_V03()
        Dim Trouve As Boolean
        Dim x As Integer

        Trouve = False

        Do
            'Boucle tant que le compteur x est inférieur à 50
            Do While x < 1000
                'Incrémente le compteur.
                x = x + 1
                'Vérifie le contenu de la cellule.
                If Cells(x, 1) = "DVP" Then
                    'Attribue la valeur Vrai si le mot est trouvé.
                    Trouve = True
                    'Anticipe la sortie de la boucle.
                    Exit Do
                End If
            Loop
        'Quitte la boucle si la variable à la valeur True.
        Loop Until Trouve = True Or x = 1000

        'Affiche un message en fonction du résultat de la recherche.
        MsgBox IIf(Trouve = True, "Trouvé ligne " & x, "Pas trouvé")
    End Sub
```

Do Loop est aussi parfois utilisée tant qu'un évènement est ou n'est pas survenu.

Cet exemple attend la fin de chargement d'une page html avant de poursuivre la procédure (Toutefois il est toujours préférable, quand c'est possible, d'utiliser les évènements spécifiques de l'application: **DocumentComplete** dans ce cas).

Vba

Vba
```vba
Sub piloterPageHtml()
    Dim IE As Object
    Dim T As Single

    T = Timer

    Set IE = CreateObject("internetExplorer.Application")
    IE.Visible = True
    'Affiche une page html dans internet Explorer
    IE.navigate "http://office.developpez.com/"

        'Attend la fin du chargement pour continuer la procédure.
        Do Until IE.readyState = 4 '(READYSTATE_COMPLETE)
            DoEvents
        Loop

    Debug.Print "Page chargée en " & (Timer - T) & " secondes."
End Sub
```

V - While Wend

L'instruction **While Wend** répète une action tant qu'une condition est vraie.

While condition

Actions

Wend

Qui peut être traduit par:

```
[Tant que] La condition à vérifier
   Les actions à effectuer
[Répéter]
```

Si la condition est vraie, les actions indiquées dans la procédure sont effectuées. Quand l'instruction **Wend** est atteinte, la procédure revient sur l'instruction **While** et la condition est de nouveau vérifiée. Si condition est toujours vraie, le processus est répété. Si la condition est fausse, l'exécution passe directement à la première ligne de code qui suit l'instruction **Wend**.

Vba

```
Sub Test_WhileWend()
    Dim i As Integer

    i = 1

    'Boucle sur les cellules de la colonne A
    'On sort de la boucle si la cellule testée (Cells(i, 1)) est vide
    While Not IsEmpty(Cells(i, 1))
        'Ecrit le contenu de la cellule dans la fenêtre d'exécution.
        Debug.Print Cells(i, 1)
        'Incrémente la variable d'une unité afin de tester la cellule suivante.
        i = i + 1
    Wend
End Sub
```

Remarque:

While Wend est incluse dans VBA pour assurer une compatibilité ascendante. Privilégiez l'instruction **Do Loop** qui permet d'exécuter une itération de manière plus structurée et plus souple (CF aide Excel).

La méthode **While wend** ne possède pas d'instruction spécifique pour la sortie anticipée de boucle.

VI - Les boucles récursives

Une procédure est dite **récursive** lorsqu'elle s'appelle elle-même.

Cette technique a des avantages mais peut aussi poser des problèmes d'espace mémoire (De la mémoire supplémentaire est utilisée à chaque fois qu'une procédure s'appelle elle-même). Une condition de sortie est également nécessaire afin de terminer la fonction récursive sinon elle peut boucler sur elle-même à l'infini.

Faites toujours vos premiers tests en mode pas à pas afin de vous assurer:

* Que la condition de sortie peut être remplie.

* Que la répétition des appels récursifs n'épuise pas l'espace mémoire disponible.

Pour optimiser les ressources mémoire:

* Faites le ménage dans vos codes: Supprimez les variables inutiles.

* **Adaptez correctement les types de données**.

* Evitez d'utiliser les types de données Variant.

* Vérifiez la structure de la procédure.

Vérifiez toujours préalablement si la récursivité ne peut pas être remplacée par des boucles imbriquées.

Cet exemple utilise la récursivité pour boucler sur le répertoire spécifié et tous ses sous-répertoires, afin de lister le nom des fichiers qu'ils contiennent.

Vba

```
Option Explicit

Sub TestListeFichiers()
    Dim Dossier As String

    'Définit le répertoire pour débuter la recherche de fichiers.
    '(Attention à ne pas indiquer un répertoire qu contient trop de sous-dossiers ou de
    'fichiers, sinon le temps de traitement va être très long).
    Dossier = "C:\Documents and Settings\mimi\dossier"

    'Appelle la procédure de recherche des fichiers
```

Vba

```
        ListeFichiers Dossier

        'Ajuste la largeur des colonnes A:E en fonction du contenu des cellules.
        Columns("A:E").AutoFit
        MsgBox "Terminé"
    End Sub



    Sub ListeFichiers(Repertoire As String)
        '
        'Nécessite d'activer la référence "Microsoft Scripting RunTime"
            'Dans l'éditeur de macros (Alt+F11):
            'Menu Outils
            'Références
            'Cochez la ligne "Microsoft Scripting RunTime".
            'Cluquez sur le bouton OK pour valider.

        Dim Fso As Scripting.FileSystemObject
        Dim SourceFolder As Scripting.Folder
        Dim SubFolder As Scripting.Folder
        Dim FileItem As Scripting.File
        Dim i As Long

        Set Fso = CreateObject("Scripting.FileSystemObject")
        Set SourceFolder = Fso.GetFolder(Repertoire)

        'Récupère le numéro de la dernière ligne vide dans la colonne A.
        i = Range("A65536").End(xlUp).Row + 1

        'Boucle sur tous les fichiers du répertoire
        For Each FileItem In SourceFolder.Files
            'Inscrit le nom du fichier dans la cellule
            Cells(i, 1) = FileItem.Name
            'Ajoute un lien hypertexte vers le fichier
            ActiveSheet.Hyperlinks.Add Anchor:=Cells(i, 1), _
                Address:=FileItem.ParentFolder & "\" & FileItem.Name
            'Indique la date de création
            Cells(i, 2) = FileItem.DateCreated
            'Indique la date de dernier acces
            Cells(i, 3) = FileItem.DateLastAccessed
            'Indique la date de dernière modification
            Cells(i, 4) = FileItem.DateLastModified
            'Nom du répertoire
            Cells(i, 5) = FileItem.ParentFolder

            i = i + 1
        Next FileItem


        '--- Appel récursif pour lister les fichier dans les sous-répertoire ---.
        For Each SubFolder In SourceFolder.subfolders
            ListeFichiers SubFolder.Path
        Next SubFolder

    End Sub
```

VII - Arrêter une boucle

Dans les chapitres précédents, nous avons vu les instructions **Exit For** et **Exit Do** pour forcer la sortie d'une boucle en fonction de conditions. Néanmoins, il y a toujours un risque de créer une boucle infinie si la condition de sortie n'est jamais remplie.

En cas d'urgence, si une macro de veut plus s'arrêter, utilisez simultanément les touches clavier: **Ctrl + Pause**.

Vous pouvez aussi utiliser la touche **Echap**.

Cette solution affiche une fenêtre d'erreur 'Exécution interrompue'. Cliquez sur le bouton **Fin** pour terminer la procédure.

Si vous souhaitez gérer les touches **Ctrl + Pause** ou **Echap** par macro, utilisez la propriété **EnableCancelKey**:

Cet exemple affiche un message personnalisé à la place de la fenêtre d'erreur, avant de sortir de la macro.

Vba

```
Sub GestionSortieBoucle_Echap()
    Dim x As Long

    On Error GoTo Fin
    Application.EnableCancelKey = xlErrorHandler

    'Crée un boucle assez longue pour vous donner le temps de tester
    'l'utilisation de la touche "Echap".
    For x = 1 To 50000
        Cells(x, 1) = x
    Next x

Fin:
   If Err.Number = 18 Then MsgBox "Opération annulée."
End Sub
```

VIII - Conclusion

La répétition d'actions va entrainer une durée de traitement plus ou moins longue en fonction:

* Du type de boucle.

* Du nombre d'itérations.

* De la logique et de la structure de votre code.

* De l'espace mémoire utilisé.

Avant d'utiliser une boucle, vérifiez qu'Excel ne dispose pas d'outils spécifiques qui peuvent être appliqués en lieu et place dans votre projet. Par exemple, pour supprimer tous les graphiques incorporés d'une feuille, plutôt que de boucler sur les objets, il est possible d'utiliser directement:

Vba

```
Feuil1.ChartObjects.Delete
```

Il convient de Choisir le type de boucle le mieux adapté à votre projet.

Faites des essais pour mesurer le temps d'exécution dans différentes configurations et en testant différentes méthodes.

Assurez vous qu'il existe au moins une condition qui permettra de terminer la boucle.

Si vous n'avez pas besoin de modifier directement les informations dans le classeur, l'utilisation des **variables tableaux** sera la méthode la plus rapide pour traiter des grands groupes de données.

Boucler sur une collection sera (généralement) plus rapide que d'utiliser un compteur.

Evitez, quand c'est possible, les boucles récursives.

Utilisez des types de données adaptés au contenu de chaque variable.

Evitez les variables **Variant** qui sont moins rapides à traiter et nécessitent plus d'espace mémoire.

La structure du code entre 2 itérations a aussi sont importance:

* Evitez au maximum les **Select** et **Activate** qui prennent beaucoup de temps.

* Désactivez la mise à jour de l'écran.

* Désactivez si besoin le recalcul automatique (N'oubliez pas de le réactiver en fin de procédure).

* Utilisez les sorties anticipées **Exit For** et **Exit Do** pour ne pas continuer une boucle alors que toutes les actions sont terminées.

IX - Remerciements

Je remercie toute l'équipe Office, et particulièrement **Starec** pour la relecture et la correction du tutoriel.

X - Téléchargement