LA GESTION DES ERREURS VBA

Les projets EXCEL VBA que nous développons sont susceptibles de **rencontrer un certain nombre de d'erreurs exécution**.

Si nous ne prenons pas soin d'inclure dans nos projets **une gestion de ces erreurs** d'exécution, les utilisateurs risquent de recevoir **des messages du système** plus ou moins clairs ou même troublants (comme la trop fameuse : « Erreur 1004 – Erreur définie par l'application ou par l'objet »).

Cette fiche « **Bonnes pratiques VBA** » s'attache à présenter de façon concise ce qu'il est nécessaire de connaître sur la gestion des erreurs d'exécution dans nos projets VBA.

(Les copies-écrans du présent document sont issues d'EXCEL version 365)

L'icône signale une remarque réclamant une **attention particulière**.

L'icône marque les **commentaires en marge** du sujet traité. Ils peuvent être ignorés sans inconvénient.

Table des matières

INTRODUCTION 2
INSTRUCTION « On Error Goto… » 3
PORTEE DE L'INSTRUCTION « On Error Goto… » 4
L'INSTRUCTION « On Error Goto 0 » 5
L'OBJET « Err» 6
Les propriétés de l'objet Err 6
Les méthodes de l'objet Err 6
INSTRUCTION « Resume… » 7
INSTRUCTION « On Error Resume Next » 8
MISE EN ŒUVRE D'UN FICHIER LOG 9

INTRODUCTION

Le but d'une gestion des erreurs dans un projet VBA consiste **à prévenir,** autant que possible, l'apparition **d'erreurs lors de l'exécution** et/ou faire en sorte **que des explications claires** sur les raisons de ces erreurs soit fournies aux utilisateurs.

Considérons que nous avons éliminé **toutes les erreurs de syntaxe, de variables ou de logique** de notre code lors d'une **phase de « débogage ».**

Si besoin, reportons-nous au document traitant de cette phase de débogage : **Excellons « Bonnes Pratiques VBA – Les outils de débogage »**

Si nous nous référons aux **principes de programmation structurée**, le langage VBA souffre de ce que l'on pourrait qualifier de « tare congénitale » du « Basic » : **l'instruction « Goto »**.

Dans nos projets VBA, il est possible (et chaudement recommandé) **d'éviter l'utilisation de l'instruction « Goto »** en utilisant les **instructions structurées** (For..Next ; Select Case ; Do Until…Loop ; …) et d'organiser nos projets en **procédures/sous-procédures/fonctions**.

En dépit de ces considérations, la mise en œuvre d'une gestion des erreurs, nous contraint à utiliser l'instruction non structurée : **« On Error Goto… ».**

Sachons que le langage **« VB.Net »** (lui-même lointain héritier de « Basic ») a introduit l'instruction **« Try…Catch…Finally »** permettant d'éradiquer définitivement l'instruction « On Error Goto ».

Il n'est pas exclu qu'une future version du langage VBA intègre une telle instruction…restons vigilants…

INSTRUCTION « On Error Goto… »

L'instruction « **On Error Goto** » nous permet **d'indiquer l'étiquette** à laquelle **VBA se débranche en cas d'erreur**.

C'est **l'instruction de base** que nous utilisons pour la gestion des erreurs :

```
Classeur 1.xlsx - Module1 (Code)
(Général)                                   ProcPrincipale
Option Explicit
Sub ProcPrincipale()
    Const cErr1 = 10001
    Const cErr2 = 10002
    Dim sParam1 As String
    On Error GoTo gestionErreurs
    'Code VBA.....

    'Appel d'une sous-procédure
    ProcSecondaire sParam1

    Exit Sub    'Pour stopper l'exécution de la procédure si aucune erreur
'Etiquette de débranchement en cas d'erreur
gestionErreurs:
    Select Case Err
        'Gestion des erreurs provoquées
        Case cErr1
            MsgBox "Une erreur de saisie s'est produite." & vbCrLf _
            & vbCrLf & "Veuillez corriger votre saisie." _
            , vbExclamation, "Avertissement"
            Resume
```

Dans le cas où **une erreur se produit dans la procédure**, **VBA exécutera la séquence de code** située après l'étiquette indiquée dans l'instruction **en ignorant tout code** situé entre l'instruction ayant provoqué l'erreur et l'étiquette.

Cette étiquette marque **le début de la séquence de gestion des erreurs** dans une procédure.

N'oublions pas **d'isoler la séquence de gestion des erreurs** par une instruction « **Exit Sub** » placée juste avant l'étiquette. Sinon, lorsque la procédure connait une fin sans erreur, **l'exécution passera par la séquence de gestion des erreurs**. Ce que, la plupart du temps, nous ne désirons pas :

```
    Dim sParam1 As String
    On Error GoTo gestionErreurs
    'Code VBA.....

    'Appel d'une sous-procédure
    ProcSecondaire sParam1

    Exit Sub    'Pour stopper l'exécution de la procédure si aucune erreur
'Etiquette de débranchement en cas d'erreur
gestionErreurs:
    Select Case Err
        'Gestion des erreurs provoquées
        Case cErr1
```

PORTEE DE L'INSTRUCTION « On Error Goto… »

Le langage VBA ne peut traiter qu'**une seule séquence de gestion d'erreurs à un instant donné**.

En conséquence, si une **sous-procédure appelée** ne possède pas **sa propre séquence de gestion d'erreurs** et qu'une erreur s'y produit, c'est la **gestion d'erreurs de la procédure appelante** qui sera déroulée par VBA.

Dans l'exemple de code ci-dessous, la **« ProcPrincipale »** possède une séquence de gestion d'erreurs commençant à l'étiquette **« gestionErreurs : ».**

La sous-procédure **« ProcSecondaire »** appelée dans « ProcPrincipale » **ne possède pas** de séquence de gestion d'erreurs. Si une erreur d'exécution s'y produit, c'est la séquence **« gestionErreurs : »** de **« ProcPrincipale »** qui est déroulée par VBA.

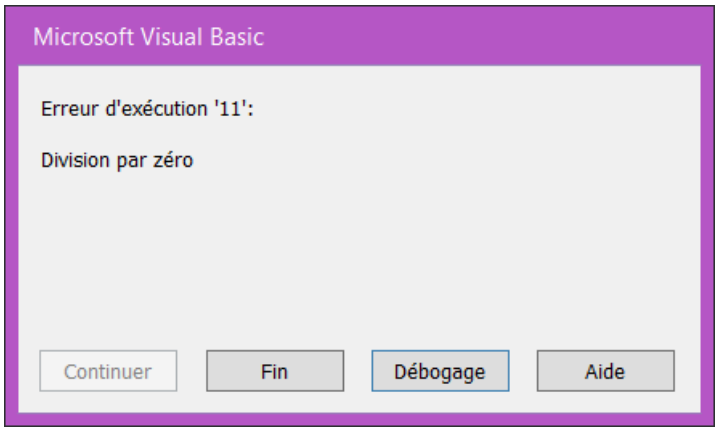
De même, la sous-procédure **« ProcTertiaire »** appelée dans « ProcSecondaire » **ne possède pas** de séquence de gestion d'erreurs. Si une erreur d'exécution s'y produit, c'est la séquence **« gestionErreurs : »** de **« ProcPrincipale »** qui est déroulée par VBA.

```
Sub ProcPrincipale()
    Dim sParam1 As String
    'On établit la gestion d'erreur
    On Error GoTo gestionErreurs

    'Appel d'une sous-procédure
    ProcSecondaire sParam1

    Exit Sub    'Pour stopper l'exécution de la procédure si aucune erreur
'Etiquette de débranchement en cas d'erreur
gestionErreurs:
    Select Case Err
        'Gestion des erreurs provoquées
        Case cErr0
            MsgBox "Traitement annulé par l'utilisateur", vbExclamation, "FIN..."

End Sub
Sub ProcSecondaire(zParm1 As String)
    Dim lParm2 As Long
    'Code VBA .....

    'Appel d'une sous-procédure
    ProcTertiaire lParm2
End Sub
Sub ProcTertiaire(zParm As Long)

    Dim lResult As Double
    lResult = 1000 / zParm
End Sub
```

L'INSTRUCTION « On Error Goto 0 »

Cette forme particulière de l'instruction « On Error Goto… » permet **d'annihiler l'effet de la dernière instruction « On Error »**.

Après cette instruction, si une erreur d'exécution se produit, VBA affiche **les messages** et la **boîte de dialogue système par défaut**. Par exemple, lors d'une erreur de division par 0 :

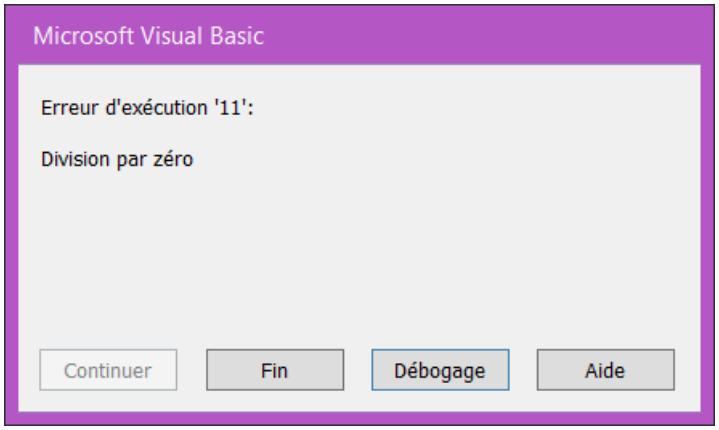


L'utilisation principale que nous avons de l'instruction « On error Goto 0 » : **mettre fin à l'effet** d'une instruction **« On Error Resume Next »** (cf. le paragraphe consacré à cette instruction).

L'OBJET « Err»

L'objet « Err » est utilisé par le langage VBA **pour fournir des détails sur l'erreur rencontrée** au cours de l'exécution.

Par exemple, comme vu ci-dessus, la division par 0 provoque, si elle n'est pas gérée, le message :



Les propriétés de l'objet Err

Nous utilisons principalement les propriétés suivantes de l'objet « Err » :

Err.Number = donne le numéro de l'erreur (dans l'exemple : « 11 »)

Err.Description = donne une description courte de l'erreur (dans l'exemple : « Division par zéro »)

Err.Source = Nom du projet d'origine dans lequel l'erreur s'est produite (non affiché dans la boîte de dialogue par défaut)

Les méthodes de l'objet Err

Les 2 méthodes de l'objet « Err » sont :

Err.Clear = efface le numéro et la description de la dernière erreur d'exécution rencontrée.

Err.Raise = permet d'affecter **une erreur propre à notre traitement** en lui donnant un numéro et éventuellement une description, une origine, …

Les numéros d'erreurs de **0 à 512 sont réservés aux erreurs VBA**. Les numéros d'erreurs **513 à 65535 nous sont disponibles** pour affecter les erreurs propres à nos traitements.

INSTRUCTION « Resume… »

L'instruction **« Resume »** a pour effet **d'annihiler la dernière erreur** d'exécution. Lorsqu'elle est suivi d'une étiquette, cela indique à VBA l'endroit où **l'exécution doit reprendre**.

Par exemple dans le code suivant, il est demandé à l'utilisateur **d'indiquer une donnée numérique** par l'intermédiaire de la méthode « InputBox ».

Lorsque la saisie de l'utilisateur **n'est pas numérique**, une erreur spécifique est déclenchée au moyen de la méthode **« Err.Raise »** (cf. le paragraphe consacré à cette méthode).

Dans la séquence de traitement des erreurs, le code relatif au traitement spécifique de cette erreur comporte l'instruction **« Resume… »** suivie d'une étiquette permettant de demander à l'utilisateur une nouvelle saisie en lui indiquant la nature son erreur :

```
    'On demande à l'utilisateur d'entrer son chiffre
    sLibelle = "Votre chiffre:"
    sTitre = "Merci d'entrer une valeur valide."
    lInitVal = 1
Saisie:
    sSaisie = InputBox(sLibelle, sTitre, lInitVal)
    'On verifie la saisie réalisée
    'Si l'utilisateur a annulé la saisie, on provoque une erreur
    If Len(sSaisie) = 0 Then
        Err.Raise cErr0
    End If
    If Not IsNumeric(sSaisie) Then
        'Dans le cas où la saisie est incorrecte, on provoque une erreur
        Err.Raise cErr1
    End If

    'Appel d'une sous-procédure
    If ProcSecondaire(sParam1) Then
        'Code VBA.....
    Else
        'Dans le cas où une erreur s'est produite dans la sous-procédure, on provoque une erreur
        Err.Raise cErr2
    End If

    Exit Sub    'Pour stopper l'exécution de la procédure si aucune erreur
'Etiquette de débranchement en cas d'erreur
gestionErreurs:
    Select Case Err
        'Gestion des erreurs provoquées
        Case cErr0
            MsgBox "Traitement annulé par l'utilisateur", vbExclamation, "FIN..."

        Case cErr1
            MsgBox "Une erreur de saisie s'est produite." & vbCrLf _
            & vbCrLf & "Veuillez corriger votre saisie." _
            , vbExclamation, "Avertissement"
            'On redemande la saisie
            Resume Saisie
```

INSTRUCTION « On Error Resume Next »

L'instruction « **On Error Resume next** » nous permet de demander à VBA **d'ignorer les erreurs d'exécution** à partir de cette instruction.

Nous devons manier l'instruction **« On Error Resume Next »** avec **grande prudence**.

La mauvaise idée : **placer cette instruction au début** d'une procédure principale, par exemple. Nous pourrions alors nous satisfaire de **ne produire aucun message d'erreur**.

Mais, ce faisant, **nous nous privons de toute information** concernant les erreurs rencontrées. Nous agissons comme ces malades qui, voulant ignorer leurs fièvres, cassent le thermomètre.

Il est certain cas où l'utilisation de « On Error Resume next » **est incontournable**. Mais alors, nous ne devons pas omettre de **mettre fin à son effet** (par une instruction « On Error Goto 0 » par exemple) **dès que possible** dans notre code.

L'exemple suivant, met en œuvre une **utilisation recommandée** de l'instruction **« On Error Resume Next »** dans le cas où l'affection d'une variable objet « Worksheet » génère une erreur à l'exécution :

```
Sub ProcPrincipale()
    Const cSheetName = "ACCUEIL"
    Dim sParam1 As String
    Dim oSheet As Worksheet
    Dim sLibelle As String, sTitre As String, lInitVal As Long
    Dim sSaisie As String

    'On ignore la gestion d'erreur
    On Error Resume Next
    'On affecte la variable objet représentant la feuille voulue
    Set oSheet = ThisWorkbook.Worksheets(cSheetName)
    If oSheet Is Nothing Then
        'Dans la cas où la feuille est inexistante, on la crée
        Set oSheet = ThisWorkbook.Worksheets.Add
        oSheet.Name = cSheetName
    End If
    'On rétablit la gestion d'erreur
    On Error GoTo gestionErreurs
```

MISE EN ŒUVRE D'UN FICHIER LOG

Nous pouvons sophistiquer notre **gestion d'erreurs** par la mise en œuvre d'un fichier **retraçant les évènements** (erreurs ou autres événements) se produisant lors de l'exécution de notre projet.

Cette mise en œuvre pouvant venir **soit en complément, soit en remplacement** d'une gestion des erreurs affichant des messages.

Cela nous permet, si nous prenons soin d'y **faire figurer tout événement pertinent**, d'analyser à postériori le comportement de notre application ainsi que les erreurs rencontrées pour les corriger.

Les principes d'une mise en œuvre d'un fichier log dans un projet VBA EXCEL sont les suivants :

- **A l'ouverture** du classeur (évènement 'Workbook_Open'), **création du fichier log** (procédure 'createLogfile' dans l'exemple joint ci-dessous).
- Lors de **la survenance d'une erreur** ou **d'un évènement** que nous voulons tracer, composition d'un message avec attribution d'un niveau de sévérité et **écriture dans le fichier log** (procédure 'writeIntoLog' dans l'exemple joint ci-dessous).
- **A la fermeture** du classeur (évènement 'Workbook_BeforeClose'), **clôture du fichier log** (procédure 'CloseLogfile' dans l'exemple joint ci-dessous).

Dans la séquence de gestion des erreurs, nous composons le message et le traçons dans la log. Par exemple :

```
gestionErreurs:
    Select Case Err
        'Gestion des erreurs provoquées
        Case cErr0
            MsgBox "Traitement annulé par l'utilisateur", vbExclamation, "FIN..."

        Case cErr1
            'On trace l'erreur la la log
            writeIntoLog "E", "Saisie de l'utilisateur non numérique"
            MsgBox "Une erreur de saisie s'est produite." & vbCrLf _
            & vbCrLf & "Veuillez corriger votre saisie." _
            , vbExclamation, "Avertissement"
            'On redemande la saisie
            Resume Saisie
```

Au moment où un évènement pertinent survient, nous composons le message et le traçons dans le log. Par exemple :

```
    'On ignore la gestion d'erreur
    On Error Resume Next
    'On affecte la variable objet représentant la feuille voulue
    Set oSheet = ThisWorkbook.Worksheets(cSheetName)
    If oSheet Is Nothing Then
        'Dans la cas où la feuille est inexistante, on la crée
        Set oSheet = ThisWorkbook.Worksheets.Add
        oSheet.Name = cSheetName
        writeIntoLog "W", "Avertissement : feuille 'Accueil' inexistante. Elle a été créée au moment de l'exécution"
    End If
    'On rétablit la gestion d'erreur
    On Error GoTo gestionErreurs
```

Exemple de mise en œuvre de fichier log :