Vk

Zeste de savoir

La gestion des erreurs en VBA

mercredi 03 juillet 2024

Table des matières

Introduction

En développant en VBA, vous avez déjà dû faire face à des erreurs d'exécution.

Ces erreurs, douloureuses, mais nécessaires, peuvent requérir une gestion bien particulière dans certains cas. Par exemple, nous pourrions imaginer une calculatrice où nous demanderions deux nombres ainsi qu'un opérateur binaire. Si l'utilisateur saisirait zéro en second nombre pour une division, le programme lèverait une erreur de division par zéro et nous pourrions capturer cette erreur pour demander un nouveau nombre à l'utilisateur.

Au cours de ce billet, nous allons voir comment fonctionne la gestion des erreurs en VBA.

C'est parti !

Pour ce billet, nous nous placerons dans l'environnement Microsoft Office.

1. Instructions

Le langage de macro de Microsoft met à disposition différentes instructions pour gérer les exceptions.

Si vous connaissez d'autres langages tels que Java, vous avez peut-être déjà utilisé des `try`, `catch` ou encore `finally`. Rien de tout cela en VBA, mais ce dernier offre tout de même des mécanismes afin de gérer les exceptions pour les capturer, mais aussi pour reprendre l'exécution du programme. Voyons cela.

1.1. Capturer l'erreur

Pour capturer une erreur, il faut identifier dans quelle instruction ou quel bloc d'instructions elle peut survenir. Si nous reprenons notre exemple de calculatrice, l'erreur surviendrait lors de l'évaluation du calcul, en divisant par zéro.

1.1.1. On Error GoTo étiquette

Dès lors, nous pouvons décider de capturer cette exception avec l'instruction `On Error GoTo étiquette`.

Le branchement de gestion d'exception doit se trouver au sein de la procédure, fonction ou propriété courante, sinon une erreur de compilation est levée.

```vba
Private Sub Exemple_On_Error_GoTo()
    On Error GoTo DivisionParZero
    Dim dResultat As Double
    dResultat = 1 / 1
    dResultat = 1 / 0 ' Affiche "Calcul impossible !"
    Exit Sub
DivisionParZero:
    Debug.Print ("Calcul impossible !")
End Sub
```

Il faut penser à ajouter une instruction pour quitter la fonction (`Exit Function`), la procédure (`Exit Sub`) ou encore la propriété (`Exit Property`) avant le bloc de gestion d'erreur sans quoi il sera aussi exécuté lorsqu'aucun problème n'a été rencontré.

1.1.2. On Error Resume Next

Une autre possibilité est de forcer le passage à l'instruction suivante en cas d'erreur. Nous avons alors le choix de gérer cette erreur de manière silencieuse ou non. Cela peut être utile pour des instructions qui peuvent lever des erreurs, mais pour qui c'est des comportements voulus que nous ne voulons pas traiter.

Voici un exemple avec la suppression du corps d'un tableau qui lève une erreur lorsque vide en VBA :

```vba
Private Sub Exemple_On_Error_Resume_Next()
    On Error Resume Next
    Sheets(1).ListObjects("Tableau1").DataBodyRange.Delete
```

```
4
5       ' Gestion silencieuse ou non en décommentant les lignes
            suivantes
6       'If Err.Number <> 0 Then
7           'Debug.Print ("Une erreur est survenue lors de la
                suppression du contenu du tableau...")
8       'End If
9       On Error GoTo 0
10  End Sub
```

Dans la documentation , il est aussi mentionné que c'est cette approche qu'il faut privilégier plutôt que `On Error GoTo étiquette` pour accéder à un objet avec `GetObject`.

Le revers de la médaille d'`On Error Resume Next`, c'est le risque d'ignorer *involontairement* des exceptions sans s'en rendre compte ! Cela peut amener des comportements ou des problèmes difficiles à cerner. Par exemple, imaginons un programme qui lit le contenu d'un fichier et utilisant cette instruction tout au début. Si une erreur survient (avec un fichier non trouvé par exemple), l'exécution du programme pourrait nous faire croire que le contenu du fichier à lire est vide. Il faut donc être particulièrement vigilant en utilisant cette instruction et l'appliquer à des blocs bien délimités comme montré ci-dessus.

1.1.3. On Error GoTo 0

Cette dernière variante de `On Error` annule l'effet des deux précédentes. Nous revenons alors à la gestion par défaut et toute erreur d'exécution produira le plantage du programme avec une fenêtre.

Cette instruction est exécutée de manière implicite en sortie de procédure, fonction et propriété.

```
1   Private Sub Exemple_On_Error_GoTo_0()
2       Dim dResultat As Double
3       On Error Resume Next
4       dResultat = 1 / 0 ' Rien
5       dResultat = 1 / 0 ' Rien
6       On Error GoTo 0
7       dResultat = 1 / 0 ' Erreur d'exécution 11 : Division par zéro
8   End Sub
```

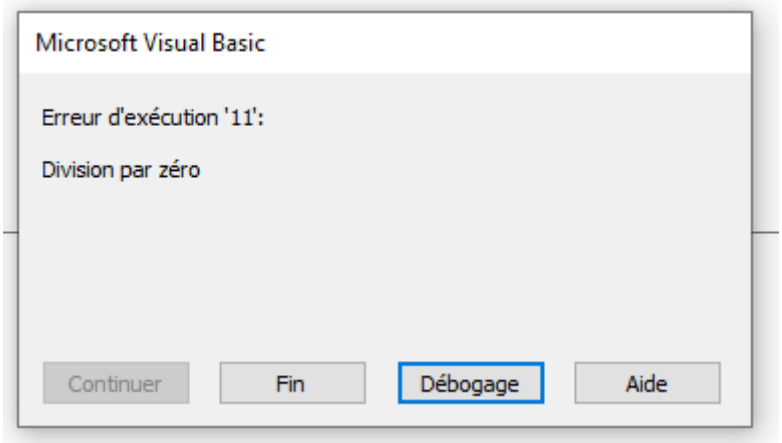

FIGURE 1.1. – Fenêtre erreur d'exécution

1.2. Reprendre l'exécution

Une fois l'erreur identifiée et gérée, nous pourrions souhaiter reprendre l'exécution. L'instruction `Resume` sert à cela avec trois possibilités.

1.2.1. Resume

`Resume` permet de reprendre à la ligne ayant généré l'exception.

```
1  Private Sub Exemple_Resume()
2      On Error GoTo GestionErreur
3      Dim dNombre2 As Double
4      Dim dResultat As Double
5      dNombre2 = 0
6      dResultat = 1 / dNombre2 ' Ligne générant une erreur la
           première fois
7      Debug.Print (dResultat) ' Affiche 1
8      Exit Sub
9  GestionErreur:
10     dNombre2 = 1
11     Resume ' Reprise à la ligne ayant généré l'erreur
12 End Sub
```

1.2.2. Resume Next

`Resume Next`, quant à elle, permet de reprendre à partir de la ligne suivante celle ayant généré l'exception.

```
Private Sub Exemple_Resume_Next()
    On Error GoTo GestionErreur
    Dim dNombre2 As Double
    Dim dResultat As Double
    dNombre2 = 0
    dResultat = 1 / dNombre2 ' Ligne générant une erreur
    Debug.Print (dResultat) ' Affiche 'inf'
    Exit Sub
GestionErreur:
    dNombre2 = 1
    Resume Next ' Reprise à la ligne suivante celle ayant généré
        l'erreur
End Sub
```

1.2.3. Resume étiquette

Enfin, `Resume étiquette` permet de reprendre l'exécution à un branchement, pourvu que celui-ci se trouve bien dans la procédure ou fonction ou propriété en cours.

```
Private Sub Exemple_Resume_GoTo_Etiquette()
    On Error GoTo GestionErreur
    Dim dNombre2 As Double
    Dim dResultat As Double
SaisieNombre:
    dNombre2 = CDbl(InputBox("Merci de saisir un nombre valide"))
        ' Erreur 'Incompatibilité de type' possible
    dResultat = 1 / dNombre2 ' Erreur 'Division par zéro' possible
    Debug.Print (dResultat)
    Exit Sub
GestionErreur:
    dNombre2 = 1
    Resume SaisieNombre
End Sub
```

Dans ce dernier exemple, remarquons que la saisie de la valeur 0 comme d'une valeur qui n'est pas un nombre entraînent la réouverture de la fenêtre de saisie. Il y a donc deux types d'erreur possibles dans ce programme : la division par zéro et l'incompatibilité de type lorsque la conversion de type (la transformation de la saisie en nombre) n'est pas possible. Pour le moment, nous ne savons pas identifier précisément le type d'erreur qui est survenu, mais nous verrons cela un peu plus loin.

1.3. Laisser l'erreur se propager pour s'en occuper ailleurs

Nous ne sommes pas obligés de gérer ou d'ignorer l'erreur au sein même de la fonction ou procédure ou propriété où elle est levée.

En effet, l'erreur se propage dans la pile des appels jusqu'à trouver un endroit où elle est prise en compte ou ignorée (sinon le programme plante). Il est ainsi possible de déléguer la gestion de l'erreur en dehors de la fonction ou procédure ou propriété où elle a été levée :

```
Private Sub Exemple_Propagation_Erreur()
    On Error GoTo GestionErreur
    Exemple_Division_Erreur
    Exit Sub
GestionErreur:
    Debug.Print Err.Number, Err.Description ' 11 Division par zéro
End Sub

Private Function Exemple_Division_Erreur() As Double
    Dim dResultat As Double
    Exemple_Division_Erreur = dResultat = 1 / 0 ' Erreur levée
End Function
```

Nous pouvons mettre encore plus de distance :

```
Private Sub Exemple_SuperPropagationErreur()
    On Error GoTo GestionErreur
    Exemple_Propagation_Erreur
    Exit Sub
GestionErreur:
    Debug.Print Err.Number, Err.Description ' 11 Division par zéro
End Sub

Private Function Exemple_Propagation_Erreur() As Double
    Exemple_Propagation_Erreur = Exemple_Division_Erreur ' Erreur
        qui se propage
End Function

Private Function Exemple_Division_Erreur() As Double
    Dim dResultat As Double
    Exemple_Division_Erreur = 1 / 0 ' Erreur levée
End Function
```

Au cours de cette première section, nous avons étudié différentes instructions dédiées à la gestion des exceptions en VBA.

2. L'objet Err

L'objet `Err` est un objet accessible de manière globale pour gérer l'erreur survenue.

Les propriétés de cet objet peuvent être remplies de façon automatique ou manuelle. De même, ses méthodes peuvent être exécutées de manière automatique ou manuelle.

2.1. Propriétés

Vous avez peut-être remarqué dans la fenêtre d'erreur d'exécution ce numéro, ce message de description ou encore ce bouton "Aide" qui nous redirige vers la page de la documentation en ligne. En fait, ce sont des propriétés ! Celles-ci sont toutes accessibles en lecture et en écriture.

2.1.1. Number

La propriété `Number` (qui est celle par défaut de l'objet) permet d'identifier le type erreur. Dans certains cas, plusieurs types d'erreur peuvent survenir, cela est donc très utile d'avoir son identifiant.

```
1  Private Sub Exemple_Propriete_Number()
2      On Error GoTo GestionErreur
3      Sheets(1).ListObjects("Tableau2").DataBodyRange.Delete
4      Exit Sub
5  GestionErreur:
6      Select Case Err.Number: ' Équivalent de Select Case Err:
7          Case 9
8              Debug.Print ("Le tableau n'a pas été trouvé")
9          Case 91
10             Debug.Print ("Le tableau est déjà vide")
11         Case Else:
12             Debug.Print ("Une autre erreur est survenue : " &
                   Err.Description)
13     End Select
14 End Sub
```

La valeur zéro correspond à l'absence d'erreur.

Pour lever une erreur personnalisée pour une classe, il faut prendre la constante `vbObjectError` pour base :

```
1  Err.Raise Number:=vbObjectError + 1001, Source:="clsUneClasse"
```

2.1.2. Description

La description de l'erreur est contenue dans sa propriété `Description`.

2.1.3. HelpFile & HelpContext

Ce sont les valeurs qui permettent d'accéder à l'aide pour l'erreur rencontrée. La première correspond au fichier tandis que la seconde correspond à la rubrique.

2.1.4. Source

La `Source` correspond à là où a eu lieu l'erreur. Cela peut-être un nom de classe ou encore un identificateur tel que le nom du projet.

2.2. Méthodes

En plus de ces propriétés, l'objet `Err` possède différentes méthodes.

2.2.1. Lever une erreur

Il est possible de lever une erreur à travers la méthode `Raise` en fournissant à minima un numéro d'erreur, les autres valeurs sont alors automatiquement déduites lorsque c'est une erreur standard.

Voici un programme où nous levons une erreur standard et une erreur personnalisée :

```
1  Private Sub Exemple_Raise()
2      On Error GoTo GestionErreur
3      Err.Raise 5
4      Err.Raise Number:=vbObjectError + 1001,
           Description:="Erreur survenue classe clsUneClasse",
           Source:="clsUneClasse"
5      Exit Sub
6  GestionErreur:
7      Debug.Print (Err.Number)
8      Debug.Print (Err.Description)
9      Debug.Print (Err.HelpFile)
10     Debug.Print (Err.HelpContext)
11     Debug.Print (Err.Source)
12     Debug.Print ("________")
13     ' 1ère erreur :
14     ' 5
15     ' Argument ou appel de procédure incorrect
16     ' C:\Program Files\Common Files\Microsoft
         Shared\VBA\VBA7.1\1036\VbLR6.chm
17     ' 1000005
18     ' VBAProject
19
20     ' 2ème erreur :
21     ' -2147220503
```

```
22      ' Erreur survenue classe clsUneClasse
23      ' C:\Program Files\Common Files\Microsoft
            Shared\VBA\VBA7.1\1036\VbLR6.chm
24      ' 1000440
25      ' clsUneClasse
26      Resume Next
27  End Sub
```

Notons que nous pouvons aussi créer une erreur à partir de celle en cours. De cette manière, nous pouvons gérer l'exception à différents niveaux, par exemple :

```
1   Private Function
        Exemple_Division_Avec_Gestion_Et_Nouvelle_Erreur(ByVal
        dNombre1 As Double, ByVal dNombre2 As Double) As Double
2       On Error GoTo GestionErreur
3       Exemple_Division_Avec_Propagation = dNombre1 / dNombre2
4       Exit Function
5   GestionErreur:
6       Debug.Print (Err.Number)
7       Err.Raise Number:=Err.Number, Description:=Err.Description,
            HelpFile:=Err.HelpFile, HelpContext:=Err.HelpContext,
            Source:=Err.Source
8   End Function
9
10  Private Sub Main()
11      On Error GoTo GestionErreur
12      Exemple_Division_Avec_Gestion_Et_Nouvelle_Erreur 1, 0
13      ' 11
14      ' Division par zéro
15      Exit Sub
16  GestionErreur:
17      Debug.Print (Err.Description)
18  End Sub
```

2.2.2. Vider l'erreur

L'objet est vidé de manière implicite en fin de procédure, fonction ou propriété, ou encore lors d'une reprise d'exécution (`Resume`) après gestion de l'erreur dans une étiquette.

Il peut aussi être vidé de manière explicite à travers sa méthode `Clear`.

Au cours de cette seconde section, nous vu comment tirer profit de l'objet global `Err`.

Conclusion

C'est déjà la fin de ce billet.

Au cours de celui-ci, nous avons vu comment fonctionne la gestion des erreurs en VBA, avec les différentes instructions d'abord et l'objet `Err` ensuite.

Toutes les erreurs ne sont pas faites pour être gérées ainsi (et d'ailleurs ce serait difficile 🍊). Il faut parfois chercher à comprendre le dysfonctionnement à l'origine du bogue ou du problème via le débogage par exemple.

À bientôt !

Quelques ressources :

— La documentation sur l'instruction On Error
— La documentation sur l'instruction Resume
— La documentation sur l'objet Err

Merci à Sarkas pour ses retours.

Liste des abréviations

VBA Visual Basic for Applications. 1