6 GESTION DES ERREURS

Il existe trois grandes catégories d'erreurs.
La première regroupe les erreurs de syntaxe. Elles sont faciles à détecter car le compilateur les signale. L'éditeur de VB les détecte même au cours de la frappe.

La deuxième partie est constituée des erreurs de logique (algorithme). Cette partie est plus importante et ces erreurs sont plus difficiles à corriger. Cela peut être des compteurs de boucles qui n'avance pas, ou des tests de sortie jamais vérifiés qui entraînent des boucles infinies.

La troisième recense les erreurs de fonctionnement. Par exemple, tenter d'ouvrir un fichier qui n'existe pas. Ces erreurs peuvent être gérées dans le code et interceptées pour ne pas interrompre le déroulement du programme. Pour être rattrapées, ces erreurs doivent être anticipées. Nous allons voir deux méthodes pour y parvenir.

6.1 L'objet Err

VB met à disposition un objet qui récupère les erreurs renvoyées par le système lors de l'exécution du programme. Cet objet stocke un numéro en fonction de l'erreur survenue, ainsi qu'une description.
Le programmeur peut ainsi gérer les erreurs en fonction des numéros retournés par *Err*.

Propriétés principales de *Err* :

Number : Valeur indiquée comme argument pour l'instruction *Error*. Il peut s'agir de n'importe quel numéro d'erreur valide.
Source : Nom du projet Visual Basic en cours.
Description : Chaîne correspondant à la valeur renvoyée par la fonction *Error* pour la propriété *Number* indiquée, si cette chaîne existe. Si la chaîne n'existe pas, l'argument Description contient le message "Erreur définie par l'application ou par l'objet".

On utilise l'objet avec l'instruction *On Error* placée en début de procédure. Cette instruction précise le comportement du programme en cas d'erreur. On utilise aussi *Resume* qui précise où reprendre l'exécution après la gestion d'une erreur.

Exemple :

```
Private Sub Toto()
On error Resume Next
        'Cette instruction précise qu'en cas d'erreur, on poursuit l'exécution à la ligne suivant celle qui a provoqué l'erreur.
        Code…

End Sub

Private Sub Toto()
On error Goto Piege_a_erreur
        Code…
```

```
	'Cette instruction précise qu'en cas d'erreur, on saute directement à l'étiquette Piege_a_erreur puis on exécute le code qui suit cette étiquette.

Piege_a_erreur :

	Code…

End Sub
```

6.2 Piège à erreurs

Cette méthode consiste à aller à une étiquette après toute erreur. On peut gérer les erreurs dans une sous partie de la procédure. On utilise l'instruction *On Error Goto Une Etiquette* pour atteindre la gestion d'erreur située en fin de procédure.

On ajoute une instruction *Exit Sub* ou *Exit Function* pour quitter la procédure sans exécuter la gestion d'erreur si la procédure se déroule normalement.

Ex :

```
Sub MaProcedure()
On Error Goto Trait_Erreur  'Etiquette à atteindre si erreur

	Code…

Trait_Fin:
	Exit Sub		'Sortie si pas d'erreur sans exécuter ce qui suit

Trait_Erreur:
	Select Case Err		'Traitement de l'erreur selon le numéro
	Case 1004
		Resume Next 'Exécution de la ligne suivant celle qui
	Case Else			'a généré l'erreur
		MsgBox "Erreur numéro : " & Err.Number & vbcrlf & _
		"Description : " & Err.Descripion
		Resume Trait_Fin
	End Select
End Sub
```

6.3 Traitement en ligne

Le principe est de tester l'objet *Err* après une ligne suspecte pour gérer une éventuelle erreur. On utilise l'instruction *On Error Resume Next*. Cette instruction précise qu'il faut exécuter la ligne suivante en cas d'erreur.

Cette technique impose d'anticiper efficacement les erreurs possibles car seules certaines lignes seront suivies d'un test de l'objet *Err*. On utilise la méthode *Clear* pour réinitialiser l'objet *Err*.

Par sécurité, on utilise un piège à erreur pour traiter les erreurs non prévues. On quitte ainsi la procédure toujours proprement.

```
Sub Toto()
On Error Resume Next        'Passe à l'instruction suivante si erreur
   ...
        'Instruction suspecte (si répertoire inexistant)
        ChDir "C:\AIGEM"

        Gestion de l'erreur
        Select Case Err
        Case 76                             'Erreur de chemin d'accès
                MkDir "C:\AIGEM"            'Création du répertoire
                ChDir "C:\AIGEM"            'Changement de répertoire
        Err.Clear                       'L'erreur est gérée on vide l'objet
        Case Else
                Goto Trait_Erreur       'erreur non gérée (cas imprévu)
        End Select
...
Exit Sub
Trait_Erreur :                  'Piège à erreur
        Select Case Err
        Case Else
                MsgBox "Erreur numéro : " & Err.Number & vbcrlf & _
                "Description : " & Err.Descripion
        End Select
End Sub
```

Voici enfin un exemple efficace de gestion d'erreur. Il permet d'exécuter une partie de code quoi qu'il arrive. Cette technique repose sur le piégeage des erreurs.

```
Sub MaProcedure()
On Error Goto Trait_Erreur  'Etiquette à atteindre si erreur

        Code…

Trait_Fin:      'Ce code est exécuté qu'il y ait des erreurs ou non
        Code…
        Exit Sub                'Sortie si pas d'erreur sans exécuter ce qui suit

Trait_Erreur:  'Etiquette atteinte sur erreur
        Select Case Err         'Traitement de l'erreur selon le numéro
        Case 1004
                Resume Next
        Case Else               'Traitement d'erreurs imprévues
                MsgBox "Erreur numéro : " & Err.Number & vbcrlf & _
                "Description : " & Err.Descripion
        Resume Trait_Fin        'Reprise de l'exécution à l'étiquette de sortie de procédure
        End Select
End Sub
```

Cette méthode présente l'avantage de toujours exécuter le code placé entre les étiquettes « Trait_Fin » et « Trait_Erreur ». En effet, si tout se déroule sans erreur, la section « Code… » va s'exécuter jusqu'à l'instruction *Exit Sub* placée après *Trait_Fin*.

Si une erreur non gérée spécifiquement survient, le cas *Case Else* sera exécuté et l'instruction *Resume Trait_Fin* renvoie l'exécution du programme à l'étiquette spécifiée.
C'est l'occasion de libérer les variables objet, de fermer les fichiers, annuler une transaction etc.

L'essentiel

N'oubliez pas que l'aide en ligne est votre premier support de développement. N'hésitez pas à la consulter fréquemment. La connaissance exhaustive de tous les événements, de toutes les propriétés et méthodes de tous les objets est illusoire. Ainsi, vous apprendrez un langage et vous formerez sur chaque nouvel objet ou contrôle.

Vous trouverez divers ouvrages pour obtenir de l'information sur ce qui a été traité ici, notamment l'accès aux données qui constitue un élément important du développement VB. Intéressez-vous à SQL, langage indispensable de manipulation de données.

Vous pouvez enfin poursuivre avec les thèmes suivants. Ils sont présentés dans un ordre cohérent et vous permettront d'aller plus loin avec Visual Basic.