Le débogage sous Visual Basic 6

par DarkVader (http://www.developpez.com/)

Date de publication : 29 octobre 2011

Dernière mise à jour :

Tout ce que vous devez savoir sur le débogage et la gestion des erreurs sous Visual Basic 6.

I - Introduction

La lecture du forum Visual Basic laisse à penser que beaucoup d'utilisateurs ne connaissent pas ou mal certaines possibilités de l'IDE VB6 concernant le traitement des erreurs ni l'éventail des possibilités offertes pour le débogage : il m'a donc semblé utile de rédiger ce tutoriel sommaire concernant les spécificités de Visual Basic 6 dans ce domaine.

Pour tout complément d'information, consultez la **MSDN**

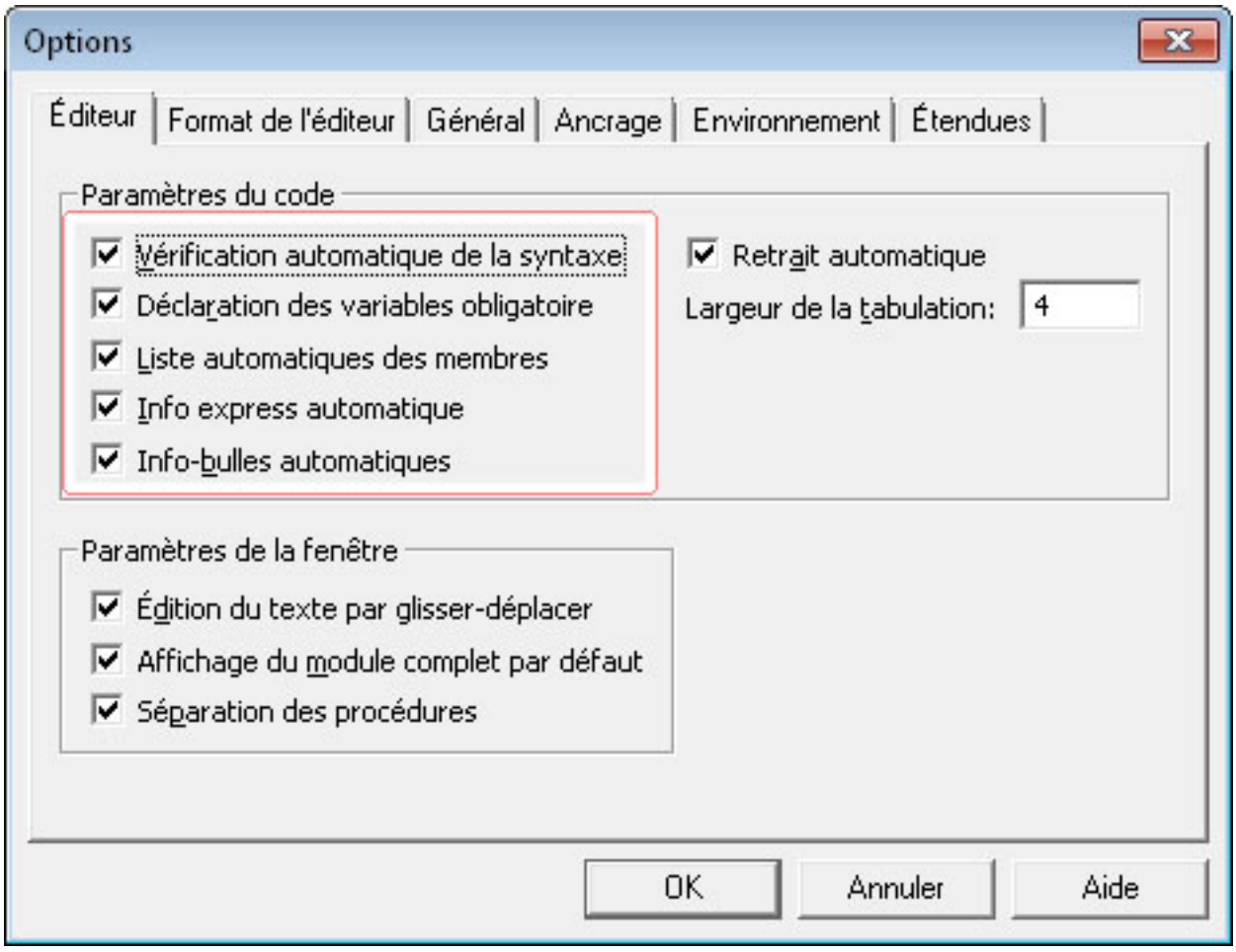
II - Les outils

L'IDE Visual Basic est riche d'outils d'assistance qui permettent notamment l'aide à l'écriture, son contrôle, la gestion personnalisée du mode runtime ou la possibilité d'interactions etc .

II-1 - Les outils d'aide à l'écriture

Menu Outils/Options (Alt OT) > Onglet Editeur

L'IDE Visual Basic dispose avec la technologie Intellisense de différents outils d'aide à l'écriture (Listes automatiques , Info-bulles, des Infos-express). Outre le fait de limiter le risque d'erreurs de syntaxe, ils ont pour propriété de devenir non fonctionnels s'il existe une erreur (défaut de typage, de paramètres etc) ***ce qui permet donc indirectement de signaler une erreur.***



Cocher toutes les options «Paramètres du code»

II-1-1 - Vérification automatique de la syntaxe

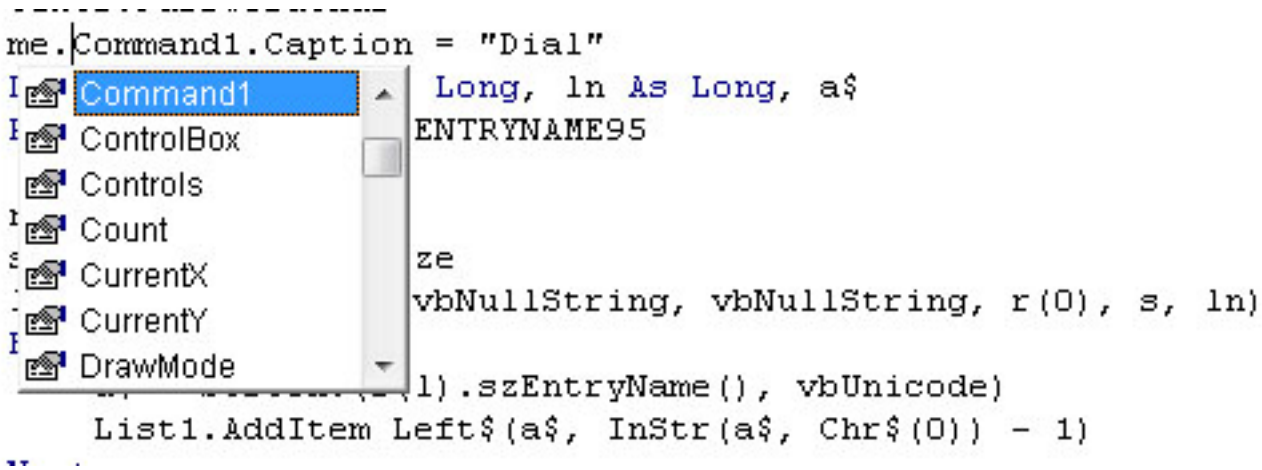
La vérification automatique de la syntaxe permet de vérifier la syntaxe à chaque ligne de code.

II-1-2 - Déclaration des variables obligatoire

Cocher systématique l' «Option Explicit» force à déclarer chaque variable en ajoutant une instruction «Option Explicit» en tête de module au moment de sa création.
L'absence de typage étant une source importante d'erreurs, il est recommandé d'effectuer le typage de chaque variable.
(Il est même recommandé de pratiquer un typage fort => non utilisation du type Variant)

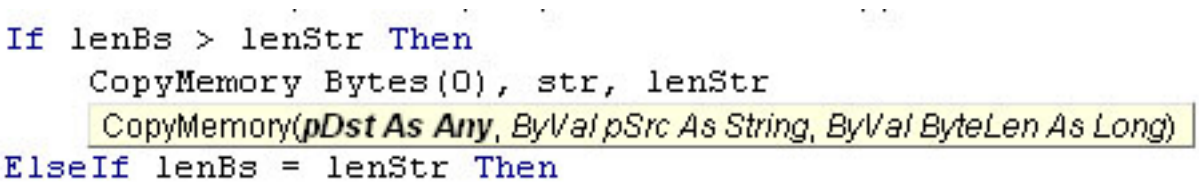
II-1-3 - Auto-completion

L'auto-completion (Liste automatique des membres) fournit au développeur dans une zone de liste les propriétés, méthodes etc du modèle objet sélectionné (racine) ce qui permet de réduire le temps de développement et assure un code syntaxiquement correct. (Caractère d'activation «.»)



II-1-4 - Info-express automatique (Ctrl Maj I/Ctrl I)

La zone d'Info-express est complémentaire de l'auto-completion; elle fournit la syntaxe de fonction, d'instruction, de méthode et des paramètres liés en indiquant le paramètre en cours d'édition.



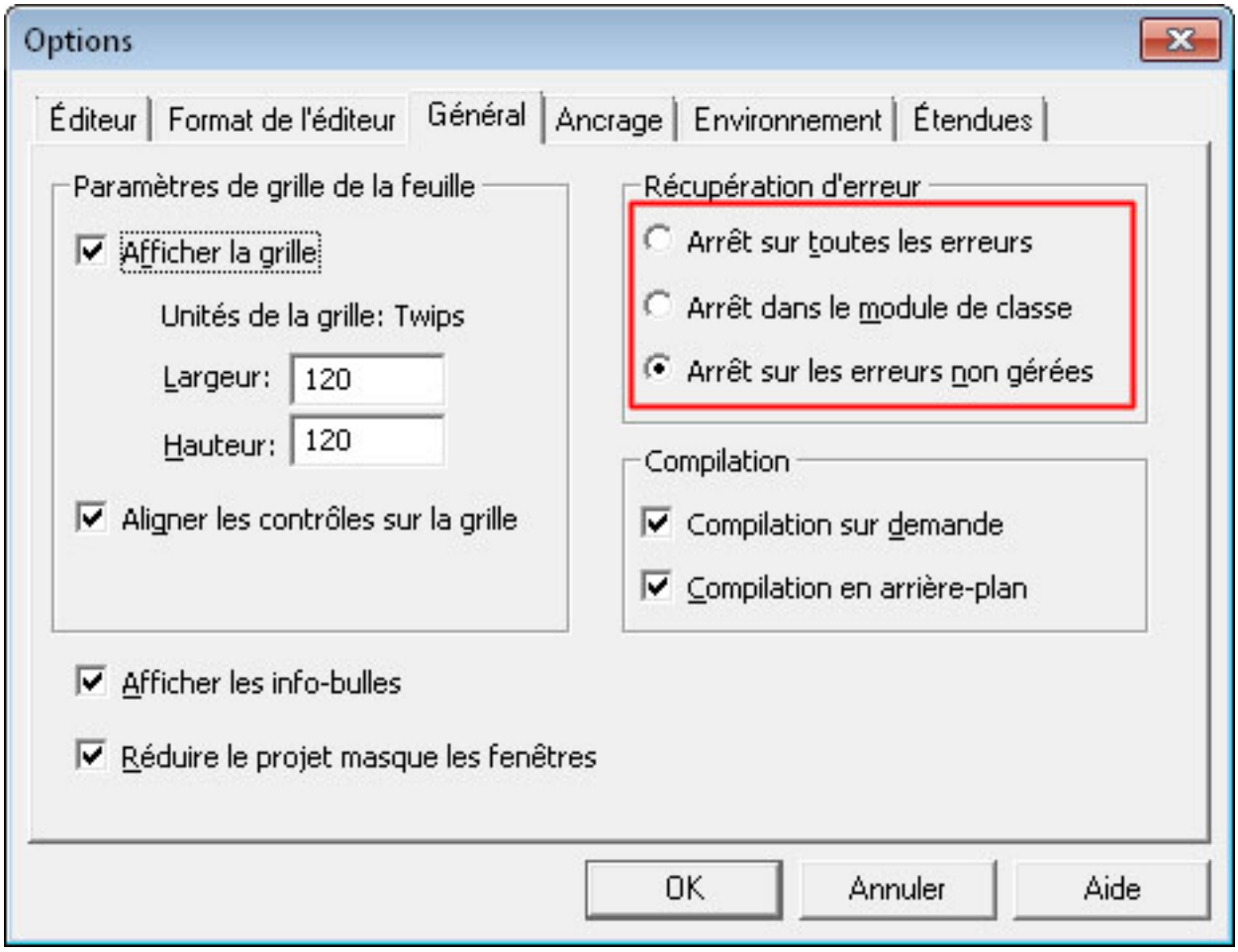
II-1-5 - Info-bulle automatique

L'outil Info-bulle fonctionne en mode arrêt ; il permet d'afficher la valeur en cours de l'expression se situant sous le curseur.

```
Dim lenStr As Long
lenBs = UBound(Bytes) - LBound(Bytes)
lenBs = 256 = LenB(StrConv(str, vbFromUnicode))
If lenBs > lenStr Then
```

II-2 - Le paramétrage des Options de gestion d'erreurs

Le paramétrage des options de l'IDE va conditionner le comportement des gestionnaire d'erreurs notamment l'activation ou la désactivation des gestionnaires



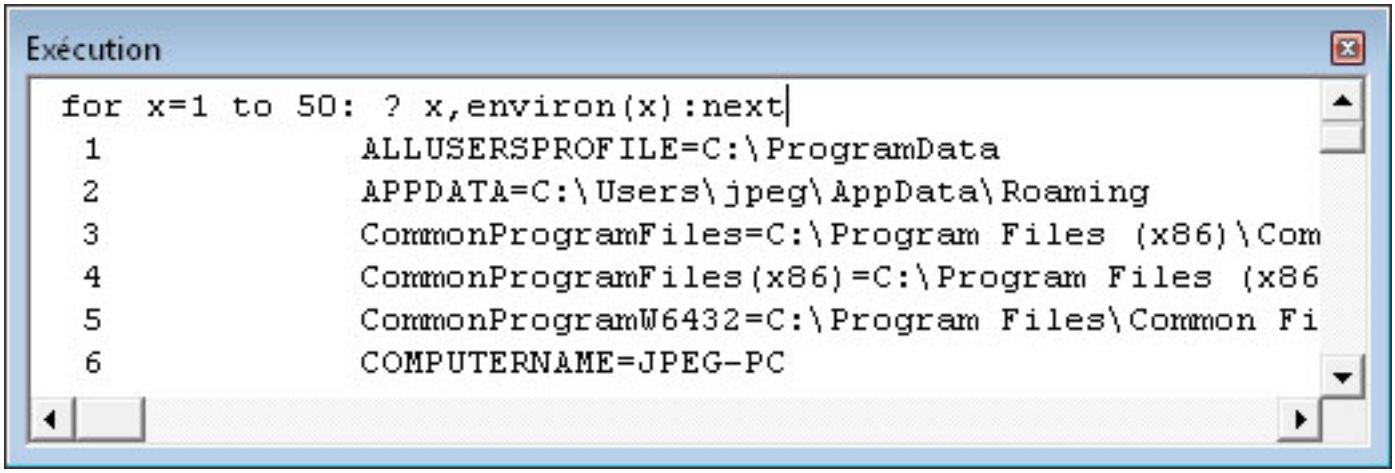
3 possibilités sont présentes sur le comportement face à une erreur :

- Arrêt sur toutes les erreurs : Désactive tous les gestionnaires d'erreur et marque un arrêt sur toutes les erreurs
- Arrêt sur les erreurs non gérées : Active les gestionnaires d'erreur et marque un arrêt uniquement sur les erreurs non gérées
- Arrêt dans les modules de classe : Se distingue de l'option précédente marquant un point d'arrêt dans le module de classe au lieu du module d'appel de la classe

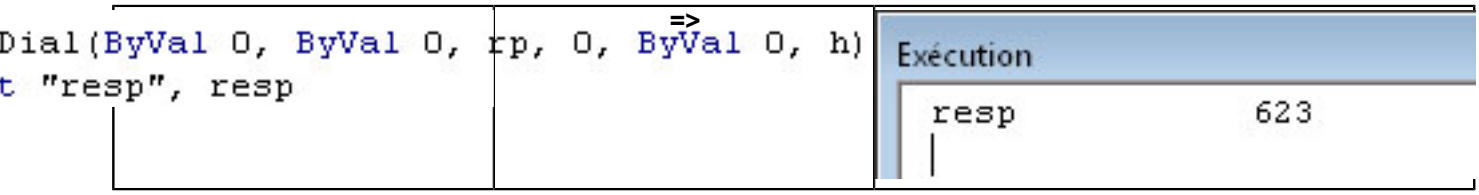
II-3 - Les outils dédiés au débogage

La fenêtre d''exécution (Ctrl G) *permet les actions suivantes :*

- Exécuter directement des instructions (pour une utilisation d'instructions multiples, utilisez une seule ligne et le séparateur d'instructions « : »)



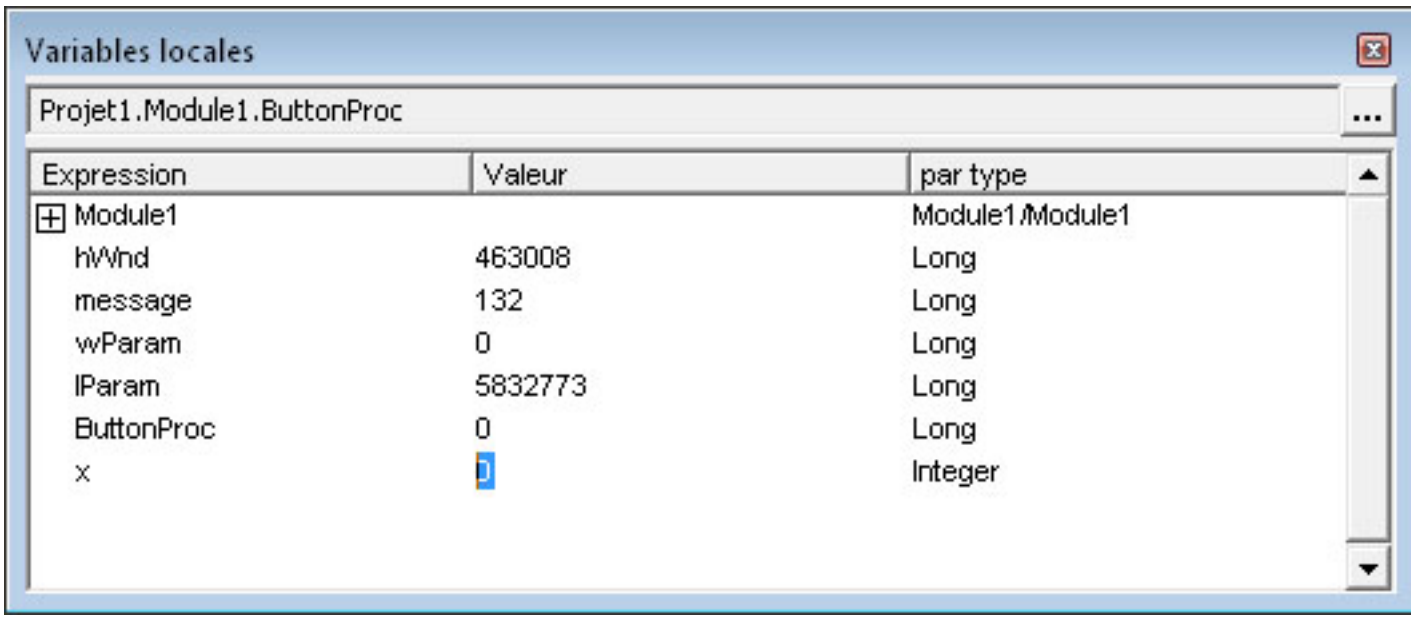
- Visualiser le résultat de partie de code en cours d'exécution en mode runtime (instruction Debug.Print)



- Lancer une procédure particulière quand le code actif s'exécute et se trouve en mode arrêt

L'utilisation de la fenêtre d'exécution est utilisable en mode arrêt lors d'une exécution en mode runtime ou directement en mode conception.

La fenêtre des variables locales (Alt IL)

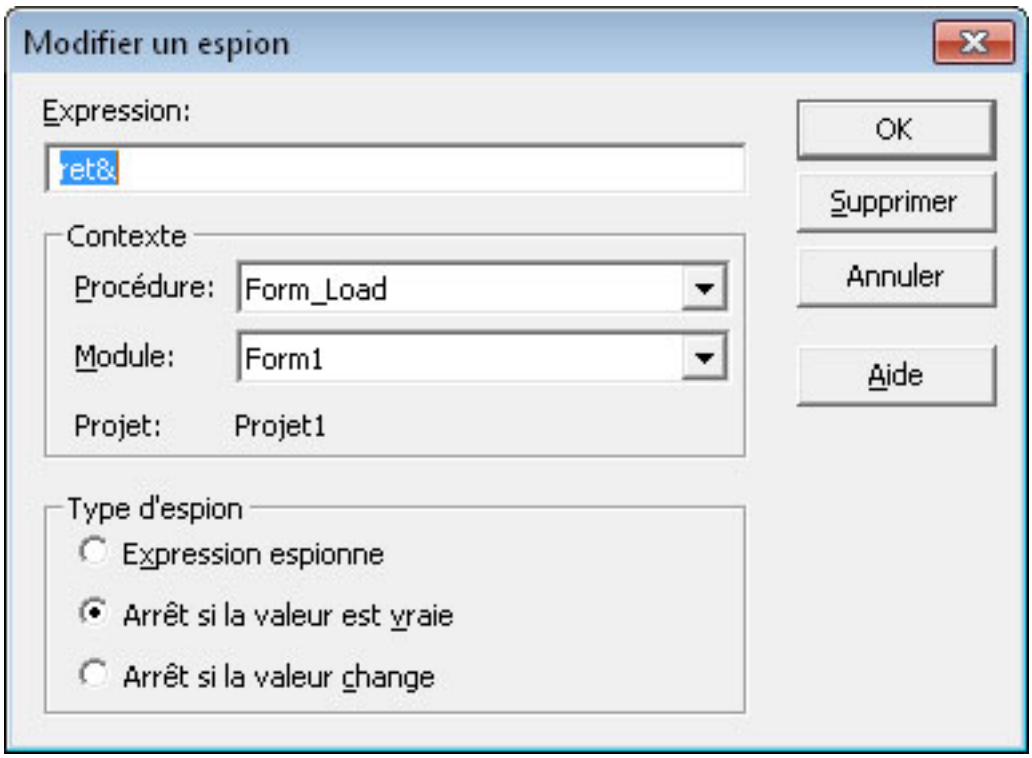


Elle permet de visualiser ou **modifier** les variables en cours

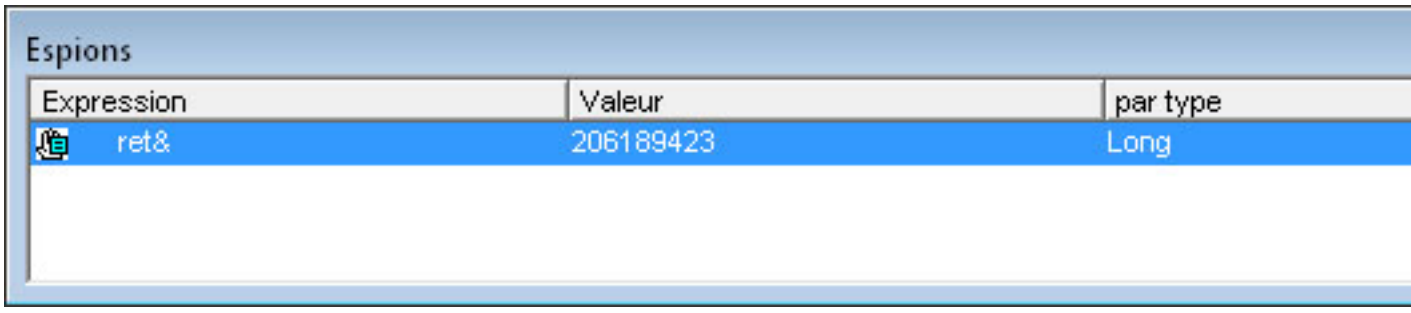
Pour modifier la valeur d'une variable dans la fenêtre Variables locales : double-cliquer sur la valeur concernée afin de l'éditer

La fenêtre Espions

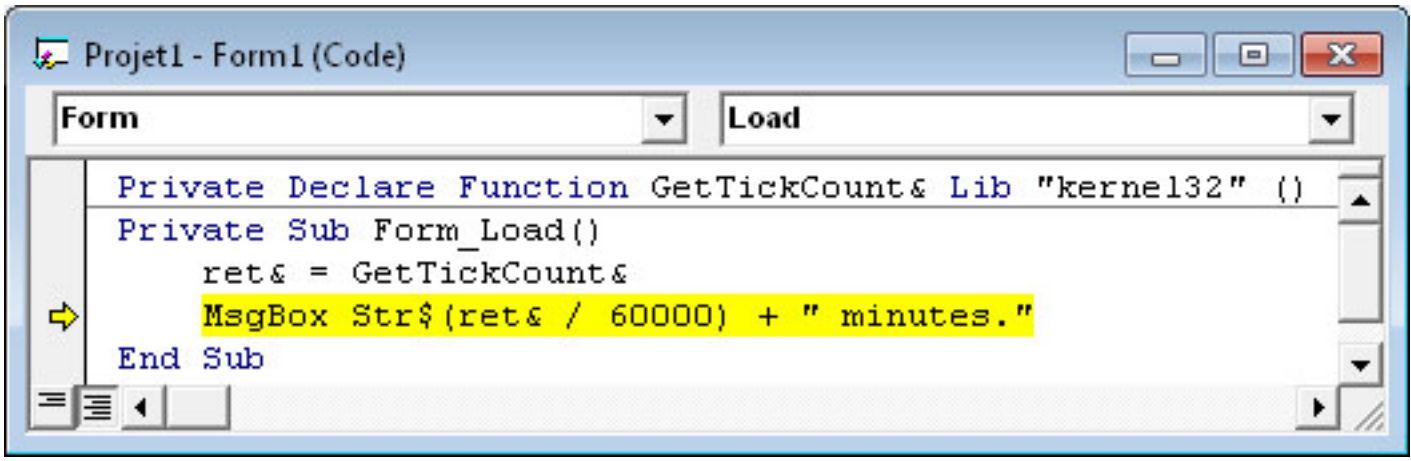
Permet de suivre la valeur de retour de propriétés, de fonctions etc. ou de déterminer une expression dont la valeur stoppera l'exécution du code en mode runtime
Les menus en rapport : Ajouter un espion - Modifier un espion



génèrera la fenêtre suivante en cours d'exécution ...



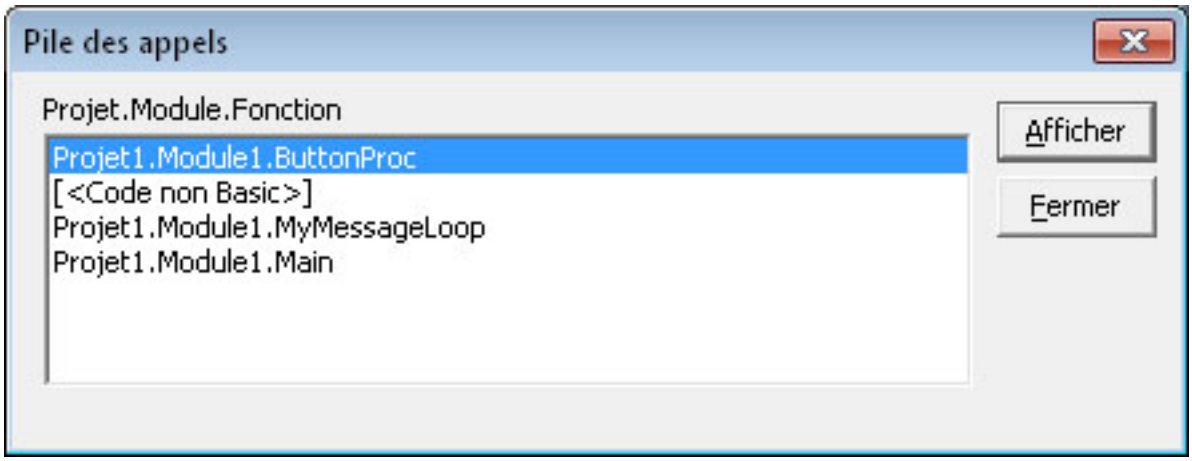
et un arrêt sur la ligne suivante.



Il est possible d'ajouter un espion en effectuant un Glisser Déposer de la variable préalablement sélectionnée dans la fenêtre Espions.

Pile des Appels (Ctrl L)

Cette fenêtre affiche la liste des procédures en cours d'exécution pendant le mode arrêt et éventuellement affiche le code de la procédure sélectionnée.
Utile pour les procédures récursives ou pour retrouver un gestionnaire d'erreur parent.



II-4 - Les utilitaires dédiés

Les outils fournis par MZ-Tools

L'addin **MZ-Tools** met à disposition 2 outils intéressants pour la gestion d'erreurs:

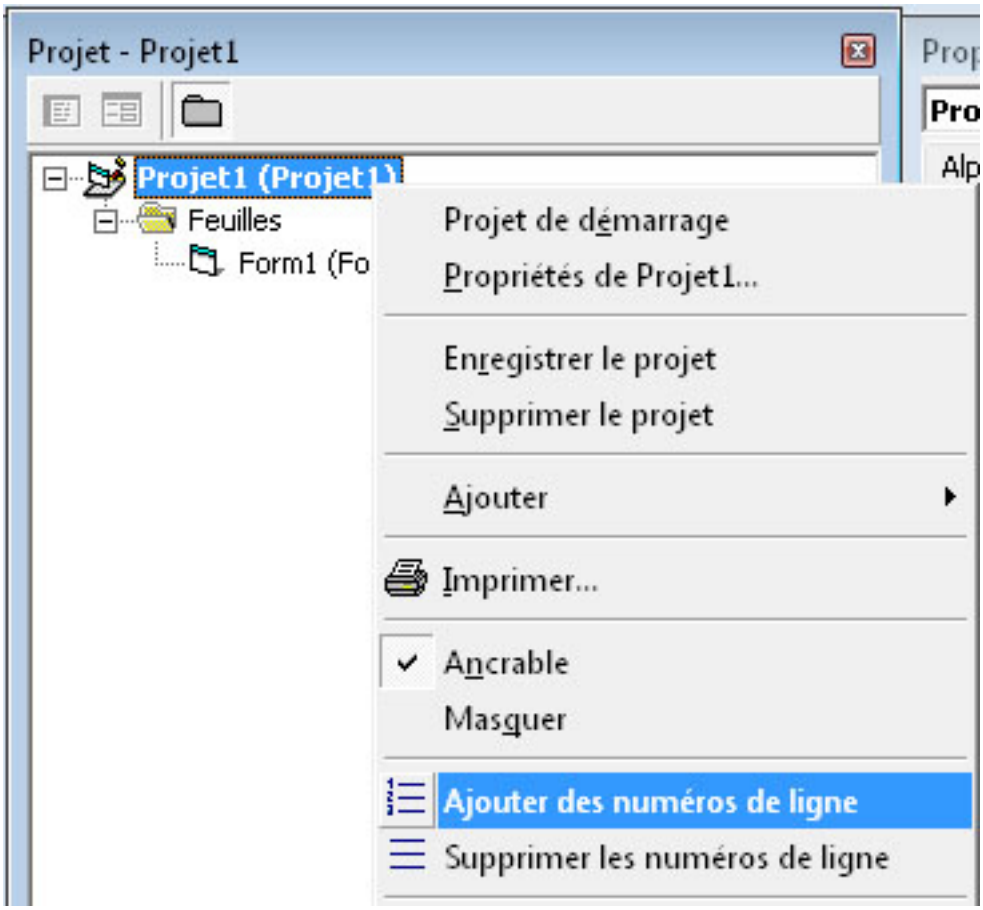
Le numéroteur de lignes

Cet outil permet d'automatiser l'indexation des lignes de code

- soit au niveau procédure,
- soit au niveau d'un module,
- soit au niveau de tout le projet.

Il est possible de déterminer dans le paramétrage de l'addin, l'incrémentation des numéros de lignes.

La numérotation au niveau projet ou module est possible en utilisant le menu contextuel (Clic droit) après sélection du contexte dans l'Explorateur de projet.



Ajouter une procédure d'erreur

Il peut être fastidieux d'ajouter une gestion d'erreur à de nombreuses procédures; cet outil répond à ce problème en suivant un modèle prédéfini en fonction de différentes variables

Les champs disponibles pour définir le modèle sont :

Variables	Descr	
{PROJECT_FILENAME}		VBP du fichier

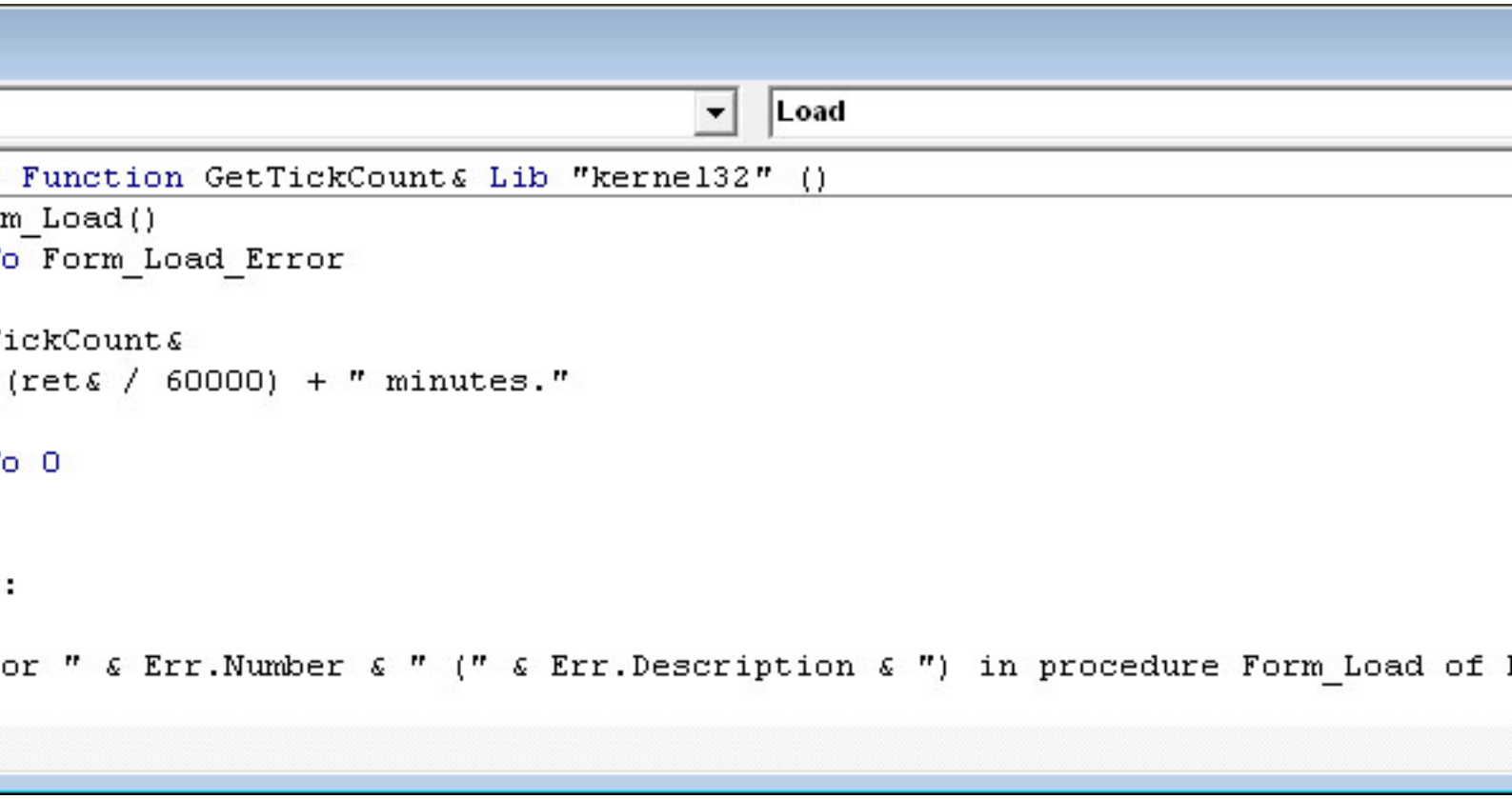
	de projet
{PROJECT_NAME}	VBProject.Name du projet
{MODULE_TYPE}	VBComponent.Type de module
{MODULE_FILENAME}	VBComponent.FileName de fichier du module
{MODULE_NAME}	VBComponent.Name du module
{FUNCTION_RETURN_TYPE_NAME}	Nom du type de retour de la procédure (ex : Long, Boolean)
{FUNCTION_RETURN_TYPE_PREFIX}	Préfixe (1ère lettre) du type de retour de la procédure
{PROCEDURE_TYPE}	Type de la procédure
{PROCEDURE_NAME}	Name de la procédure
{PROCEDURE_BODY}	Corps de procédure (représente le code de la procédure après le premier bloc de déclaration)

Le modèle est à définir dans l'interface des Options MZ-Tools (Onglet «Gestionnaire d'erreur»)

Soit, par exemple, le modèle suivant :

```
On Error GoTo {PROCEDURE_NAME}_Error

{PROCEDURE_BODY}

On Error GoTo 0
Exit {PROCEDURE_TYPE}

{PROCEDURE_NAME}_Error:

MsgBox "Error " & Err.Number & " (" & Err.Description & ") in procedure {PROCEDURE_NAME} of
 {MODULE_TYPE} {MODULE_NAME}"
```

qui génèrera le résultat suivant :



NB: Les lignes ajoutées sont marquées d'un signet dans la marge.

L'outil «Signet» permet de signaler une ligne en ajoutant un tag dans la marge.

III - Gestion des erreurs par le code

III.1 - Les instructions de gestion

On error ... *Instruction*

L'instruction On Error supporte plusieurs syntaxes :

- On Error Goto *Etiquette* : permet de rerouter l'exécution vers un bloc de gestion de l'exception levée, placé en général immédiatement après une instruction Exit Sub/Exit Function/Exit Property
- On Error Goto 0 : Supprime toute gestion d'erreur préalablement installée.
- On Error Resume Next : permet de générer un gestionnaire d'erreurs «passif» : le contrôle de l'exécution est transmis directement à la ligne suivante sans qu'aucun traitement ne soit effectué


Utiliser impérativement une variable intermédiaire si une instruction de test doit être effectuée sous le contrôle d'une instruction On Error Resume Next.

```
' exemple :
On Error Resume Next
ret= Val(myString)
If ret<>0 Then
    '.../...
endif

' et non pas :
If Val(myString) Then
```


La portée d'une instruction On Error ... est limitée à la procédure courante.

Resume *Instruction*
Provoque une réinitialisation de l'object Err (equiv Err.Clear) et la poursuite de l'exécution après la gestion de l'exception :

- Resume : l'exécution reprend à la ligne où l'interruption a eut lieu
- Resume Next : l'exécution reprend à la ligne suivante en ignorant la ligne ayant levé l'exception
- Resume *Line* : l'exécution reprend à une ligne spécifique en fonction d'un numéro de ligne ou d'une étiquette

Error *Instruction*
Assure la compatibilité avec les versions précédentes; préférer Err.Raise

IsError *Expression*
Renvoie Vrai si *expression* est une valeur d'erreur

CVErr *Function*
Permet de créer des erreurs définies par l'utilisateur dans des procédures créées par l'utilisateur. (voir également la constante vbObjectError)

Err *Object*


La portée de l'object Err est limitée à la procédure à laquelle est associée un gestionnaire d'erreur «On Error ...»

Propriétés	Type	Lect/Ecr.	Description
Description	String	RW	Description d'un object Err
Number	Long	RW	ID
Source	String	RW	Nom de l'object(module) ou de l'application à l'origine de l'erreur.
HelpContext	String	RW	Id du contexte d'aide associé.
HelpFile	String	RW	Chemin vers un fichier d'aide
LastDllError	Long	R	Dernier code d'erreur produit par un appel à DLL

Méthodes	Description
Clear	Réinitialise l'object Err
Raise	Génère une erreur

ERL *Function*
Retourne le n° de ligne (Long) sur lequel s'est produit la dernière erreur levée (portée procédure)

Debug *Object*

Méthodes	Description
Print	Retourne la valeur d'une expression dans la fenêtre d'exécution quand le code s'exécute en mode runtime
Assert	Arrête l'exécution en mode runtime sur la ligne concernée quand la valeur de l'expression est fausse (Ignorée en mode compilée)

III.2 - Créer un gestionnaire d'erreurs

La création d'un gestionnaire d'erreurs «actif» consiste à élaborer une gestion au cas par cas afin de répondre à toutes les erreurs susceptibles de se produire.
Illustration par l'exemple ....

```
Option Explicit

' Nécessite que l'option de l'Ide «Arrêt sur les erreurs non gérées» soit activée
Private Sub Form_Load()
    Dim strErrnum As String
    Dim intErr    As Integer
    Dim intReturn As Integer

    On Error GoTo Exemple1_Error

10    intErr = 3 / intReturn    ' (div 0)
20    intErr = 10 ^ 31          ' dépassement de capacité

30    strErrnum = 71   ' disque non prêt
40    intErr = Val(strErrnum)
50    Err.Raise Number:=intErr

Exit Sub

Exemple1_Error:
    Select Case Err.Number
        Case 6
            Resume Next ' Erreur non corrigée : Passer à la ligne suivante
```

```
        Case 11
            intReturn = 1 ' Correction de l'erreur ...
            MsgBox "=> La valeur 0 va être remplacée par la valeur 1"
            Resume ' Reprendre l'exécution

        Case 68

  If MsgBox(Err.Description, vbRetryCancel) = vbRetry Then Resume ' Exécuter à nouveau ou annuler ?

        Case Else ' Signaler toute autre erreur et sortie du code de la procédure incriminée
            MsgBox Err.Description ,,  "Form_Load (Line " & Erl & ")"

    End Select
End Sub
```

III.3 - Le choix de l'externalisation

Le choix du mode de sortie de l'erreur dépend du contexte :

- la boîte de dialogue (Msgbox) : pour interagir avec l'utilisateur final ou permettre d'entrer en mode pas à pas pendant la mise au point
- la fenêtre d'exécution (Debug.Print) : permet de signaler une erreur sans interrompre le déroulement du programme
- le fichier journal : il a le mérite de ne pas être éphémère, de transmettre des informations non déformées et pouvoir être traité automatiquement - a réserver en production ou dans le cadre de mise au point suite à erreur fatale (dans ce cadre, l'écriture doit être optimisée)
- le presse-papier : ce peut être une solution intermédiaire entre la fenêtre de Debug et le fichier log pour la phase de mise au point
- la console : peut éventuellement présenter un intérêt lorsque qu'une quantité importante d'information doit être présentée à l'utilisateur final
- l'imprimante : accessoirement, peut être utilisée pour analyser la génération de résultats erronés.

III.4 - Gestionnaire d'erreurs centralisé

Etablir un gestionnaire centralisé consiste à réaliser une ou plusieurs procédures spécifiques de traitement des exceptions qu'il suffira ensuite d'appeler lors de l'interception de l'erreur.

Comme pour le choix du mode de sortie, c'est le contexte qui déterminera le choix entre gestionnaire en ligne ou et gestionnaire centralisé :

- un gestionnaire centralisé présente l'avantage de ne pas multiplier les actions similaires, en conséquence, il présente l'avantage que sa modification soit simplifiée puisque répercutée sur l'ensemble de l'application ; c'est même un investissement si le gestionnaire est réutilisable dans des projets différents.
- un gestionnaire en ligne est au contraire destiné au cas particuliers.

L'un comme l'autre peuvent éventuellement dépendre d'un fichier de ressources dédié qui sera aisément traductible (ex: vb6xx.dll pour Visual Basic.)

IV - Spécificités en mode compilé

IV.1 - Les options et paramètres de compilation

> Propriétés du projet (Alt PP) > Onglet Compilation > Options avancées

conditionne le comportement de l'exécutable compilé face à une exception.
Si elles permettent d'optimiser la vitesse d'exécution en supprimant des contrôles internes, leurs utilisations risquent également de créer de graves dysfonctionnement pour peu que la gestion d'erreurs ne soit pas des plus strictes.

Néophytes, s'abstenir !!

IV.2 - Debogage symbolique

> Propriétés du projet (Alt PP) > Onglet Compilation

Générer les informations de débogage symbolique :
Cette option permet de générer automatiquement à la compilation un fichier pdb utilisable notamment par VC++ afin de déboguer l'exécutable préalablement compilé en mode natif
Ces informations sont générée chaque fois qu'un fichier OBJ est créé par le compilateur ; elles contiennent les informations de type, de prototype des fonctions destinées au débogueur.

IV.3 - Remarques

Le déploiement et l'installation d'un exécutable nécessite de respecter quelques règles incontournables pour éviter certaines erreurs liées à l'installation :

- Lorsque qu'un exécutable inclut des dépendances ActiveX, il importe de conserver les fichiers dep associés ainsi que les dépendances avec les sources car ils décrivent les éventuelles dépendances imbriquées qu'il sera ultérieurement difficile d'identifier sans eux.
- Il est essentiel de réaliser (et de conserver avec les sources) un Setup d'installation car il n'est pas acquis que la version du système d'exploitation sur lequel l'installation s'effectuera distribue les dépendances du système sur lequel a été compilé l'exécutable (par exemple, Vista/Seven ne distribue pas une partie des dll/ocx distribuée sous Win2000/XP)
- Rappelez à l'utilisateur qu'il est essentiel avant l'installation de fermer les autres programmes et d'effectuer l'installation en mode administrateur.
- Il est nécessaire de tester l'exécutable sous les différents OS disponibles afin de connaître les éventuelles contraintes d'installation ou incompatibilités.(par exemple, Installation en mode compatible sous Seven 32 ou 64 bits)

L'Observateur d'évènements de Windows et les journaux liés peuvent être une source d'information complémentaires sur les erreurs.

V - Le débogage

V.1 - Mise au point d'un programme

V.1-1 - L'exécution en mode runtime :

Tout contrôle d'un exécutable commence par une exécution en mode runtime ...
Personne n'imaginerait compiler directement un exécutable et le distribuer sans cette étape intermédiaire.

L'IDE VB6 permet d'exécuter du code en mode «runtime» ***tout en permettant d'intervenir sur celui-ci lors de son exécution*** *dès qu'il entre en mode arrêt.*

Cette propriété essentielle permet :

- La modification du code existant,
- L'ajout de lignes d'instructions,
- L'ajout ou la modification de déclarations
- La modification de valeur de variables
- Le saut d'instructions, de procédures etc.

La réunion de ces possibilités laisse imaginer la puissance offerte pour la mise au point.

*Il est possible dans le même temps, **d'exécuter du code en parallèle depuis la fenêtre d'Exécution***
=> l'exécution des instructions, procédures ou fonctions se répercutera sur le code global en mode arrêt

L'usage de cette technique a toutefois des limites :

- Il faut l'éviter dans les procédures de rappels au risque d'un crash de l'IDE.
- Il n'est pas question d'effectuer des modifications importantes dans ce contexte car la stabilité de l'IDE est en relation avec le cumul des erreurs interceptées, leur gravité et les modifications opérées consécutivement.

V.1-2 - Tester la stabilité :

Tester la stabilité (la robustesse) d'un exécutable est également indispensable car si un code s'exécute sans problème jusqu'à son terme ou a contrario permet de mettre en évidence les erreurs, rien ne prouve qu'il s'exécutera correctement en condition extrême (charge CPU, saturation mémoire etc.)
Outre des outils professionnels dédiés à cet usage, il est possible d'exécuter un test de charge simple qui consiste à appeler consécutivement une même routine (ou un ensemble de routines) un grand nombre de fois - on observera en parallèle l'utilisation des ressources.

exemple :

```
Dim x   As Long, ret As Long
Const k As Long = 1000

On Error Goto cath_Err

For x=1 To k
    ret = myFunction()
Next

cath_Err:
    Debug.Print ret
    If x<=k Then Debug.Print "Error at " & k & " cycles"
```

Un second test simple consiste à renouveler plusieurs fois la routine précédente et comparer le résultat final retourné à chaque test.

V.1-3 - Erreur inattendue:

C'est l'erreur qui se produit lors d'une nième exécution, voir même aléatoirement sans qu'on puisse de premier abord en identifier l'origine.

On pourra tenter d'identifier si l'environnement pose problème - il faudra dans ce cadre :

- travailler avec un environnement restreint (fermeture de tous les programmes, désactivation des éventuels compléments,)
- voir si nécessaire utiliser une session de l'OS en mode sans échec,
- ou accessoirement tester si l'erreur se reproduit sous différents systèmes d'exploitation.

V.2 - Exécuter le code

Il existe 2 modes d'exécution :

- le mode Pas à pas (F8) qui permet d'exécuter le programme ligne à ligne
- le mode normal (F5) qui exécute le programme d'une traite jusqu'à ce qu'une erreur ou un point d'arrêt soit éventuellement rencontré.

Les 2 modes peuvent être utilisés consécutivement au sein d'une même session.

Une fois en mode arrêt, il est possible d'exécuter le code par tranche en utilisant :

- Mode Pas à pas principal **(Maj F8)**
- Mode Pas à pas sortant **(Ctrl Maj F8)**
- Exécuter jusqu'au curseur **(Ctrl F8)**

Arrêter l'exécution sur une ligne particulière :

- Le point d'arrêt (clic dans la marge de la ligne concernée) : il permet de stopper l'exécution juste avant l'exécution d'une ligne précise
- L'instruction Stop est identique au point d'arrêt
- En fonction d'un test dans le code (cf l'instruction Debug.Assert)
- Arrêts conditionnels en fonction d'un Espion (cf Outils)

L'instruction Stop ne doit pas apparaître dans un exécutable compilé.

=> Durant la phase de mise au point, il peut être intéressant d'utiliser les paramètres de compilation conditionnels (Alt PP > onglet Créer)
afin de gérer automatiquement la non compilation des instructions d'aide au débogage (Stop, Msgbox, Debug.Print etc)

exemple: Arguments de compilation conditionnelle > *NOCOMPILATION=1*

```
# If NOCOMPILATION Then
    Stop
#End If
```

V.3 - Cas particuliers des exécutions d'instances

Si l'exécution en mode runtime d'un exécutable standart (Exe) ne pose pas de problème particuliers, il est moins évident d'exécuter le code des Dll ActiveX, controls et Addins qui ne peut être exécuté directement dans l'Ide puisque une instance doit être crée.
Leur débogage nécessite donc quelques artifices ...

Utilisez le paramétrage : (Outils/Options> Onglet Général) : Arrêt sur les erreurs non gérées ou « Arrêt dans les modules de classe

V.3-1 - Control ActiveX

- Créer un nouveau projet Exe Standart (Ctrl N) ; celui-ci servira de projet support pour le test
- Créer un groupe de projet en ajoutant le projet du control à tester (Alt FA) => les contrôles sont disponibles dans la Boite à Outils du projet de test.

Après avoir placé si nécessaire, un point d'arrêt à l'endroit où commencer l'exécution en mode Pas à Pas (ligne par ligne)

- déposer le control à tester sur la feuille du projet de test => la partie «Concepteur» du code s'exécute ...
- Exécutez le projet de test en mode Runtime (F5) => la partie du code «Utilisateur» s'exécute ...

L'ordre d'ouverture des projets dans le groupe détermine le projet qui s'exécutera en mode runtime.

V.3-2 - Dll ActiveX

Pratiquer comme pour un Control en utilisant un groupe de projet, mais en ajoutant une Référence (Alt PR) au projet dans le projet de test.
=> Une entrée *«Nom_du_projet» (Chemin d'accès : Chemin_du_projet.vbp)* a été ajouté à la liste des références disponibles.

Il existe toutefois une alternative à l'utilisation d'un groupe de projets :

- Ajouter un module standart au projet à tester
- Ajouter une procédure de test dans laquelle est initialisée la classe à tester
- Exécuter la procédure en l'appelant depuis la fenêtre d'Exécution

V.3-3 - AddIn

L'exécution de l'addin est lancée en mode normal depuis le projet de l'addin (F5) : l'instance en cours se met en attente de connection
=> la connection à l'instance en cours s'effectuera en ouvrant une instance de l'application liée.


Afin de ne pas avoir à gérer l'installation de l'addin à l'ouverture de l'application cliente :
*=> Définir le mode de démarrage de l'addin (Comportement Initial) à «**Startup**»*

VI - Conclusion

La gestion des erreurs est un vaste sujet, incontournable pour toute application, et les outils de mise au point sous VB6 sont nombreux.
En faire le tour nécessiterait la rédaction d'un ouvrage- les lecteurs désireux d'approfondir le sujet peuvent toutefois trouver plus d'information en consultant la MSDN.

Il existe également un projet de démo «Errors.vbp» illustrant la gestion d'erreurs dans le répertoire de la MSDN (*path_to_MSDN*\MSDN98\98VSa\1036\SAMPLES\VB98\Errors\ERRORS.VBP)

Tous mes remerciements à ceux qui ont bien voulu prendre quelques minutes de leur temps pour me relire et finaliser cet article.
notamment ThierryAIM et Pierre Fauconnier.