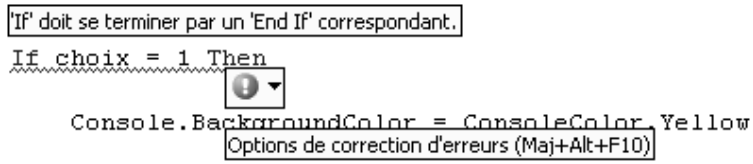
A. Les différents types d'erreurs

Pour un développeur, les erreurs sont une des principales sources de stress. En fait, nous pouvons classer ces erreurs en trois catégories. Regardons chacune d'entre elles et les solutions disponibles pour les traiter.

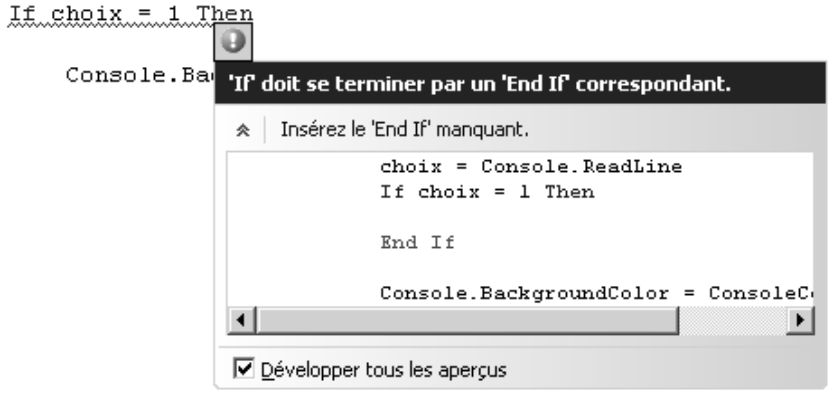
1. Les erreurs de syntaxe

Ce type d'erreur se produit au moment de la compilation, lorsqu'un mot clé du langage est mal orthographié. Très fréquentes avec les premiers outils de développement où l'éditeur de code et le compilateur étaient deux entités séparées, elles deviennent de plus en plus rares avec les environnements tels que Visual Studio. La plupart de ces environnements proposent une analyse syntaxique au fur et à mesure de la saisie du code. De ce point de vue, Visual Studio propose de nombreuses fonctionnalités nous permettant d'éliminer ces erreurs.

Il surveille, par exemple, que chaque instruction `If` est bien terminée par un `End If`.

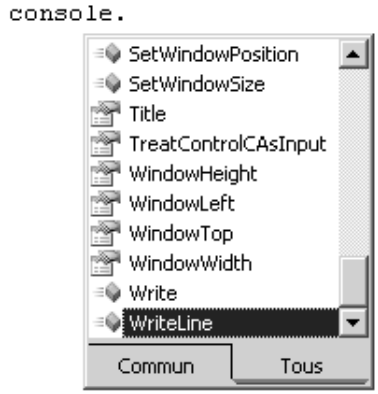


Si une erreur de syntaxe est détectée, Visual Basic propose les solutions possibles pour corriger cette erreur. Elles sont affichées en cliquant sur l'icône associée à l'erreur.

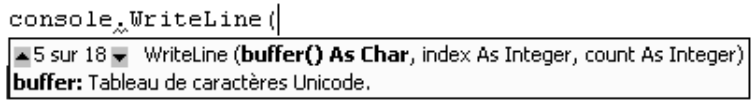


D'autre part les "fautes d'orthographe" dans les noms de propriétés ou de méthodes sont facilement éliminées grâce aux fonctionnalités `IntelliSense`. `IntelliSense` prend en charge les fonctionnalités ci-après.

- L'affichage automatique de la liste des membres disponibles :

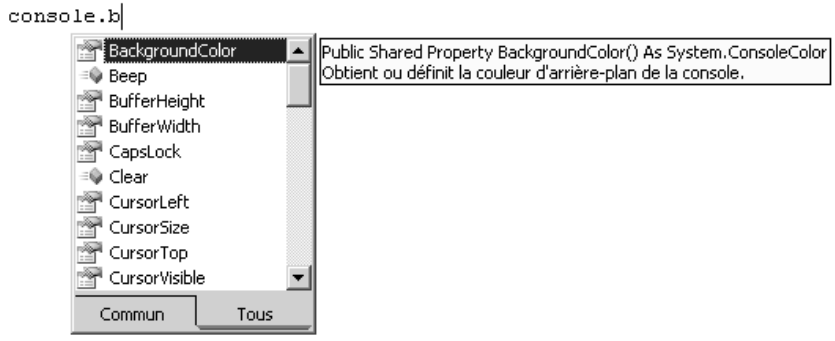


- L'affichage de la liste des paramètres à fournir pour l'appel d'une procédure ou fonction :



Si plusieurs surcharges sont disponibles, `IntelliSense` affiche leur nombre et vous permet de les parcourir en utilisant les flèches haut et bas du clavier.

- L'affichage d'informations ponctuelles sur membres d'une classe :

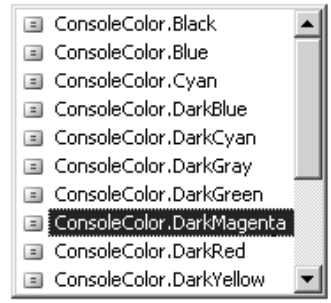


- Le complément automatique de mots : commencez à saisir un début de mot puis utilisez la combinaison de touches [Ctrl][Espace] pour afficher tout ce que vous pouvez utiliser comme mot, à cet emplacement, commençant par les caractères déjà saisis. S'il n'y a qu'une seule possibilité, le mot est ajouté automatiquement, sinon sélectionnez-le dans la liste et validez avec la touche [Tab].

© Editions ENI - All rights reserved

- L'affichage de la liste des valeurs possibles pour une propriété de type énumération.

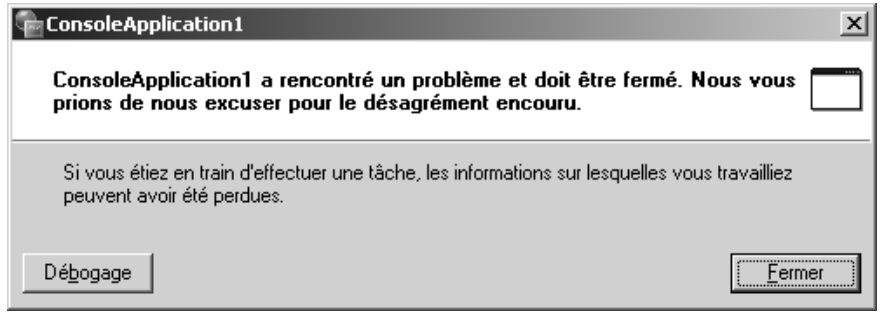
```
If choix = 1 Then

    Console.BackgroundColor=
End If
```



Avec toutes ces fonctionnalités, il est pratiquement impossible de générer des fautes de syntaxe dans le code.

2. Les erreurs d'exécution

Ces erreurs apparaissent après la compilation, lorsque vous lancez l'exécution de votre application. La syntaxe du code est correcte mais l'environnement de votre application ne permet pas l'exécution d'une instruction utilisée dans votre application. C'est, par exemple, le cas si vous essayez d'ouvrir un fichier qui n'existe pas sur le disque de votre machine. Vous obtiendrez sûrement une boîte de dialogue de ce type.



Ce type de boîte de dialogue n'est pas très sympathique pour l'utilisateur !

Heureusement, Visual Basic permet la récupération de ce type d'erreur et évite ainsi l'affichage de cette inquiétante boîte de dialogue. Deux techniques sont disponibles pour la gestion de ce type d'erreurs :

- la gestion en ligne ;
- les exceptions.

Nous détaillerons cela un peu plus loin dans ce chapitre.

Les erreurs de logique

Les pires ennemis des développeurs. Tout se compile sans problème, tout s'exécute sans problème et pourtant "ça ne marche pas" !!!

Il convient, dans ce cas, de revoir la logique de fonctionnement de l'application. Les outils de débogage nous permettent de suivre le déroulement de l'application, de placer des points d'arrêt, de visualiser le contenu des variables, etc.

B. Traitement des erreurs

Deux techniques sont disponibles pour le traitement des erreurs dans Visual Basic :

- la gestion en ligne ;
- le traitement par exception.

1. La gestion en ligne

L'instruction `On Error` est l'élément de base de la gestion des erreurs en ligne. Lorsque cette instruction est exécutée, dans une procédure ou fonction, elle active la gestion des erreurs pour cette procédure ou fonction. Il convient cependant d'indiquer comment notre gestionnaire doit réagir lorsqu'une instruction déclenche une erreur. Deux solutions :

```
On error resume next
```

L'exécution du code va se poursuivre par la ligne suivant celle qui a provoqué l'erreur.

```
On error goto etiquette
```

L'exécution du code va se poursuivre par la ligne repérée par `etiquette`.

La syntaxe est donc la suivante :

```
Private Sub Nom_de_procedure()
   On Error Goto gestionErreurs
      ...
      "Instructions dangereuses"
  ...
Exit Sub
gestionErreurs:
  ...
  code de gestion des erreurs
  ...
End Sub
```

L'étiquette vers laquelle le gestionnaire d'erreurs va rediriger l'exécution doit se trouver dans la même procédure que l'instruction `on error goto`. L'instruction `Exit sub` est obligatoire pour que le code de gestion d'erreurs ne soit pas exécuté à la suite des instructions normales mais seulement en cas d'erreur.

© Editions ENI - All rights reserved

Le code du gestionnaire d'erreur doit déterminer la conduite à tenir en cas d'erreur.
Trois solutions :

`Resume`

On essaie à nouveau d'exécuter la ligne qui a provoqué l'erreur.

`resume next`

On continue l'exécution par la ligne suivant celle qui a provoqué l'erreur.

`exit sub` ou `exit function`

On abandonne l'exécution de cette procédure ou fonction.

Typiquement, le gestionnaire d'erreurs affichera une boîte de dialogue demandant à l'utilisateur ce qu'il souhaite.
En fonction de sa réponse, on utilisera l'une ou l'autre des solutions.

```
Private Sub OuvertureFichier()
        On Error GoTo gestionErreurs
        My.Computer.FileSystem.OpenTextFileReader("c:\essai")
        Exit Sub
gestionErreurs:
        Dim reponse As Integer
        reponse = MsgBox("impossible de lire le fichier",
MsgBoxStyle.AbortRetryIgnore)
        Select Case reponse
            Case MsgBoxResult.Retry
                Resume
            Case MsgBoxResult.Ignore
                Resume Next
            Case MsgBoxResult.Abort
                Exit Sub
        End Select
    End Sub
```

Il nous reste encore un problème à résoudre : notre gestionnaire d'erreurs réagira quelle que soit l'erreur. Pour pouvoir déterminer quelle erreur vient de se produire dans l'application, nous avons à notre disposition l'objet `err` qui nous fournit des informations sur la dernière erreur apparue. Cet objet contient, entre autres, deux propriétés, `number` et `description`, nous permettant d'obtenir le code de l'erreur et sa description. Nous pouvons donc modifier notre code pour réagir de façon différente en fonction de l'erreur.

```
Private Sub OuvertureFichierBis()
        On Error GoTo gestionErreurs
        My.Computer.FileSystem.OpenTextFileReader("a:\essai")
        Exit Sub
gestionErreurs:
        Dim reponse As Integer
        Console.WriteLine(Err.Number)
```

```
        Stop
        If Err.Number = 53 Then
            reponse = MsgBox("impossible de trouver le fichier", MsgBoxStyle.
AbortRetryIgnore)
            Select Case reponse
                Case MsgBoxResult.Retry
                    Resume
                Case MsgBoxResult.Ignore
                    Resume Next
                Case MsgBoxResult.Abort
                    Exit Sub
            End Select
        End If
        If Err.Number = 57 Then
            reponse = MsgBox("insérer une disquette dans le lecteur",
MsgBoxStyle.OKCancel)
            Select Case reponse
                Case MsgBoxResult.OK
                    Resume
                Case MsgBoxResult.Cancel
                    Exit Sub
            End Select
        End If
    End Sub
```

Un gestionnaire d'erreurs peut être désactivé en utilisant l'instruction `on error goto 0`. Si une erreur se produit après cette instruction, elle n'est pas récupérée et l'application s'arrête.

En fait, l'application ne s'arrête pas immédiatement mais Visual Basic recherche dans les fonctions appelantes si un gestionnaire d'erreur est actif et s'il en trouve un, il lui confie la gestion de l'erreur.

```
Private Sub procedure1()
        On Error GoTo gestionErreurs
        procedure2()
        Exit Sub
gestionErreurs:
        MsgBox("erreur d'execution")
    End Sub
    Private Sub procedure2()
        Dim x, y As Integer
        x = 0
        y = (1 / x)
    End Sub
```

© Editions ENI - All rights reserved