Utilisation des classes VBScript dans les pages ASP

Préambule

Ce tutoriel se veut être une introduction à l'utilisation des classes VBScript dans les pages ASP. Il résulte de la méconnaissance des développeurs ASP dans ce domaine et du désir de faire connaître un concept que l'on retrouve dans tous les langages évolués.
Il fait la synthèse de divers articles parus à ce sujet, la plupart du temps en anglais.
Malgré un souci d'exhaustivité, il se peut que j'aie oublié d'aborder certains points avancés. Si tel est le cas, merci de m'en faire part par MP en cliquant sur le lien au-dessus.

Je tiens à remercier particulièrement Johann Heymes et Henry Cesbron Lavau qui, par leur relecture attentive de l'article, m'ont permis à partir de leurs propositions de formuler l'introduction théorique de façon claire et précise.

Sommaire

Un peu de théorie

Qu'est-ce qu'une classe, un objet, une instance ?

Une classe est une méthode d'encapsulation de données : propriétés (variables) et méthodes (procédures). C'est un concept clé de la POO (Programmation Orientée Objet). Les classes permettent de construire des types complexes que ne possèdent pas d'emblée les langages ; le VBScript n'échappe pas à cette règle, d'autant moins que c'est un langage faiblement typé.
Concrètement, comment définir une classe ? Imaginons par exemple un véhicule. Ce véhicule possède des propriétés, comme par exemple sa couleur, son poids, sa longueur, sa hauteur. Il possède également des méthodes : il peut avancer, reculer, stopper. Tous les véhicules ont ces caractéristiques : c'est le concept de classe. Une classe représente une entité, c'est à dire une représentation *informatique* de quelque chose qui existe dans le monde réel. Toutefois, mon véhicule n'est certainement pas le même que le vôtre : couleur

différente, poids différent, etc... Cela introduit le concept d'objet : ma voiture est un objet qui appartient à la classe véhicule, de même que la vôtre, même s'il existe des différences entre elles.
En programmation, le fait de déclarer un nouvel objet de classe s'appelle instancier. Un **objet** est donc une **instance** de **classe**.

Héritage

Un autre concept important de la POO est la notion d'héritage. Qu'entend-on par là ? Reprenons l'exemple de la classe véhicule. Ma voiture est un objet de cette classe. Mais ma bicyclette est aussi un objet de cette classe. Il existe cependant des différences assez importantes entre les 2. Pour éviter donc de créer un objet *trop général*, il m'est possible de définir une *sous-classe* qui va reprendre les carcatéristiques de ma classe véhicule mais qui va l'*étendre* pour en ajouter d'autres. On parle ainsi d'héritage. En effet ma sous-classe (véhicule_non_motorisé par exemple) conserve les méthodes et propriétés de la classe de base.
Concept intéressant s'il en est mais malheureusement inexistant en VBScript. il existe d'autres concepts qui découlent de celui d'héritage, mais comme VBScript ne supporte pas l'héritage, il est inutile de s'épancher plus longuement dessus.

L'utilisation des classes en VBScript aborde donc les concepts de la POO mais on ne peut malgré tout pas parler de Programmation Orientée Objet pour VBscript. Les classes ont été introduites avec la version 5.0 de VBScript.

Ma première classe

Voyons maintenant comment déclarer une classe et comment instancier une classe.
On utilise pour déclarer une classe les mots clé ***Class*** et ***End Class***.

```
<%
Class ClassTest

End Class
%>
```

La création d'une instance de classe se fait à l'aide du mot clé ***New*** et l'affectation à une variable, comme pour les objets COM, à l'aide du mot clé ***Set***.

```
<%
Class ClassTest

End Class

'      Création      d'une      instance      de      la      classe      ClassTest
Set instClassTest = New ClassTest

'        Destruction        de        l'instance        créée        précédemment
Set instClassTest = Nothing
%>
```

Cette classe telle qu'elle est définie pour l'instant ne fait absolument rien. Nous allons donc lui définir des propriétés.

La portée : *Private/Public*

Il y a deux cas à prendre en compte. Comme dit plus haut, une classe est une méthode d'encapsulation. Qui dit encapsulation dit donc que l'on va définir des propriétés (variables) qui seront accessibles depuis l'extérieur de la classe, et d'autres uniquement à l'intérieur de la classe.
Si je reprends l'exemple de la classe véhicule, peut m'importe de savoir la façon dont va être calculé le poids, ce qui m'importe, c'est de récupérer cette valeur, et uniquement cette valeur finale, pas les valeurs intermédiaires qui auront permis d'obtenir celle-ci. C'est la notion de portée. On va pouvoir déclarer des propriétés (variables) publiques, c'est à dire accessibles dans et à l'extérieur de la classe, et des propriétés privées, c'est à dire uniquement accessibles dans la classe. Comme pour l'instant je m'attache uniquement aux propriétés, c'est pour cette raison que j'ai omis de parler des méthodes, mais le même principe s'applique également à celles-ci.
La distinction va donc s'effectuer à l'aide des mots-clé ***Private*** et ***Public***.

```
<%
Class Personne
  Public prenom
  Private surnom

End                                                                    Class
%>
```

Les propriétés et méthodes d'un objet (mais vous connaissez déjà) s'écrivent à l'aide du point, sur le modèle

```
variable = instanceobjet.propriete
instanceobjet.propriete = variable
instanceobjet.methode()
variable = instanceobjet.methode()
```

On va donc pouvoir modifier et afficher la propriété nom

```
<%
Class Personne
  Public prenom
  Private surnom

End                                                                    Class

Set instPersonne                                                = New Personne
instPersonne.prenom                                 =                  "jérôme"
Response.Write "Bonjour              "              &          instPersonne.prenom
Set instPersonne                                                    = Nothing
%>
```

Si vous essayez de faire la même chose avec la propriété *surnom*, étant donné que celle-ci est privée, vous obtiendrez le message d'erreur suivant :

```
Erreur d'exécution Microsoft VBScript error '800a01b6'

Cet objet ne gère pas cette propriété ou cette méthode: 'surnom'
```

Quelques remarques :
- Par défaut, si vous ne le précisez pas à l'aide du mot-clé ***Public***, c'est à dire si vous utilisez le mot-clé ***Dim*** pour déclarer vos variables de classe, la portée est publique.
- Il n'est pas possible de combiner les mots-clé ***Private/Public*** avec le mot-clé ***Dim***.
- Il n'est pas possible d'utiliser le mot-clé ***Redim*** pour déclarer une variable de classe.
- Si vous voulez déclarer une variable de classe comme étant une variable tableau, il faut la déclarer à l'aide d'un des 3 mots-clé ***Dim/Public/Private*** puis la redimensionner à l'aide du mot-clé ***Redim*** dans une procédure.

Le constructeur : *class_initialize*

Un constructeur est une procédure qui va être appelée au moment où une instance va être créée. Cela permet notamment d'initialiser des variables dans la classe avec une valeur par défaut.
En VBScript, on dispose de la procédure ***Class_Initialize()***

```
<%
Class Personne
  Public prenom
  Private surnom

  Private Sub Class_Initialize()
    Response.Write "La classe est initialisée :<br>"
    prenom = "inconnu"
  End Sub

End Class

Set instPersonne = New Personne
Response.Write "Bonjour " & instPersonne.prenom
instPersonne.prenom = "jérôme"
Response.Write "Bonjour " & instPersonne.prenom
Set instPersonne = Nothing
%>
```

Ceci va donc donner la sortie suivante à l'écran :

```
La classe est initialisée :
Bonjour inconnu
Bonjour jérôme
```

Nota : La procédure ***Class_Initialize()*** ne possède pas d'arguments

Le destructeur : *class_terminate*

Un destructeur est une procédure qui va être appelée au moment où l'instance va être détruite.
En VBScript, on dispose de la procédure ***Class_Terminate()***

```
<%
Class Personne
  Public prenom
  Private surnom
```

```
  Private                                                    Sub Class_Initialize()
    Response.Write "La          classe          est          initialisée          :<br>"
    prenom                                  =                                  "inconnu"
  End                                                                             Sub

  Private                                                  Sub Class_Terminate()
    Response.Write "La              classe              est              détruite<br>"
  End                                                                             Sub

End                                                                             Class

Set instPersonne                                                  = New Personne
Response.Write "Bonjour              "              &              instPersonne.prenom
instPersonne.prenom                             =                             "jérôme"
Response.Write "Bonjour              "              &              instPersonne.prenom
Set instPersonne                                                          = Nothing
%>
```

Ceci va donc donner la sortie suivante à l'écran :

```
La                  classe                  est                  initialisée                  :
Bonjour                                                                                  inconnu
Bonjour                                                                                  jérôme
La classe est détruite
```

Nota : Comme la procédure ***Class_Initialize()***, la procédure ***Class_Terminate()*** ne possède pas d'arguments.
Si vous omettez de détruire explicitement l'instance de classe créée (ce qui est toujours déconseillé), celle-ci le sera automatiquement à la fin de l'exécution du script et donc l'affichage provoqué par la procédure ***Class_Terminate()*** (les instructions définies dans celle-ci) s'effectuera en fin de document.

En VBScript, on ne parle pas de constructeur ni de destructeur, mais d'événements. ***Class_Initialize()*** et ***Class_Terminate()*** sont les événements de la classe.

Les fonctions accesseur

Les fonctions accesseur sont des fonctions qui vont permettre d'accéder (lecture/écriture) aux variables de la classe avec un seul point d'accès, c'est à dire via une seule section de code.
Imaginons par exemple une classe qui possèderait une propriété *prixHT*. La valeur que nous fournirions à la classe serait donc un prix hors taxes. Différentes fonctions effectueraient des calculs à l'intérieur de la classe en prenant à chaque fois le prix TTC, c'est à dire le prix HT plus 20.6 %. Lorsque le taux change, il faut changer dans toutes les fonctions. Avec une fonction accesseur, nous calculons déjà le prix TTC et c'est cela que la classe utilise, donc une seule ligne de code à changer en cas de changement de taux.
Les fonctions accesseur en VBScript s'écrivent d'une manière un peu particulière. Voir la première d'entre-elles ci-dessous.

Property Let

Cette procédure (sans valeur de retour) est définie ainsi dans la documentation VBScript (les crochets dans le code suivant signifient que c'est optionnel).

```
[Public | Private] Property                                    Let name ([arglist,] value)
  [statements]
```

```
[Exit Property]
[statements]
End Property
```

Il ne faut pas perdre de vue qu'il s'agit bien ici d'une procédure, cela veut donc dire qu'il est possible d'y effectuer des instructions (statements) comme dans n'importe quelle procédure. Simplement, on y accède dans le code comme s'il s'agissait d'une propriété (variable). L'avantage est que l'on peut passer plus d'une valeur ([arglist,]), l'important étant que ce qui va se trouver à droite du signe **=** dans l'expression est la valeur (value) de la propriété, et donc lors de la définition de la propriété dans la classe, il faut bien respecter l'ordre des arguments. Cette propriété est en **lecture seule**.
Si on reprend le code de la classe précédente, on peut maintenant l'écrire (un peu remanié)

```
<%
Class Personne
  Private PrStrPrenom
  Private PrStrSurnom

  Private Sub Class_Initialize()
    Response.Write "La classe est initialisée :<br>"
    PrStrPrenom = "inconnu"
    PrStrSurnom = "l'inconnu"
  End Sub

  Private Sub Class_Terminate()
    Response.Write "La classe est détruite<br>"
  End Sub

  Public Property Let Prenom(StrPrenom)
    PrStrPrenom = StrPrenom
  End Property

  Public Property Let Surnom(StrSurnom)
    PrStrSurnom = StrSurnom
  End Property

End Class

Set instPersonne = New Personne
instPersonne.Prenom = "jérôme"
instPersonne.Surnom = "goldorak" ' goldorak n'est pas mon vrai surnom, c'est juste pour l'exemple ;))
Set instPersonne = Nothing
%>
```

J'ai utilisé ici uniquement la valeur et aucun argument. Voyons un exemple qui n'a aucun intérêt en soi mais qui va permettre de comprendre comment utiliser des arguments :

```
<%
Class Personne
  Private PrStrPrenom
  Private PrStrSurnom
  Private PrStrTitreNom

  Private Sub Class_Initialize()
    Response.Write "La classe est initialisée :<br>"
    PrStrPrenom = "inconnu"
    PrStrSurnom = "l'inconnu"
```

```
  End                                                                    Sub

  Private                                                  Sub Class_Terminate()
    Response.Write "La              classe              est              détruite<br>"
  End                                                                    Sub

  Public        Property        Let SexSitNom(StrSexe,        BlnMarie,        StrNom)
    '    StrSexe    peut    prendre    les    valeurs    "m",    "M",    "f",    "F"
    ' BlnMarie est un booléen et peut donc prendre les valeurs True ou False
    Dim StrTitre '        variable        locale        à        la        fonction

    If                  Lcase(StrSexe)                  =                  "m" Then
      StrTitre                                =                                "Mr"
    Else
      If BlnMarie Then
        StrTitre                              =                              "Mme"
      Else
        StrTitre                              =                             "Melle"
      End                                                                      If
    End                                                                        If
    PrStrTitreNom   =   StrTitre   &   "   "   &   PrStrPrenom   &   "   "   &   StrNom
  End                                                                    Property

  Public                        Property                        Let Prenom(StrPrenom)
    PrStrPrenom                             =                             StrPrenom
  End                                                                    Property

  Public                        Property                        Let Surnom(StrSurnom)
    PrStrSurnom                             =                             StrSurnom
  End                                                                    Property

End                                                                        Class

Set instPersonne                                                 = New Personne
instPersonne.Prenom                             =                        "jérôme"
instPersonne.SexSitNom("m",             False)              =             "Sauvain"
instPersonne.Surnom = "goldorak" ' goldorak n'est pas mon vrai surnom, c'est juste
pour                              l'exemple                                    ;))
Set instPersonne                                                          = Nothing
%>
```

Dans l'état actuel, il n'existe aucun moyen de *récupérer* les valeurs que l'on a saisit, ni les éventuelles modifications qui ont été faites sur elles. En effet, les variables PrStrPrenom, PrStrSurnom et PrStrTitreNom qui contiennent les valeurs saisies sont déclarées comme privées et donc accessibles uniquement dans la classe.
Voyons donc comment accéder à ces valeurs.

Property Get

Cette procédure (avec valeur de retour) est définie ainsi dans la documentation VBScript (les crochets dans le code suivant signifient que c'est optionnel).

```
[Public [Default]                    | Private] Property                    Get name [(arglist)]
  [statements]
  [[Set] name = expression]
  [Exit                                                                      Property]
  [statements]
  [[Set] name = expression]
End Property
```

Le fonctionnement est sensiblement le même que pour ***Let***, à la différence que maintenant vous allez manipuler la propriété à droite du signe égal dans une expression. Elle permet donc d'affecter une valeur à une variable du script (en

dehors ou dans la classe) ou d'afficher cette valeur.
Il est courant d'utiliser comme nom pour une procédure ***Get*** le nom déjà donné à une procédure ***Let***, ceci afin que ces procédures remplissent parfaitement leur rôle de point d'accès.
Toutefois, il est important de noter que dans ce cas, vous devez avoir le même nombre d'arguments (arglist) déclarés dans la procédure ***Get*** que vous en avez déclaré dans la procédure ***Let***. Cette propriété est en **écriture seule**.

```
<%
Class Personne
  Private PrStrPrenom
  Private PrStrSurnom
  Private PrStrTitreNom

  Private                                                Sub Class_Initialize()
    Response.Write "La         classe         est         initialisée         :<br>"
    PrStrPrenom                                =                              "inconnu"
    PrStrSurnom                                =                              "l'inconnu"
  End                                                                                Sub

  Private                                                Sub Class_Terminate()
    Response.Write "La             classe             est             détruite<br>"
  End                                                                                Sub

  '***************                          LET                          **************
  Public        Property        Let SexSitNom(StrSexe,        BlnMarie,        StrNom)
    '    StrSexe    peut    prendre    les    valeurs    "m",    "M",    "f",    "F"
    ' BlnMarie est un booléen et peut donc prendre les valeurs True ou False
    Dim StrTitre '        variable        locale        à        la        fonction

    If                  Lcase(StrSexe)                  =                  "m" Then
      StrTitre                                  =                              "Mr"
    Else
      If BlnMarie Then
        StrTitre                                =                              "Mme"
      Else
        StrTitre                                =                              "Melle"
      End                                                                            If
    End                                                                              If
    PrStrTitreNom   =   StrTitre   &   "   "   &   PrStrPrenom   &   "   "   &   StrNom
  End                                                                        Property

  Public                      Property                      Let Prenom(StrPrenom)
    PrStrPrenom                                =                          StrPrenom
  End                                                                        Property

  Public                      Property                      Let Surnom(StrSurnom)
    PrStrSurnom                                =                          StrSurnom
  End                                                                        Property

  '***************                          GET                          **************
  Public  Property  Get TitreNom() '  ici  les  parenthèses  peuvent  être  omises
    TitreNom                                  =                          PrStrTitreNom
  End                                                                        Property

  Public        Property        Get Surnom() '        ici        aussi        évidemment
    Surnom                                    =                          PrStrSurnom
  End                                                                        Property

End                                                                            Class

Set instPersonne                                                  = New Personne
instPersonne.Prenom                                =                          "jérôme"
instPersonne.SexSitNom("m",             False)             =             "Sauvain"
```

```
instPersonne.Surnom = "goldorak" ' goldorak n'est pas mon vrai surnom, c'est juste
pour                                l'exemple                                   ;))

Response.Write "Bonjour              "             &             instPersonne.TitreNom
Response.Write ",   ou   peut-être   préférez-vous   que   je   vous   appelle   '"   &
instPersonne.Surnom                  &                  "'                  ?<br>"
Set instPersonne                                                        = Nothing
%>
```

Ce qui va donner comme résultat à l'écran :

```
La                  classe                  est                  initialisée                  :
Bonjour Mr jérôme Sauvain, ou peut-être préférez-vous que je vous appelle 'goldorak' ?
La classe est détruite
```

Quelques remarques :

- Si nous définissions une procédure ***Get*** avec le nom *SexSitNom*, il faudrait donc déclarer 2 arguments, pas nécéssairement les mêmes que dans la procédure ***Let***, c'est à dire pas forcément de même signification ni de même type, le traitement de ces 2 arguments pouvant être totalement différent. Par conséquent, lors de l'utilisation de la propriété dans le script, il faudrait passer 2 valeurs :

```
<%
Class Personne
  ...
  '***************                        LET                        **************
  Public       Property       Let SexSitNom(StrSexe,       BlnMarie,       StrNom)
    ...
  End                                                                   Property

  ...
  '***************                        GET                        **************
  Public              Property             Get SexSitNom(arg1,              arg2)
    SexSitNom       =       arg1       &       PrStrTitreNom       &       arg2
  End                                                                   Property

  ...
End                                                                     Class

Set instPersonne                                                  = New Personne
instPersonne.Prenom                          =                          "jérôme"
instPersonne.SexSitNom("m",             False)             =             "Sauvain"
instPersonne.Surnom                          =                        "goldorak"

Response.Write instPersonne.SexSitNom("|",          "|")          &          "<br>"
Set instPersonne                                                        = Nothing
%>
```

Affichage à l'écran :

```
La                  classe                  est                  initialisée                  :
|Mr                                  jérôme                                  Sauvain|
La classe est détruite
```

- Il n'est pas obligatoire que la procédure ***Property Get*** ne se comporte pas réellement comme une propriété, c'est à dire qu'elle se trouve à droite dans une expression. Il suffit qu'elle ne renvoie pas de valeur. Toutefois, pour des raisons de lisibilité et de compréhension du code, je vous déconseille fortement de l'utiliser de cette façon

```
<%
Class Personne
  ...

  '***************                          GET                          **************
  Public                              Property                             Get Prenom()
    PrStrTitreNom       =       PrStrTitreNom       &       "       bla       bla"
    Response.Write PrStrPrenom                       &                        "<br>"
  End                                                                     Property

  ...
End                                                                           Class

Set instPersonne                                                  = New Personne
instPersonne.Prenom                              =                      "jérôme"
instPersonne.SexSitNom("m",               False)             =             "Sauvain"
instPersonne.Surnom                              =                    "goldorak"

instPersonne.Prenom
Response.Write instPersonne.SexSitNom("|",          "|")          &          "<br>"
Set instPersonne                                                        = Nothing
%>
```

Affichage à l'écran :

```
La                  classe              est                 initialisée             :
jérôme
|Mr                 jérôme              Sauvain             bla                     bla|
La classe est détruite
```

- L'emploi du mot-clé Set dans la définition de la procédure va permettre de définir la propriété comme étant de type objet et non de type *Variant*. Par exemple :

```
<%
Class ClsFichier

  Public                              Property                                Get FSO()
    Set FSO                     = Server.CreateObject("Scripting.FileSystemObject")
  End                                                                       Property

End                                                                           Class

Set instClsFichier                                                = New ClsFichier
Set fso                                 =                      instClsFichier.FSO
Set instClsFichier                                                      = Nothing
%>
```

Property Set

Cette procédure (sans valeur de retour) est définie ainsi dans la documentation VBScript (les crochets dans le code suivant signifient que c'est optionnel).

```
[Public | Private] Property                                  Set name ([arglist,] reference)
  [statements]
  [Exit                                                                    Property]
  [statements]
End Property
```

Comme pour ***Let***, la procédure ***Property Set*** va permettre de définir une propriété en lecture, c'est à dire que l'on va écrire à gauche du signe = dans une expression. La grande différence avec ***Let***, c'est que l'on ne va pas lui affecter une valeur de type *Variant* mais une référence à un objet. Cet objet peut-être un

objet COM (comme un objet ADO connection), ou bien une instance de classe. Cette propriété est en **lecture seule**.
Voyons son utilisation sur l'exemple suivant : cette classe va permettre d'instancier un objet **FileSystem**, d'écrire une ligne de texte dans un fichier, et de lire le contenu de ce fichier. Il ne s'agit pas de la meilleure implémentation possible mais c'est un exemple je pense assez parlant.

```
<%
Class ClsFichier

  Private file,                                                    scfso

  Public                          Property                      Set FSO(RefFSO)
    Set scfso                                 =                        RefFSO
  End                                                                Property

  Public     Property     Let OpenFile(IOMode,     CreateIfNotExist,     NomFichier)
    Set file     =     scfso.OpenTextFile(NomFichier,     IOMode,     CreateIfNotExist)
  End                                                                Property

  Public                        Property                      Let WriteTexte(Texte)
    file.writeline         Texte         &         "         "         &         Now()
  End                                                                Property

  Public                          Property                      Get ReadTexte()
    ReadTexte                               =                     file.readall
  End                                                                Property

End                                                                  Class

Set instClsFichier                                              = New ClsFichier
Set insClsFichier.FSO           = Server.CreateObject("Scripting.FileSystemObject")
insClsFichier.OpenFile(8,             true)             =             "c:\file.txt"
insClsFichier.WriteTexte                          =                        "toto"
insClsFichier.OpenFile(1,             true)             =             "c:\file.txt"
Response.Write insClsFichier.ReadTexte

Set insClsFichier                                                    = Nothing
%>
```

Les fonctions/procédures

Les fonctions (procédure avec valeur de retour : ***Function***) et procédures (sans valeur de retour : ***Sub***), qu'elles soient déclarées privées ou publiques, vont constituer les méthodes de la classe.
Comme pour toutes les autres déclarations effectuées dans la classe, elles sont publiques par défaut sauf si vous le préciser avec le mot-clé ***Private*** devant les mots-clé***Function/Sub***.
On pourrait, en reprenant l'exemple précédant, écrire une méthode CloseFile qui fermerait le fichier et détruirait l'instance créée à ce fichier dans la classe, et qui renverrait un message d'erreur ou de confirmation. On pourrait également réécrire OpenFile pour en faire non plus une propriété mais une méthode.

```
<%
Class ClsFichier

  Private file,                                                    scfso

  Public                          Property                      Set FSO(RefFSO)
    Set scfso                                 =                        RefFSO
```

```
  End                                                                    Property

  Public                        Property                          Let WriteTexte(Texte)
    file.writeline        Texte         &         "         "         &         Now()
  End                                                                    Property

  Public                           Property                           Get ReadTexte()
    ReadTexte                                =                          file.readall
  End                                                                        Property

  Public          Sub OpenFile(NomFichier,          IOMode,          CreateIfNotExist)
    Set file     =     scfso.OpenTextFile(NomFichier,     IOMode,     CreateIfNotExist)
  End                                                                          Sub

  Public                                                          Function CloseFile()
    On                      Error                       Resume                      Next
    file.Close
    Set file                                                                  = Nothing
    If                                  Err.Number                              Then
      CloseFile = "Erreur n° " & Err.Number & " lors de la fermeture du fichier : "
& Err.Description
    Else
      CloseFile                  =                  "Fichier                   fermé"
    End                                                                          If
  End                                                                     Function

End                                                                        Class

Set instClsFichier                                                = New ClsFichier
Set insClsFichier.FSO           = Server.CreateObject("Scripting.FileSystemObject")
insClsFichier.OpenFile "c:\file.txt", 8, true ' pas de parenthèses car c'est une
procédure
insClsFichier.WriteTexte                          =                          "toto"
Response.Write insClsFichier.CloseFile()                  &                  "<br>"
insClsFichier.OpenFile              "c:\file.txt",              1,              true
Response.Write insClsFichier.ReadTexte

Set insClsFichier                                                         = Nothing
%>
```

Nous aurions pû faire de la procédure *OpenFile* une fonction avec valeur de retour de la même façon que pour *CloseFile*.

Remarque générale : A aucun endroit dans tous les codes donnés n'apparaît l'instruction ***Exit procédure*** ou procédure peut être ***Property/Sub/Function***. Toutefois, sachez (si ce n'est pas le cas) que cette instruction vous permet d'arrêter le traitement de la procédure en question et d'en sortir.

Cas pratiques

Classe comme fonction

Une possibilité intéressante qu'offrent les classes est de combler une lacune de VBScript. En effet, les fonctions/procédures en VBScript n'acceptent pas d'arguments optionnels. Grâce aux classes, vous pouvez pallier ce manque. C'est à dire que l'on construit une classe ne contenant que des variables de classe publiques (ou bien privées avec des procédures ***Property*** publiques), un événement d'initialisation, et une procédure (avec ou sans valeur de retour).

```
<%
Class ClsFonction
```

```
  Public Prenom,                                                    Nom

  Private                                              Sub Class_Initialize()
    PrStrPrenom                          =                          "Pifou"
    PrStrNom                    =                  "le                chien"
  End                                                                  Sub

  Public                                                        Sub Welcome()
    Response.Write "Bonjour    "    &    Prenom    &    "    "    &    Nom    &    "<br>"
  End                                                                  Sub

End                                                                  Class

Set instClsFonction                                    = New ClsFonction
instClsFonction.Nom                          =                  "Sauvain"
instClsFonction.Welcome
Set instClsFonction                                              = Nothing
%>
```

Pseudo-héritage

Attention, je parle bien de pseudo-héritage et pas d'héritage, l'héritage, comme dit dans l'introduction, n'existant pas en VBScript. Qu'entend-je alors par pseudo-héritage ? Tout simplement la possibilité de déclarer une instance de classe A à l'intérieur d'une classe B et donc de pouvoir utiliser les propriétés et méthodes des classes A et B avec un objet de la classe B. Vous n'avez rien compris ? Voici un exemple très simple qui va permettre d'éclaircir mon propos.

```
<%
Class ClsPere

  Private PNom

  Private                                              Sub Class_Initialize()
    PNom                                 =                        "père"
  End                                                                  Sub

  Public                          Property                   Let LetPNom(un_nom)
    PNom                                 =                        un_nom
  End                                                              Property

  Public                             Property                    Get GetPNom()
    GetPNom                                  =                        PNom
  End                                                              Property

End                                                                  Class


Class ClsFils
  Private instClsPere,                                                FNom

  Private                                              Sub Class_Initialize()
    Set instClsPere = New ClsPere ' instanciation d'un objet de la classe père dans
la                                  classe                           fils
  End                                                                  Sub

  Public                          Property                   Let LetFNom(un_nom)
    FNom                                 =                        un_nom
  End                                                              Property

  Public                             Property                    Get GetFNom()
    If                                                      IsEmpty(FNom) Then
      '  si  la  variable  FNom  n'a  pas  été  explicitement  définie  à  l'aide  de
```

```
la propriété LetFNom, on retourne la valeur contenue dans la propriété GetPNom de
la                                    classe                                    père
      GetFNom                          =                          instClsPere.GetPNom
    Else
      GetFNom                                  =                                  Fnom
    End                                                                          If
  End                                                                         Property

  Private                                                          Sub Class_Terminate()
    Set instClsPere                                                          = Nothing
  End                                                                           Sub

End                                                                            Class


Set instClsFils                    =                    New                    ClsFils
Response.Write "bonjour       "       &       instClsFils.GetFNom       &       "<br>"
instClsFils.LetFNom                          =                                 "fils"
Response.Write "bonjour       "       &       instClsFils.GetFNom       &       "<br>"
Set instClsFils                                                             = Nothing
%>
```

Ce code va produire la sortie suivante à l'écran :

```
bonjour                                                                        père
bonjour fils
```

Tel qu'est définie la classe fils, nous ne pouvons malheureusement pas utiliser les méthodes et propriétés de la classe père en dehors de la classe fils, c'est à dire qu'il n'est pas possible d'écrire :

```
Response.Write "bonjour " & instClsFils.GetPNom & "<br>"
```

Pour pouvoir écrire ce genre de code, il faudrait définir la propriété GetPNom à l'intérieur de la classe fils, celle-ci n'étant qu'une interface pour accéder à la propriété GetPNom de la classe père... ou bien il faudrait que VBScript supporte l'héritage. Concrètement par rapport au code présent, il suffirait de renommer la propriété GetFNom de la classe fils en GetPNom. Toutefois, ceci est très lourd car il faudrait faire de même pour toutes les propriétés et méthodes de la classe père, ce qui au bout du compte reviendrait presque à réécrire (pas tout à fait quand même) la classe père au sein de la classe fils. Autrement dit, pourquoi faire deux classes ?
La solution consiste plutôt à doter la classe fils d'une propriété qui permet d'accéder à la classe père (en dehors de la classe) et donc à toutes ses propriétés et méthodes. Il suffit pour cela d'ajouter la propriété suivante dans la classe fils :

```
  Public                        Property                        Get GetinstClsPere()
    Set GetinstClsPere                              =                        instClsPere
  End Property
```

Nous pouvons donc maintenant utiliser les propriétés et méthodes des deux classes, toutefois l'écriture du code pour manipuler celles de la classe père via un objet de la classe fils va être un tout petit peu différente :

```
Set instClsFils                    =                    New                    ClsFils
Response.Write "bonjour    "    &    instClsFils.GetinstClsPere.GetPNom    &    "<br>"
instClsFils.GetinstClsPere.LetPNom                          =                    "papa"
```

```
Response.Write "bonjour    "    &    instClsFils.GetinstClsPere.GetPNom    &    "<br>"
Response.Write "bonjour      "      &      instClsFils.GetFNom      &      "<br>"
instClsFils.LetFNom                              =                              "fils"
Response.Write "bonjour      "      &      instClsFils.GetFNom      &      "<br>"
Set instClsFils                                                                = Nothing
%>
```

Ce qui aura pour effet de produire l'affichage :

```
bonjour                                                                    père
bonjour                                                                    papa
bonjour                                                                    papa
bonjour fils
```

Conclusion

Les exemples présentés ici sont relativement simples. Toutefois, vos classes peuvent être nettement plus complexes, posséder toute une batterie de procédures Property et autres procédures Sub et Function.
Vous pouvez instancier à l'intérieur même des classes des objets COM, d'autres classes, des objets incorporés du langage ASP, bref seule votre imagination et l'utilisation que vous allez faire de votre classe limitent sa complexité.
Les exemples présentés combinent dans le même script la définition de la classe et son instanciation. Ce n'est évidemment pas l'usage courant. Il vaut mieux séparer les deux et écrire le code de définition de la classe dans un fichier (nomdelaclasse.asp) afin de pouvoir réutiliser cette classe dans plusieurs scripts, et donc d'inclure ce fichier dans chaque script qui nécéssite l'utilisation de la classe.
Les classes ne remplacent pas les composants COM (dll et exe) qui sont plus rapides d'exécution (car ils fonctionnent dans un processus différent de celui de la page), mais elles sont parfois indispensables :
- votre hébergeur ne possède pas tel ou tel composant et refuse de l'installer, il faut alors vous débrouiller par vous-même et une classe est certainement la meilleure solution
- si vous avez l'objectif de créer un composant (surtout si vous l'écrivez en Visual Basic), cela vous permet de tester la validité du code très simplement et rapidement (pas besoin de compiler, enregistrer et redémarrer IIS à chaque fois) et d'effectuer les changements nécéssaires
- cela vous permet également d'avoir un code portable car indépendant des composants installés sur le serveur qui héberge votre application ; vous pouvez donc très facilement*déménager* votre application d'un serveur à un autre
Les principaux désavantages par rapport à un composant COM sont la moins grande puissance et moins grande rapidité d'exécution.
L'autre désavantage est la non persistence des classes. Une classe (ou plutôt un objet de classe) n'existe que dans le script qui l'utilise, ses propriétés sont donc perdues d'une page à une autre. De même il n'est pas possible de placer un objet de classe dans une variable de session ou d'application. Voir à ce propos cet article (en anglais) qui propose diverses méthodes pour obtenir une pseudo-persistence.

Pour résumer (très) rapidement, les classes permettent d'effectuer des opérations complexes à travers une interface simple, donc... à vos claviers.