Vk

Zeste de savoir

Les objets en VBA

jeudi 07 novembre 2024

Table des matières

Introduction

Les objets ne sont décidément pas des valeurs ordinaires.

Pour rappel, un objet est une instance d'une classe.

Durant ce billet, nous allons développer encore quelques points afin de mieux les comprendre.

1. Spécificités

Un objet a quelques spécificités.

1.1. Un objet peut être stocké dans une variable de type générique

Le type générique `Object` permet de référencer n'importe quelle instance.

Par défaut sa valeur est une référence nulle (`Nothing`).

```
1 Dim oClient1 As Object ' Pour liaison tardive
2 Debug.Print (oClient1 Is Nothing) ' Vrai
3 Set oClient1 = New clsClient
```

Cela peut servir quand nous voulons faire de la liaison tardive dont nous parlerons à la fin de ce billet.

1.2. Un objet adresse une zone mémoire

Il est assez logique qu'un objet occupe une certaine place en mémoire vu qu'il peut être composé de tout un tas de propriétés.

En fait, la variable ou propriété référençant cet objet ne contient pas directement toutes ces valeurs mais contient l'adresse vers la zone en mémoire de celles-ci (ou l'adresse nulle `Nothing` quand pas encore initialisée).

C'est pour cela que nous pouvons référencer plusieurs fois un même objet :

```
1  Private Sub Exemple_Objet_et_Memoire()
2      Dim oClient1 As clsClient
3      Dim oClient2 As clsClient
4      Dim oClient3 As clsClient
5
6      Set oClient1 = New clsClient
7      Set oClient2 = oClient1
8      Set oClient3 = New clsClient
9
10     oClient1.sPrenom = "Gabriel"
11     oClient1.sNom = "Dupont"
12     oClient3.sPrenom = "Gabriel"
13     oClient3.sNom = "Dupont"
14
15     Debug.Print (oClient2.sNomComplet) ' Gabriel DUPONT
16     Debug.Print (oClient2 Is oClient1) ' Vrai
17
18     Debug.Print (oClient2.sNomComplet) ' Gabriel DUPONT
19     Debug.Print (oClient3 Is oClient1) ' Faux
20 End Sub
```

Dans l'exemple ci-dessus, l'opérateur `Is` retourne d'abord `True` parce qu'`oClient2` et `oClient1` font références au même objet puis `False` car `oClient3` et `oClient1` ne font pas références au même objet.

Autre conséquence, les objets sont toujours passés par référence en VBA. L'utilisation de `ByVal` ou `ByRef` détermine juste si cette adresse est passée par valeur ou par référence.

1.3. Un objet doit parfois pouvoir être comparé à un autre

Nous venons de nous servir de l'opérateur `Is` afin de comparer deux adresses. Néanmoins, comparer l'adresse d'objets ne suffit pas à évaluer s'il sont identiques.

En effet, deux objets pourraient être considérés comme identiques parce qu'ils ont des propriétés égales après tout. Dans notre exemple, nous aurions pu considérer qu'`oClient3` et `oClient1` étaient aussi identiques car ils avaient le même nom complet (pour faire simple).

Ainsi, il faudra parfois implémenter soi-même une procédure de comparaison, évaluant et retournant un booléen en résultat comme ceci :

```
' Dans module de classe clsClient
Public Function IsEqual(ByRef oClient As clsClient)
    IsEqual = (sNomComplet = oClient.sNomComplet)
End Function
```

```
' Dans procédure de test
Debug.Print (oClient3.IsEqual(oClient1)) ' Vrai
```

1.4. Un objet a un type

Nous avons déjà dit qu'un objet avait un type. Au cours de programmes, il peut être intéressant de vérifier celui-ci, par exemple lors de l'utilisation du type générique `Object`.

Tout d'abord, nous pouvons nous assurer qu'une variable ou propriété contient bien un objet via la fonction `VarType` retournant la valeur `vbObject` (9) dans ce cas. Ensuite, nous pouvons recourir à l'opérateur `TypeOf ... Is ...` pour littéralement vérifier que le type d'une variable ou propriété est bien de la classe indiquée.

```
Dim oClient1 As clsClient
Set oClient1 = New clsClient

Debug.Print (VarType(oClient1)) ' 9 (Objet)
Debug.Print (TypeOf oClient1 Is Object) ' Vrai
Debug.Print (TypeOf oClient1 Is clsClient) ' Vrai
Debug.Print (TypeOf oClient1 Is clsAdresse) ' Faux`
```

1.5. Un objet a une durée de vie

Une instance est liée à une variable ou à une propriété via l'instruction `Set`.

Derrière, il y a alors un compteur qui va suivre le nombre de variables ou propriétés référençant cette instance.

Ce compteur peut être incrémenté en référençant un objet déjà existant avec une autre variable ou propriété :

```
Dim oClient1 As clsClient
Dim oClient2 As clsClient

Set oClient1 = New clsClient ' Compteur client1 = 1
Set oClient2 = oClient1 ' Compteur client1 = 2
```

Ce compteur peut être décrémenté implicitement via les fins de portée ou encore en changeant l'instance liée. Ce compteur peut aussi être décrémenté explicitement en affectant la valeur `Nothing`. Cela peut être une bonne pratique de le faire explicitement pour avoir visuellement l'instruction de fin de vie (si l'objet n'est pas encore référencé ailleurs bien entendu).

Dès que le compteur tombe à zéro, l'objet est détruit. La mémoire occupée est alors automatiquement libérée.

```
' Libération implicite en référençant un nouvel objet
Set oClient2 = New clsClient ' Compteur client1 = 1, compteur
    client2 = 1
' Libération explicite
Set oClient1 = Nothing ' Compteur client1 = 0 donc mémoire libérée
' Libération implicite en fin de procédure de client2 (fin de
    portée) donc mémoire libérée
```

Cela fonctionne aussi ainsi avec les objets stockés dans des structures de données. Par exemple, si une collection contient des dictionnaires et que ces dictionnaires ne sont pas référencés par des variables ou propriétés, ni contenus ailleurs, ils seront implicitement détruits lorsque le compteur de références à la collection tombera à zéro.

Pour résumer, un objet ne vit que tant qu'il est référencé au moins une fois. Sinon, il est automatiquement détruit.

Au cours de cette section, nous avons vu quelques spécificités concernant les objets.

2. Instanciation explicite vs auto-instanciation

En VBA, il y a deux façons d'instancier des objets.

2.1. Instanciation explicite

La première est celle que nous avons utilisée jusqu'à présent : l'instanciation explicite dont voici quelques exemples :

```vba
Dim oClient1 As clsClient ' Déclaration
Set oClient1 = New clsClient ' Instanciation explicite
oClient1.sNom = "DUPONT"

Dim ocTableau(0) As clsClient ' Déclaration
Set ocTableau(0) = New clsClient ' Instanciation explicite
```

2.2. Auto-instanciation

La seconde est l'auto-instanciation. Elle se fait en ajoutant `New` au moment de la déclaration comme ceci :

```vba
Dim oClient2 As New clsClient ' Déclaration en indiquant
    auto-instanciation
oClient2.sNom = "DUPONT" ' instanciation dès utilisation et que
    valeur vide
```

L'objet est alors automatiquement instancié lorsqu'utilisé et étant à `Nothing`.

> ! Cette version a l'avantage d'être plus courte, mais elle offre moins de contrôle sur la vie de l'objet puisque nous ne gérons pas quand il est instancié et que nous ne pouvons pas vérifier s'il a été déférencé, car vérifier son adresse via `Is` a pour effet de l'instancier de nouveau dans le cas où il était à `Nothing` !

```vba
Set oClient2 = Nothing
oClient2.sPrenom = "Gabriel" ' Nouvelle instanciation
Set oClient2 = Nothing
Debug.Print (oClient2 Is Nothing) ' Faux car nouvelle instanciation
```

Nous venons de voir les différents types d'instanciation en VBA.

3. Liaison anticipée vs liaison tardive

Il existe deux types de liaison en VBA.

Le type de liaison définit comment les propriétés et méthodes d'un objet vont être appelées lors de l'exécution du code.

3.1. Liaison anticipée

En liaison anticipée, les propriétés et méthodes seront appelées au travers leur adresse mémoire.

C'est celle que nous avons utilisée jusqu'à présent. Il faut seulement déclarer la variable avec la bonne classe associée :

```
1 Dim cNotes As Collection ' Liaison anticipée
2 Set cNotes = New Collection
3
4 Dim dictFruits As Scripting.Dictionary ' Liaison anticipée
5 Set dictFruits = New Scripting.Dictionary
```

Lorsque nous voulons utiliser certaines classes en liaison anticipée, nous devons également ajouter une bibliothèque au projet VBA comme dans le cas des dictionnaires par exemple.

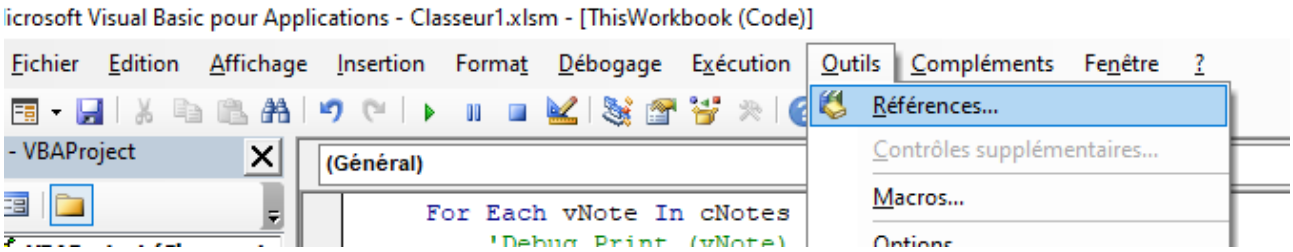

FIGURE 3.1. – Accès à la fenêtre des Références.

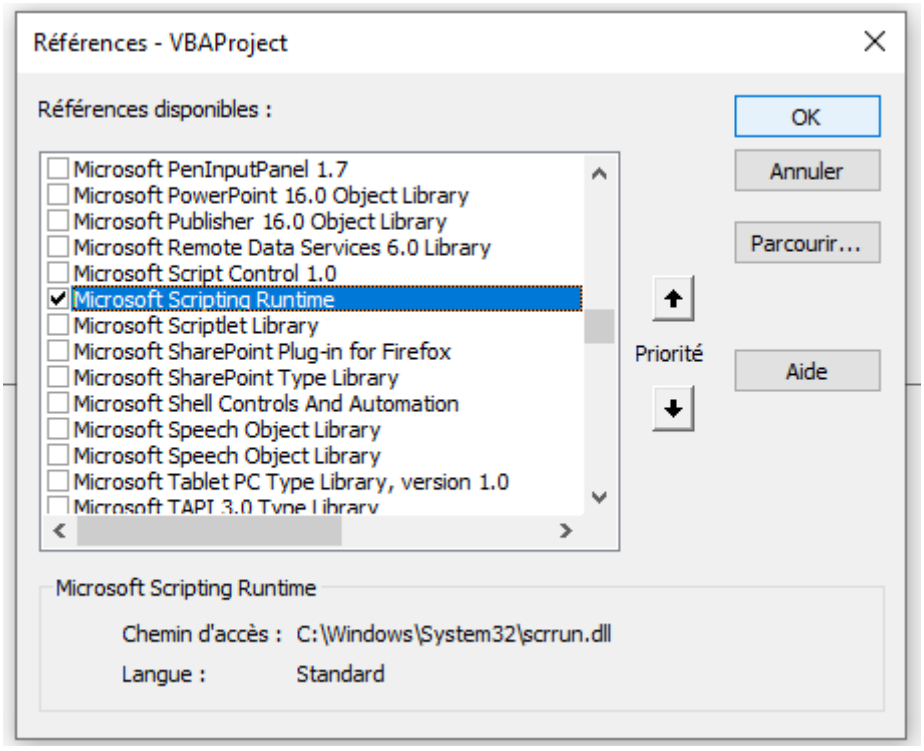

FIGURE 3.2. – Ajout référence Microsoft Scripting Runtime pour la liaison anticipée des dictionnaires.

C'est l'approche la plus performante et recommandée par Microsoft. En outre, elle permet l'aide au développement (IntelliSense) du VBE.

3.2. Liaison tardive

En liaison tardive, les propriétés et méthodes seront appelées dynamiquementau moyen d'un identificateur.

Il faut seulement déclarer la variable avec la classe générique `Object`. En général, nous utiliserons également les fonctions `CreateObject` ou encore `GetObject` :

```
1 Dim cNotes As Object ' Liaison tardive
2 Set cNotes = New Collection
3
4 Dim dictFruits As Object ' Liaison anticipée
5 Set dictFruits = CreateObject('Scripting.Dictionary')
```

La fonction `CreateObject` a pour effet de créer et retourner un objet pour un identifiant composé de la bibliothèque et de la classe voulues.

Rappelons qu'avec la liaison tardive, il n'y a pas besoin d'ajouter la bibliothèque au projet VBA pour utiliser l'objet voulu (si tant est que celui-ci implémente bien le nécessaire pour la liaison tardive).

Ce type de liaison peut être nécessaire dans certains cas (problèmes de comptabilité de composants par exemple).

Au fil de cette section, nous avons vu les différences entre liaison anticipée et liaison tardive.

Conclusion

Voilà, nous en avons appris un peu plus sur les objets en VBA avec certaines de leurs spécificités, les différentes instanciations et les différentes liaisons.

À bientôt !

Quelques ressources :

- La documentation concernant la création de variables d'objet
- La documentation concernant les règles de gestion des nombres de références
- La documentation concernant les liaisons
- La documentation concernant la fonction CreateObject