Esprit EDGE - Support de formation

Sujet : Initiation au VBA ESPRIT

Auteur : RIVIER Enora
Révisé par : Ayoub MERABET
Dernière révision : 15/06/2026
Pour toutes questions techniques : sav@delta-first.com
Version de Esprit applicable : Esprit EDGE 2026.1

Table des matières

Index des Illustrations et Tableaux

1. Introduction

V.B.A© ou **Visual Basic for Application**, est un environnement de développement informatique intégré. Il permet à l'utilisateur d'un logiciel d'automatiser des tâches à travers les passerelles[1] laissées ouvertes par les développeurs du dit logiciel.

Initialement porté sur les produits de la suite **Microsoft Office©**, cet environnement s'est démocratisé sur beaucoup de logiciel basés sur la technologie **Microsoft©**. Par sa simplicité d'utilisation et son intégration au plus près de l'utilisateur finale, **V.B.A©** permet l'optimisation de l'usage d'un logiciel et une grande personnalisation.

L'objectif de cette formation est de permettre aux utilisateurs **ESPRIT©** de mettre un pied dans le vaste monde du développement spécifique et ainsi personnaliser complètement l'usage de son logiciel.

1 Passerelles, ou A.P.I (Application Productivity Interface)

2. Ce qu'il faut tout d'abord savoir

2.1. L'algorithmie

Théorie sommaire

L'algorithmie est l'étude de la résolution de problème par la mise en œuvre de suites d'opérations élémentaires selon un processus défini aboutissant à une solution. Elle crée des algorithmes.

L'algorithme modélise un problème et sa solution par une suite d'action simples articulées par des tests et aboutissant forcement à un résultat. Il est souvent représenté par un logigramme :

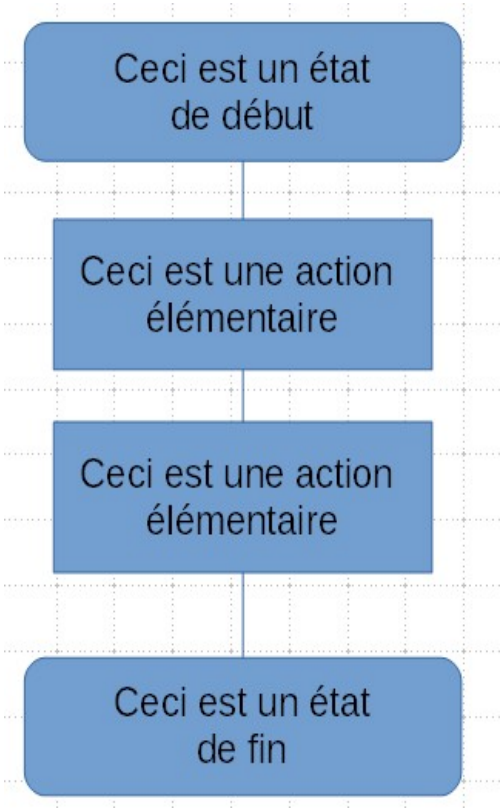


Illustration 1: Logigramme d'algorithme simple

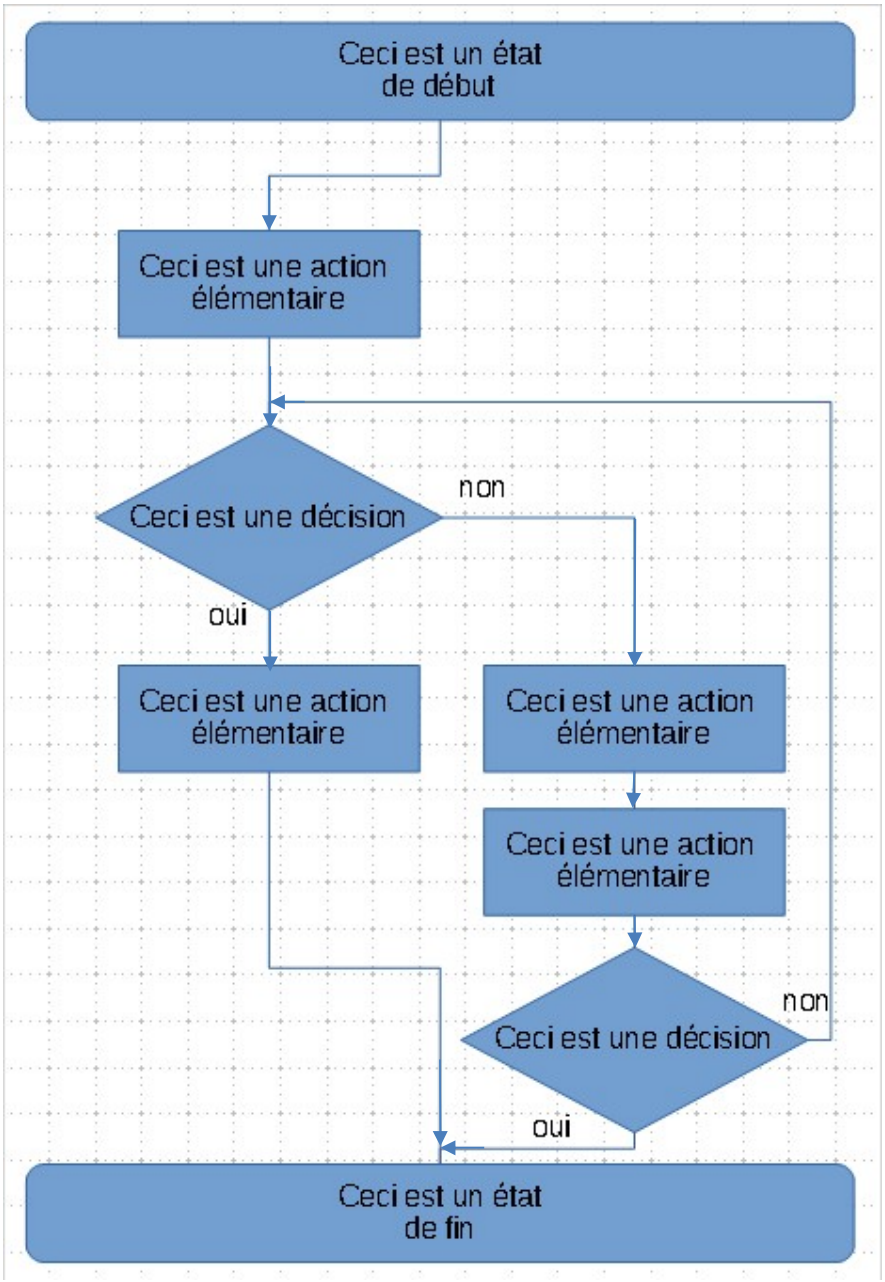


Illustration 2: Logigramme d'algorithme complexe

Cas pratiques (exercices)

Ex 1 :

« Un facteur a un courrier pour un destinataire dont il connaît le nom et l'adresse. Il se situe à la bonne adresse devant plusieurs boites aux lettres étiquetées avec un nom unique sur chacune. Avec quel algorithme est-il possible de modéliser la série d'action menant à la solution, c'est à dire le courrier dans la bonne boite ?

Ex 2 :

« Un humain en position debout doit parcourir une certaine longueur en marchant, la longueur finale étant matérialisée par une ligne blanche traversante sa route. Avec quel algorithme est-il possible de modéliser la marche du sujet jusqu'à attendre cette ligne d'arrivée ? »

2.2. Notions d'analyse fonctionnelle

L'analyse fonctionnelle consiste à modéliser un système par sa fonction principale. Lors de la conduite de cette analyse, il convient de simplifier au maximum ce système pourne garder que trois éléments : la fonction, les données d'entrée et les données de sortie.

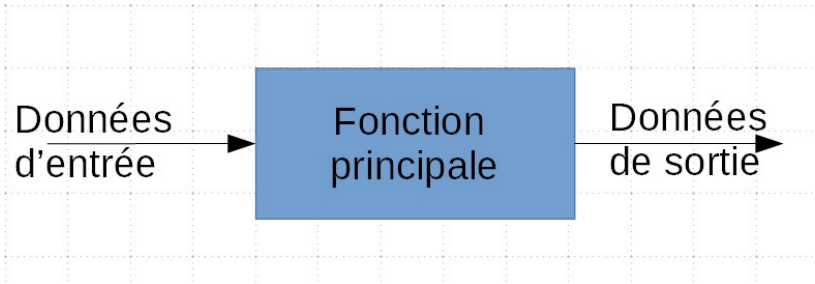


Illustration 3: schéma d'analyse fonctionnel

Illustration 4: Exemple de système (F.A.O) analysé fonctionnellement

Ainsi faite, cette méthode permet d'assembler une série de système simple afin de constituer l'analyse de système plus complexes. Cela se nomme l'encapsulation. Tout le développement informatique est basé sur ce concept qui permet d'architecturer la programmation selon le niveau d'abstraction désiré.

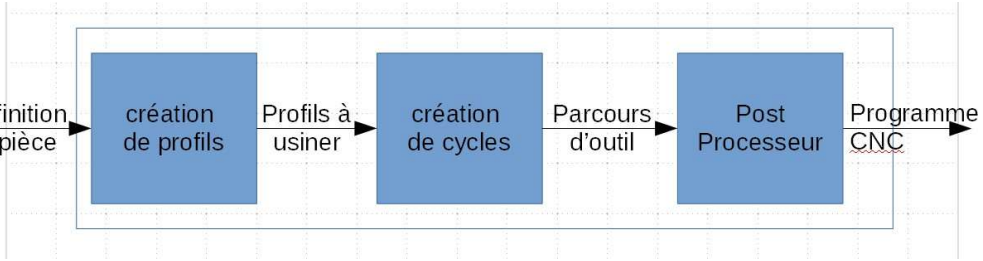


Illustration 5: Exemple d'analyse avec systèmes encapsulés (F.A.O.)

2.3. Accès et aperçu de l'éditeur V.B.A pour ESPRIT

Après le lancement d'**ESPRIT**, l'accès à l'éditeur **V.B.A** se situe dans le menu *Outils→Macro→Editeur Visual Basic*. Il est à noter que le raccourci **ALT+F11** appelle la fonction directement.

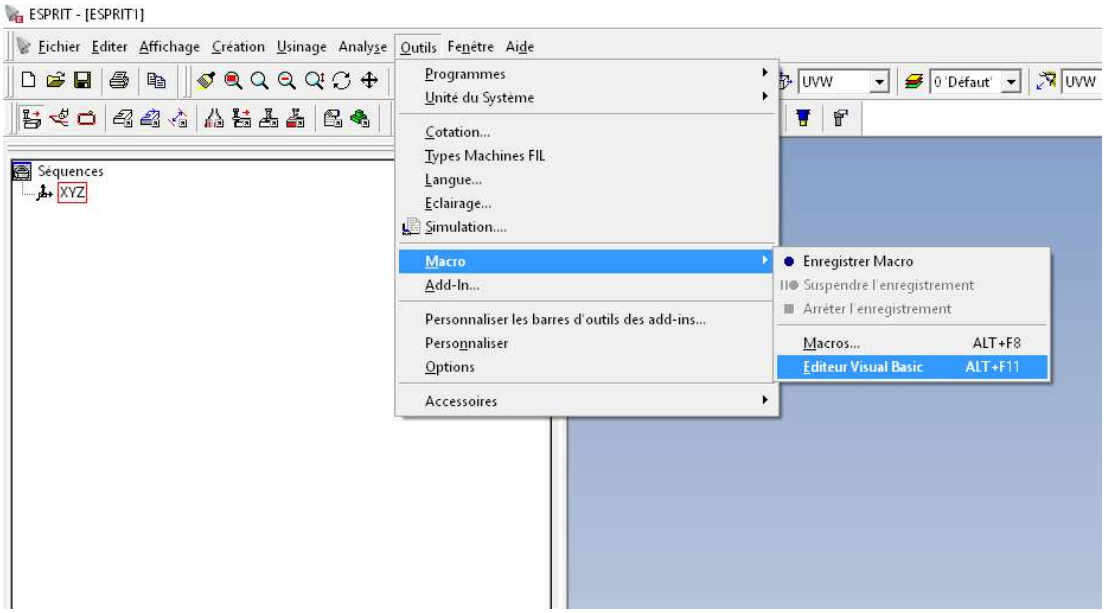


Illustration 6: Accès à l'éditeur V.B.A dans Esprit

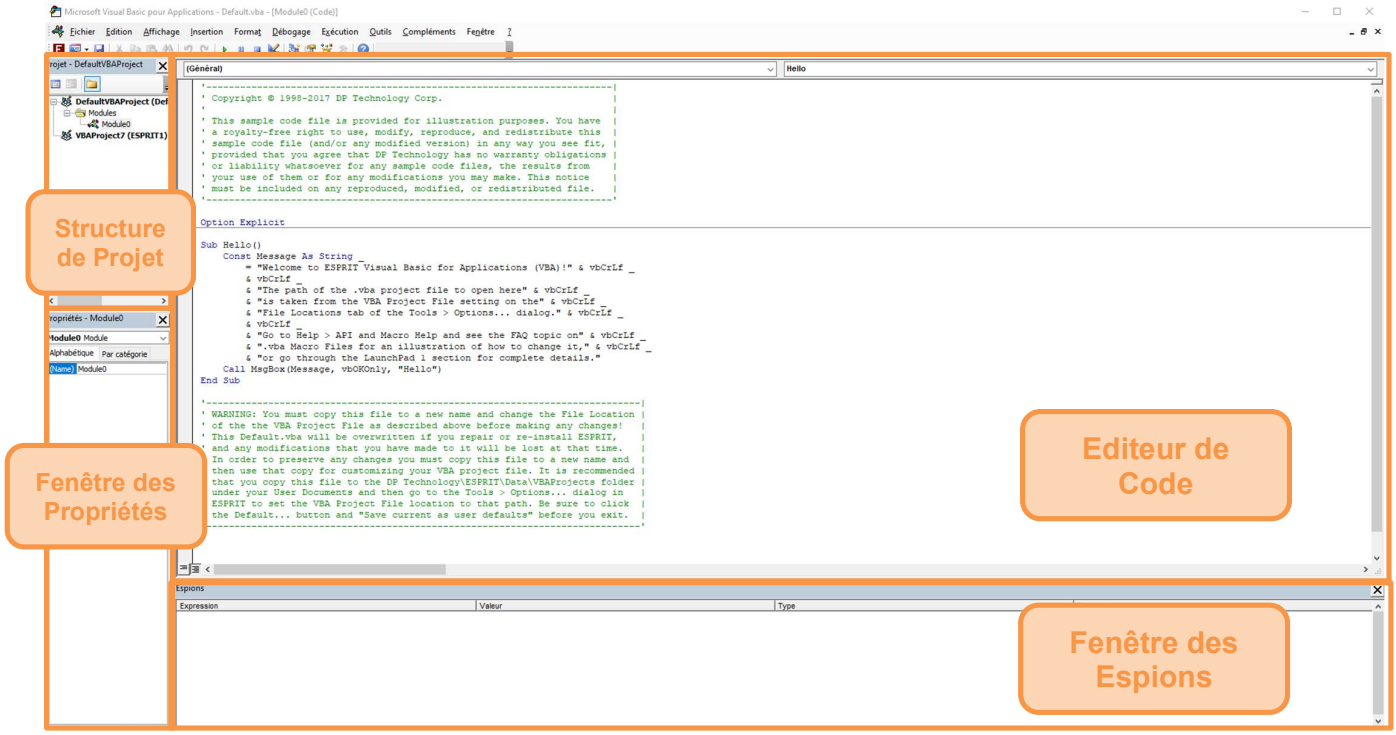


Illustration 7: Aperçu de l'interface de l'éditeur V.B.A

2.4. Documentation

L'aide d'**ESPRIT** sur les API et macros (en anglais) contient énormément d'information sur les API en elles-même mais aussi des tutoriaux, didacticiels et exemples de macros.

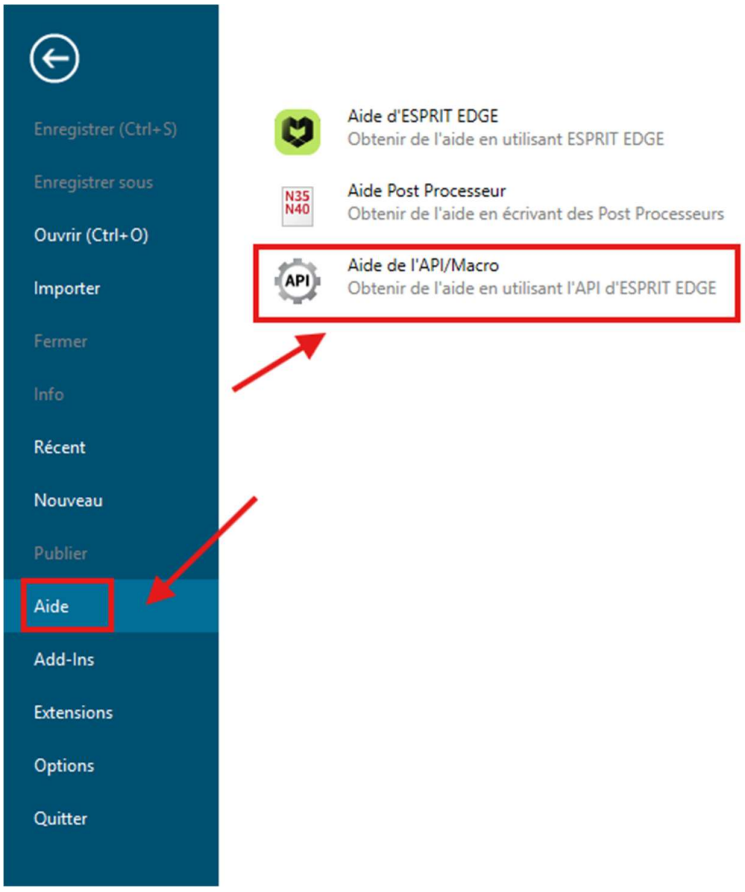


Illustration 8: Accès à la documentation V.B.A pour Esprit

L'éditeur **V.B.A** contient également une aide documentant le langage Visual Basic. Elle est accessible via le menu ' ? ' comme le montre l'illustration ci-après :

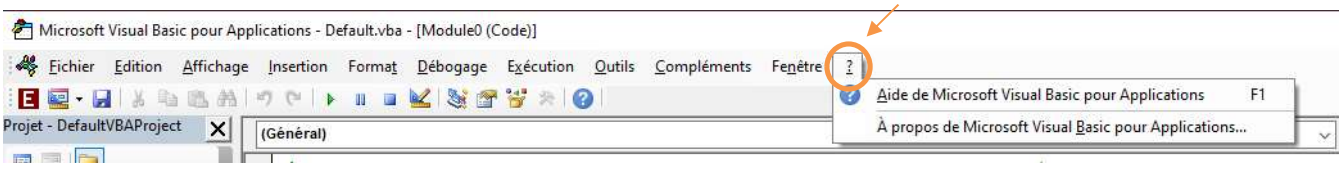


Illustration 9: Accès à la documentation intégrée à l'éditeur V.B.A

Sur Internet, d'autres aides sont disponibles :

- Google (www.google.fr) : Ce moteur de recherche référence les entrées des sites documentaires, rendant très efficace la recherche sur un sujet de développement particulier.

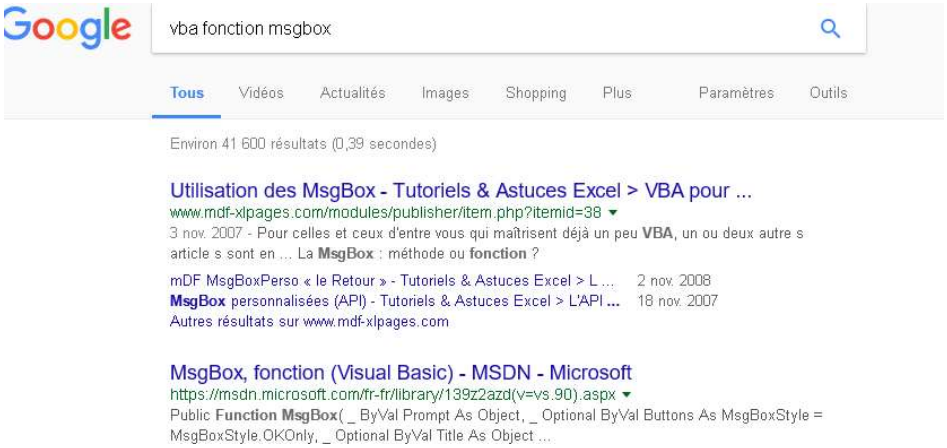


Illustration 10: Google comme référence d'aide à la programmation

- MSDN (msdn.microsoft.com/fr-fr/) : site documentaire de référence.

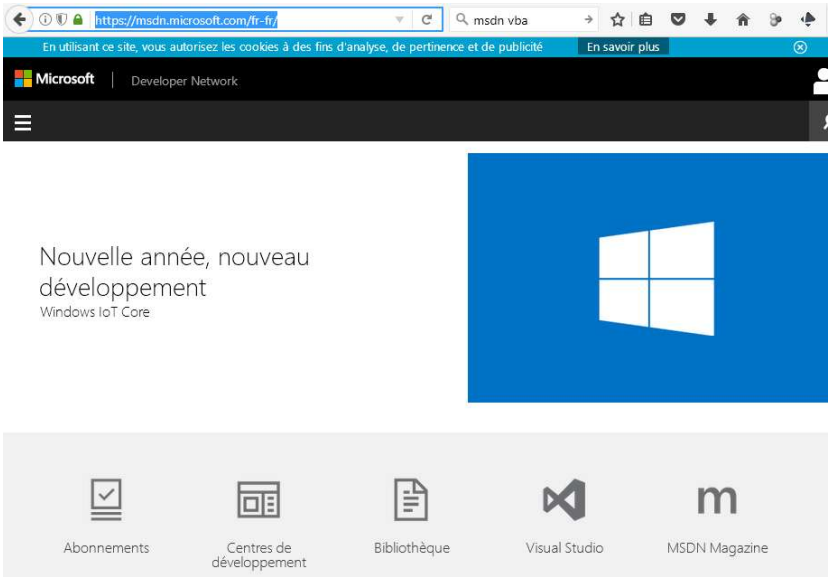


Illustration 11: Site de la MSDN

- Developpez.net (www.developpez.net) : site communautaire regorgeant d'aide, de tutoriaux et de documentation.

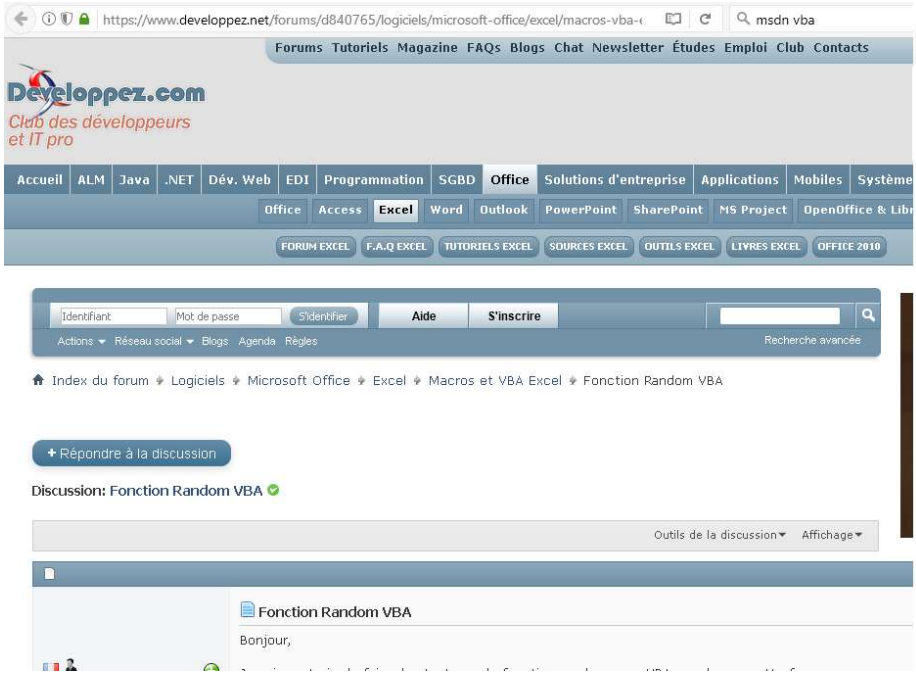


Illustration 12: Site Developpez.com

3. Introduction au V.B.A

3.1. Les types de variables :

Ceux sont les éléments de base de la programmation, les variables vont permettre de stocker les données nécessaires à l'exécution du programme. Il existe plusieurs types en fonction de l'usage :

Type de variable	Contenu
Byte	Entier non signé
Boolean	True ou False
Integer	Tout Entier
Long	tout entier
Currency	Tout Entier
Date	Tout Entier spécial
Single	Nombre décimal
Double	Nombre décimal
Decimal	Nombre décimal
String	Texte
Variant	Tout
Object	Spécial

Tableau 1: Les différents types de variables

Ces différents types vont permettre de stocker des plages de valeurs différentes et impacteront l'usage en mémoire comme le montre le tableau ci-après.

Type de donnée	Espace mémoire	Intervalle de valeur
Byte	1 byte	0 à 255
Boolean	2 bytes[1]	True ou False
Integer	2 bytes	-32,768 à 32,767
Long	4 bytes	-2,147,483,648 à 2,147,483,647
Currency	8 bytes	-922,337,203,685,477.5808 à 922,337,203,685,477.5807
Date	8 bytes	January 1, 100 à December 31, 9999
Single	4 bytes	-3.402823E38 à -1.401298E-45 pour les valeurs négatives; 1.401298E-45 à 3.402823E38 pour les valeurs positives
Double	8 bytes	-1.79769313486232E308 à -4.94065645841247E-324 pour les valeurs négatives; 4.94065645841247E-324 à 1.79769313486232E308 pour les valeurs positives
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 sans point décimal; +/-7.9228162514264337593543950335 avec 28 chiffres après la virgule;la plus petite valeur non nulle est +/-0.0000000000000000000000000001
String (longueur fixe)[2]	longueur de caractère	De 1 à environ 65,400 caractères
String (longeurvariable) [2]	10 bytes + longueur de caractère	De 0 à environ 2 milliards de caractères
Variant (numérique)	16 bytes	toute valeur numérique jusqu'à la portée d'un double
Variant (texte)	22 bytes + longueur de caractère	même portée qu'un string de longueur variable
Object	4 bytes	Tout référence mémoire[3]
User Defined (using Type)	4 bytes + l'espace nécessaires aux élements	Même portée que les types constituant le type personnalisé

Tableau 2: Les différents types de donnée

3.2. Objets

3.2.1 Généralités

V.B.A est un langage dit « orienté objet ». Il est donc possible de manipuler des classes, soit une famille d'objet, et des objets, instances de classes qu'ils représentent. Tousles membres d'une classe ont des attributs en commun mais sont tous uniques. En tentant une analogie avec la vie courante, nous pourrions dire par exemple que « voiture automobile » est une classe, « compact cabriolet » est une sous classe et que le compact cabriolet garé devant chez moi est un objet de la classe « voiture automobile ». La classe « voiture » indique que les objets de cette classe auront 4 roues et seront motorisés.

Dans **ESPRIT**, il existe une classe générale appelée *« ESPRIT »* contenant de nombreuses autres classes comme « point ».

```
Dim P As Esprit.Point
Set P = Document.Points.Add(10, 15, 0)
```

Dans l'exemple ci dessus, *Esprit.Point* est une classe et P un objet instanciant cette classe. Il est à noté qu'un objet s'initialise avec le mot clef 'Set'. P a en commun avec tous les points de la classe, entre autre, trois propriétés qui sont X, Y et Z. Les objets d'une même classe ont en commun des propriétés et des méthodes.

3.2.2 Propriétés (properties)

Tout objet a des propriétés. Ce sont des attributs, des valeurs, accessible en lecture ou en écriture.

La syntaxe utilisable en écriture est la suivante : objet.propriété= nouvelleValeur

La syntaxe utilisable en lecture est la suivante : maVariable = objet.propriété

Il est possible d'utiliser la valeur d'une propriété d'un objet directement dans un test logique. Par exemple :

```
Sub PropertyExample1() Dim P
    As Esprit.Point
    Set P = Document.Points.Add(3, 4, 0)
    P.X = 6
    P.Y = P.X * 4 / 3IfP.Y =
    8Then
                CallMsgBox("P.Y is 8")
    EndIf
End Sub
```

3.3.3 Méthodes (Methods)

Un objet peut avec des méthodes soit des fonctions intégrées à l'objet en lui-même. Ces méthodes permettent souvent de modifier l'objet en lui-même ou être des fonctions déclenchant une série d'actions plus complexe. Plusieurs syntaxes sont possibles pour les actionner :

- Si la méthode ne demande pas d'argument ou ne retourne pas de valeur, il est possible de l'appeler directement en écrivant son nom à la suite de l'objet conteneur.

```
objet.NomDeLaMethode
```

Par exemple, l'objet 'document' d'**ESPRIT** a une méthode '.save' pour déclencher l'enregistrement du document :

```
Document.Save
```

- Si la méthode demande un argument et qu'elle ne retourne pas de valeur, il faut utiliser lemot clef V.B.A : 'Call'.

```
Call objet.NomDeLaMethode(argument)
```

Par exemple, la méthode 'saveAs' de l'objet 'document' permet d'enregistrer le document sous un autre nom et demande le chemin complet de ce nouveau nom.

```
Call document.SaveAs("C:\Program Files\D.P.Technology\Esprit\
data\Examples\MethodExample.esp")
```

- Si la méthode retourne une valeur, il faut assigner cette valeur à une variable ou utiliser le mot clef : 'Call'

```
Dim maVariable as letypeRetourneParLaMethode Set
maVariable = objet.NomDeLaMethode(argument)Call
objet.NomDeLaMethode(argument)
```

Par exemple, la méthode ‘getCookie()’ de l’objet ‘Addin’ ne demande pas d’argument etretourne un entier long. Cela donnerait :

```
Dim MyCookie As Long
MyCookie = Application.AddIn.GetCookie
```

Autre exemple, les méthodes ‘Add’ et ‘item’ des objet-collection retournent toujours un objet.Il convient donc de déclarer et d’initialiser ainsi une variable objet-collection

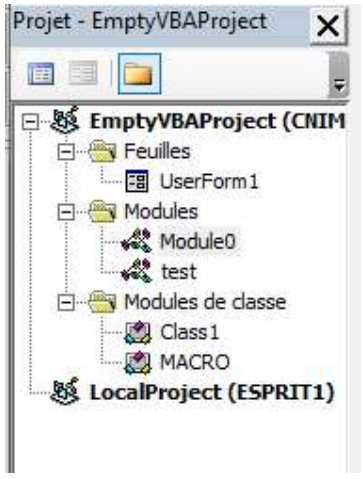
```
Dim monObjet As objet
Set monObjet = objetCollection.Add(arguments)' ou
Set monObjet = objetCollection.item(index)
```

L’exemple suivant exploite les méthodes de l’objet-collection ‘points’ :

```
Dim MyPoint As Esprit.Point
Set MyPoint = Document.Points.Add(1, 2, 3)Dim
MyOtherPoint As Esprit.Point
Set MyOtherPoint = Document.Points.Item(1)
```

3.3.4 Module et Navigation

Dans la fenetre Editeur de Visual Basic, on retrouve l’onglet de navigation qui permet d’alterner entre différence module. Cela permet de structure sa programmation. Les modules ont ici un but organisationnel dans la construction d’un programme, et il en existe 3 types.
Le module classique, qui fera souvent office de programme principal (principalement des appels des autres modules. Le module de classe, qui permettra l’exploitation des différentes librairies d’événements d’esprit. Et enfin le module de « user form » qui permet de créer différentes interfaces graphiques IHM (Interface Homme Machine).



3.3. Les bibliothèques (Libraries)

Les bibliothèques sont les recueils de l'ensemble des classes. Elles organisent les classes par thèmes et sont intégrables dans tout environnement de développement. Les A.P.I d'**Esprit** sont organisées en plusieurs bibliothèques comme le montre le tableau de la page suivante.

Bibliothèque	Description brève
Esprit	Contient l'ensemble des objets graphiques et tout autre objet en support de l'espacegraphique d'ESPRIT
EspritComBase	Contient quelques utilitaires de base pour retrouver des informations d'échec lors desreconnaissances de séquences avec dépouille.
EspritCommands	Contient les objet nécessaires à la création d'addins pour se connecter à l'applicationESPRIT
EspritConstants	Contient la plupart des constantes pour travailler avec les autres bibliothèques
EspritDesigner	Contient des objets nécessaires pour créer des extensions à ESPRIT
EspritFeatures	Contient des objets utilisés pour définir des séquences dans ESPRIT.
EspritFileAPILib	Contient un utilitaire pour retrouver les propriétés de fichier esp sans ouvrir dans ESPRIT.
EspritGeometry	Contient des objets pour traiter le parcours d'outil
EspritGeometryBase	Contient des objets de géométrie de base pour traiter le parcours d'outil
EspritGeometryRoutines	Contient des utilitaires pour traiter les objets de EspritGeometryBase
EspritGraphicsIO	Contient les objets pour les entrées/sorties du matériel graphique.
EspritImages	Contient des méthodes pour retrouver les images bitmap et les icones depuis les ressourcesd'ESPRIT.
EspritMenus	Contient les objets pour atteindre les menus de l'application ESPRIT.
EspritProperties	Contient les objets pour traiter les propriétés personnalisés des objets graphiques.
EspritSimulation	Contient les objets de la simulation du parcours d'outil.
EspritSolids	Contient les objets pour retrouver des informations sur les solides.
EspritSolidsHoles	Contient les objets pour faire de la reconnaissance de trou sur solides.
EspritSort	Contient des utilitaires pour trier des chaines de points.
EspritStrings	Contient des méthodes pour retrouver du texte à partir des ressources ESPRIT.
EspritTechnology	Contient tous les outils et objets technologiques utilisés pour définir les opérationsd'usinage dans ESPRIT.

EspritTools	Fournit une interface pour les collections d'outils dans le document ESPRIT.
EspritUtils	Contient certains utilitaires pour faciliter le travail avec les API

Tableau 3: Les différents types de bibliothèque

3.4. Quelques exemples de macro

3.4.1 Bonjour

```
Sub Hello()
    Const Message As String _
        = "coucou"
    Call MsgBox(Message, vbOKOnly, "Hello")End Sub
```

Cette simple macro fait appel à l'objet msgBox qui permet d'afficher une boite de messageà l'utilisateur avec un texte et un bouton de validation

3.4.2 CreateCircle

```
Sub createCircle()
    Dim monpt As Esprit.Point
    Set monpt = Document.GetPoint(10, 20, 0)
    Call Document.Circles.Add(monpt, 10)Call
    Document.Refresh
End Sub
```

Cette macro fait appel à plusieurs objets **ESPRIT** dont '*Document.circles*' qui propose la méthode *Add* pour ajouter un cercle au document **ESPRIT**. Cette méthode demande deux arguments, un '*Esprit.point'* et un '*double*' pour rayon comme l'indique l'aide sur les API et macro.

```
Function Add(CenterPoint As IPoint, Radius As Double) As Circle
```

4. Passerelles vers Microsoft Excel©

Microsoft Excel possède également des A.P.I permettant de manipuler ses objets et intéragir depuis **ESPRIT** sur des feuilles de calculs, cellules ou tout autre élément de l'application qui serait exposé dans la bibliothèque **Microsoft Excel**.

4.1 Activation de la bibliothèque Microsoft Excel dans le V.B.A ESPRIT

Parce que le lien vers cette bibliothèque n'est pas chargé par défaut dans l'environnement de développement Visual Basics d'**ESPRIT**, il convient de l'activer via le menu Outils > Références.

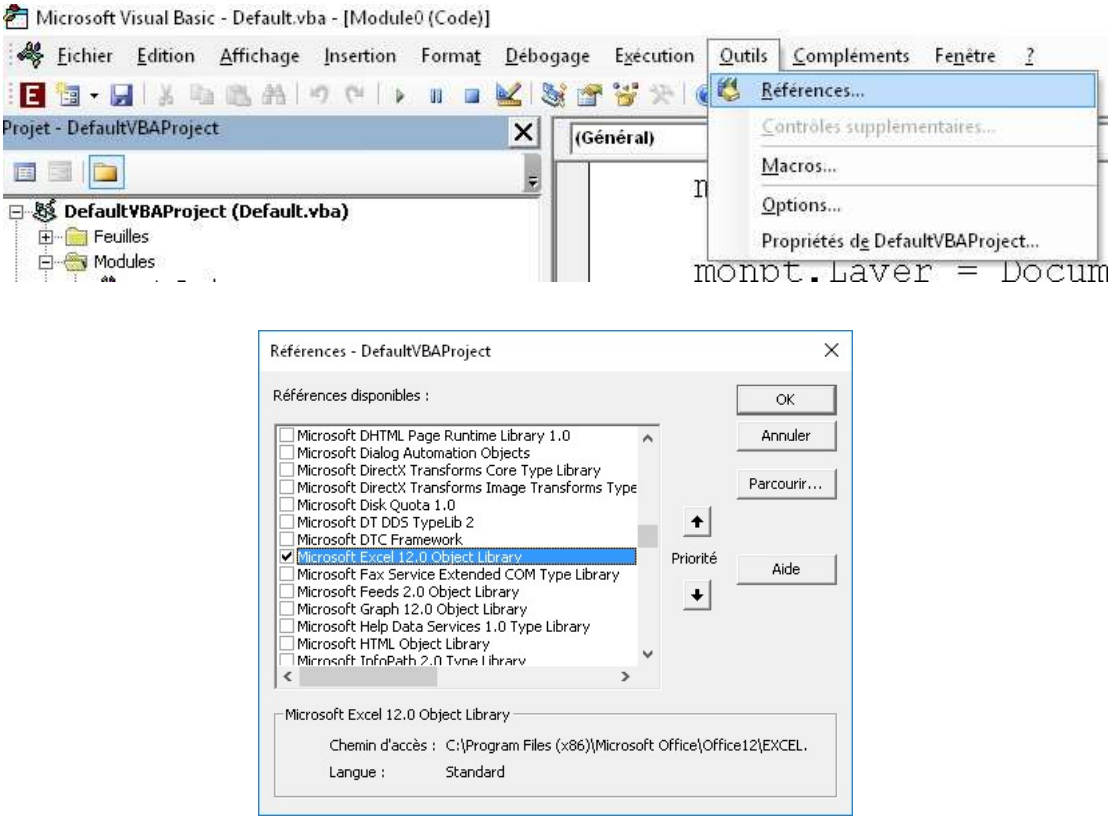


*Illustration 13: Activation Bibliothèque **Microsoft Excel***

Il est à noter que le nom peut changer en fonction de la version de **Microsoft Office** qui est installée.

4.2 Quelques objets Excel

Afin d'ouvrir une instance d'**Excel**, il est possible déclarer un objet de type *Excel.application* et de l'initialiser :

```
Dim XlApp As Excel.Application
Set XlApp = New Excel.Application
XlApp.Visible = True
```

Il devient alors possible de créer un classeur par l'initialisation d'une variable de type *Excel.Workbook* :

```
Dim xlwb As Excel.Workbook
Set xlwb = XlApp.Workbooks.Add
```

4.3 Exemple de macro

Le Tutorial 143 de l'aide « *ESPRIT API and Macro Help* »: intitulé « *Reading and Writing Excel Spreadsheets* » présente un bon exemple d'export de donnés **ESPRIT** vers une feuille de calcul **Excel**.

```
Sub WritePointsToExcel()
  ' set up the Excel application'
  Dim XlApp As Excel.Application Set XlApp =
  New Excel.ApplicationXlApp.Visible =True
  Call XlApp.Workbooks.Add'
  ' set up a sheet for the points'
  Dim WS As Excel.Worksheet Set
  WS = XlApp.ActiveSheetWS.Name
  = "Points"
  '
  ' write the header row'
  WS.Range("A1").Value = "Key"
  WS.Range("B1").Value = "X"
  WS.Range("C1").Value = "Y"
  WS.Range("D1").Value = "Z"'
  ' now loop through all of the points'
```

```
    Dim IAs Long, PAs Esprit.Point, RowNumberAs StringFor I = 1To
    Document.Points.Count
      Set P = Document.Points.Item(I)'
      ' write the point data'
      RowNumber = Trim(Str(I + 1)) WS.Range("A" &
      RowNumber).Value = P.KeyWS.Range("B" &
      RowNumber).Value = P.X WS.Range("C" &
      RowNumber).Value = P.Y WS.Range("D" &
      RowNumber).Value = P.Z
    Next'
    ' cleanup'
    Set WS = Nothing Set
    XlApp = Nothing
End Sub
```

Autre exemple, qui permet de lire des informations depuis Excel à destination de ESPRIT :

```
Sub ReadPointsFromExcel()'
    ' prompt the user for the point spreadsheet to openDim
    PointDataFilenameAs String
    Load frmCommonDialog1
    frmCommonDialog1.Hide
    With frmCommonDialog1.CommonDialog1
                                                .InitDir                                    =
Application.Configuration.GetFileDirectory(espFileTypeEspritDrawings)
        .Filename = ""
        .Filter = "Point Data Spreadsheet (*.xls)|*.xls"
        .FilterIndex = 1
        .DialogTitle = "Open Point Data Spreadsheet"
        .ShowOpen
    End With
    PointDataFilename = frmCommonDialog1.CommonDialog1.FilenameIf
    PointDataFilename = ""Then Exit Sub '
    ' set up the Excel application'
    Dim XlApp As Excel.Application Set XlApp =
    New Excel.ApplicationDim WBAs
    Excel.Workbook
    Set WB = XlApp.Workbooks.Open(PointDataFilename)Dim WS
    As Excel.Worksheet
    Set WS = WB.ActiveSheet'
    ' read the data'
```

```
    Dim RowNumberAs Long
    Dim XAs Double, YAs Double, ZAs DoubleDo
      RowNumber = RowNumber + 1'
      ' if first cell is empty, that is end of data'
      If WS.Range("A" & RowNumber).Value = ""Then Exit DoX =
      Val(WS.Range("A" & RowNumber).Value)
      Y = Val(WS.Range("B" & RowNumber).Value) Z =
      Val(WS.Range("C" & RowNumber).Value)' make the
      point
      '
      Call DefaultAttributes(Document.Points.Add(X, Y, Z))
    Loop
    ' cleanup
    WB.Close
    Set WS = Nothing Set
    WB = Nothing Set XlApp
    = Nothing
End Sub
```