


Aide-mémoire LibreOffice

LibreOffice Basic

Types structurés

Inter.



v. 1.01 – 03/11/2019

Rédigé avec LibreOffice v. 5.3.3 – Plateforme : Toutes

Tableaux (Arrays)

Regroupe des éléments ayant un rapport entre eux. Les éléments sont indexés (Long).

Dimensions

Un tableau peut comporter plusieurs dimensions (maxi : 60).

Déclarer

⚠ La base d'indexation est 0 (zéro) ! Modifiable par Option Base 1 (peu courant).

Ci-dessous : ix = index.

Tableaux statiques	Caractéristiques prédéfinies
1 dimension (« vecteur »)	
Dim T(ixMin To ixMax) As UnType	Dim T(1 To 12) As UnType 1 dimension de 12 emplacements mémoire, indexés de 1 à 12.
ou Dim T(ixMax) As UnType	Dim T(9) : 1 dimension de 10 éléments, indexés de 0 à 9.
Plusieurs dimensions	
Dim T(ixMin1 To ixMax1, ixMin2 To ixMax2) As UnType	Plusieurs dimensions (exemple avec 2). Dim T(1 To 12, 1 To 31) As Integer 2 dimensions (réserve 12 et 31 éléments)
ou Dim T(ixMax1, ixMax2, ...) As UnType	Dim T(2,4) As String : 2 dimensions. 3 éléments pour la 1 ^è , 5 pour la 2 ^è .

Tableaux dynamiques	Caractéristiques connues à l'exécution
Dim Tablo() As UnType Dim Tablo As Variant	Déclare un tableau de dimension inconnue. Un ReDim sera nécessaire par la suite.

☑ Déclaré de type Variant, un tableau peut contenir des éléments de types différents.

Tableaux imbriqués (Nested arrays/Jagged arrays)

Ou tableaux de tableaux. Ex : utilisés pour accéder aux données des plages dans Calc.

Un tableau externe (de variants) a pour éléments des tableaux de données :

```
Dim T As Variant
T = Array(Array(1, 2, 3), Array(10, 20, 30), Array(7, 8, 9))
T(0)(0) vaut 1; T(2)(2) vaut 9, etc.
```

Accéder à un élément d'un tableau

```
Par son index :
Dim Tablo(9) As Integer           Dim Tablo(11, 31) As Integer
Tablo(5) = 123                   Tablo(5, 28) = 123
```

Fonctions et instructions relatives aux tableaux

Option Base 1	(instruction en début de module – applicable au module courant) Force l'index des tableaux à débuter à 1 au lieu de 0.
IsArray()	Renvoie True si la variable est d'un type tableau. OK = IsArray(Tablo)
Array()	Renvoie un tableau créé à partir de valeurs discrètes. Tablo = Array("Un", 2, Now()) 'ici, tableau de variants
Redim	(instruction) Redimensionne un tableau Avec perte des données : Redim Tablo(dimension) Sans perte de données : Redim Preserve Tablo(dimension)
Erase	(Instruction) Efface le contenu du tableau. Erase Tablo Si tableau dynamique, libère la mémoire.
LBound()	Renvoie la borne inférieure. Par défaut sur la 1 ^{ère} dimension, sinon préciser : LBound(Tablo, 2)
UBound()	Renvoie la borne supérieure (idem). ☑ Le tableau n'a pas de dimension définie si UBound(Tablo) = -1 et LBound(Tablo) = 0
Split()	Crée un tableau (vecteur) par découpe d'une chaîne sur un délimiteur. T=Split("C:/file.txt", "/") → T(0)="C:", T(1)="file.txt"
Join()	Réalise l'opération inverse de Split() : fusionne des éléments d'un tableau (vecteur) pour initialiser une chaîne. Join(T, " ") → "C: file.txt"

Vérifier la validité d'un tableau

Soit Tablo une variable « tableau ». Elle est manipulable si elle passe les trois tests :

1. La variable existe-t-elle ? Not IsNull(Tablo)
2. S'agit-il d'un tableau ? IsArray(Tablo)
3. La dimension du tableau est-elle définie ? UBound(Tablo) >= LBound(Tablo)

Parcourir un tableau à 1 dimension (vecteur)

Par ses index

```
Dim Tablo(9) As Integer, i As Long
For i = LBound(Tablo) To UBound(Tablo)
    Print Tablo(i)
Next i
```

Par ses éléments

```
Dim Val As 'type compatible avec les éléments
For Each Val In Tablo
    Print Val
Next
```

Parcourir un tableau à 2 dimensions

```
Dim Tablo(2, 4) '3 lignes, 4 colonnes
Dim i As Long, j As Long
For i = LBound(Tablo) To UBound(Tablo)
    For j = LBound(Tablo, 2) To UBound(Tablo, 2)
        Print Tablo(i, j)
    Next j
Next i
```

Trier des tableaux

☑ Pas de fonctionnalité prédéfinie (chercher QuickSort sur le web).

Dupliquer des tableaux

Cas général Une boucle (For...Next) qui recopie les valeurs entre les 2 tableaux.
Peut être looong !

Astuce

```
Par affectation puis ReDim Preserve :
Tablo2 = Tablo1 'les deux var. -> les mêmes données
ReDim Preserve Tablo2(Taille) ' -> 2 tableaux différents
☑ S'applique uniquement aux tableaux de valeurs simples (non objet).
```

Utiliser les tableaux avec des sous-programmes

Comme paramètre d'une Sub

```
Sub avec paramètre tableau Sub UseArray(ByRef Tablo() As String)
Appel de la Sub Dim Tablo(9) As String
UseArray(Tablo)
```

Comme résultat d'une Fonction

```
Fonction avec résultat tableau Function GetArray() As Integer()
GetArray = UnTabloEntiers()
Retour de la fonction Dim MonTablo As Integer
MonTablo = GetArray()
```

Tableaux et plages de classeur

(voir A-M n°3)

Types personnalisés (Custom types)

Permettent d'agréger plusieurs valeurs (membres) dans un type de donnée unique. Simplifient la manipulation de données, le passage de param. ou le retour de fonction.

⚠ Type des membres : n'importe quel type simple ou personnalisé (sauf tableau).

Déclaration du type

```
Type MonTypePerso Type Evenement
UnMembre As TypeSimple Nom As String
AutreMembre As AutreTypeSimple DateHeure As Date
End Type End Type
```

Déclaration des variables de type personnalisé

```
Dim UneVar As MonTypePerso
```

⚠ Limitation : un type personnalisé n'est visible que dans le module où il est déclaré. La déclaration As MonType n'est donc possible qu'au sein du même module que celui où est déclaré le type MonType. Voir Fonction « usine » (Factory) / Accesseur (accessor).

Utilisation

Affectation

```
VarPerso.UnMembre = UneValeur
```

Lecture

```
UneVar = VarPerso.AutreMembre
```

Le mot-clé With

```
Permet d'abrégier les références aux With VarPerso
éléments .UnMembre = UneValeur
End With
☑ Notez la présence du point.
```

Collections

Structure qui permet l'accès rapide aux données en les indexant.

Une collection est manipulée comme un type Object.

Les éléments stockés peuvent être de n'importe quel type, y compris Object.

⚠ La clé d'indexation est de type String. Dans une collection, chaque clé est unique.

Déclarer une collection

```
Dim oColl As New Collection
```

Ajouter un élément

```
Avec clé oColl.Add(Elt, "LaClé")
☑ Accès ultérieur par clé ou par index.
☑ La clé n'est pas sensible à la casse.
☑ Il est possible de préciser l'emplacement de l'ajout au moyen de Before/After :
oColl.Add(Elt, "LaClé", After:="Clé0")
oColl.Add(Elt)
☑ Accès ultérieur par index seulement.
```

Sans clé

Vérifier l'exist^e d'un élément

Tenter d'accéder et traiter l'erreur éventuelle (voir ci-dessous).

Accéder à un élément

```
Par sa clé Valeur = oColl("LaClé")
```

Par son index Valeur = oColl.Item(Index)

Remplacer un élément

Même clé, nouvelle valeur d'élément.
☑ Supprimer puis Ajouter.

Supprimer un élément

```
Par sa clé oColl.Remove("LaClé")
```

Par son index oColl.Remove(Index)

Supprimer tous les éléments

```
ReDim oColl As New Collection
```

Nombre d'éléments

```
Nombre = oColl.Count
```

Vérifier l'existence d'un élément

Tenter d'accéder à la clé et traiter l'erreur éventuelle.

```
Function ExistsItem(ByRef pColl As Object, pKey As String) As Boolean
Dim Item As Variant, Exists As Boolean
On Local Error Goto ErrHandler
Exists = False
Item = pColl(pKey) 'si pKey n'existe pas -> ErrHandler:
Exists = True
ErrHandler:
'do nothing
ExistsItem = Exists
End Function
```

Parcourir une collection

Vous pouvez récupérer les éléments d'une collection en la parcourant.

☑ Il n'existe pas de possibilité de lister les clés.

Par index

```
For i = 1 To oColl.Count
    Valeur = oColl.Item(i) 'accès à la donnée
Next i
```

Par accès direct aux éléments

```
Dim Element As 'type compatible avec les éléments de la collection.
For Each Element In oColl
    'faire qqch avec Element (donnée)
Next
```

Créer des classes en Basic

Embryon de programmation orientée objet (POO).

☞ Limites : pas d'héritage (utilisez la délégation), pas de polymorphisme !

Une classe LibreOffice Basic pourrait donc être vue comme un type personnalisé amélioré auquel on ajoute un comportement (fonctions et sous-programmes).

Vocabulaire

Module de classe	Module de code destiné à contenir les déclarations de la classe.
Classe	Type qui permet de créer (instancier) des variables objet.
Événements	Deux événements peuvent être interceptés : la création et la destruction de l'objet.
Membre	Variable interne à une classe.
Propriété	Reflète l' état de l'objet.
Méthode	Réalise une action sur l'objet.
Instance	Objet créé à partir d'un type classe.

Spécifier une classe

Les spécifications d'une classe (membres, événements, propriétés, méthodes) se font à l'intérieur d'un module de code dédié. Dans LibreOffice Basic, ce module ne se distingue d'un module standard que par ses options initiales.

☞ Conseil : adoptez une convention de nommage pour les modules de classes.

Options initiales

En début de module de classe, précisez les options :

```
Option Explicit 'comme d'habitude
Option Compatible
Option ClassModule
```

Variables membres

Elles sont internes, donc déclarées `Private`.

☞ Les membres d'une classe ne devraient **jamais** être appelés directement à travers l'instance mais uniquement au travers de propriétés créées à cet effet.

```
Private mName As String
Private mSheet As Object
```

Événements (Events)

Ce sont deux sous-programmes internes, donc déclarés `Private`.

Constructeur `Private Sub Class_Initialize()`

Pour initialiser l'objet en cours de création.

Destructeur `Private Sub Class_Terminate()`

Pour nettoyer les composants internes d'un objet en cours de destruction.

☞ **Faille de sécurité.** Il est **très fortement** déconseillé d'inclure ce destructeur dans vos classes : en raison d'un bug d'implémentation dans VisualBasic, `Class_Terminate()` constitue une faille de sécurité et, comme telle, est rejeté par les antivirus (voir CVE-2018-8174).

☞ **Limitation** : il n'est pas possible de passer des paramètres à ces sous-programmes.

Propriétés (Properties)

Propriété = **état** de l'objet.

Elles sont visibles de l'extérieur, donc déclarées `Public`.

En lecture (Get) (tous types de données, y compris objets)
`Public Property Get Name() As String`
 Name = mName
`End Property`

En écriture (Let) (tous types de données sauf objets)
`Public Property Let Name(ByRef pName As String)`
 mName = pName
`End Property`

En écriture (Set) (objets seulement)
`Public Property Set Sheet(ByRef pSheet As Object)`
 Set mSheet = pSheet
`End Property`

☞ Il est possible de quitter prématurément une propriété par `Exit Property`

Une classe peut comporter des propriétés en lecture et en écriture : écrivez successivement les deux propriétés `Get` et `Let/Set` si nécessaire.

Les propriétés peuvent accéder aux membres, aux propriétés et aux méthodes de la classe.

Méthodes (Methods)

Méthode = **action** sur/de l'objet.

`Sub` et `Function` propres à la classe, internes (`Private`) ou visibles (`Public`). S'écrivent comme des `Sub` ou des `Function` standard précédées du mot-clé `Public` ou `Private`.

Elles ont accès aux membres et aux propriétés de la classe.

Utiliser des classes en Basic

Déclarer/créer un objet

```
Instanciation immédiate     Set MonObjet = MaClasse
Instanciation différée     { Dim MonObjet As New MaClasse
(au 1er appel)             { MonObjet = New MaClasse
```

Le constructeur de la classe est appelé au moment de l'instanciation de l'objet.

La déclaration d'un objet `As New MaClasse` n'est pas possible en dehors de la bibliothèque dans laquelle le module de classe existe. Vous devrez généralement déclarer l'objet `As Object`.

☞ **Limitation** : Une classe n'est **pas** visible en dehors de la bibliothèque dans laquelle elle est déclarée. Voir Fonction « usine » (Factory) / Accesseur (accessor).

Accéder aux propriétés et méthodes d'un objet

Syntaxe d'accès aux éléments d'un objet : `objet.propriété` ou `objet.méthode`

Comme pour les types personnalisés, vous pouvez utiliser la syntaxe `With...End With`

Libérer un objet

Lorsqu'un objet n'est plus nécessaire, vous pouvez le détruire : `Set oObjet = Nothing`
Le destructeur de la classe est appelé à ce moment là.

☞ Cette instruction n'est pas strictement nécessaire dans des `Sub` ou des `Function`, puisque leurs variables internes sont détruites en sortie. Cependant la destruction effective de la variable n'est alors pas sous votre contrôle.

L'instruction `Set oObjet = Nothing` :
– montre clairement l'intention,
– assure le contrôle du moment de la destruction de l'objet.

Fonction « usine » (Factory) / Accesseur (accessor)

Question de la visibilité des types personnalisés et des classes

- Un **type personnalisé** n'est visible que dans le **module** où il est déclaré.
 - Une **classe** n'est visible que dans la **bibliothèque** où elle est déclarée.
- Pour contourner ce problème, préparez une fonction « usine » (ou accesseur) pour créer les variables. Cette fonction pourra être appelée de n'importe quel module/bibliothèque.

	Utilisation	Visibilité	Usine créée dans	Déclaré
Type perso	As MonType	Même module	Même module	As Variant
Classe	As MaClasse	Même bibli	Même bibli, autre module	As Object

Création de la fonction « usine » / accesseur

☞ La fonction usine peut aussi être utilisée pour initialiser la variable.

Types personnalisés

```
Function CreateMonTypePerso() As MonTypePerso
    Dim oVar As MonTypePerso
    CreateMonTypePerso = oVar
End Function
```

Classes

```
Function CreateMaClasse() As MaClasse
    Dim oVar As New MaClasse
    CreateMaClasse = oVar
End Function
```

Utilisation de la fonction « usine » / accesseur

Types personnalisés

```
Dim MaVar As Variant
MaVar = CreateMonTypePerso()
```

Classes

```
Dim MaVar As Object
MaVar = CreateMaClasse()
```

Crédits

Auteur : Jean-François Nifenecker – jean-francois.nifenecker@laposte.net

Nous sommes comme des nains assis sur des épaules de géants. Si nous voyons plus de choses et plus lointaines qu'eux, ce n'est pas à cause de la perspicacité de notre vue, ni de notre grandeur, c'est parce que nous sommes élevés par eux. (Bernard de Chartres [attr.])

Historique

Version	Date	Commentaires
1.0	20/04/2019	Première version
1.01	03/11/2019	Corrections

Licence

Cet aide-mémoire est placé sous licence

Creative Commons BY-SA v3 (fr).

Informations

<https://creativecommons.org/licenses/by-sa/3.0/fr/>

