Aide-mémoire LibreOffice

LibreOffice Basic
Les bases

Débutant

v. 2.0 – 06/04/2021

Rédigé avec LibreOffice v. 7.0.5 – Plateforme : Toutes

Généralités

☞ Le temps de développement : Coder 20 % – **Maintenir 80 %** → rédigez proprement !

Nommer les entités

Variables, constantes, subs, fonctions, modules, bibliothèques doivent être identifiées.
Caractères permis : lettres non accentuées, chiffres, souligné (_ ou « tiret du 8 »).

☞ Un nom ne peut pas commencer par un chiffre ni contenir d'espace.
⚠ N'utilisez pas de mots-clefs du langage pour nommer les entités !

Des noms **lisibles**	`CamelCase, Intercaler_des_separateurs`
Des noms **explicites**	`EstCellule(), EnregistrerClasseur()`

Commentaires

`'` (apostrophe) ou `REM`. Ce qui les suit est un commentaire.

☞ Les commentaires sont aussi importants que le code ! Seulement pour la **ligne courante**.

Indenter le code

Lecture facilitée. Indentez chaque niveau de bloc de code : **Espace** / **Tabulation**.

Plusieurs instructions sur la même ligne

Utilisez le caractère « `:` » (deux-points) pour séparer des instructions sur une ligne :

```
Dim MaVar As Integer : MaVar = 1234
```

Continuer une instruction à la ligne suivante

Deux **derniers** caractères de la ligne : _ (espace + souligné).

Variables

Variable : emplacement mémoire. Son contenu peut être modifié au cours de l'exécution.

⚠ Par défaut, la déclaration des variables n'est pas obligatoire, mais cette pratique est **dangereuse** (doublons en cas de fautes de frappe).
`Option Explicit` en début de **module** oblige à déclarer les variables.

Déclarer des variables

Variables simples

`Dim MaVar As UnType`	Ex : `Dim MonTexte As String`
`Dim A As Byte, B As String`	(déclarations multiples)
`Dim MaVar As Integer : MaVar = 123`	(déclaration + initialisation)

Variables tableaux

☞ Voir A-M n°2B « Types structurés »

Affectation de variables non objet

```
MaVar = UneValeur
```

☞ Si `UneValeur` n'est **pas** de même type que `MaVar`, il y a souvent *transtypage* automatique. Il est préférable de transtyper explicitement (fonctions `CXxx()`. Voir A-M n°5).

Création/Affectation de variables objet

☞ Voir A-M n°2B « Les types structurés »

Portée (visibilité) des variables

La déclaration…	donne visibilité…
`Dim MaVar As UnType`	Dans le sous-programme ou module courant.
`Static MaVar As UnType`	Dans le sous-programme. ☞ Valeur persistante entre appels.
`Private MaVar As UnType`	Dans le module courant.
`Public MaVar As UnType`	Dans la bibliothèque courante.
`Global MaVar As UnType`	Dans toutes les bibliothèques. ⚠ Valeur persistante entre exécutions !

Types

Spécifie les valeurs que peut porter une variable ou retourner une fonction.

Types simples prédéfinis

Type	Description	Val. init.
`Boolean`	Valeurs logiques `True` / `False` (Vrai/Faux). ☞ Peut être vu comme `False` = 0 ; `True` = autres entiers (ex : -1).	`False`
`Byte`	Nombres entiers (8 bits), de 0 à 255.	`0`
`Currency`	Nombres monétaires (4 décimales).	`0.0000`
`(Decimal)`	Sous-type de `Variant`, obtenu par `CDec(string)` 28 chiffres (p.entière + p.décimale). De 1×10^{-28} à $7{,}9 \times 10^{28}$. ☞ Utilisé avec les fonctions de l'API qui utilisent des entiers 64bits. ⚠ Les débordements n'entraînent pas d'erreur d'exécution.	`n/a`
`Date`	Dates et heures. En réalité : nombres réels. La date de référence (`0.0`) est le `30/12/1899` à `00:00`.	`0.0`
`Double`	Nombres réels (64 bits).	`0.0`
`Integer`	Nombres entiers (16 bits), de `-32 768` à `+32 767`	`0`
`Long`	Nombres entiers (32 bits), de `-2 147 483 648` à `+2 147 483 647`	`0`
`Object`	Objets. Permet de manipuler les objets LibreOffice.	`Null`
`Single`	Nombres réels (32 bits).	`0.0`
`String`	Texte (0 à 65 545 caractères). Les chaînes sont délimitées par des `"`.	`""`
`Variant`	N'importe quel type, y compris objet.	`Empty`

Voir aussi le tableau de Compatibilité des types principaux.

☞ Si type non précisé : `Variant` implicite.
☞ Les valeurs entières peuvent être en base hexadécimale.
Préfixez ces valeurs avec &H. Ex : `&HFF` (décimal 255). Utile pour les couleurs.
☞ Affectez des valeurs initiales plutôt que compter sur des initialisations implicites.
⚠ Attention aux erreurs d'arrondi dans les calculs sur des nombres réels !

Tableaux, types personnalisés, Collections et Objets

☞ Voir A-M n°2B « Types structurés »

Empty, Null et Nothing

`Empty`	Variable non encore initialisée. Assignation `Empty` possible.
`Null`	Contenu non valide. Assignation `Null` possible.
`Nothing`	(objets seulement) Pas/plus de référence à l'objet. Assignation possible.

Fonctions

`IsEmpty(UneVar)`	La variable est vide.
`IsNull(UnObjet)`	La donnée n'est pas utilisable.

Opérateurs

Booléens

`Not` Non `And` Et `Or` Ou (inclusif) `Xor` Ou exclusif

Comparaison (renvoient `True` ou `False`)

=	Égal strictement	<	Inférieur strictement	<=	Inférieur ou égal
<>	Différent	>	Supérieur strictement	>=	Supérieur ou égal

⚠ Attention aux comparaisons de nombres réels (erreurs d'arrondi) !

Numériques

+	Addition	-	Soustraction
*	Multiplication	/	Division
\	Division entière	`Mod`	Modulo (reste de la division entière)
^	Élévation à la puissance		

Texte

&	Concaténation (fusion) de chaînes (« + » est possible ; à éviter).

Constantes

Constante : emplacement mémoire, valeur **fixe** (immuable tout au long de l'exécution).

Déclarer des constantes

```
Const UNE_CONST = Valeur
```

☞ `Valeur` doit être d'un type simple, non tableau, non objet.

Nommer les constantes

Il est habituel de nommer les constantes en toutes majuscules.

Portée (visibilité) des constantes

La déclaration…	donne visibilité…
`Const MACONST = Valeur`	Dans le sous-programme ou module courant.
`Public MACONST = Valeur`	Dans la bibliothèque courante.
`Global MACONST = Valeur`	Dans toutes les bibliothèques.

Chemins des fichiers

Pour être multi plate-formes, les chemins des fichiers peuvent être exprimés au format URL :
`file:///support/chemin/vers/fichier.txt`.
Deux fonctions existent pour passer du format URL au format natif du système :

De natif à URL	`NomURL = ConvertToURL(NomFichierNatif)`
De URL à natif	`Nom = ConvertFromURL(NomFichierURL)`
Exemple (Windows)	Nom natif : `C:\MonRepertoire\Fichier.odt` Nom URL : `file:///C:/MonRepertoire/Fichier.odt`

☞ **Le format URL**
Un URL (*Uniform Resource Locator*) indique l'adresse d'un document ou d'un serveur.
Structure générale d'un URL : `service://nom_hôte:port/chemin/page#marque`

Sous-programmes

⚠ Correspondance arguments ↔ paramètres, en nombre, en position et en type.
☞ Sortie de sous-programme prématurée : `Exit Sub`, `Exit Function`

Sub

Exécute une action.

☞ Conseil de nommage : verbe à l'infinitif : `FaireXxx`, `LireXxx`, etc.

Déclaration
Structure

```
Sub NomDeLaSub(parametres)
  'instructions
End Sub
```

Utilisation `NomDeLaSub(arguments)`. Si pas d'argument : `NomDeLaSub()`

Function

Renvoie une valeur.

☞ Conseil de nommage : verbe à l'indicatif : `LisXxx()`, `EstXxx()`, etc.

Déclaration
Structure

```
Function NomFonction(parametres) As UnType
  'instructions
  'quelque part, définir la valeur de retour :
  NomFonction = UneValeur
End Function
```

Utilisation

```
UneVar = NomFonction(arguments)
```

Si pas d'argument : `UneVar = NomFonction()`

☞ Une `Function` peut être appelée comme une Sub (sans lire la valeur de retour).

Paramètres

Paramètre valeur attendue selon déclaration du sous-programme.
Argument valeur réellement passée par le programme appelant.

Utilisation

```
Ex:  MaSub(ByRef Param As Long, ByVal AutreParam As Long, _
         Optional ByRef UnParam As Object)
```

`ByRef`
Par référence (défaut). Le paramètre reçu **pointe** vers l'argument passé .
⚠ Toute modification de la valeur d'un paramètre `ByRef` au sein de l'appelé est répercutée dans l'appelant.

`ByVal`
Par valeur. Le paramètre est une **copie** de l'argument passé.
☞ Les modifications de valeur restent locales à l'appelé.

`Optional`
Paramètre **facultatif**. Cumulable avec `ByRef`/`ByVal`.
☞ Tester son absence avec `If IsMissing(UnParam) Then …`
L'identifiant `UnParam` reste utilisable dans le sous-programme.
Donner une valeur par défaut à un paramètre optionnel :

```
If IsMissing(UnParam) Then UnParam = UneValeur
```

Structures de contrôle

Boucles

Répéter une série d'instructions selon une condition.

☞ Sortie de boucle prématurée possible par `Exit For` ou `Exit Do` selon le cas.

For ... Next

Pour chaque valeur du compteur ...

```
For i = Debut To Fin [Step
    Increment]
  'instructions
Next i
```

Il faut connaître les bornes du compteur.
Par défaut, l'incrément `Step` est de `1`.

☞ Les compteurs sont souvent `i`, `j`, `k`, etc.

☠ Le compteur ne doit **jamais** être modifié par des instructions de la boucle !

For Each ... Next

Pour chaque élément ...

```
For Each element In UnObjet
  'faire qqch avec element
Next element
```

Ne nécessite pas de connaître le nombre d'éléments.
`element` doit être d'un type compatible.

Do While ... Loop

Faire ... Tant que

```
Do While Condition
  'instructions
Loop
```

`While` : `Condition` évaluée en **premier**.

☠ Attention aux boucles infinies (`Condition` jamais réalisée)

ou aussi...

```
While Condition
  'instructions
Wend
```

☞ *Ancienne syntaxe autorisée pour compatibilité.*
Ne supporte pas `Exit` : **à éviter !**

Do Loop ... Until

Faire ... Jusqu'à

```
Do
  'instructions
Loop Until Condition
```

`Until` : `Condition` évaluée en **dernier**.

☠ Attention aux boucles infinies (`Condition` jamais réalisée)

Tests conditionnels

Branchement, aiguillage qui permet d'agir en fonction d'un état, d'une situation.

If Then (une ligne)

```
If Condition Then UneInstruction
```

Si `Condition` Alors ...

If Then [Else] (multiligne)

```
If Condition Then
  'InstructionsAlors
Else
  'InstructionsSinon
End If
```

Si `Condition` Alors ... Sinon ...
`Else` est facultative. `End If` est obligatoire.

If ElseIf

```
If Condition Then
  'InstructionsAlors1
ElseIf AutreCondition Then
  'InstructionsAlors2
Else
  'InstructionsSinon
End If
```

Si `Condition` Alors ... SinonSi ... Sinon ...
Permet d'éviter des `If` imbriqués multiples.

Select

```
Select Case UneVariable
  Case Valeur : FaireCa()
  Case UneValeur
    'instructions pour UneValeur
  Case Val1, Val2 To Val3
    'instructions pour les valeurs
  Case Else
    'instructions pour les autres cas
End Select
```

Choisir parmi plusieurs possibilités pour la valeur de `UneVariable`.

Charger une bibliothèque de code

Par mesure de lisibilité, organisez votre code en plusieurs **bibliothèques** (A-M n°1).

☞ Seule la bibliothèque de code **Standard** est chargée à l'ouverture d'un document. Les autres doivent être chargées explicitement pour utiliser leur code.

☠ Noms des bibliothèques : respectez la casse !

Charger depuis un conteneur local (document)

Vérifier l'existence
```
Existe = BasicLibraries.hasByName("MaBibli")
```
Charger
```
BasicLibraries.loadLibrary("MaBibli")
```

Charger depuis un conteneur global

Idem mais `BasicLibraries` est remplacé par `GlobalScope.BasicLibraries`.

☠ Attention aux **collisions** d'identifiants entre bibliothèques ! Vous pouvez qualifier les noms sous la forme : `conteneur.bibliothèque.module.nom` (en tout ou en partie).
Ex : `GlobalScope.Tools.Strings.ClearMultiDimArray(MonTablo, 3)`

Appeler la commande associée à un menu LibreOffice

Principe

Utiliser le `Dispatcher`, associé à la commande de menu UNO voulue.

Connaître les commandes de menus UNO

Listes de commandes de menus UNO : voir fichiers `menubar.xml` dans le répertoire d'installation de LibreOffice (selon OS), sous `share/config/soffice.cfg/modules`. Sous-répertoire `menubar` du module voulu (ex : `sglobal/menubar/menubar.xml`, etc.).
Les commandes commencent toutes par `.uno:`
Ex : `".uno:Open"` (**Fichier > Ouvrir**), `".uno:OptionsTreeDialog"` (**Outils > Options**), etc.

Squelette du programme à exécuter

```
Dim Frame As Variant : Frame  = ThisComponent.CurrentController.Frame
Dim Dispatch As Object
Dim Args() As Variant       'contenu dépendant du contexte
Dim UnoCmd As String : UnoCmd = 'la commande UNO à exécuter
Dispatch = createUnoService("com.sun.star.frame.DispatchHelper")
Dispatch.executeDispatch(Frame, UnoCmd, "", 0, Args())
```

Exemples

(seules les parties modifiées sont montrées)

Ex1. Appeler l'aperçu avant impression

```
Dispatch.executeDispatch(Frame, ".uno:PrintPreview", "", 0, Args())
```

Ex2. Afficher/masquer le volet latéral

```
Dim Args(0) As New com.sun.star.beans.PropertyValue
Args(0).Name  = "Sidebar"
Args(0).Value = True        'ou False selon objectif
Dispatch.executeDispatch(Frame, ".uno:Sidebar", "", 0, Args())
```

Gérer les erreurs

En Basic la gestion des erreurs repose sur :
- des instructions `On Error Xxx` (et `On Local Error Xxx`) : intercepter les erreurs ;
- des fonctions `Err`, `Erl` et `Error` : infos sur **dernière** erreur rencontrée.

Fonctions d'information sur une erreur

`Err`	Le code de l'erreur intervenue. ☞ Le code d'erreur `0` (zéro) = « pas d'erreur ». Utilisez `If Err Then ...` pour tester l'existence d'une erreur.
`Error`	Le texte du message descriptif de l'erreur.
`Erl`	Le numéro de ligne où l'erreur s'est produite.

On Error – Interception globale des erreurs

☠ L'interception des erreurs par `OnError` est active **tant qu'elle n'a pas été annulée**.

`On Error Goto MonEtiquette`	Active l'interception des erreurs. En cas d'erreur, l'exécution se poursuit à `MonEtiquette:` ☞ Dans le corps du programme, définir l'étiquette `MonEtiquette:` (attention au « deux-points »).
`On Error Resume Next`	Active l'interception des erreurs. En cas d'erreur, l'exécution se poursuit à la prochaine instruction.
`On Error Goto 0`	Annule l'interception des erreurs.

On Local Error – Interception locale des erreurs

Dans un sous-programme, on peut préférer `On Local Error Xxx` (même syntaxe) car elle ne nécessite pas le recours à `On Error Goto 0` pour annuler l'interception des erreurs : l'annulation est automatique en quittant la Sub ou la Function.

☞ `On Local Error Goto Xxx` **a préséance** sur un `On Error Goto Xxx` en place.

Méthodes de lancement d'une macro

▼ Méthode	LibreOffice	Type de document	Document courant
Par barre d'outils		●	●
Par menu		●	●
Par raccourci	●	●	
Par événement	●		●

Compatibilité des types principaux

Possibilités d'affectation entre variables ou valeurs retours de fonctions.

Cible ▶ / Source ▼	Integer	Long	Single	Double	Currency	Decimal	Date	String	Object	Boolean	Variant	Byte
Integer	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
Long	!	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	!
Single	O!	O!	✓	✓	✓	✓	✓	✓	✗	!	✓	O!
Double	O!	O!	O!	✓	✓	✓	✓	✓	✗	!	✓	O!
Currency	O!	O!	!	✓	✓	✓	O	✓	✗	!	✓	O!
Decimal	O!	O!	O!	O	O!	✓	O	✓	✗	O!	✓	O!
Date	O!	O!	!	✓	✓	O!	✓	✓	✗	!	✓	O!
String	O!	O!	O!	O!	O!	✓	O!	✓	✗	O!	✓	O!
Object	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
Boolean	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
Variant	O!	O!	O!	O!	O!	✓	O!	O!	✓	O!	✓	O!
Byte	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓

Compatibilité

✓ Compatible O Risque de perte ! Risque de débordement ✗ Non compatible

Lecture

- Le contenu d'une variable **source** de type `Double` peut être assigné à une variable **cible** des types `Double`, `Currency`, `Date`, et `Variant`, sans perte.
- Une variable **cible** de type `Double` peut recevoir sans perte des données des types `Integer`, `Long`, `Single`, `Double`, `Date` et `Byte`.

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
2.0	06/04/2021	Modif. de la numérotation (2A) ; refonte (certains types déplacés vers A-M n°2B) ; révision de la mise en forme.