Guía de referencia LibreOffice

LibreOffice Basic n.º 2
Las bases

Principiante

v. 2.0 – 20/04/2019

Redactado con LibreOffice v. 5.3.3 – Plataforma : Todas

Generalidades

☞ Tiempo de desarrollo : Escritura de Código 20 % – **Mantenimiento 80 %**

Nombrar las entidades

Variables, constantes, subs, funciones, módulos y bibliotecas deben estar identificadas.
Caracteres permitidos : letras no acentuadas, cifras, o guión bajo (_).

☞ Un nombre no puede empezar por una cifra ni contener espacios.

☠ No use palabras reservadas del lenguaje para nombrar las entidades !

Nombres **legibles** `«CamelCase» o Intercalar_separadores`
Nombres **explicitos** `LaCelda(), GuardarCarpeta()`

Comentarios

`'` (apóstrofo) o `REM`. Indican que lo siguiente es un comentario.

☞ Los comentarios son tan importantes como el código ☠ - Sólo abarcan una linea

Indentar el código

Para facilitar la lectura Indentar cada nivel de bloque de código: **Espacio** o **Tabulador**.

Varias instrucciones sobre la misma linea

Utilizar el carácter « **:** » (dos puntos) para separar instrucciones en una misma linea :

```
Dim MiVar As Integer : MiVar = 123
```

Continuar una instrucción en la linea siguiente

Los dos **últimos** caracteres de la linea: _ (espacio + subrayado).

Variables

Variable : se sitúa en memoria, su contenido se puede modificar durante la ejecución.

☠ Por defecto, la declaración de variables no es obligatoria pero no declararlas es una práctica **peligrosa** (pueden crearse duplicados en caso se errores tipográficos).
Con `«Option Explicit»` al inicio del módulo se obliga a declarar las variables.

Declaración de variables

Variables simples

`Dim MiVar As Un_Tipo`	Ej : `Dim MiTexto As String`
`Dim A As Byte, B As String`	(declaraciones múltiples)
`Dim MiVar As Integer : MiVar = 123`	(declaración + inicialización)

Variables matriz

☞ Vea la *Guía de referencia nº9 Tipos estructurados*

Asignación de variables no objeto

```
MiVar = UnValor
```

☞ Si un valor no es del mismo tipo que `MyVar`, se hace una conversión automática.
Es preferible convertir el tipo explicitamente (funciones `Cxxx()`. Vea la *Guía ref. n°5*.

Creación/Asignación de variables objeto

☞ Vea la Guía de referencia nº9 « Tipos estructurados »

Alcance (visibilidad) de las variables

Declaración…	Visibilidad ...
`Dim MiVar As UnTipo`	En la subrutina o módulo actual.
`Static MiVar As UnTipo`	En la subrutina.
	☞ Valor persistente entre llamadas.
`Private MiVar As UnTipo`	En el módulo actual.
`Public MiVar As UnTipo`	En la biblioteca actual.
`Global MiVar As UnTipo`	En todas las bibliotecas.
	☠ Valor persistente entre ejecuciones !

Tipos

Especifican los valores que puede tener una variable o retorno de una función.

Tipos simples predefinidos

Tipo	Descripción	Valor inicial
`Boolean`	Valores lógicos `True` / `False` (Verdadero/Falso)	`False`
	☞ Puede verse como `False` = 0 ; `True` = 1 (u otros enteros)	
`Byte`	Números enteros (8 bits), de 0 a 255.	`0`
`Currency`	Números moneda (4 decimales).	`0.0000`
`(Decimal)`	Sub-tipo de variant, obtenido por `CDec(string)` 28 cifras (p.entera + p.decimal). De `1 x 10`$^{-28}$ a `7.9 x 10`^{28} (precisión máxima 28 decimales).	`n/a`
	☞ Utilizado con las funciones de la API (usan enteros de 64bits).	
	☠ Los desbordamientos no provocan error en tiempo de ejecución.	
`Date`	Fechas y horas. En realidad : números reales. La fecha de referencia (`0.0`) es el `30/12/1899` a las `00:00`.	`0.0`
`Double`	Números reales (64 bits).	`0.0`
`Integer`	Números enteros (16 bits), de `-32.768` a `+32.767`	`0`
`Long`	Enteros de (32 bits), de `-2.147.483.648` a `+ 2.147.483.647`	`0`
`Object`	Objetos. Permite la manipulación de los objetos de LibreOffice.	`Null`
`Single`	Números reales (32 bits).	`0.0`
`String`	Texto (0 a 65.545 caracteres). Las cadenas se delimitan con `"` (comillas dobles normales (no tipográficas).	`""`
`Variant`	No importa el tipo, incluye también el objeto.	`Empty`

Ver también la tabla de Compatibilidad de los tipos principales.

☞ Si un tipo no se declara : obtendrá un tipo `Variant` de manera implícita.

☞ Los valores enteros pueden estar en base hexadecimal.
Prefijar estos valores con `&H`. Ej : `&HFF` (décimal `255`). Util para los colores.

☞ Asigne valores iniciales en lugar de contar con su iniciación implícita.

☠ Cuidado con los errores de redondeo en los cálculos con numeros reales

Matrices, tipos personalizados, Colecciones y Objetos

☞ Ver Recordatorio nº9 « Tipos estructurados »

Empty, Null y Nothing

`Empty`	Variable no inicializada. Posible asignacion `Empty`.
`Null`	Contenido no válido. Posible asignación `Null`.
`Nothing`	(sólo objetos) Sin referencia al objeto. Asignación posible.

Funciones

`IsEmpty(UnaVariable)`	La variable está vacía.
`IsNull(UnObjeto)`	El dato no se puede utilizar (no hay datos).

Operadores

Booleanos

`Not` No `And` Y `Or` O (inclusivo) `Xor` O exclusivo

Comparadores (devuelven `True` o `False`)

`=`	Estrictamente igual	`<`	Estrictamente menor	`<=`	Menor o igual
`<>`	Distinto de	`>`	Estrictamente mayor	`>=`	Mayor o igual

☠ Cuidado con las comparaciones de números reales (errores en redondeos) !

Numéricos

`+`	Suma	`-`	Resta
`*`	Multiplicación	`/`	División
`\`	División entera	`Mod`	Módulo (resto de división entera)
`^`	Élevación a una potencia		

Texto

`&`	Concatenación (fusion) de cadenas , también se puede usar (« **+** » pero mejor evitarlo porque es la suma y con números puede dar un resultado no deseado).

Constantes

Constante : se sitúan en memoria, valor **fijo** (inmutable durante toda la ejecución).

Declación de constantes

```
Const UNA_CONSTANTE = UnValor
```

☞ `UnValor` debe ser de un tipo simple, no puede ser matriz ni objeto

Nombre de las constantes

Es habitual nombrar las constantes con todos los caracteres en mayúsculas.

Alcance (visibilidad) de las constantes

Declaración…	Visibilidad…
`Const MICONST = UnValor`	En la subrutian o módulo actual.
`Public MICONST = UnValor`	En la biblioteca actual.
`Global MICONST = UnValor`	En todas las bibliotecas.

Rutas de los archivos

Al ser multi plataforma, las rutas a los archivos pueden visualizarse en formato nativo o en formato URL : `file:///soporte/ruta/al/archivo.txt`. Pero en las instrucciones se debe usar el formato URL
Hay dos funciones para pasar de un formato a otro :

De nativo a URL `NombreURL = ConvertToURL(NombreArchivoNativo)`
De URL a nativo `Nombre = ConvertFromURL(NombreArchivoURL)`
Ejemplo (Windows)
Nombre nativo : `C:\MiDirectorio\Archivo.odt`
Nobre URL : file:///C:/MiDirectorio/Archivo.odt

☞ **El formato URL**
Un URL (*Uniform Resource Locator*) indica la dirección de un documento o servidor.
Estructura general : `servicio://anfitrión:puerto/ruta/página#marca`
(según el caso, ciertos elementos pueden no estar presentes). Un URL puede ser una dirección FTP, de internet (HTTP), de archivo o de correo electrónico.

Subrutinas

☠ Respete la concordancia de argumentos ↔ en número, en orden y en tipo.

☞ Salida prematura de subrutina o función : `Exit Sub`, `Exit Function`

Sub

Ejecuta una acción.

☆ Sugerencia de Nombre : verbo en infinitivo ; Hacer`Xxx`, `LeerXxx`, etc.

Declaración `Sub NombreDeLaSub(parámetros)`

Estructura
```
Sub NombreDeLaSub(parámetros)
  'instrucciones
End Sub
```

Uso `NombreDeSub(argumentos)`. Sin argumentos : `NombreDeLaSub()`

Function

Devuelve un valor.

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

Declaración `Function NombreFuncion(parámetros) As UnTipo`

Estructura
```
Function NombreFuncion(parámetros) As UnTipo
  'instrucciones
  'definir el valor de retorno en algún punto:
  NombreFuncion = UnValor
End Function
```

Uso
```
UnaVar = NombreFuncion(argumentos)
```
Si no hay argumentos : `UnaVar = NombreFuncion()`

☞ S puede llamar a una Función como una subrutina (sin usar su valor de retorno).

Parámetros

Parametro Valor esperado según la declaración de la subrutina
Argumento Valor realmente pasado por la llamada desde la subrutina

Uso

```
Ej : MiSub(ByRef Param As Long, ByVal OtroParam As Long, _
       Optional ByRef UnParam As Object)
```

`ByRef` **Por referencia** (por defecto). El parámetro recibido **apunta** al argumento pasado por la llamada

☠ Toda modificación del valor de un parámetro `ByRef` dentro de la llamada repercute en la subrutina que hace la llamada

`ByVal` **Por valor**. El parámetro es una **copia** del argumento pasado

☞ Las modificaciones de valor se quedan dentro de la subrutina que hace la llamada.

`Optional` Parámetro **opcional.**

☞ Comprobar su ausencia con `If IsMissing(UnParam) Then …`
El Identificador siempre es utilizable dentro de la subrutina

Dar un valor por defecto a un parámetro opcional :
`If IsMissing(UnParam) Then UnParam = UnValor`

Estructuras de control

Bucles

Repetición de una serie de instrucciones según una condición.

☞ Es posible salir del bucle prematuramente mediante `Exit For` o `Exit Do`

For … Next (Para cada valor del contador …)

```
For i = inicio To Fin [paso de incremento]
  'instrucciones
Next i
```

Es necesario saber los límites del contador.
Por defecto el paso de incremento es `1`.

☞ Los contadores suelen ser `i`, `j`, `k`, etc.

☠ El contador no debe ser **nunca** modificado por las instrucciones del bucle !

For Each … Next (Por cada elemento …)

```
For Each elemento In UnObjeto
  'hacer algo con el elemento
Next
```

No es necesario saber el número de elementos.
`elemento` debe ser un tipo compatible.

Do While … Loop (Hacer … Mientras que)

```
Do While Condición
  'instrucciones
Loop
```

`Condición` evaluada al **principio**.

☠ Cuidado con los bucles infinitos (`Condición` que nunca se cumple)

o también…

```
While Condición
  'instrucciones
Wend
```

☞ *Sintaxis antigua, autorizada por compatibilidad.*
No acepta `Exit` : **Evite su uso !**

Do Loop ... Until (Hacer … hasta que)

```
Do
  'instrucciones
Loop Until Condición
```

`Condición evaluada al` **final.**

☠ Cuidado con los bucles infinitos (`Condición` que nunca se cumple)

Comprobaciones condicionales

Derivación que permite actuar según un estado o una situación.

If (solo)

```
If Condición Then UnaInstrucción
```

Si se cumple la condición, Entonces …

If Then [Else]

```
If Condición Then
  'Instrucciones si se cumple
Else
'Instrucciones Si no se cumple
End If
```

Si se cumple la condicíón, Entonces …
Sino …
(La condición **`Else`** es facultativa).

If Elself

```
If Condición Then
  'Instrucciones1
ElseIf OtraCondición Then
  'Instrucciones2
Else
  'Instrucciones en otro caso
End If
```

Si se cumple la condición, Entonces...,
Sino … Entonces …
Sino …
Permite evitar la anidación de múltiples `If`

Select

```
Select Case UnaVariable
  Case Valor : HacerEsto()
  Case UnValor
    'instrucciones para UnValor
  Case Val1, Val2 To Val3
    'instrucciones para varios valores
  Case Else
    'instrucciones para otros casos
End Select
```

Elección entre varias posibilidades en función del valor de "UnaVariable."

Cargar una biblioteca de código

Para una mejor lectura, organice su código en varias **bibliotecas** (*Guía ref. n°1*).

☞ Al abrir un documento, sólo se carga la biblioteca de código **Standard**. Las otras deben ser cargadas explicitamente para usar su código.

☠ Los nombres de las bibliotecas son sensibles a las Mayúsculas !

Cargar desde un contenedor local (en el documento)

Verificar su existencia `If BasicLibraries.hasByName("Mibiblioteca")then`
Carga `BasicLibraries.loadLibrary("Mibiblioteca")`

Cargar desde un contenedor global (En el programa)

Igual, pero **`BasicLibraries`** se debe reemplazar por **`GlobalScope.BasicLibraries`**.

☠ Cuidado con las **colisiones** de indentificadores entre bibliotecas ! Puede calificar los nombres : `conteneudor.biblioteca.modulo.nombre` (parcial o totalmente).
Ej : `GlobalScope.Tools.Strings.ClearMultiDimArray(MiTabla, 3)`

Llamar a un comando asociado a un menú LibreOffice

Principio

Uso del `Dispatcher`, asociado al comando del menu UNO elegido.

Conocer los comandos de menú UNO

Hay un listado de comandos de menú UNO : ver archivos `menubar.xml` el el directorio de instalación de LibreOffice (según S.O.), en `share/config/soffice.cfg/modules`. Subdirectorio menubar del módulo (ej. : `sglobal/menubar/menubar.xml`, etc.).
Todos los comandos comienzan por `.uno:` Ej : `".uno:Open"` (Archivo ► Abrir), `".uno:OptionsTreeDialog"` (Herramientas ► Opciones), etc.

Esqueleto del programa a ejecutar

```
Dim Frame As Variant
Dim Dispatch As Object
Dim Args() As Variant       'contenido dependiente del contexto
Dim UnoCmd As String
Frame  = ThisComponent.CurrentController.Frame
UnoCmd = 'El comando UNO a ejecutar (abajo)
Dispatch = createUnoService("com.sun.star.frame.DispatchHelper")
Dispatch.executeDispatch(Frame, UnoCmd, "", 0, Args())
```

Ejemplos

(sólo se muesran las partes modificadas)

Ej.1 Llamar a la vista previa de Impresión

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

Ej.2 Mostrar/ocultar la barra lateral

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

Gestión de errores

En Basic la gestión de errores se basa en :
- las instrucciones `On Error Xxx` (y `On Local Error Xxx`) : interceptan los errores;
- las funciones `Err`, `Erl` et `Error` : informan del **último** error encontrado.

Funciones de información de un error

`Err`	El código del error intervenido. ☞ El código de error `0` (cero) = « sin error ». Utilice : `If Err Then …` para comprobar existencia de error.
`Error`	El texto del mensaje descriptivo del error.
`Erl`	El número de linea donde se produjo el error.

On Error – Interceptación global de errores

☠ La interceptación de errores con `OnError` **queda activa hasta que se anula**.

`On Error Goto MiEtiqueta`	Activa la interceptación de errores. En caso de error la ejecución continua en `MiEtiqueta:` ☞ En el cuerpo del programa se debe definir la etiqueta `MiEtiqueta:` (usando « dos puntos »).
`On Error Resume Next`	Activa la interceptación de errores. En caso de error la ejecución continua en la siguiente instrucción.
`On Error Goto 0`	Anula la interceptación de errores.

On Local Error – Interceptación local de errores

En una subrutina, se puede preferir el uso de `On Local Error Xxx` (misma sintaxis) puesto que no necesita recurrir a `On Error Goto 0` para anular la interceptación de errores : la anulación es automática al salir de la rutina o función.

☞ `On Local Error Goto Xxx` **tiene precedencia** sobre `On Error Goto Xxx` .

Métodos de ejecución de una macro

▼ Método	LibreOffice	Tipo de documento	Documento abierto
Barra de herramientas		●	●
Menú		●	●
Por atajo	●	●	
Por evento	●		●

Compatibilidad de los tipos principales

Destino ► / Origen ▼	Integer	Long	Single	Double	Currency	Decimal	Date	String	Object	Boolean	Variant	Byte
Integer	●	●	●	●	●	●	●	●	x	●	●	!
Long	!	●	●	●	●	●	●	●	x	●	●	!
Single	○!	○!	●	●	●	●	●	●	x	!	●	○!
Double	○!	○!	○!	●	●	●	●	●	x	!	●	○!
Currency	○!	○!	!	●	●	●	○	●	x	!	●	○!
Decimal	○!	○!	○!	○	○!	●	○	●	x	○!	●	○!
Date	○!	○!	!	●	●	○!	●	●	x	!	●	○!
String	○!	○!	○!	○!	○!	●	○!	●	x	○!	●	○!
Object	x	x	x	x	x	x	x	x	●	x	●	x
Boolean	●	●	●	●	●	●	●	●	x	●	●	○
Variant	○!	○!	○!	○!	○!	●	○!	○!	●	○!	●	○!
Byte	●	●	●	●	●	●	●	●	x	●	●	●

Compatibilidad

● Compatible ○ Posible pérdida ! Posible desbordamiento x incompatible

Lectura

- El contenido de una variable **origen** de tipo `Double` puede ser asignada a una variable **destino** de tipo `Double`, `Currency`, `Date`, o `Variant`, sin pérdida.
- Una variable **destino** de tipo doble puede recibir sin pérdida de datos los tipos `Integer`, `Long`, `Single`, `Double`, `Date` o `Byte`.

Creditos

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

Somos como enanos sentados sobre los hombros de gigantes. Si vemos más cosas y más lejanas que ellos, no es por la perspicacia de nuestra visión, ni por nuestra grandeza, sino porque son ellos los que nos elevan. (Bernard de Chartres [atribuido])

Historial

Versión	Fecha	Comentarios
2,0	20/04/20197	Rediseño (algunos tipos trasladados a *Guia ref. n ° 9*)
	05/04/2021	Traducción al español : B. Antonio Fernández

El documento original se puede obtener en la Wiki francesa de publicaciones de L.O.