LibreOffice

Fichas de referencia

LO Basic

Introducción

Este documento es una recopilación y adaptación de los documentos originales, en Francés, que se puede obtener en la página de Publicaciones en la Wiki de LibreOffice (fr).

Si bien el formato original de los documentos originales está pensado para ocupar el mínimo espacio, cada ficha en una hoja de papel impresa, aquí se ha cambiado el formato para que la visualización sea más fácil al consultar las fichas en formato electrónico (PDF o HTML).

También se pueden obtener las fichas en el formato reducido en español (aunque en una versión anterior) en la página de Publicaciones en la Wiki de LibreOffice (es)

N del T. - Al igual que su autor, espero que estas páginas sean útiles.

Créditos

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

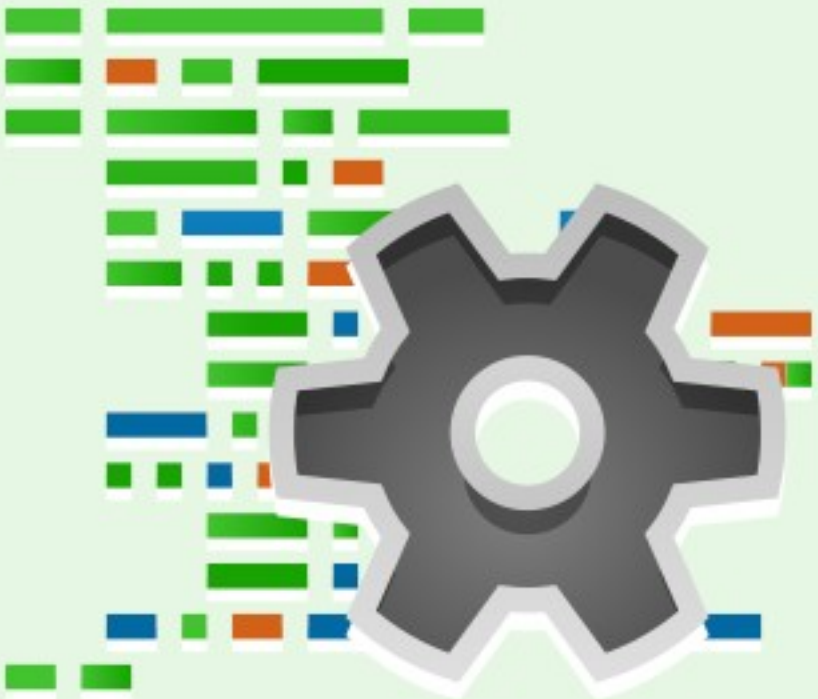
Traducción y adaptación: B. Antonio Fernández

Somos como enanos a hombros de gigantes. Si alcanzamos a ver más cosas y más lejanas que ellos, no es por la perspicacia de nuestra visión, ni por nuestra grandeza, sino porque son ellos quienes nos elevan. (atribuido a Bernard de Chartres)

Contenido

Ficha nº 1.
El EDI

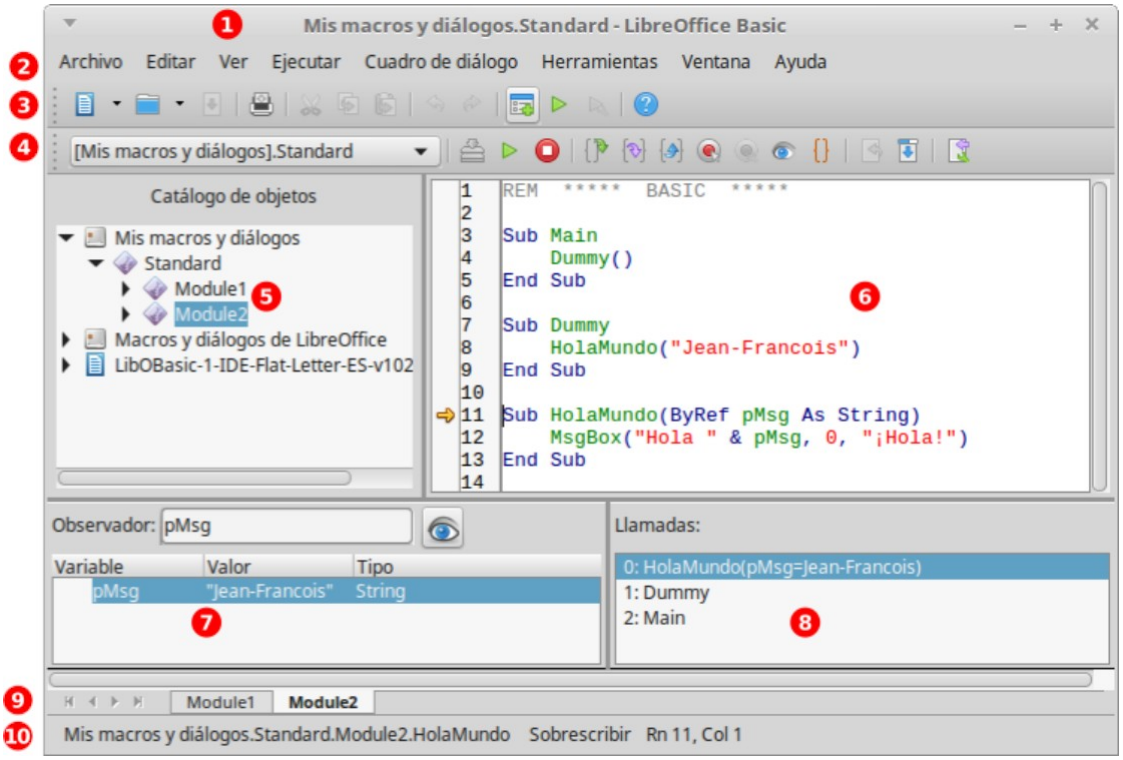
Entorno de desarrollo integrado - Nivel: Principiante



Acceso al EDI

El EDI (Entorno de desarrollo integrado) es un conjunto de herramientas para escribir y depurar código de macros, para acceder a la interfaz , vaya al menú **Herramientas > Macros > Editar macros** o bien a **Herramientas > Macros > Organizar macros > LibreOffice Basic** (Alt+F11), seleccione el módulo que se desee y pulse el botón *Editar*.

Figura 1: Interfaz del Entorno de desarrollo



La ventana del EDI se compone de 10 zonas:

(1)	Barra de Título	Nombre del contenedor y librería seleccionada.
(2)	Menú	Menú dedicado a la gestión de macros.
(3)	Barra de herramientas Estándar	Muestra herramientas comunes para organizar y editar el código
(4)	Barra de Macros	Herramientas para la edición y depuración de código.
(5)	Catálogo de objetos	Vista en árbol de los contenedores, muestra librerías, módulos y procedimientos.
(6)	Editor de código	Área principal para edición de código, con resaltado de sintaxis, gestión de puntos de interrupción y control de ejecución.
(7)	Panel Observador	Control de los contenidos de variables (observador).
(8)	Panel Llamadas	Muestra las llamadas a procedimientos y sus parámetros.
(9)	Barra Pestañas	Lista y gestiona los módulos de la librería en uso.
(10)	Barra de Estado	Estado en tiempo de ejecución.

Los paneles principales (*Catálogo* ❺, *Editor* ❻, *Observador* ❼ y *Llamadas* ❽) se pueden desanclar.

F6 permite pasar de una zona otra.

Barras de herramientas

Si alguna barra no está visible: vaya al menú **Ver > Barras de herramientas**.

Los iconos pueden variar en función del sistema operativo y tema de iconos.

Estándar

Dos botones destacables:

Seleccionar macro	Abre el diálogo *Macros de Basic*.
Módulos	Abre el diálogo *Organizador de macros BASIC*.

Macros

Biblioteca actual	Selección de la biblioteca que se usará.
Compilar	Verifica la sintaxis.
Ejecutar F5	Ejecuta desde el principio el procedimiento en el que está el cursor.
Detener ⇧+F5	Detiene la ejecución en curso.
Pasar al siguiente ⇧+F8	Ejecución paso a paso (La marca de ejecución no sale del procedimiento aunque haya llamadas a otro procedimiento)
Entrar en el proceso F8	Ejecución paso a paso (La marca de ejecución sale del procedimiento cuando hay llamadas a otro procedimiento)
Salir del proceso Ctrl+⇧+F8	Ejecución paso a paso (la marca de ejecución vuelve al procedimiento que inició la llamada).
Act./Desac. punto de interrupción F9	Activa/Desactiva el punto de interrupción de la linea bajo el cursor.
Gestionar puntos de interrupción	Abre el diálogo Gestión de puntos de Interrupción.
Act./Des. inspección F7	Activa/Desactiva la inspección de variables.
Buscar paréntesis	
Importar Basic	Importa código fuente de un archivo *.bas
Exportar Basic	Exporta el código basic a un archivo *.bas
Importar diálogo	Importa el código de diálogo de un archivo *.xdl.

Cuadro de diálogo

Esta barra remplaza la barra anterior cuando se edita un diálogo.

Insertar controles	Muestra el Cuadro de herramientas para insertar controles
Importar diálogo	Importa el código de diálogo de un archivo *.xdl (XML).
Exportrar diálogo	Exporta el código de diálogo a un archivo *.xdl (XML).

Cuadro de herramientas

Esta barra aparece al crear o editar un diálogo, muestra el conjunto de controles que puede utilizar en sus diálogos. Sólo se mencionan tres botones destacables.

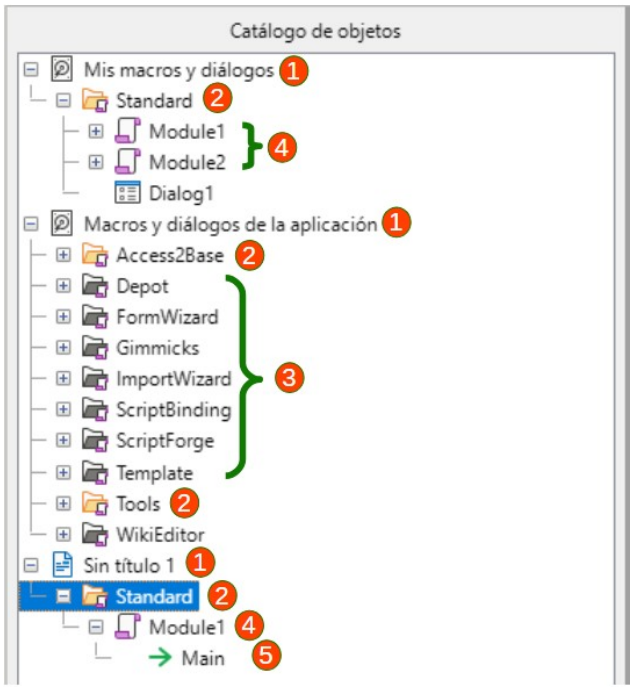
	Seleccionar elemento	Se entra en modo de selección.
	Gestionar idiomas	Permite crear diálogos multilíngues.
	Previsualizar el diálogo	Para comprobar el diálogo (Esc para salir).

Catálogo de objetos

Muestra los tres contenedores básicos y su contenido: bibliotecas, módulos y macros.

Contenedores  **(1)**

Figura 2: Catálogo de objetos



- **Mis macros y diálogos**: Específico del usuario, para todos los documentos. Solo accesible por el usuario de la sesión abierta.
- **Macros y diálogos de la aplicación**: Macros almacenadas dentro del contenedor global de LibreOffice. Como tal, pueden ser visualizadas y utilizadas por cualquiera, forman parte de la instalación de LibreOffice.
- **Sin título 1**: pertenecen al documento abierto.

Bibliotecas  **(2)**

En color = cargadas **(2)** , en gris = sin cargar **(3)**

- La biblioteca Standard: A excepción de *Macros y diálogos de la aplicación*, todo contenedor tiene una biblioteca *Standard*.

 La biblioteca *Standard* se carga siempre, al abrir el programa o el documento: No se puede eliminar, no se puede sobreescribir mediante la importación de código y no se pude cifrar.

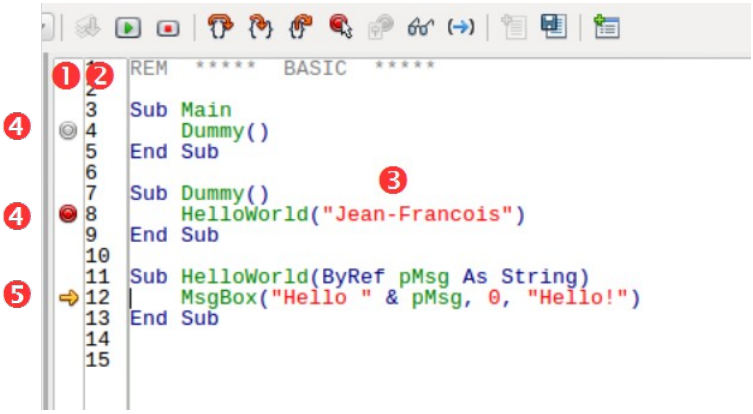
Módulos  **(4)**

Agrupan las **Subrutinas y funciones o procedimientos (5)**

Editor de código

Se utiliza en el desarrollo del código (con resaltado de sintaxis) y su depuración.

Figura 3: Editor de código



Elementos del panel de edición

(1)	Canal	Margen lateral en el que se colocan las marcas de punto de interrupción **(4)** y se muestra el punto de ejecución **(5)**.
(2)	Numeros de línea	Permiten una navegación más fácil.
(3)	Éditor	Para escribir el código Basic (con resaltado de sintaxis).

Sangría de líneas

Utilice las teclas ⇥ o ⇧+⇥ para aumentar/disminuír la sangría de líneas.

☞ Estos atajos sirven para aplicar sangria a varias lineas seleccionadas.

Coloreado de sintaxis

Los colores se pueden cambiar por unos preestablecidos en **Ver > Combinación de colores** o por otros más personalizados en **Herramientas > Opciones > LibreOffice > Apariencia** Sección **Personalizaciones, Editor de Basic**.

Mostrar/ocultar los números de línea

Muestre u oculte los números de línea mediante el menú **Ver > Números de renglón**.

☞ Para la gestión de puntos de interrupción y retomar la ejecución vea *Depuración*.

Ir al renglón

El atajo Ctrl+L abre un diálogo que permite desplazarse a la linea especificada.

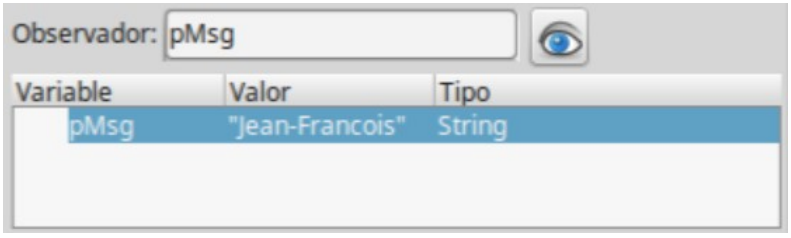
Panel de Observador

Permite visualizar el contenido de variables durante la ejecución (vea *Depuración*).

Tiene tres columnas (ajustables en anchura) :

- Nombre de la variable, Valor y Tipo

Figura 4: Panel de Observador



Añadir un observador

1. En el código seleccione la variable o constante para inspeccionar (puede ingresar su nombre en el campo *Observador* del panel)
2. haga clic en el botón *Activar observador* o pulse F7

Eliminar un observador

Seleccione el observador que desea retirar y haga clic en el botón *Eliminar observador* .

Panel Llamadas

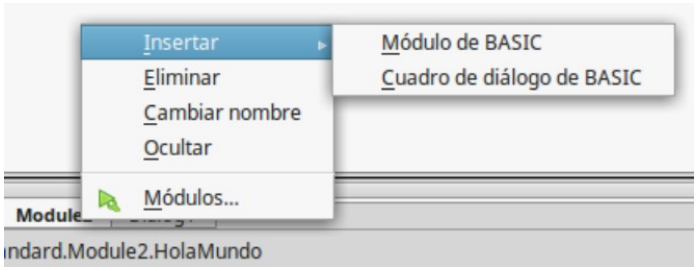
Figura 5: Panel Llamadas

Durante la ejecución paso a paso crea un historial de llamadas a los procedimientos, Estas van apareciendo según su ejecución (la más antigua abajo), y muestra los parámetros de ejecución. el procedimiento en curso se numera con 0 y el resto de manera consecutiva.

Barra de pestañas

Muestra la lista de los módulos y diálogos de la biblioteca en uso.

Figura 6: Menú en barra de pestañas



Los módulos se muestran en orden alfabético de izquierda a derecha. Haga clic derecho en la barra de pestañas para gestionar los módulos

Insertar > Módulo de Basic	Crea un nuevo módulo (*ModuleN*).
Insertar > Diálogo de Basic	Crea un nuevo diálogo Basic (*DialogN*).
Eliminar	Elimina el módulo o diálogo seleccionado.
Cambiar nombre	Renombra el módulo o diálogo seleccionado.
Ocultar	Oculta temporalmente el módulo o diálogo seleccionado.
Módulos	Abre el diálogo *Organizador de macros*.

Barra de Estado

De izquierda a derecha indica:

- Nombre completo del procedimiento donde se encuentra el cursor.
- El Número de linea (*Rn*) y columna (*Col*) donde se encuentra el cursor.
- Modo de edición del código (se cambia al modo de *Sobrescritura* al pulsar Ins).
- Deslizador y porcentaje de zum.

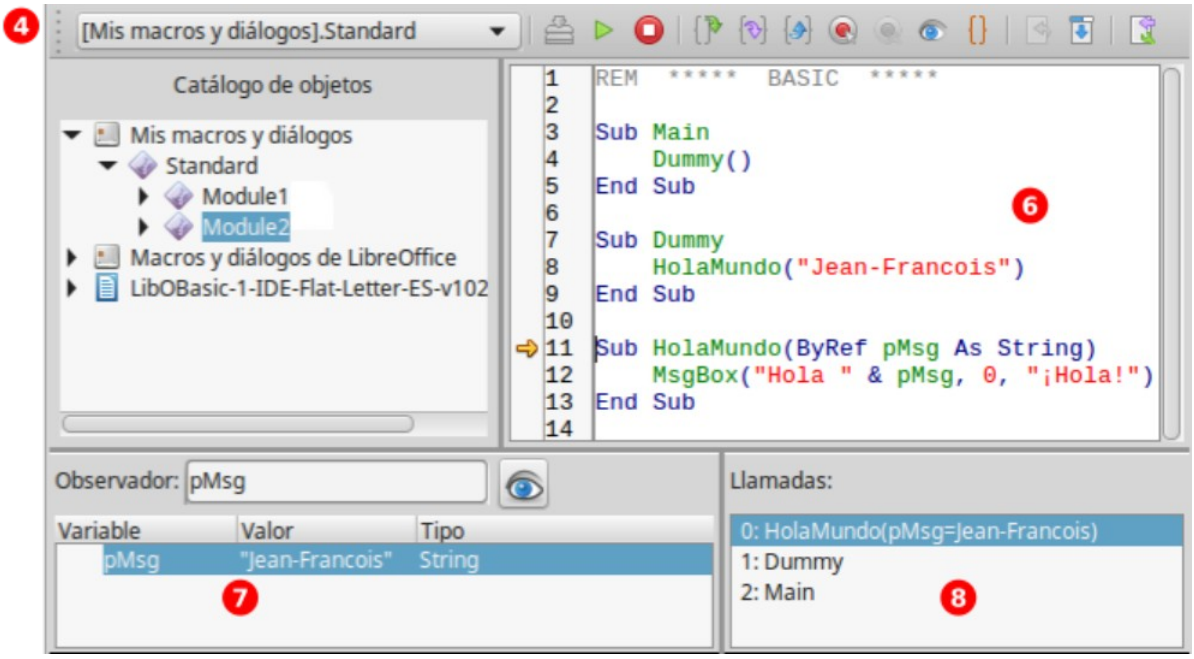
Depuración

Depuración es la comprobación de la ejecución de una macro

> Esta operación se efectúa mediante la ejecución de la parte elegida en modo paso a paso, mientras se observa el contexto de ejecución (Valores de variables) .

La depuración se apoya simultáneamente en los tres paneles: El Editor de código **(6)**, el panel Llamadas **(7)** y el panel Observador **(8)**. Se realiza mediante el control de los botones paso a paso (barra de herramientas *Macros* **(4)**) y los puntos de interrupción.

Figura 7: Depuración



Modos de ejecución

Cinco botones controlan la manera en que se ejecuta el código. La marca de ejecución → permite conocer su avance en el editor.

> Si ha iniciado la ejecución (en cualquier modo) cuando el cursor se encuentra en un procedimiento que necesita que se le pasen parámetros (no opcionales), recibirá el mensaje de error: *El argumento no es opcional*.

	Ejecutar ⇧ + F5	Ejecuta el procedimiento donde se encuentra el cursor. La ejecución se detiene al encontrar un punto de interrupción.
	Detener F5	Detiene la ejecución.
	Pasar al siguiente ⇧ + F8	Ejecución paso a paso (línea tras línea) sin entrar en otros procedimientos.
	Entrar en proceso F8	Ejecución paso a paso (saliendo a otros procedimientos cuando hay llamadas).
	Salir del proceso Ctrl + ⇧ + F8	Ejecución paso a paso volviendo al procedimiento que inició la llamada.

Gestionar puntos de interrupción

Punto de Interrupción: Marca donde se detiene la ejecución del código. Permite examinar el contexto de ejecución en ese momento (valores de los testigos).

> Durante la depuración se pueden realizar las siguientes acciones:

Añadir punto de interrupción

Doble clic en el canal al lado de la linea en que quiera insertar el punto o F9 (en la línea con el cursor).

Eliminar punto de interrupción

Doble clic en el canal sobre el punto de interrupción o (F9 en la línea).

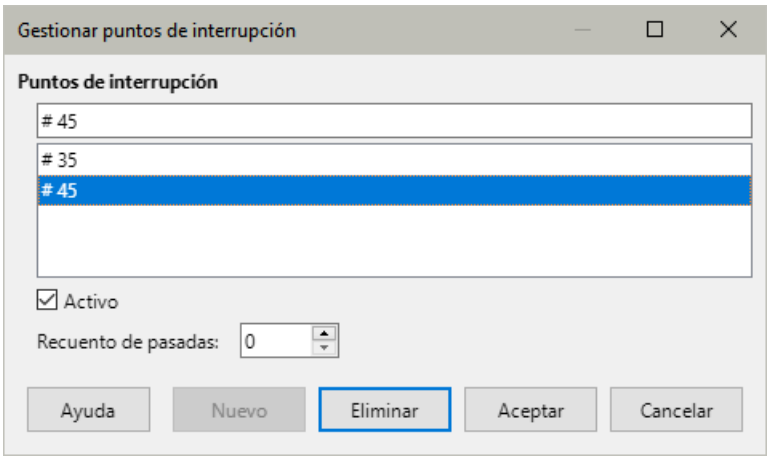
Desactivar/Activar punto de interrupción

Haga clic derecho en el canal sobre el punto de interrupción y seleccione *Activo*. El icono del punto de interrupción cambia entre activo e inactivo .

Diálogo Gestionar puntos de interrupción

Haga clic derecho sobre el cusor de ejecución o punto de interrupción existente y seleccione *Gestionar puntos de interrupción* o botón de la barra de herramientas.

Figura 8: Diálogo Gestionar Puntos de Interrupción



Elementos del diálogo Gestionar puntos de interrupción

(Campo de inserción y lista)	Números de linea en la que se encuentran los puntos de interrupción. Seleccione el que le interese o cree uno nuevo.
Activo	Marque o desmarque para activar/desactivar el punto seleccionado.
Recuento de pasadas	Permite desactivar el punto de interrupción después del número de pasadas definido.
Nuevo	Escriba un número de linea en el campo superior y pulse este botón para un nuevo punto de interrupción.
Eliminar	Elimina el punto de interrupción seleccionado.

Personalizar el EDI

LibreOffice permite personalizar el EDI, mediante el menú: **Herramientas > Opciones > LibreOffice > EDI de Basic**

Activar compleción de código	El IDE completa automáticamente los métodos de objeto Basic. No se aplica a objetos personalizados y la opción *Usar tipos extendidos* debe estar activada.
Corrección automática	Corrige la sintaxis de las palabras clave y nombres de variables
Cerrar comillas automáticamente	Al escribir comillas dobles de apertura el IDE agrega las de cierre.
Cerrar paréntesis automáticamente	Al escribir paréntesis de apertura, el IDE agrega los de cierre.

Cerrar procedimientos atomáticamente	Al declarar un procedimiento `Sub Xxx` o `Function Xxx`, se agrega automáticamente `End Sub` o `End Function`.
Usar tipos extendidos	(Necesario para la compleción de código). Permite utilizar tipos de objetos UNO como tipos Basic válidos.

Trucos y consejos

Copiar una biblioteca de un contenedor a otro

1. Abra el documento/contenedor origen.
2. Abra el *Organizador de macros*  , pestaña *Bibliotecas*.
3. **Exportar > Exportar como biblioteca Basic**.
4. Abra el documento/contenedor destino y pulse *Importar*.

Mover / copiar módulos de una biblioteca a otra

(En el mismo documento o entre documentos/contenedores)

1. Abra los dos documentos/contenedores origen y destino.
2. Abra el *Organizador de macros*.
3. Arrastre y suelte desde el origen al destino.

 De manera predeterminada, los módulos se mueven. Para copiarlos mantenga pulsada la tecla Ctrl mientras los arrastra.

Ocultar los módulos

Permite simplificar la lista de pestañas ocultando los módulos temporalmente: Haga clic derecho en la pestaña y elija *Ocultar*. Para visualizar de nuevo la pestaña oculta de un módulo haga clic derecho, elija *Módulos* y seleccione el módulo oculto.

Cifrar bibliotecas

El cifrado de bibliotecas permite proteger el código.

 Se puede cifrar cualquier biblioteca excepto la biblioteca *Standard*.

1. Abra el *Organizador de macros*, pestaña *Bibliotecas*.
2. Seleccione la ubicación y la biblioteca.
3. Pulse el botón *Contraseña*.
4. Introduzca la contraseña, confirme la contraseña y valide.

 **Advertencia -** El cifrado es muy sólido (AES) : **¡Si olvida la contraseña es irremediable!**

 Conserve siempre una versión no cifrada.

Limpiar una importación de MSOffice

 ¡Trabaje siempre con una copia del archivo!

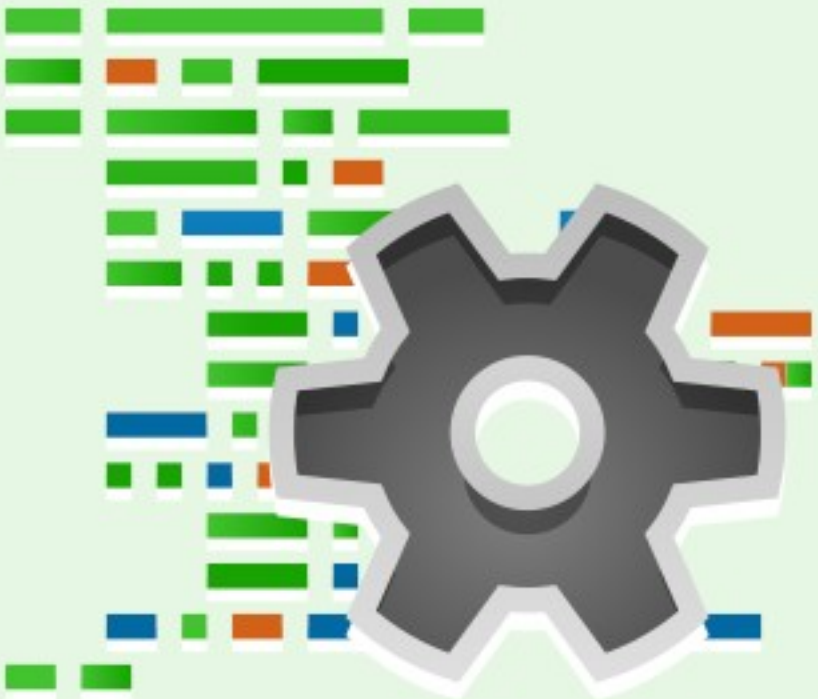
1. Elimine las bibliotecas y módulos innecesarios procedentes de la importación.
2. Guarde el archivo como «texto sin formato» (ej : `.fods`).
3. Abra el archivo con un editor de texto que admita `XML`.
4. Busque la etiqueta `<office:scripts>` y elimine toda la sección entre `<ooo:library-embedded ooo:name="Standard">` y `</ooo:library-embedded>`.
5. Guarde el archivo, abra el archivo con LibreOffice y recree el árbol de bibliotecas/módulos (copie/pegue el código deseado desde el documento original al nuevo).

Atajos de teclado

Diálogo de acceso a macros	Alt + F11	Detener ejecución	⇧ + F5
Ir a una linea	Ctrl + L	Entrar en proceso	F8
Punto de interrupción	F9	Pasar al siguiente	⇧ + F8
Activar inspección	F7	Salir de proceso	Ctrl + ⇧ + F8
Ejecutar	F5	Moverse entre zonas	F6

Ficha nº 2.
Las Bases

Nivel: Principiante



Generalidades

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

Nombrar las entidades

Las variables, constantes, subrutinas, funciones, módulos y bibliotecas deben estar identificados con un nombre adecuado. Los caracteres permitidos son: 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(), Guardar_Carpeta()`

Indentar el código

Se recomienda indentar cada nivel de código con espacios o tabuladores para facilitar la lectura.

Comentarios

Los comentarios **solo abarcan una linea** y sirven para clarificar o anular las instrucciones de la macro. Son tan importantes como el código

El apóstrofo `'` o `REM`. Indican que lo siguiente es un comentario y por tanto, no se ejecutará el código (si lo escrito son instrucciones).

Comentar varias líneas

Seleccione las líneas que desee comentar y utilice el atajo de teclado Ctrl + Alt + C

Varias instrucciones en la misma linea

Para incluir instrucciones distintas en una misma linea se utiliza el carácter de dos puntos (`:`)

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

Continuar una instrucción en la linea siguiente

Se pueden fraccionar líneas muy largas, de manera que una instrucción continúe en la siguiente línea. Los **dos últimos** caracteres de la linea deben ser espacio + subrayado (`_`)

Variables

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

La declaración de variables no es obligatoria pero no declararlas es una práctica peligrosa: Se pueden crear duplicados en caso de errores tipográficos.

Con `Option Explicit` al inicio del módulo se muestra un error al ejecutar instrucciones si la variable no ha sido declarada. Esto es muy útil para controlar las variables.

Declaración de variables

Variables sencillas

`Dim MiVar As Un_Tipo` – Ejemplo : `Dim MiTexto As String`

`Dim A As Byte, B As String` – En una misma línea se separan con comas (`,`).

`Dim MiVar As Integer : MiVar = 123` – Declaración y asignación en una única línea.

Variables compuestas (matrices)

Vea la «Ficha nº 3. Tipos estructurados»

Asignación de variables (no Objeto)

```
MiVar = UnValor
```

☞ Si `UnValor` no es del mismo tipo que `MyVar`, se hace una conversión automática.
Es preferible convertir el tipo explicitamente (funciones typecast). Vea la «Ficha n° 6».

Creación/Asignación de variables de tipo Objeto

☞ Vea la «Ficha 3, Tipos estructurados»

Alcance (visibilidad) de las variables

Declaración...	Visibilidad ...
`Dim MiVar As UnTipo`	En la subrutina o módulo.
`Static MiVar As UnTipo`	En la subrutina. ☞ Persiste entre llamadas a otras rutinas.
`Private MiVar As UnTipo`	En el módulo.
`Public MiVar As UnTipo`	En la biblioteca.
`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	V. inicial
`Boolean`	Valores lógicos `True` / `False` (Verdadero / Falso) Puede verse como `False` = 0 ; `True` = 1 (u otros enteros)	`False`
`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 1×10^{-28} a 7.9×10^{28} (precisión máxima 28 decimales). ☞ Utilizado con las funciones de la API (en enteros de 64bits). ⚠ Los desbordamientos no provocan errores de ejecución.	`n/a`
`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 rectas (") (no tipográficas).	`""`
`Variant`	Sirve para cualquier tipo, incluye también el objeto.	`Empty`

Vea 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 expresarse en base hexadecimal. Prefije estos valores con `&H`.
Ejemplo: `&HFF` (= 255 en decimal). Útil para los colores.
Asigne valores iniciales en lugar de contar con su iniciación implícita.

Cuidado con los errores de redondeo en los cálculos con números reales

Matrices, tipos personalizados, Colecciones y Objetos

Vea la «Ficha nº 3, Tipos estructurados»

Empty, Null y Nothing

`Empty`	Variable no inicializada. Posible asignación `Empty`.
`Null`	Contenido no válido. Posible asignación `Null`.
`Nothing`	Sólo en objetos, sin referencia al objeto. Posible asignación.

Comprobaciones

Con la función `IsEmpty(UnaVariable)` se puede comprobar si la variable está vacía.

Con la función `IsNull(UnObjeto)` se puede comprobar si se puede utilizar (existe) un objeto.

Operadores

Booleanos

`NOT` No	`AND` Y	`OR` O (inclusivo)	`XOR` O exclusivo

Comparaciones (devuelven `True` o `False`)

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

Atención a las comparaciones de números reales (pueden producir errores por redondeos)

Numéricos

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

Texto

`&`	Concatenación de texto

También se puede usar el signo **+** pero no es recomendable porque es la suma y puede dar un resultado no deseado al intentar concatenar números.

Constantes

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

Declación de constantes

```
Const MI_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

`Const MICONST = UnValor` – Visible en la subrutian o módulo actual.
`Public MICONST = UnValor` – Visible en la biblioteca actual.
`Global MICONST = UnValor` – Visible en todas las bibliotecas.

Rutas de los archivos

Al ser multi plataforma, las rutas a los archivos pueden visualizarse en formato nativo o URL :

`file:///soporte/ruta/al/archivo.txt.`

⚠ En las instrucciones se debe usar siempre 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:

Nombre nativo (en Windows): `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

Las Subrutinas ejecutan una acción.

💡 Sugerencia de Nombre : verbo en infinitivo ; HacerXxx, LeerXxx, etc.

Declaración

```
Sub NombreDeLaSub(parámetros)
```

Estructura

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

Uso

`NombreDeLaSub(argumentos)` o si no hay argumentos: `NombreDeLaSub()`

Function

Las funciones ejecutan una acción y también devuelven un valor.

💡 Sugerencia de Nombre : verbo en indicativo : `ListaXxx()`, `EstXxx()`, etc.

Declaración

```
Function NombreFuncion(parámetros) As UnTipo
```

Estructura

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

Uso

```
UnaVariable = N_Funcion(arg) O si no hay argumentos: UnaVariable = N_Funcion()
```


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

Parámetros o argumentos

Realmente un parámetro es lo mismo que un argumento. La diferencia está en el nombre que está condicionado por dónde se presenta:

Argumento – Al hacer una llamada a otra subrutina, se pasa el valor como argumento
Parámetro – El valor que ha pasado una subrutina, necesario para la función o subrutina .

Ejemplo de Uso

```
MiSub(ByRef Param1 As Long, ByVal Param2 As Long, Optional ByRef Param3 As
Object)
```

`ByRef` (Por referencia - es el valor predeterminado). 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. Es decir:
Si desde una subrutina se hace una llamada a otra subrutina con un argumento de valor `1` y la segunda subrutina cambia ese valor a `5`, al volver a la primera, el valor es `5`

`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 predeterminado 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 …)

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

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


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 …)

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

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

Do While … Loop / While … Wend (Hacer algo … Mientras que)

`Condición` evaluada al principio de cada pasada.

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

o también la sintaxis antigua:

```
While Condición
  'instrucciones
Wend
```

> ⚠ **La sintaxis antigua,** está permitida por compatibilidad pero se debe evitar su uso. `No acepta salida de bucle (Exit).`

Do Loop ... Until (Hacer algo … hasta que)

`Condición` evaluada al final de cada pasada..

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

> ⚠ **Tenga Cuidado con los bucles infinitos (Condición que nunca se cumple)**

Comprobaciones condicionales

Derivación que permite actuar según un estado o una situación.

If en una línea (If … Then)

Si se cumple la condición se ejecuta una instrucción

```
If Condición Then UnaInstrucción
```

If en varias líneas (If … Then … [Else] .. End If)

Si se cumple la condición se ejecutan unas instrucciones.

```
If Condición Then
  'Instrucciones si se cumple
Else ' La instrucción Else es facultariva.
  'Instrucciones si la condición no se cumple
End If
```

If… Then … ElseIf … Else … End If

Permite evitar la anidación de varios `If`

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

Select Case … End Select

Elección entre varias posibilidades en función del valor de «`UnaVariable`».

```
Select Case UnaVariable
  Case Valor : 'Instrucciones para «Valor»
  Case otro_Valor
    instrucciones para «otro_Valor»
  Case Val1, Val2 To Val3
    'instrucciones para varios valores
  Case Else
    'instrucciones para otros casos
End Select
```

Cargar bibliotecas de código Basic

Para una mejor lectura, organice su código en varias bibliotecas (Ficha nº1, El EDI).

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)

```
If BasicLibraries.hasByName("Mibiblioteca")then 'Verificar si existe
   BasicLibraries.loadLibrary("Mibiblioteca")'Carga
```

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 de la siguiente manera : `contenedor.biblioteca.modulo.nombre` (parcial o totalmente).
Ejemplo : `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.). También se puede consultar la siguiente página https://wiki.documentfoundation.org/Development/DispatchCommands

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 entrecomillado
  Dispatch = createUnoService("com.sun.star.frame.DispatchHelper")
  Dispatch.executeDispatch(Frame, UnoCmd, "", 0, Args())
```

Ejemplos

En los ejemplos sólo se muestran las partes modificadas

Ej.1 Llamada a la vista previa de Impresión

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

Ej.2 Mostrar/ocultar la barra lateral

Algunos comandos Uno necesitan argumentos:

```
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` y `Error` : informan del último error encontrado.

Funciones de información de error

`Err` – El código del error intervenido.
`Error` – El texto del mensaje descriptivo del error.
`Erl` – El número de linea donde se produjo el error.

☞ El código de error `0` (cero) es igual a «sin errores».
Utilice : `If Err = [#] Then …` para comprobar existencia de un error de número `#`.

Interceptación global de errores

On Error

⚠ 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 bloque de código se debe definir la etiqueta `MiEtiqueta:`

(usando los dos puntos al final del nombre de la etiqueta).

`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

En una subrutina, se puede utilizar `On Local Error Xxx` (misma sintaxis). En bastantes casos es preferible 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 activo
Barra de herramientas		✔	✔
Menú		✔	✔
Atajo de teclado	✔	✔	
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**	✔	✔	✔

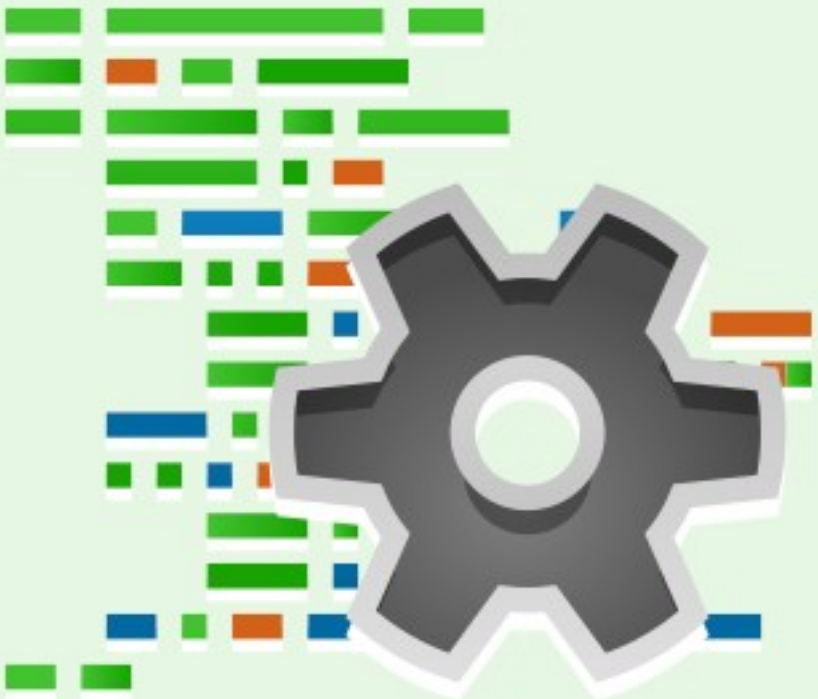
✔ Compatible	○ Posible pérdida	**!** Posible desbordamiento	**x** incompatible

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`.

Ficha nº 3.
Tipos Estructurados

Nivel: Intermedio



Matrices

Agrupación de elementos similares, aplicando un índice interno (Long).

Dimensiones

Una matriz puede contener varias dimensiones (max: 60).

Declaración

⚠ La base predet. del índice es `0` (cero) Modificable por `Option Base 1` (poco usado).

En la tabla siguiente: el índice se expresa como `ix`.

Matrices estáticas	Características predefinidas
Una dimension («vector»)	
`Dim T(ixMin To ixMax) As` UnTipo	`Dim T(1 To 12) As` UnTipo 1 dimensión, 12 elementos, (`ix = 1`)
`Dim T(ixMax) As` UnTipo	`Dim T(9)`: 1 dimensión, 10 elementos, (`íx = 0`) (se empieza a contar desde cero)
Varias dimensiones	
`Dim T(ixMin1 To ixMax1, ixMin2 To ixMax2) As` UnTipo	Varias dimensiones (en el ejemplo 2). `Dim T(1 To 12, 1 To 31) As Integer` 2 dimensiones, 12 y 31 elementos (`ix = 1`)
`Dim T(ixMax1, ixMax2, … ) As` UnTipo	`Dim T(2,4) As String`: 2 dimensiones. 3 elementos para la 1ª, 5 para la 2ª.(`ix = 0`)

Matrices dinámicas	***Las dimensiones se definen durante ejecución.***
`Dim Array1() As` UnTipo `Dim Array1 As Variant`	Matriz de dimensiones desconocidas. Después será necesario usar `ReDim`

☞ Declarada de tipo `Variant`, una matriz puede contener elementos de distintos tipos.

Matrices anidadas (Nested arrays/Jagged matrices)

O matrices de matrices. Ej: usadas para acceder a los intervalos de datos en Calc.

Una matriz externa (de variants) tiene por elementos matrices de datos:

```
Dim T As Variant
Anidada = Array(Array(1, 2, 3), Array(10, 20, 30), Array(7, 8, 9))
```

`Anidada(0)(0)` vale `1`; - `Anidada(2)(2)` vale `9`, etc.

Acceso a un elemento de la matriz

Por índice :

```
Dim Matriz(9) As Integer
Matriz(5) = 123 '(modifica el 5º elem).
```

```
Dim Matriz(11, 31) As Integer
Matriz(5, 28) = 123
```

Funciones e instrucciones relativas a la matriz

- `Option Base 1` :Instrucción al inicio de módulo (sólo se aplica donde se declara). Fuerza a los índices de las matrices a empezar por 1 en lugar de 0.
- `IsArray()`: devuelve `True` si la variable es de tipo matriz (array).
  - `OK = IsArray(Matriz)` (`OK` obtiene el valor true)

- `Array()`: devuelve la matriz creada con valores variados
  - `Matriz = Array("Uno", 2, Now())` (matriz de variants)
- `Redim`: (instrucción) Redimensiona una matriz
  - `Redim Matriz(dimension)` – Con pérdida de datos
  - `Redim Preserve Matriz(dimension)` – Sin pérdida de datos
- `Erase`: (Instrucción) Borra el contenido de la matriz
  - `Erase Matriz` : si la matriz es dinámica la libera de la memoria.
- `Lbound()`: devuelve el límite inferior si no se indica es el índice de la primera dimensión o indicando la dimensión:
  - `LBound(Matriz, 2)`
- `Ubound()`: devuelve el límite superior (igual que en el caso anterior).

☞ No tiene dimensiones definidas si `UBound(Matriz)=-1` y `LBound(Matriz)=0`

- `Split()`: crea una matriz (de una dimensión) cortando una cadena de texto mediante un delimitador.
  - `T = Split("C:/file.txt", "/")` -► `T(0)="C:"` y `T(1)="file.txt"`
- `Join()`: fusiona los elementos de una matriz (de una dimensión) para obtener una cadena.
  - `Join(T, "|")` -► `"C:|file.txt"` (aquí se usa el delimitador `|`)

Verificación de una matriz

Una variable «Matriz» sólo se procesa si pasa las siguientes (3) comprobaciones:

¿Existe la variable? – `Not IsNull(Matriz)`
¿Es una matriz? – `IsArray(Matriz)`
¿Están definidas sus dimensiones? – `UBound(Matriz) >= LBound(Matriz)`

Recorrer una matriz de una dimensión (vector)

Por sus índices

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

Por sus elementos

```
Dim Element As 'tipo compatible con todos los elementos
  For Each Elemento In Matriz
    Print Elemento
  Next
```

Recorrer una matriz de dos dimensiones

```
Dim Matriz(2, 4)   '3 filas, 5 columnas
Dim i As Long, j As Long
  For i = LBound(Matriz) To Ubound(Matriz)
    For j = Lbound(Matriz, 2) To UBound(Matriz, 2)
      Print Matriz(i, j)
    Next j
  Next i
```

Ordenación de los elementos de matrices

☞ No existe un método predefinido, (busque «QuickSort» o «Bubblesort» en la web).

Duplicar matrices

Método General

Un bucle (`For … Next`) que copie los valores entre 2 matrices (el proceso puede ser muy largo)

Truco

Por asignación y después usando `ReDim Preserve`:

```
Matriz2 = Matriz1  'Dos matrices -> mismos valores
ReDim Preserve Matriz2(Tamaño)  '-> 2 matrices distintas
```

☞ Se aplica sólo a matrices de valores simples (no objeto).

Usar una matriz en subrutinas

Como parámetro de una subrutina

Sub con parámetro matriz)

```
Sub UsarArray(ByRef Matriz() As String
```

Llamada a la subrutina

```
Dim Matriz(9) As String
UsarArray(Matriz)
```

Como resultado de una función

Función con resultado matriz

```
Function ObtenArray() As Integer()
  ObtenArray = MatrizDeEnteros()'Después de crear la «MatrizDeEnteros»
End function
```

Llamada a la función desde una sub

```
Dim Matriz As Integer
  Matriz = ObtenArray()
```

Tablas e intervalos de libros de cálculo

(Vea la Ficha nº 4. Calc)

Tipos personalizados (Custom types)

Permiten añadir varios valores (miembros) en un tipo de valor único.

Simplifican la manipulación de datos, paso de parámetros o resultado de función.

⚠ Tipo de miembros: sin definir o personalizado (**excepto** matriz).

Declaración de tipo

```
Type MiTipoPerso
  UnMiembro As TipoSimple
  OtroMiembro As OtroTipoSimple
End Type
' -----------------
Type Suceso
  Nombre As String
  FechaHora As Date
End Type
```

Declaración de una variable de un tipo personalizado

```
Dim UnaVar As MiTipoPerso
```

Limitacion: un tipo personalizado sólo es visible en el módulo en que se declara. La declaración As MiTipo por tanto sólo es posible en el seno del mismo módulo donde se ha declarado el tipo MiTipo. Vea Función de «fábrica» (Factory) / Accesor (accessor).

Uso

Asignación: `VarPerso.UnMiembro = UnValor`

Lectura: `UnaVar = VarPerso.OtroMiembro`

La palabra clave `With` – Permite agrupar las referencias a los elementos.

```
With VarPerso
  .UnMiembro = UnValor 'Observe la presencia del punto.
End With
```

Colecciones

Estructura que permite el acceso rápido a los datos al indexarlos. Una colección se manipula como un tipo `Object`. Los elementos almacenados pueden ser de cualquier tipo, incluido `Object`.

La clave de índice es de tipo `String`. En una colección cada clave es **única**.

Declarar una colección

```
Dim oColec As New Collection
```

Añadir un elemento

Con clave: `oColec.Add(Elt, "Clave")`

Acceso posterior por clave o por índice.
La clave no es sensible a mayúsculas.
Es posible precisar la ubicación del elemento añadido mediante `Before/After`:
`oColl.Add(Elt, "Clave", After:="Clave0")`

Sin clave: `oColl.Add(Elt)`

Acceso posterior sólo por índice.

Accceder a un elemento

– Por su clave: `Valor = oColl("Clave")`

– Por su índice: `Valor = oColl.Item(Index)`

Remplazar un elemento

Misma clave, nuevo valor de elemento.

Eliminar y después **Añadir**

Eliminar un elemento

– Por su clave: `oColl.Remove("Clave")`

– Por su índice: `oColl.Remove(Index)`

Eliminar todos los elementos

```
ReDim oColl As New Collection
```

Número de elementos

```
Numero = oColl.Count
```

Verificación de la existencia de un elemento

Intento de acceso y tratamiento del error eventual (la función siguiente devuelve true o false)

```
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 no existe -> ErrHandler:
  Exists = True
ErrHandler:
    'do nothing
  ExistsItem = Exists
End Function
```

Recorrer una colección

Puede obtener los elementos de una colección recorriéndola.

☞ No hay posibilidad de listar las claves.

Por índices

```
  For i = 1 To oColl.Count
    Valor = oColl.Item(i)  'acceso al dato
  Next i
```

Por acceso directo a los elementos

```
Dim Element As 'tipo compatible con los elementos de la colección.
  For Each Element In oColl
    'Hacer algo con el elemento(dato)
  Next
```

Creación de clases en Basic

Embrión de programación orientada a objetos (POO) (OOP en inglés).

☞ Limites: No tiene herencia (utilice la delegación), No tiene polimorfismo

Una clase LibreOffice Basic podría entenderse como un tipo personalizado mejorado, al que se añade un comportamiento (funciones o subrutinas).

Vocabulario

Módulo de clase	**Módulo de código** destinado a contener las declaraciones de la clase
Clase	**Tipo** que permite crear (instanciar) variables objeto.
Sucesos	Métodos para interceptar la creación y destrucción del objeto.
Miembro	**Variable** interna de una clase.
Propiedad	Refleja el estado del objeto.
Método	Realiza una acción con el objeto.
Instancia	**Objeto** creado a partir de un tipo clase.

Especificar una clase

Las especificaciones de una clase (miembros, propiedades, métodos) se hacen en el interior de un módulo de código dedicado. En LibreOffice Basic, este módulo no se distingue de un módulo estándar más que por sus opciones iniciales.

☞ Consejo adopte una convención de nombres para los módulos de clase.

Opciones iniciales

Al principio del modulo de clase indique las opciones:

```
Option Explicit  'como habitualmente
Option Compatible
Option ClassModule
```

Variables miembros

Son internas, por tanto declaradas `Private`.

> Los miembros de una clase sólo deben llamarse a través de propiedades creadas para este propósito nunca diréctamente a través de la instancia,.

```
Private mNombre As String
Private mSheet As Object
```

Sucesos (Events)

Son dos subrutinas internas por tanto, declaradas `Private`.

Constructor: `Private Sub Class_Initialize()` Para iniciar el objeto durante su creación.

Destructor: `Private Sub Class_Terminate()` Para purgar sus componentes internos durante su destrucción.

> Fallo de seguridad. Se recomienda encarecidamente no incluir este destructor en sus clases: debido a un error de implementación en VisualBasic, `Class_Terminate ()` constituye una vulnerabilidad de seguridad y, como tal, es rechazado por los antivirus (consulte el certificado CVE-2018-8174).

> **Limitación**: no es posible pasar parámetros a estas subrutinas.

Propiedades (Properties)

Propiedad = estado del objeto. Son visibles desde el exterior, por tanto declarados `Public`.

De lectura (Get)

```
(todo tipo de datos, incluidos objetos)
Public Property Get Name() As String
  Name = mName
 End Property
```

De escritura (Let)

```
(todo tipo de datos excepto objetos)
Public Property Let Name(ByRef pName As String)
  mName = pName
End Property
```

De escritura (Set)

```
(sólo objetos)
Public Property Set Sheet(ByRef pSheet As Object)
  Set mSheet = pSheet
End Property
```

> Es posible abandonar prematuarmente una propiedad mediante `Exit Property`

Una clase puede contener propiedades de lectura y escritura: Escriba sucesivamente las dos propiedades `Get` y `Let`/`Set` si es necesario.

Las propiedades pueden acceder a los miembros, propiedades y métodos de la clase.

Métodos (Methods)

Método = action sobre/de el objeto.

`Sub` y `Function` propias de la clase, internas (`Private`) o visibles (`Public`). Se escriben como `Sub` o `Function` estándar precedidas de la palabra clave `Public` o `Private`. Estas tienen acceso a los miembros y propiedades de la clase.

Utilización de las clases en Basic

Declarar/crear un objeto

Instanciación inmediata

```
Set MiObjeto = MiClase
```

Instanciación diferida (1ª llamada)

```
Dim MiObjeto As New MiClase : MiObjeto = New MiClase
```

El constructor de la clase se llama en el momento de la instanciación del objeto.

No se puede declarar un objeto `As New MiClase` fuera de la bibliloteca en la que el módulo clase existe. Por lo general deberá declarar el objeto `As Object`.

⚠ **Limitación**: Una clase no es visible fuera de la biblioteca en la que se declara.
Vea: Función de «fábrica» (Factory) / Accesor (accessor).

Acceder a las propiedades y métodos de un objeto

Sintaxis de acceso a los elementos de un objeto:

`objeto.propiedad` u `objeto.metodo`

Como para los tipos personalizados, puede utilizar la sintaxis `With..End With`

Liberar un objeto

Cuando un objeto no sea ya necesario, puede destruirlo: `Set oObjet = Nothing`

En ese momento se hace una llamada al destructor de la clase

☞ Esta instrucción no es totalmente necesaria en `Sub` o `Function`, puesto que sus variables internas se destruyen a la salida. Sin embargo, la destrucción efectiva de la variable no está bajo su control.
La instrucción `Set oObjet = Nothing`:
– muestra claramente la intención.
– asegura el control del momento de destrucción del objeto.

Función de «fábrica» (Factory) / Accesor (accessor)

Cuestión de visibilidad de tipos y clases personalizados

- Un **tipo personalizado** sólo es visible en el **módulo** donde se ha declarado.
- Una **clase** sólo es visible en la **biblioteca** donde se ha declarado

Para solucionar este problema, cree una función «fábrica» (o accesor) para crear las variables. A esta función se la podrá llamar desde cualquier módulo o biblioteca.

	Uso	Visibilidad	Fábrica creada en	Declarado
Tipo personal.	`As MiTipo`	Mismo módulo	Mismo módulo	`As Variant`
Clase	`As MiClase`	Misma biblioteca	Misma biblioteca, otro módulo	`As Object`

Creación de la función «fábrica» / accesor

⚠ La función fábrica puede también utilizarse para inicializar la variable.

Tipos personalizados

```
Function CreateMiTipoPerso() As MiTipoPerso
  Dim oVar As MiTipoPerso
  CreateMiTipoPerso = oVar
End Function
```

Clases

```
Function CreateMiClase() As MiClase
  Dim oVar As New MiClase
  CreateMiClase = oVar
End Function
```

Uso de la función «fábrica» / accesor

Tipos personalizados

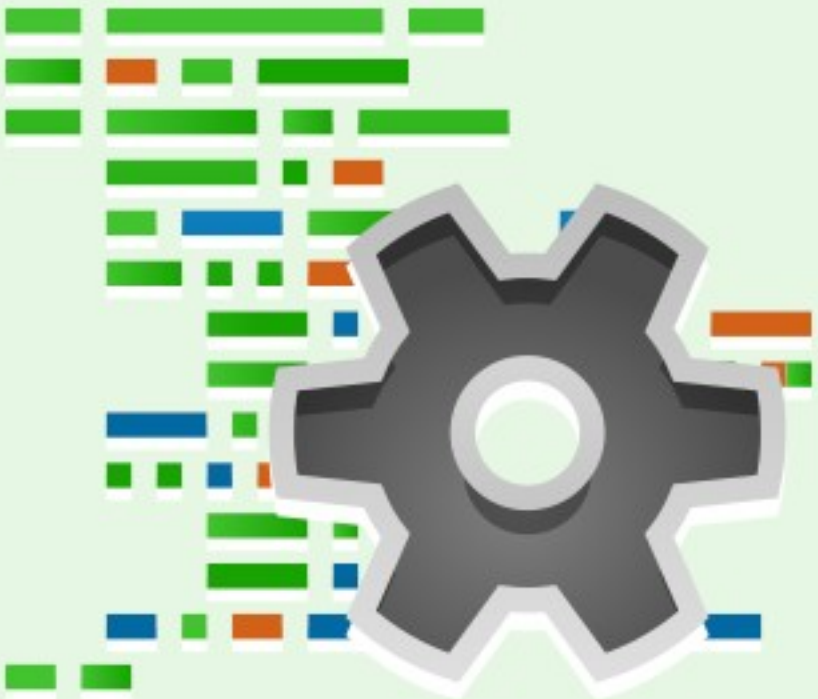
```
Dim MiVar As Variant
  MiVar = CreateMiTypoPerso()
```

Clases

```
Dim MiVar As Object
  MiVar = CreateMiClase()
```

Ficha nº 4.
Calc

Nivel: Principiante



Documentos de LibreOffice

Document activo

```
Dim Doc As Object
  Doc = ThisComponent
```

Abrir un documento existente

Modo visible

```
Dim Doc As Object, RutaDoc As String
Dim Props() 'este array no se inicializa pero es necesario definirlo
  RutaDoc = ConvertToURL("C:\Ruta\a\UnaHoja.ods")
  Doc = StarDesktop.loadComponentFromURL(RutaDoc,"_blank",0,Props())
```

Modo invisible

```
Dim Doc As Object, RutaDoc As String
Dim Props(0) As New com.sun.star.beans.PropertyValue
  RutaDoc = ConvertToURL("C:\Ruta\a\UnaHojaDeCalculo.ods")
  Props(0).Name = "Hidden"  'el documento se abre como "oculto"
  Props(0).Value = True
  Doc = StarDesktop.loadComponentFromURL(RutaDoc,"_blank",0,Props())
```

Hacerlo visible (después , en la misma macro)

```
  Doc.CurrentController.Frame.ContainerWindow.Visible = True
  Doc.CurrentController.Frame.ContainerWindow.toFront()
```

Crear una nueva hoja de cálculo

A partir (1) de la plantilla predeterminada o (2) de una plantilla especifica.

```
Dim Doc As Object
Dim Props() 'este array no se inicializa  pero es necesario
  Modelo = "private:factory/scalc"  '(1)
  Modelo = ConvertToURL("C:\Ruta\a\UnaPlantillaDeHoja.ots")  '(2)
  Doc = StarDesktop.loadComponentFromURL(Modelo,"_blank",0,Props())
```

Guardar un documento

El documento ya existe

(**Archivo > Guardar**) Utilice el método `store` del documento : `ThisComponent.store`

El documento todavía no se ha guardado

(**Archivo > Guardar como**)

```
Dim Doc As Object, RutaDoc As String
Dim Props()     'las propiedades de guardado (vacías)
  RutaDoc = convertToURL("C:\Ruta\a\UnaHojaDeCálculo.ods")
  Doc.storeAsURL(RutaDoc, Props())
```

☞ Si se guarda como un documento ya existente, este pasa a ser el documento activo.

Guardar una copia

Como antes pero con `Doc.storeToURL(RutaDoc, Props())`

☞ Al guardarlo como copia, la copia **NO** pasa a estar activa, se sigue usando el original

Guardar con contraseña (cualquiera de los casos anteriores)

Añada una propiedad `Password` y asígnele el valor adecuado.

```
Props(n).Name  = "Password" '(n es la enumeración de propiedades)
Props(n).Value = "mypwd"
```

Cerrar un documento

Utilice el método close del objeto documento :

```
ThisComponent.close(True)
```

Información del documento

El objeto documento contiene las propiedades:

Location	El directorio de almacenamiento del documento. ☞ Cadena vacía si no se ha guardado.
`DocumentProperties (Objeto)`	Propiedades complementarias.

DocumentProperties

`Author`	Nombre del autor.	`ModifyDate`	Fecha última modificación
`CreationDate`	Fecha de creación.	`Subject`	Asunto (String).
`Description`	Comentarios.	`Title`	Título (String).
`ModifiedBy`	Usuario que ha modificado el documento.	`UserDefinedProperties`	Propriedades personalizadas (Objeto).

¿Es una hoja de cálculo?

El objeto `Doc` se refiere al documento (`Doc = ThisComponent`).

```
CalcOK = Doc.SupportsService("com.sun.star.sheet.SpreadsheetDocument")
```

Calc – Funcionalidades generales

El objeto `Doc` se refiere al documento (`Doc = ThisComponent`).

Cálculo automático

¿Está activo? (Booleano)	`Auto = Doc.isAutomaticCalculationEnabled`
Désactivar	`Doc.enableAutomaticCalculation(False)`
Activar	`Doc.enableAutomaticCalculation(True)`
Forzar el recálculo de fórmulas	`Doc.calculate '(sólo fórmulas no actualizadas)` `Doc.calculateAll '(recalcula todo)`

Proteger el libro

¿Libro protegido? (Booleano)	`Test = Doc.isProtected`
Proteger el documento	`Doc.protect(Contraseña) 'puede estar vacía`
Desproteger el documento	`Doc.unprotect(Contraseña)`

Hojas (sheets)

El objeto `Doc` se refiere al documento (`Doc = ThisComponent`).

Acceder a las hojas

Se puede trabajar con los objetos hojas (se empieza contando por 0) :

Hoja activa	`Hoja = Doc.CurrentController.ActiveSheet`
Lista de hojas	`Hojas = Doc.Sheets`

Número de hojas	`NumHojas = Doc.Sheets.Count`
Objeto hoja (por su índice)	`Hoja = Doc.Sheets(número_índice)`
Objeto hoja (por su nombre)	`Hoja = Doc.Sheets.getByName(Nombre_Hoja)`
Verificar la existencia (nombre)	`Exist = Doc.Sheets.hasByName(Nombre_Hoja)`
Índice de una hoja	`Indice = Hoja.RangeAddress.Sheet`

Modificar las hojas

`p` es la posición en el documento (índice, empezando por 0).

Añadir una hoja con Nombre	`Doc.Sheets.insertNewByName(Nombre, p)`
Eliminar una hoja	`Doc.Sheets.removeByName(Nombre_Hoja)`
Duplicar una hoja	`Doc.Sheets.copyByName(NomOrigen, NomDestino, p)`
Mover una hoja	`Doc.Sheets.moveByName(Nombre_Hoja, p)`

Gestionar las hojas

Hoja se refiere a un objeto `Sheet` (hoja).

Activar una hoja	`Hoja = Doc.CurrentController.ActiveSheet`
Ocultar/Mostrar una Hoja	`Hoja.IsVisible = False 'True`
Comprobar la protección	`Protegida = Hoja.IsProtected`
Proteger una hoja	`Hoja.protect(contraseña) '(puede estar vacía)`
Desproteger una hoja	`Hoja.unprotect(contraseña)`
Colorear la pestaña	`Hoja.tabColor = RGB(255, 255, 0)`

Enlazar una hoja

Enlazar a un archivo (ej : CSV)	`Hoja.link(URL, "", "Text - txt - csv (StarCalc)", _` `  Filtre, com.sun.star.sheet.SheetLinkMode.VALUE)`
Eliminar enlace	`Hoja.setLinkMode(com.sun.star.sheet.SheetLinkMode.NONE)`

Encontrar la última fila/columna utilizada

Hoja es el objeto `Sheet` (hoja) donde buscar. `Fila` y `Col` son informaciones buscadas

```
Dim oCur As Object 'cursor sobre la celda
Dim oRango As Object 'el rango usado
Dim Fila As Long, Col As Long
  oCur = Hoja.createCursorByRange(Hoja.getCellRangeByName("A1"))
  oCur.gotoEndOfUsedArea(True)
  oRango = Hoja.getCellRangeByName(oCur.AbsoluteName)
  Fila = oRango.RangeAddress.EndRow
  Col = oRango.RangeAddress.EndColumn
```

Celdas (cells)

Acceder a las celdas

Hoja se refiere a un objeto `Sheet` (hoja). Se puede acceder a un objeto `Cell` (celda):

Por coordenadas	`Celda = Hoja.getCellRangeByName("A4")`
Por nombre	`Celda = Hoja.getCellRangeByName("IVA")`

Por coordenadas numéricas X e Y (índice 0)	`Celda = Hoja.getCellByPosition(0,3)` `' X=0 (columna A) ; Y=3 (Fila 4)`

Acceder a la celda activa

`Doc` un objeto documento y `ActiveCel` el objeto celda seleccionado

```
If Doc.currentSelection.supportsService _
 ("com.sun.star.sheet.SheetCell") Then
 'comprobamos que sea una celda lo seleccionado
 ActiveCel = Doc.currentSelection
End If
```

Seleccionar una celda

```
ThisComponent.CurrentController.select(Cel)
```

Coordenadas de una celda

Coordenadas (Object)	`Coord = Cel.CellAddress`
Coordenadas absolutas (String)	`Coord = Cel.AbsoluteName`
Rango de la hoja (Integer)	`NumH = Cel.CellAddress.Sheet`
Rango de la columna (Long)	`NumC = Cel.CellAddress.Column`
Rango de la fila (Long)	`NumF = Cel.CellAddress.Row`
hoja a que pertenece (Objeto)	`Hoja = Cel.Spreadsheet`

(Des)Proteger las celdas

La propiedad `Cel.CellProtection` utiliza valores booleanos :

Proteger contra modificaciones	`CellProtection.IsLocked = True`
Ocultar la fórmula	`CellProtection.IsFormulaHidden = True`
Ocultar la celda	`CellProtection.IsHidden = True`
Evitar impresión de la celda	`CellProtection.IsPrintHidden = True`

Acceder al contenido de una celda

Propiedades

obtener un contenido texto	`MiTexto = Cel.String`
Obtener un contenido numérico	`UnNumero = Cel.Value`
Obtener una fórmula (US)	`LaFormula = Cel.Formula`
Obtener una fórmula (regionalización)	`LaFormula = Cel.FormulaLocal`
Obtener el tipo de contenido	`ElTipo = Cel.Type`
Vaciar una celda	`Cel.String = ""`

Tipo de contenido (propiedad `Type`)

Las constantes `com.sun.star.table.CellContentType.XXX` permiten conocer el tipo de información que contienen las celdas (`Cel.Type`, se muestra abajo) :

`EMPTY`	Celda vacía	`VALUE`	Valor numérico
`TEXT`	Texto	`FORMULA`	Formula

Escribir en una celda

Reemplazar el **texto** existente	`Cel.String = "Cucú !"`
Reemplazar el **valor** existente	`Cel.Value = 1,234`
Reemplazar la **fórmula** existente	`Cel.Formula = "= AND(A1="SI";A2="NO")"`
Reemplazar la **fórmula** existente (es)	`Cel.FormulaLocal = "=Y(A1="SI";A2="NO")"`

Rangos (ranges)

Rango = conjunto de celdas (puede referirse a una única celda): `Dim Rango As Object`

Acceder a los rangos

Hoja se refiere a un objeto `Sheet` (hoja). Accedemos a un objeto `Range` (Rango)

Por sus coordenadas	`Rango = Hoja.getCellRangeByName("C2:G14")`
Por su nombre	`Rango = Hoja.getCellRangeByName("NombreDeRango")`
Por coordenadas (X1, Y1, X2, Y2)	`Rango = Hoja.getCellRangeByPosition(2,1,6,13)`
Arbitrariamente (ej 3.a hoja documento)	`Rango = ThisComponent.Sheets.getCellRangeByPosition _ (2, 1, 6, 13, 2)`

Acceder al rango activo

De la misma manera que se accede a la celda activa (pero verificar antes) `"com.sun.star.sheet.SheetCellRange"` o `"[...].SheetCellRanges"`.

Seleccionar un rango

```
ThisComponent.CurrentController.select(MiRango)  MiRango es un objeto.
```

Coordenadas de un rango

Coordenadas (Object)	`Coord = MiRango.RangeAddress`
Rango de la hoja (Integer)	`Rang  = MiRango.RangeAddress.Sheet`
Columna de inicio (Long) (ángulo superior / izquierdo)	`NumCHG = MiRango.RangeAddress.StartColumn`
Fila de inicio (Long) (ángulo superior / izquierdo)	`NumLHG = MiRango.RangeAddress.StartRow`
Columna final (Long) (ángulo inferior/derecho)	`NumCBD = MiRango.RangeAddress.EndColumn`
Fila final (Long) (ángulo inferior/derecho)	`NumLBD = MiRango.RangeAddress.EndRow`
Hoja contenedora (Objeto)	`Hoja = MiRango.Spreadsheet`
Coordenadas absolutas (String)	`Coord = MiRango.AbsoluteName`

Rangos con nombre

El objeto `Doc` se refiere al documento. `Dim MisRangos As Object`

Los rangos con nombre	`MisRangos = Doc.NamedRanges`
Número (Long)	`Num = LosRangos.Count`
Acceder a un rango (por índice)	`MiRango = LosRangos(index)`

Verificar existencia de un rango (nombre)	`Existe = LosRangos.hasByName(Nombre)`
Acceder a un rango (por nombre)	`MiRango = LosRangos.getByName(Nombre)`
Añadir un rango: `Coord`: coordenadas del rango `CellRef`: objeto celda de réferencia	`LosRangos.addNewByName(Nombre,Coord, CellRef.CellAddress, 0)`
Eliminar un rango (por nombre)	`LosRangos.removeByName(Nombre)`

Borrar el contenido de un rango

Borrar el contenido de MiRango	`MiRango.clearContents(ModoBorrar)`

`ModoBorrar` es un valor que determina el tipo de borrado. Las constantes `com.sun.star.sheet.CellFlags.XXX` permiten la elección (se pueden combinar con +) :

`ANNOTATION`	Comentarios	`STRING`	Texto
`DATETIME`	Números formato fecha-hora	`VALUE`	Números (excepto fecha-hora)
`FORMULA`	Fórmulas		

Acceder al contenido de las celdas de un rango

`MiRango.DataArray` est un array de los **valores** de las celdas para `MiRango`.

Copiar el contenido de un rango en otro

Dos arrays Origen y destino, **de las mismas dimensiones.**

Copiar el contenido (**valores**) Orig. a Dest.	`Destino.DataArray = Origen.DataArray`

Escribir valores en un rango (desde un array a un rango)

`MiRango` es un objeto rango y `Tabla` un array, **de las mismas dimensiones**,

```
Dim Tabla As Variant
Tabla = MiRango.DataArray 'Tabla obtiene las dimensiones del rango
'(transferir los valores a los elementos del array)
MiRango.DataArray = Tabla
```

`.DataArray` puede ser un array anidado : utilice `.DataArray(i)(j)` si es el caso

Recorrer las celdas de un rango

Cree una enumeración desde una colección (`Rango.Celdas`). La cual se recorre llamando a sus propiedades `hasMoreElements` y `NextElement`:

```
Dim Rangos As Object
Rangos = ThisComponent.createInstance("com.sun.star.sheet.SheetCellRanges")
Rangos.insertByName("", MiRango)
LEnum = Rangos.Cells.CreateEnumeration
Do While LEnum.hasMoreElements
  Celda = Lenum.NextElement
  'hacer algo en cada objeto Celda
Loop
```

Las celdas vacías **No** se recorren !

Rangos : varios

Fusionar las celdas de MiRango	`MiRango.Merge`

Tipos de rangos

Según el modo de acceso a un rango, este implementa uno de los servicios :

`com.sun.star.sheet.SheetCell`	`com.sun.star.sheet.SheetCellRange`
`com.sun.star.table.CellRange`	`com.sun.star.sheet.SheetCellRanges`
`com.sun.star.sheet.NamedRange`	

⚠ dependiendo del servicio implementado, los rangos deben ser utilizados de manera diferente. Compruebe por medio del método `supportsService()`.

¿Rango o celda?

Para conocer el tipo de objeto, compruebe el objeto rango/celda con `supportsService()`:

`If MiObj.supportsService(nomb_servicio)Then…` Remplace Nombre de servicio con:

¿Celda ?	`"com.sun.star.sheet.SheetCell"` ①
¿Rango simple ?	`"com.sun.star.sheet.SheetCellRange"` ④
¿Rango múltiple ?	`"com.sun.star.sheet.SheetCellRanges"` ⑤

☞ compruebe siempre celda antes que rango (una celda es también un rango simple)

Filas / Columnas (rows/columns)

Filas y columnas son propiedades de los objetos `Sheet` y `Range`.

Generalidades

Filas (Objeto `LasFilas`)	`LasFilas = MiRango.Rows`
Columnas (Objeto `LasColumnas`)	`LasColumnas = MiRango.Columns`
Número de elementos	`NumF = MiRango.Rows.Count` `NumC = MiRango.Columns.Count`
Una fila (Objeto `Fila`) (base 0)	`Fila = MiRango.Rows(index)`
Una columna (Objeto `Columna`) (base 0)	`Columna = MiRango.Columns(index)`

Propiedades de filas / columnas

Se aplican a los objetos Fila o Filas / Columna o Columna.

Es Visible u oculta (Boolean)	`IsVisible = True`
Longitud óptima o fija (Boolean)	`OptimalWidth = True`

Insertar / eliminar filas / columnas

Use el objeto `Sitio`. `PosInicio` y `Num` son la posición inicial y el número de filas / columnas que quiera insertar / eliminar (Long).

Insertar	`Sitio.insertByIndex(PosInicio, Num)`
Borrar	`Sitio.clearContents(Modoborrado)'vea «Borrar el contenido de rango»`
Eliminar	`Sitio.removeByIndex(PosInicio, Num)`

Fijar filas / columnas

☞ Sólo en un documento visible.

Use el objeto Controlador : `MiContrlr = ThisComponent.CurrentController`

¿hay alguna fija ?	`Fijas = MiContrlr.hasFrozenPanes`
Fijar (X, Y)	`MiContrlr.freezeAtPosition(1,2)`
Liberar	`MiContrlr.freezeAtPosition(0,0)`

Llamar a una función Calc

Utilice el servicio `"com.sun.star.sheet.FunctionAccess"`

Principio

```
Dim FCalc As Object , Resultado As (según contexto)
Dim Params As (según contexto), NombreFunc As String
  FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
  Resultado = FCalc.callFunction(Nombre_Función, Parámetros)
```


El nombre, los parámetros y el tipo de resultado dependen de la función elegida.


El nombre de la función debe ser su nombre en **inglés**.
Para averigurarlo, cambie temporalmente la presentación de nombres de funciones en **Herramientas > Opciones > LibreOffice Calc > Formula**, **Usar nombres [...] inglés**.

Ejemplo 1, función `SUMA()`

```
Dim FCalc As Object, Resultado As Long
  FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
  Resultado = FCalc.callFunction("SUM", Array(1, 55, 321, 8))
```

Ejemplo 2, función `REDONDEAR()`

```
Dim FCalc As Object, Resultado As Double
Dim Params(1) As Variant
  Params(0) = 1,2345 'número que se quiere redondear
  Params(1) = 3      '3 decimales
  FCalc = CreateUnoService("com.sun.star.sheet.FunctionAccess")
  Resultado = FCalc.callFunction("ROUND", Params())
```

Crear una función Calc

Creación

Ejemplo : calculo de la superficie de un trapecio = (lado mayor + lado menor / 2) * altura

```
Function SupTrapecio(BMay As Double, BMen As Double, H As Double)_ As Double
  SurpTrapecio = ((BMay + BMen) / 2) * H
End Function
```

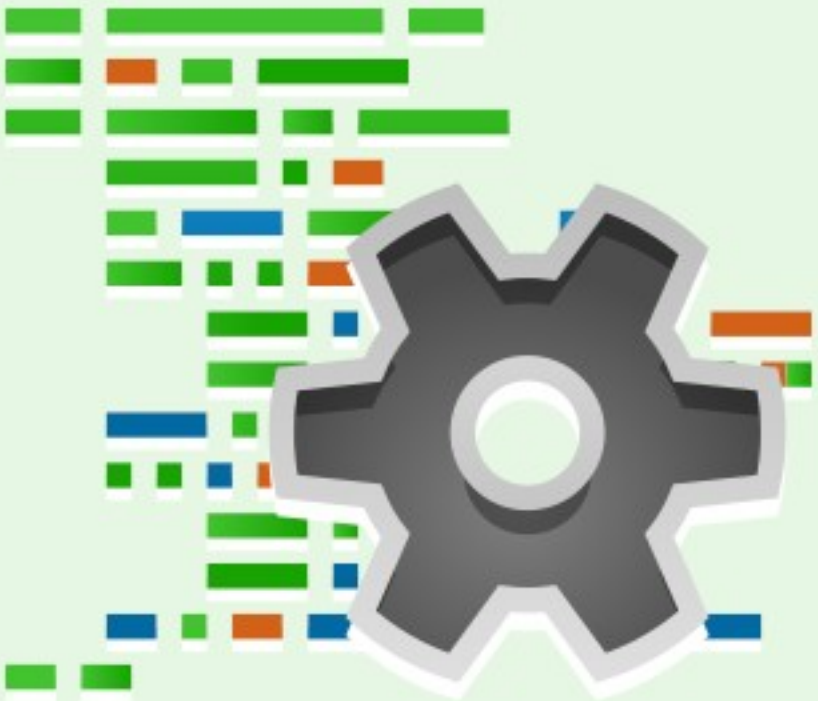
Uso en Calc

Si `A2` es la base mayor, `A3` la base menor y `A4` la altura, la superficie del trapecio se obtiene insertando la fórmula siguiente en una celda: `=SUPTRAPECIO(A2;A3;A4)`


- La macro recibe los **valores** de los argumentos y no el objeto celda.
- La macro **reenvía un valor,** No puede actuar sobre una celda
- La función se debe colocar en una biblioteca accesible en el momento de su uso (ej. : `Standard` del documento o del usuario) (sino devolverá el error `#VALOR!`)

Ficha nº 5.
Sucesos

Nivel: Intermedio



Los sucesos

Los objetos manipulados por LibreOffice reconocen diversos tipos de suceso, que se desencadenan en diferentes situaciones. Puede interceptar estos sucesos para darles el tratamiento deseado.

Objetos contenedores de sucesos

La aplicación	Objetos del documento	
Documentos	Objetos OLE	Texto Automático
Formularios y sus controles	Imágenes	Zonas de Imágenes
Diálogos Basic y sus controles	Cuadros	Hiperenlaces

Asociar un suceso a una macro

Principio

1. **Cree la macro** que desee ejecutar según el modelo:

```
Sub NombreDeLaMacro()
End Sub
```

Use un Nombre de macro de acorde con el objeto, la acción y el tipo de suceso.
ejemplo: `Sub EnBoton_OK_Click()`

   La subrutina puede contener parámetros. vea más abajo.

2. Seleccione el objeto contenedor del suceso que quiera interceptar.
3. Acceda a su configuración (método variable en función del objeto).
4. Elija el suceso que quiera interceptar (vea la lista en este guía de referencia).
5. Seleccione la macro para ejecutar con el desencadenante del suceso.

Obtener información del elemento desencadenante

El procedimiento de ejecución puede consultar el parámetro recibido para obtener informaciones complementarias sobre el suceso:

```
Sub RespuestaSuceso(ByRef Event As Object)
End Sub
```

La estructura y las propiedades del objeto `Event` dependen del tipo de suceso que desencadena la llamada al procedimiento (información más adelante).

Usos estándar para acceder a los controles de diálogo Basic:

Para acceder a	Consultar
(Objeto) **control** que hace la llamada	`Event.Source`
(Objeto) **modelo** del control	`Event.Source.Model`
(Objeto) **diálogo** propietario del control	`Event.Source.Context`

Categorías y propiedades de los sucesos

Hay cuatro categorías de sucesos de los diálogos (columna **Cat** en última tabla):

Ratón	**M**	0,00 €
Teclado	**K**	Desencadenados por las secuencias de teclas del teclado
Foco	**F**	Ejecutados cuando el foco cambia de un control a otro.

Específicos	**S**	Relacionados con ciertos controles.

☞ Las constantes citadas a continuación se deben usar vigilando las Mayúsculas.

Sucesos asociados al ratón

Las coordenadas se miden en píxeles a partir del ángulo superior izquierdo del control.

vea la estructura `com.sun.star.awt.MouseEvent`.

Buttons (short)	El botón pulsado (`com.sun.star.awt.MouseButton`).
X (long) , **Y** (long)	Coordenadas X e Y respectivas del cursor del ratón
ClickCount (long)	Número de clics asociados a un suceso del ratón. Si LibreOffice puede responder lo bastante rápido, `ClickCount` es igual a `1` para un doble-clic, (sólo un suceso).
PopupTrigger (boolean)	`True` si hay menú contextual

Constantes definidas en `com.sun.star.awt.MouseButton`.

`LEFT`	Botón izquierdo.	`RIGHT`	Botón derecho.	`MIDDLE`	Botón central.

☞ Los sucesos VBA `Click` y `Doubleclick` no están disponibles en LibreOffice Basic. Puede utilizar el suceso LibreOffice Basic `Botón del ratón soltado` en lugar del suceso `Click` e imitar el suceso `Doubleclick` modificando la lógica de la aplicación.

Sucesos asociados al Teclado

Los sucesos del teclado se asocian a pulsaciones lógicas y no a pulsaciones físicas.

☞ Combinación de teclas = un único suceso.
Una acción única sobre una tecla de modificación (ej: ⇧ o Alt) no crea ningún suceso independiente.

Los objetos `Event` asociados al Teclado tienen las siguientes propiedades:

KeyCode (short)	El código de la tecla pulsada (`com.sun.star.awt.Key.XXX`). Las teclas muertas ⇧, Ctrl o Alt no modifican el código.
KeyChar (String)	El carácter elegido (teniendo en cuenta las teclas modificadoras).
KeyFunc (Integer)	Funcionalidad de la tecla según la constante en `com.sun.star.awt.KeyFunction.XXXX`.
Modifiers (Integer)	Indica si se ha pulsado una tecla muerta (vea las constantes `com.sun.star.awt.KeyModifier.XXX`).

Resumen de las constantes definidas en `com.sun.star.awt.Key.XXX`

`NUM0 à NUM9`	Cifras	`RETURN`	↵	`POINT`	.
`A à Z`	Letras	`ESCAPE`	Esc	`COMMA`	,
`F1 à F26`	T. de función	`TAB`	⇆	`LESS`	<
`UP`	↑	`BACKSPACE`	←	`GREATER`	>
`DOWN`	↓	`SPACE`	␣	`EQUAL`	=
`LEFT`	←	`INSERT`	Ins	`CUT`	Ctrl + X
`RIGHT`	→	`DELETE`	Del	`COPY`	Ctrl + C
`HOME`	Home	`ADD`	+	`PASTE`	Ctrl + V
`END`	End	`SUBTRACT`	-	`HELP`	F1
`PAGE UP`	RePág	`MULTIPLY`	*	`MENU`	⊞

PAGEDOWN	AvPág	DIVIDE	/	CONTEXTMENU	

Los siguientes códigos indentifican teclas físicas.

Constantes com.sun.star.awt.KeyFunc.XXX

DONTKNOW	Desconocida	CUT	Cortar	CLOSE	Cerrar
NEW	Nuevo	COPY	Copiar	QUIT	Salir
OPEN	Abrir	PASTE	Pegar	PROPERTIES	Propiedades
SAVE	Guardar	UNDO	Deshacer	FIND	Búscar
SAVEAS	Guardar como	REDO	Rehacer	FINDBACKWARD	Buscar (hacia atrás)
PRINT	Imprimir	REPEAT	Repetir	FRONT	Primer plano

Constantes com.sun.star.awt.KeyModifier.XXX (acumulables con + o And)

MOD1	Ctrl	MOD2	Alt	SHIFT	⇧

Sucesos asociados al Foco

Los objetos Event asociados al foco tienen las siguientes propiedades:

FocusFlags (short)	La razón del cambio de foco. (Vea constantes com.sun.star.awt.FocusChangeReason).
NextFocus (Object)	El objeto que recibe el foco (sólo para «al perder foco»).
Temporary (Boolean)	El foco se ha perdido temporalmente.

Constantes definidas en com.sun.star.awt.FocusChangeReason

TAB	Tabulador pulsado.	BACKWARD	Control anterior.
CURSOR	Una tecla de dirección pulsada.	AROUND	Del último al 1er control (adelante o del 1º al último (atrás).
MNEMONIC	Una tecla de atajo pulsada.	UNIQUEMNEMONIC	Una tecla de atajo a un sólo control pulsada.
FORWARD	Control siguiente.		

Sucesos específicos asociados a ciertos controles

Algunos sucesos se pueden desencadenar mediante una acción en determinados controles (ej. Botones de opción). No se ejecuta ninguna acción para determinar si el estado del control se ha modificado realmente. Para evitar este tipo de « suceso ciego », guarde el valor antiguo del control en una variable global y verifique si el valor se modifica en el momento que se produzca dicho suceso.

Propiedades del suceso Item Status Changed:

Selected (long)	La entrada esta seleccionada.
Highlighted (long)	La entrada está realzada
ItemId (long)	El identificador de la entrada

Sucesos asociados a los documentos

Propiedades del suceso entrante (vea apartado siguiente):

EventName (string)	El nombre del suceso.
Source (object)	El documento que causa el suceso.

`EventName` (string)	El nombre del suceso.
`ViewController` (object)	El controlador de visualización en cuestión si no hay: `Null`.
`Supplement` (variant)	Las informaciones complementarias, si no hay: `Empty`.

Sucesos asociados al documento o a la aplicación

Herramientas > Personalizar > Eventos

Sucesos disponibles

Suceso	La macro asignada se ejecuta ...
Iniciar la aplicación	Después del inicio de la aplicación.
Cerrar la aplicación	Antes de cerrar la aplicación.
Documento creado	Después de **Archivo > Nuevo.**
Documento nuevo	Después de creacción de un documento a partir de una plantilla
La carga del doc. ha finalizado	Después de recargar un documento.
Abrir documento	Después de **Archivo > Abrir.**
Se cerrará el documento	Antes de cerrar un documento.
Documento cerrado	Después de cerrar un documento. ☞ El suceso `Documento cerrado` puede también desencadenarse al guardar el documento (con nombre ya existente).
Vista creada	Después de la creación de la vista del documento.
La vista se cerrará	Antes de cerrar la vista del documento.
Vista cerrada	Antes del cierre de la vista del documento.
Activar documento	Después de que el documento ocupe el 1er plano.
Desactivar documento	Después que otro documento ocupe el 1er plano
Guardar documento	Antes de **Archivo > Guardar**, si el nombre de documento ya existe ☞ vea `Documento cerrado`.
El documento se ha guardado	Después de **Archivo > Guardar**, si el nombre del documento ya existe.
Falló el guardado del documento	Después de un error al guardar el documento.
Guardar documento como	Antes de **Archivo Guardar como** o **Archivo > Guardar** si el nombre de documento no existe.
El documento se ha guardado como	Después de **Archivo > Guardar como**, o **Archivo > guardar** si el nombre de documento no existe.
Falló « Guardar como »	Después de un error al **Guardar como.**
Almacenar o exportar copia del documento.	
Se creó una copia del documento.	Después de la creación de la copia.
Falló la creación de una copia del d.	Después de un error al crear la copia.
Impresión de documento	Después del cierre del diálogo **Imprimir**, pero antes de la impresión propiamente dicha.

Suceso	La macro asignada se ejecuta ...
Se ha cambiado el estado «modificado»	Después de que el estado « modificado » haya cambiado.
Título del documento cambiado	Después de la modificación del título del documento.
Inició la impresión de las cartas modelo	Después del cierre del diálogo imprimir pero antes de la impresión propiamente dicha.
Finalizó la impresión de las cartas modelo	Después de la impresión de cartas modelo.
Inició la fusión de campos en el formulario	
Finalizó la fusión de campos en el formulario	
Modificación del contador de paginas	En el momento del cambio del número de páginas

☞ Los sucesos `Vista` se desencadenan al cambiar la vista:
Modo de previsualización o Nueva ventana.

Secuencias de sucesos en el documento

Apertura de un documento existente (cualquier método)

Abrir el documento > Vista creada

Cierre del documento activo (cualquier método)

La vista se cerrará > Se cerrará el documento > Vista cerrada > Documento cerrado

Creacción de un documento a partir de una plantilla

Nuevo documento > Vista creada

Interacciones con los objetos del documento

Propiedades del objeto > pestaña Macro, botón Macro, botón Sucesos, etc.

Suceso	Desencadenante	Objeto OLE	Imágen	Cuadro	Texto autom.	Mapa-Imag.	Hipervínculo
Clic sobre el objeto	Objeto seleccionado.	●	●	●			
Ratón sobre el objeto	El ratón se desplaza sobre el objeto.	●	●	●		●	●
Activar Hipervínculo	El hipervínculo asignado se activa.	●	●	●			●
El ratón sale del objeto	El puntero del ratón se desplaza fuera del objeto.	●	●	●		●	●
Carga de imagen finalizada	La carga de imágenes ha finalizado.		●				
Carga de imagen interrumpida	El usuario ha interrumpido a carga de imágenes al cargar la página (por ej.).		●				
Error al cargar la imagen	La carga de imágenes ha fallado (ej. no se encuentra la imagen).		●				
Escritura de caracteres alfanuméricos	Escritura de un texto con el teclado.			●			

Suceso	Desencadenante	Objeto OLE	Imágen	Cuadro	Texto autom.	Mapa-Imag.	Hipervínculo
Escritura de caracteres No alfanuméricos	Escritura de caracteres no imprimibles (tabulación, entrar, etc.).			●			
Modificación del tamaño de un cuadro	El tamaño de un cuadro se ha modificado mediante el ratón.			●			
Desplazamiento de un cuadro	Se ha desplazado un cuadro mediante el ratón.			●			
Antes de la inserción de Texto automático	Antes de insertar un bloque de texto automático.				●		
Después de la inserción de Texto automático	Después de insertar un bloque de texto automático.				●		

Sucesos asociados a las hojas de Calc

Editar > Hoja > Sucesos (o clic derecho en la pestaña y despues **Sucesos de hoja)**

Suceso	La macro asignada se ejecuta ...
Activar documento	Después de la presentación del documento en 1er plano.
Desactivar documento	Después de la presentación de otro documento en 1er plano.
Selección cambiada	Después de cambiar la selección.
Doble-clic	Después de un doble clic en una celda.
Clic derecho	Después de un clic derecho en un rango.
Fórmulas calculadas	Después del recálculo de las fórmulas
Contenido cambiado	Después de la modificación del contenido de una celda

Sucesos asociados a un formulario

Propiedades del control, pestaña **Sucesos**

Sucesos estandard que no pertenecen a una base de datos

Cat	Suceso	La macro asignada se ejecuta...
F	**Recepción de foco**	Cuando un control recibe el foco
F	**Al perder el foco**	Cuando un control pierde el foco
K	**Tecla pulsada**	Cuando el usuario pulsa una tecla mientras el control tiene el foco
K	**Después de haber pulsado una tecla**	Cuando el usuario suelta una tecla mientras el control tiene el foco
M	**Ratón dentro**	Cuando el ratón se encuentra en el interior del control
M	**Mover el ratón por medio del teclado**	Cuando el ratón se desplaza a la vez que se pulsa una tecla. Por ejemplo durante una acción de arrastrar - soltar (otra tecla determina el modo de mover o copiar).
M	**Movimiento del ratón**	Cuando el ratón se desplaza sobre el control

Cat	Suceso	La macro asignada se ejecuta…
M	**Botón del ratón pulsado**	Si se pulsa el botón del ratón mientras que el puntero del ratón se encuentra sobre el control
M	**Botón del ratón soltado**	Si se suelta el botón del ratón mientras que el puntero del ratón se encuentra sobre el control.
M	**Ratón fuera**	Cuando el ratón se encuentra fuera del control

☞ El suceso `Botón del ratón pulsado` sirve también para las llamadas al **menú contextual.**
La propiedad `PopupTrigger` del suceso es `TRUE`.
Si la llamada se hace con el clic derecho, el suceso se desencadena dos veces:
1) en la llamada al menú contextual y 2) en el mismo clic. Si sólo le interesa usar el clic descarte las llamadas `PopupTrigger`.

Sucesos de los controles de cuadros de diálogo

Además de los sucesos estándar indicados, algunos controles ofrecen:

Cat	Suceso	La macro asignada se ejecuta…
	Ejecutar una acción	Iniciar una acción. Por ej. si el formulario contiene un botón **Enviar**, el proceso de envío representa la acción ☞ como opción a la respuesta de clic sobre el control.
KM	**Estado modificado**	Cuando se ha modificado el estado del control
M	**Valor de desplazamiento o incremento /decremento.**	Como respuesta a la acción sobre una barra de desplazamiento o un «spinbutton».

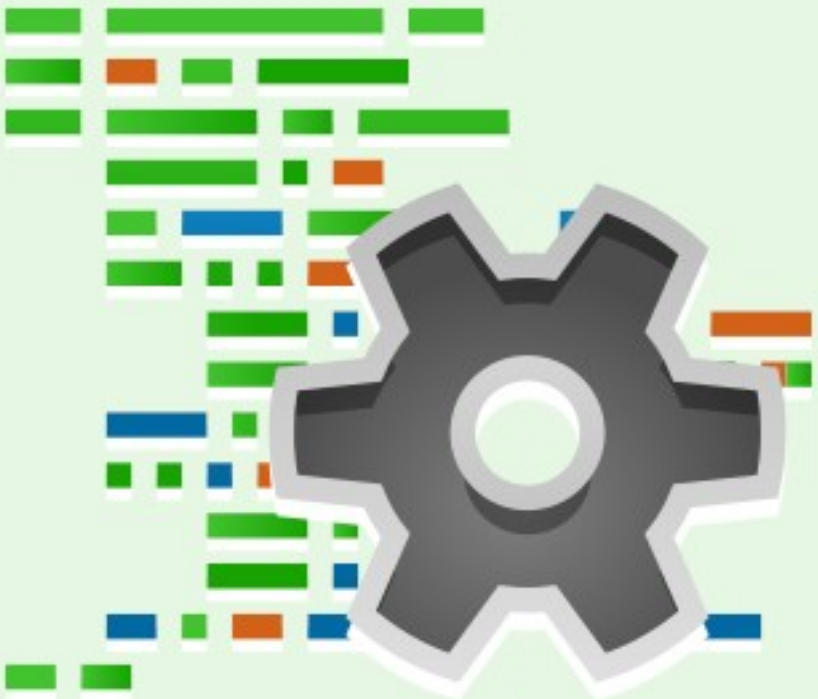
Sucesos asociados a una base de datos

Suceso	La macro asignada se ejecuta...
Después de agregar un registro	Después de la inclusión del registro activo.
Después de la modificación del registro	Inmediatamente después de la modificación del indicador del registo activo.
Antes de la modificación del registro	Antes de que el registo activo sea modificado, permite solicitar una confirmación.
Antes del envío	Antes de que los datos del formulario se envíen.
Antes de actualizar	Antes de que el contenido del control modificado por el usuario escriba en la base de datos, Por ejemplo para impedir que esta acción envíe el valor `FALSE`.
Después de actualizar	Después de que el control modificado por el usuario escriba en la base de datos.
Antes de reestablecer	Antes de reestablecer un formulario, por ej. para impedir una acción de reenvío `FALSE`. Un formulario se reestablece cuando alguna de las siguientes condiciones se cumple: El usuario pulsa un botón (HTML) definido como botón **Reestablecer**. Un registro nuevo y vacío se crea en un formulario asociado a una fuente de base de datos. Por ejemplo en el último registro, tiene que pulsar sobre el botón **Registro siguiente.**

Suceso	La macro asignada se ejecuta...
Después de restablecer	Después de reestablecer el formulario.
Antes del cambio de un registro	Antes de la modificación del indicador del registro activo. Por ejemplo, para impedir que esta acción envie un valor `FALSE`.
Antes de la descarga	Antes de que el formulario se descargue, (se separe de su fuente de base de datos).
Antes de la recarga	Antes de la recarga del formulario. (El contenido de la base de datos aún no se ha actualizado).
Confirmar eliminación	Cuando los datos se han eliminado del formulario, por ejemplo para solicitar una confirmación.
Al cargar	Después de la carga del formulario.
Al descargar	Inmediatamente después de la descarga del formulario (se separe de su fuente de base de datos).
Al recargar	Inmediatamente después de la recarga del formulario. (Ya se ha actualizado el contenido de los datos).
Rellenar los campos (parámetros)	Cuando el formulario que ha de ser cargado contiene parámetros que tengan que ser rellenados (SQL), si un parámetro no se puede rellenar, se llama a este suceso. Por ejemplo, la fuente de datos puede ser: SELECT * FROM address WHERE name=:name siendo:`name` un parámetro a rellenar en la carga del form. el parámetro se rellena automáticamente en la medida de lo posible a partir de un formulario de nivel superior.
Ha ocurrido un error	Si ocurre un error durante el acceso a la base de datos. ☞ Se aplica a los formularios, cuadros de lista y cuadros combinados.

Ficha nº 6.
Biblioteca de Ejecución

Nivel: Principiante



Opciones de ejecución

Especificaciones, **por módulo**, antes de cualquier código ejecutable.

`Option Explicit`	Impone la declaración explicita de variables.
`Option Compatible`	Para que LibO Basic se comporte como VBA.
`Option VBASupport 1`	Activa la compatibilidad o el apoyo a VBA.
`Option Base 1`	Indexar los arrays empezando por 1 en lugar de `0`.
`Option ClassModule`	Utilizado para crear clases (+ `Option Compatible`).

Constantes Basic

`True`	Verdadero (Booleano)	`Empty`	Variable no inicializada.
`False`	Falso (Booleano)	`Null`	Variable sin contenido útil
`Pi`	`3.14159265358979` (Doble)	`Nothing`	(objetos) elimina la asignación anterior.

Funciones

☞ Sintaxis general de las funciones `Resultado = NombreFunción(argumentos)`

Funciones, Cadenas de caracteres (tipo String)

`Asc()` Chr()	Devuelve el valor Ascii (del 1er carácter) de una cadena de texto. `Asc("Azerty")` → 65
`Chr()` Asc()	Devuelve el carácter al que pertenece el código Ascii expresado. `Chr(65)` → `"A"` *Vea `Asc()`, Tabla Ascii.
`ConvertFromURL()` ConvertToURL()	Convierte un nombre de archivo del formato URL al formato del S.O. Formato URL`protocolo:///host/ruta/al/archivo.ext` `ConvertFromURL("file:///c:/rep/archivo.ods")`→ C:\rep\archivo.ods Linux: `file:///home/user/rep/archivo.ods`
`ConvertToURL()` ConvertFromURL()	Convierte un nombre de archivo del formato del S.O. al formato URL `ConvertToURL("C:\rep\archivo.ods")` → file:///c:/rep/archivo.ods
`Cstr(), Str()` Str(), Val()	Convierte un numéro en cadena de texto.`Str(2.65)` → `"2.65"` ☞ El separador decimal en basic es el punto. Atentos a los espacios
`Format()`	Covierte un número en texto con el formato de una máscara. 14/4/2021, `Format(Now(), "yyyy")` → `"2021"` *Vea Colecciones.
`InStr()`	Busca en una cadena de texto la posición de otra cadena y devuelve el número que ocupa. * Si no la encuentra, devuelve `0`. `InStr("LibreOffice", "Office")` → 6
`Join()` Split()	Une los elementos de un array para convertirlos en una cadena de texto `Array1 = Array("C:", "Rep", "SsRep", "MiArchivo.ods")` `Join(Array1, "\")` → "C:\Rep\SsRep\MiArchivo.ods"
`Lcase()` Ucase()	Convierte una cadena a minúsculas `LCase("LibreOffice")` → "libreoffice"
`Left()` Mid(),Right()	`Left(cadena, N)`Extrae N caracteres de la cadena desde la izq. `Left("LibreOffice", 5)` → "Libre"
`Len()`	Devuelve el número de caracteres de una cadena. `Len("LibreOffice")` → 11

`Ltrim()` Rtrim(), Trim()	Elimina los espacios a la izquierda de una cadena (iniciales).
`Mid()` Left(),Right()	`Mid(cadena, P, N)`. Extrae `N` caracteres. Del interior de una cadena a partir de la posición `P`. `Mid("14/7/2017", 4, 1)` → `"7"`
`Right()` Mid(),Left()	`Right(cadena, N)`. Extae `N` caracteres de una cadena desde la derecha. `Right("LibreOffice", 6)` → "Office"
`Rtrim()` Ltrim(), Trim()	Elimina los espacios a la derecha de una cadena (finales)
`Space()` String()	Devuelve una cadena constituida por N espacios. `Space(3)` → `"   "`

`Split()`
Join()
	Convierte una cadena de texto en un array separando los elementos por un carácter especificado
	`MiCadena = "C:\Rep\SsRep\Archivo.ods"`
	`Split(MiCadena, "\")` → array `:"C:", "Rep", "SsRep", "Archivo.ods"`

`Str()`	Es lo mismo que `CStr()`
`StrComp()`	Compara dos cadenas y devuelve un entero (resultado de comparación).
`String()` Space()	Crea una cadena de N veces un carácter. `String(4, "Y")` → `"YYYY"` *Vea `Space()`
`Trim()` LTrim(), RTrim()	Elimina los espacios a Izquierda y derecha de una cadena.
`Ucase()` LCase()	Convierte una cadena en Mayúsculas. `UCase("LibreOffice")` → `"LIBREOFFICE"`
`Val()` Str(), CStr()	Convierte una cadena en un valor numérico (`0` si no se puede convertir) `Val("12.34")` → `12,34` (El separador decimal en basic es el punto)

Funciones Numéricas

`Abs()`	Devuelve el valor absoluto de un número.
`Exp()`	Exponencial. (Eleva a una potencia).
`Fix()`	Devuelve la parte entera de un número (sin redondearlo).
`Hex()`	Devuelve el valor hexadecimal de un número decimal.
`Int()`	Devuelve la parte entera de un número (redondeado a la baja).
`Log()`	Devuelve el logaritmo de un número.
`Oct()`	Devuelve el valor octal de un número decimal.
`Randomize()`	Inicializa un generador de números aleatorios *para la función `Rnd()`).
`Rnd()`	Devuelve un número aleatorio entre 0 y 1. *Vea `Randomize()`
`Sgn()`	Devuelve el signo de un número.
`Sqr()`	Calcula la raiz cuadrada de un número.

Funciones Trigonométricas

`Ángulos en radianes. (use la formula) radianes = (grados * Pi)/180`

`Atn()` Arco tangente.	`Cos()` Coseno.	`Tan()` Tangente.

Funciones fecha / hora

Formato de fecha «UNO»

La API LibreOffice utiliza de manera habitual las fechas «Uno» `com.sun.star.util.DateTime` (`.Date` o bien `.Time`), que se estructura así:

`IsUTC`	`True` si la zona hor. es UTC.	`Hours`	Horas (0-23).
`Year`	No del año	`Minutes`	Minutos (0-59).
`Month`	No del més (`0` si está vacía).	`Seconds`	Segundos (0-59).
`Day`	No del día (`0` si está vacía).	`NanoSeconds`	Nanosegundos.

☞ Date ↔ Uno Date: utilice las funciones de conversión `CDateXxx` siguientes.

Funciones fecha / hora

`CDateFromISO()`	Convierte una cadena fecha en formato ISO al formato Date *formato iso = (AAAAMMJJ). Date = en función de config. local `CDateFromISO("20170714")` → 14/07/2017.
`CDateFromUnoDate()`	Convierte una estructura UNO `com.sun.star.util.Date` en un valor de tipo Date.
`CDateFromUnoDateTime()`	Convierte una estructura UNO `com.sun.star.util.DateTime` en un valor de tipo Date.
`CDateFromUnoTime()`	Convierte una estructura UNO `com.sun.star.util.Time` en un valor de tipo Date.
`CDateToISO()`	Convierte la fecha en formato Date al formato ISO 14/7/2017, `CDateToISO(Now())` → `"20170714"`
`CDateToUnoDate()`	Convierte la fecha a la forma estructurada UNO `com.sun.star.util.Date`.
`CDateToUnoDateTime()`	Convierte fecha y hora a la forma estructurada UNO `com.sun.star.util.DateTime`.
`CDateToUnoTime()`	Convierte hora a la forma estructurada UNO `com.sun.star.util.Time`.
`Date()` Now() y Time()	Devuelve la fecha actual (tipo Date). `Date(Now())`
`DateAdd()`	Devuelve una nueva fecha calculada por una fecha de inicio , una máscara de tiempo y un criterio de ajuste (±). 14/7/2017, `DateAdd("m", -1, Now())` → 14/6/2017

Formatos de unidades de fecha:

`yyyy`	Año	`m`	Mes	`w`	Día de la semana	`d`	Día	`m`	Minuto
`q`	Trimestre	`y`	Día del año	`ww`	Semana del año	`h`	Hora	`s`	Segundo

`DateDiff()`	Calcula la diferencia entre dos fechas, expresada en una unidad de fecha `DateDiff("m", "14/8/2017", "14/7/2017")` → 1
`DatePart()`	Devuelve la parte especificada de la fecha (unidad elegida) ** `DatePart("q", "14/7/2017")` → 3
`DateSerial()`	Devuelve un valor fecha para una fecha calculada por tres elementos separados por coma (año, mes, día). `DateSerial(2017,7,14)` → 14/7/2017
`DateValue()`	Convierte una cadena de texto a un formato tipo Date `DateValue("14/7/2017")` → 14/07/2017 (tipo Date)

`Day()`	Devuelve el número del día del mes. `Day("14/7/2017")` → `14`
`Hour()`	Devuelve la hora `Hour(Now())` → `12` (si es mediodía)
`Minute()`	Devuelve los minutos `Minute(Now())` → `0` (si es en punto)
`Month()`	Devuelve el número del mes. `Month("14/7/2017")` → `7`
`Now()` Date(), Time()	Devuelve fecha y hora actual (formato Date).
`Second()`	Devuelve los segundos de un valor de tipo Date. `Second(Now())` → `0 (si es la hora en punto)`
`Time()`	Devuelve la hora actual en tipo Date. *Vea `Date()`, `Now()`
`Timer()`	Devuelve un Double (número de segundos desde la medianoche). ☞ Asigne `Timer()` a una variable antes de usarlo
`TimeSerial()`	Devuelve un valor date calculado desde (hora, minuto y segundo) separados por coma `TimeSerial(12,25,14)` → `12:25:14` (Date)
`TimeValue()`	Convierte una cadena de texto válida en un valor tipo Date. `TimeValue("12:25:14")` → `12:25:14` (tipo Date)
`Wait`	(instrucción) hace una pausa de un tiempo en millisegundos. `Wait 1000` → pausa de 1 segundo.
`WeekDay()`	Devuelve el número del día de lasemana (`1` = domingo). `Weekday("14/7/2017")` → `6` (viernes)
`Year()`	Devuelve el número del año `Year("14/7/2017")` → `2017`

Funciones, Colores

Los colores se almacenan en variables `Long`.

`Red()`, `Green()`, `Blue()`	Extraen los valores de los componentes de un color
`RGB()`	Devuelve el número (Long) corresp. a un color expresado en componentes `RGB(128,0,0)` → `8388608` (Ladrillo oscuro)

Funciones, Arrays

`Array()`	Crea un array a partir de valores discretos. `MiArray = Array("Uno", 2, Now())`		
`DimArray()`	Equivalente a `Array()` `MiArray = DimArray("Uno", 2, Now())` ☞ Use esta opción si está activa la declaración implicita de variables.		
`Erase`	(Instruccion) Borra el contendo del array. Si es dinámico lo libera de memoria. `Erase MiArray`		
`LBound()`	Límite inferior.	`UBound()`	Límite superior.

Funciones de consulta de tipo

Estas funciones permiten obtener información de las variables

De cualquier variable

`TypeName()`	Devuelve una cadena de texto detallando la variable indicada.
`VarType()`	Devuelve un valor numérico relativo a la variable indicada.
`IsUnoStruct()`	Devuelve `True` si la variable indicada es una estructura UNO.

Las dos primeras funciones devuelven los valores indicados en la tabla:

Tipo	Nombre	Tipo	Nombre	Tipo	Nombre	Tipo	Nombre	Tipo	Nombre
0	Empty	3	Long	6	Currency	9	Object	17	Byte
1	Null	4	Single	7	Date	11	Boolean	37	Decimal
2	Integer	5	Double	8	String	12	Variant		

De tipo variants

Devuelven `True` según el tipo real detectado.

Función	Verifica el tipo	Función	Verifica el tipo
`IsArray()`	Array	`IsNull()`	`Null` (sin datos).
`IsDate()`	Date.	`IsNumeric()`	Valor numérico.
`IsEmpty()`	Variable no inicializada	`IsObject()`	Objeto OLE.
`IsError()`	Valor de error.	`IsUNOStruct()`	`True` si es estructura UNO.

De tipo objeto

(Vea Estructuras y objetos UNO)

Funciones de conversión de tipos (typecast)

Convierten un valor de un tipo (compatible) a otro. El nombre de la función refleja el tipo destino.

☞ Legibilidad del código: Utilice con preferencia un typecast explicito a uno implícito.

`CBool()`	a `Boolean`	`CDbl()`	a `Double`	`CSng()`	a `Single`
`CByte()`	a `Byte`	`CDec()`	a `Decimal`	`CStr()`	a `String`
`CCur()`	a `Currency`	`CInt()`	a `Integer`	`CVar()`	a `Variant`
`CDate()`	a `Date`	`CLng()`	a `Long`	`CVErr()`	a `Variant (Error)`

Estructuras y objetos UNO

`CreateUnoService(Nomb)`	Crea un servicio UNO. ☛ Nomb sensible a mayúsculas
`IsUNOStruct()`	`True` si es una estructura UNO.
`(struct.)Dbg_Properties`	Devuelve el nombre de la estructura UNO (`String`).
`HasUnoInterfaces()`	`True` si el objeto UNO promueve Interfaces.
`(obj.)SupportsService()`	`True` si el `objeto` promueve el servicio aumentado (`String`).
`EqualUnoObjects(o1, o2)`	`True` si dos var. se refieren a la misa instancia del objeto.

Funciones de información de errores

`Erl`	Nº de línea de error	`Error`	Mensaje asociado al error	`Err`	Código del error

Otras Funciones

`GetGUIType()`	Devuelve un valor que indica el sistema operativo usado				
	`1` Windows	`4` OSX o Linux		`3` MacOS	
`GetSolarVersion()`	Defvelve la versión de LibreOffice.				
`IsMissing()`	Comprueba si se ha omitido un parámetro opcional (sub o func.)				

Llamadas a comandos del sistema

Syntaxis del comando: `Shell(Comando, Estilo, Param, Sinc)`

`Comando`	El comando que se desea ejecutar (`String`).	
`Estilo`	Se refiere a la ventana donde ejecutar el comando (`Integer`). Opciones:	
	`0`	El programa recibe el foco y su ventana está oculta.
	`1`	El programa recibe el foco y se ejecuta en una ventana estándar.
	`2`	El programa recibe el foco y se ejecuta en una ventana minimizada.
	`3`	El programa recibe el foco y se ejecuta en una ventana maximizada.
	`4`	El programa se inicia en una ventana sin foco.
	`6`	El programa se inicia minimizado, el foco se queda en la ventana actual.
	`10`	El programa se inicia en modo pantalla completa.
`Param`	Parámetros de ejecución pasados como argumento al programa (`String`).	
`Sinc`	Señal de seguimiento de la ejecución	
	`True`	Esperar al final de la ejecución del comando.
	`False`	No esperar al final de la ejecución del comando.

Función Format – Máscaras de formato

La función `Format()` convierte un núm. en cadena, (formato según máscara de formato). Una **máscara de formato** es una cadena que puede ser mostrada en tres secciones separadas por punto y coma `val>0;val<0;val=0`. *Una sección sola = todos los números.

> ☞ Los números se formatean según la configuración de **Herramientas >Opciones > Idiomas y regiones > Generales >Formatos**.

Números

`0`	Cifra obligatoria en la posición (0 si no se indica)	`%`	El resultado se muestra en formato de porcentaje.
`#`	Cifra opcional	`E- E+`	Formato científico.
`.`	Separador decimal	`e- e+`	Formato científico.
`+- ()` `espacio`	Caracteres literales se muestran igual en el resultado.	`\`	Caracter de escape: el car. que sigue se muestra en el resultado.

Fechas

`D` o `DD`	N° del día (1 o 2 car.)	`Q` o `QQ`	N° trimestre (1 o 2 car.)
`NNN`	Nombre del día.	`W` o `WW`	N° semana (1 o 2 car.).
`M` o `MM`	N° del mes (1 o 2 car.)	`h` o `hh`	Hora (1 o 2 car.)
`MMMM`	Nombre del mes.	`m` o `mm`	Minutos (1 o 2 car.)
`YY` o `YYYY`	N° del año (2 o 4 car.)	`s` o `ss`	Segundos (1 o 2 car.)

Tabla Ascii

Dec	Hex	Val	Dec	Hex	Val	Dec	Hex	Val	Dec	Hex	Val	Dec	Hex	Val
0	0	NUL	26	1A	SUB	52	34	4	78	4E	N	104	68	h
1	1	SOH	27	1B	ESC	53	35	5	79	4F	O	105	69	i
2	2	STX	28	1C	FS	54	36	6	80	50	P	106	6A	j
3	3	ETX	29	1D	GS	55	37	7	81	51	Q	107	6B	k
4	4	EOT	30	1E	RS	56	38	8	82	52	R	108	6C	l
5	5	ENQ	31	1F	US	57	39	9	83	53	S	109	6D	m
6	6	ACK	32	20	SPC	58	3A	:	84	54	T	110	6E	n
7	7	BEL	33	21	!	59	3B	;	85	55	U	111	6F	o
8	8	BS	34	22	"	60	3C	<	86	56	V	112	70	p
9	9	HT	35	23	#	61	3D	=	87	57	W	113	71	q
10	0A	LF	36	24	$	62	3E	>	88	58	X	114	72	r
11	0B	VT	37	25	%	63	3F	?	89	59	Y	115	73	s
12	0C	FF	38	26	&	64	40	@	90	5A	Z	116	74	t
13	0D	CR	39	27	'	65	41	A	91	5B	[	117	75	u
14	0E	SO	40	28	(	66	42	B	92	5C	\	118	76	v
15	0F	SI	41	29	)	67	43	C	93	5D	]	119	77	w
16	10	DLE	42	2A	*	68	44	D	94	5E	^	120	78	x
17	11	DC1	43	2B	+	69	45	E	95	5F	_	121	79	y
18	12	DC2	44	2C	,	70	46	F	96	60	`	122	7A	z
19	13	DC3	45	2D	-	71	47	G	97	61	a	123	7B	{
20	14	DC4	46	2E	.	72	48	H	98	62	b	124	7C	|
21	15	NAK	47	2F	/	73	49	I	99	63	c	125	7D	}
22	16	SYN	48	30	0	74	4A	J	100	64	d	126	7E	~
23	17	ETB	49	31	1	75	4B	K	101	65	e	127	7F	DEL
24	18	CAN	50	32	2	76	4C	L	102	66	f			
25	19	EM	51	33	3	77	4D	M	103	67	g			

Soporte VBA

La compatibilidad con VBA no es completa.

Opciones de entorno

Herramientas > Opciones > Cargar / guardar > Propiedades de VBA

Cargar el código Basic	Carga y guarda el código VBA de un documento MSOffice en un módulo especial LibreOffice Basic.
Código ejecutable	El código VBA se cargará preparado para su ejecución.
Guardar el Basic original	El código VBA en el documento se guarda aparte durante la carga del documento en LibreOffice.

Opciones de ejecución

Para usar la compatibilidad con VBA indique `Option VBASupport 1` y `Option Compatible` en la cabecera del módulo.

Funciones VBA

`AscW`	`FV()`	`IRR()`	`Round()`
`ChrW`	`Input()`	`Me()`	`RTL()`
`DDB()`	`InStrRev()`	`MIRR()`	`StrReverse()`
`FormatDateTime()`	`IPmt()`	`NPer()`	`WeekDayName()`

Encuentre más detalles en la ayuda en linea

Instrucciones VBA

`Enum`

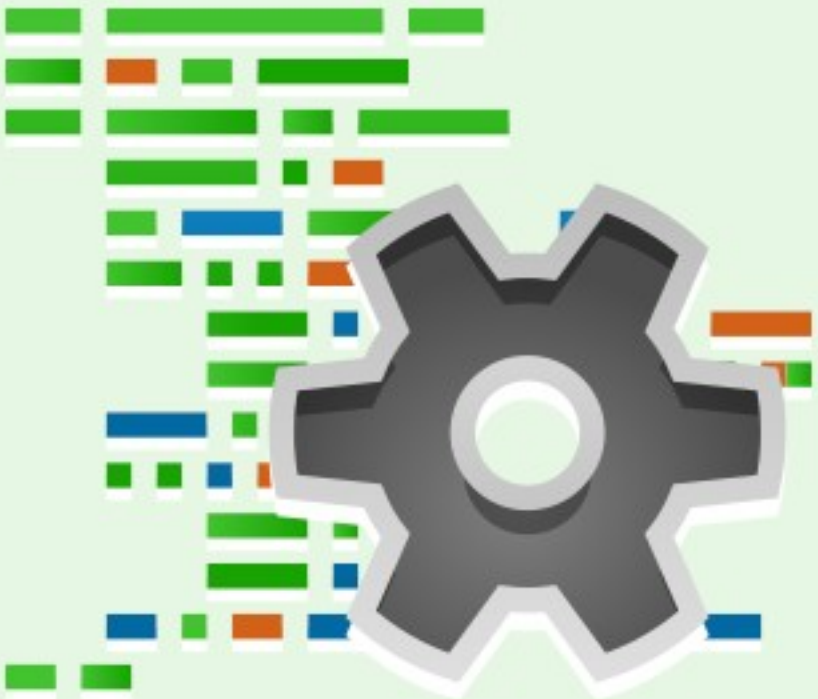
```
Enum MiEnumeracion
  WIND0WS   = 1 ' Windows
  OS2PM     = 2 ' OS/2 Presentation Manager
  MACINTOSH = 3 ' Macintosh
  MOTIF     = 4 ' Motif Window Manager / Unix-like
  OPENLOOK  = 5 ' Open Look / Unix-like
End Enum
```

A los valores enumerados se les asigna el tipo `Long`.


Los nombres de las enumeraciones y sus valores tienen que ser **únicos** en una biblioteca.

Ficha nº 7.
Diálogos

Nivel: Avanzado



Diálogos Basic

Mostrar un mensaje simple

`Print "Hola a todo el mundo !"` (utilizado para depurar)

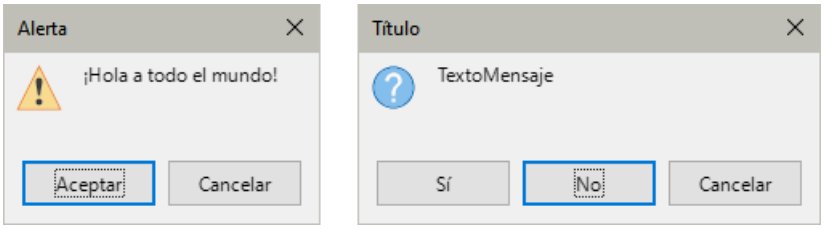
☞ Al pulsar **Cancelar** provoca la detención del programa y la apertura de la ventana del EDI

Mostrar un mensaje de Información

```
MsgBox("TextoMensaje"[, CódigoDiálogo[, "Título"]])
```

☞ Puede usar un salto de linea `Chr(10)` o de párrafo `Chr(13)` en el texto del mensaje.

Figura 9: Mensajes sencillos



Mostrar un mensaje y leer la respuesta

`Respuesta = MsgBox("TextoMensaje"[, CódigoDiálogo[, "Título"]]) donde:`

`Respuesta`: devuelve un entero según la elección del usuario.
`CódigoDiálogo`: botones a mostrar + icono + botón predet. ej: (,3+32+256,) o lo mismo (,291,)

`Botones a mostrar`		
(0) Aceptar (1) Aceptar, Anular	(2) Detener, Reintentar, Ignorar (3) Sí, No, Cancelar	(4) Si, No (5) Reintentar, Cancelar
`Icono`		
(0) ninguno (16) Alerta	(32) Pregunta (48) Avertencia	(64) Información
`Botón predeterminado`		
(0) primero	(256) segundo	(512) último
`Valores de retorno`		
(1) Aceptar (2) Cancelar (3) Interrumpir	(4) Reintentar (5) Ignorar	(6) Si (7) No

InputBox()

`InputBox("Mensaje" [, "Título"[, "Valor predeterminado"]])`

Devuelve una cadena de texto, si se cancela devuelve una cadena vacía.

Diálogos de la API

☞ Los tipos `FilePicker` y `FolderPicker` están relacionados con **Herramienas > Opciones > LibreOffice > General > Usar los diálogos de L.O**.

Tipos de diálogo provistos por la API

Selección de archivo:`FilePicker`

`com.sun.star.ui.dialogs.FilePicker`	Según la configuración de LibreOffice
`com.sun.star.ui.dialogs.OfficeFilePicker`	Fuerza el estilo de LibreOffice.
`com.sun.star.ui.dialogs.SystemFilePicker`	Fuerza el estilo del S.O.
Selección de directorio: `FolderPicker`	
`com.sun.star.ui.dialogs.FolderPicker`	Según la configuración
`com.sun.star.ui.dialogs.OfficeFolderPicker`	Fuerza el estilo de LibreOffice.
`com.sun.star.ui.dialogs.SystemFolderPicker`	Fuerza el estilo del S.O.

El Objeto FilePicker (o OfficeFilePicker o SystemFilePicker)

```
oFilePicker = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
```

`AppendFilter()`	(2 argumentos.) `appendFilter("NombreLiteral", "*.xyz")` `oFilePicker.appendFilter("Documents ODF", "*.odt;*.ods")`
`CurrentFilter`	El filtro predeterm., entre los filtros elegidos por `AppendFilter` (nombre litteral) o el filtro elegido por el usuario.
`DefaultName`	Nombre predeterm. para el archivo a guardar.
`DisplayDirectory`	El directorion inicial o elegido por el usuario.
`Execute`	Transfiere el flujo de ejecución al diálogo y provee de un codigo de retorno (ver constantes de códigos de retorno)
`Files(0)`	El archivo (en modo único archivo)
`getSelectedFiles()`	Array de los archivos seleccionados (en modo multiseleccion).

`initialize()`	Elección del tipo de diálogo (ver constantes de tipo).
	`Dim FPType(0) As Integer`
	`FPType(0) = 'la constante de tipo`
	`oFilePicker.initialize(FPType())`

`MultiSelectionMode`	Permite (`True`) o desactiva (`False`) la selección de varios archivos. (`False` es el predeterminado)
`Title`	El título del diálogo.

Constantes de tipos de FilePicker

`com.sun.star.ui.dialogs.TemplateDescription.XXX:`

`FILEOPEN_SIMPLE`	`0`	Apertura simple.
`FILESAVE_SIMPLE`	`1`	Guardado simple.
`FILESAVE_AUTOEXTENSION_PASSWORD`	`2`	Guardado: extensión automática + contra.
`FILESAVE_AUTOEXTENSION_PASSWORD_FILTEROPTIONS`	`3`	Guardado: extensión automática + contraseña + opciones filtro.
`FILESAVE_AUTOEXTENSION_SELECTION`	`4`	Guardado: extensión automática + selec.
`FILESAVE_AUTOEXTENSION_TEMPLATE`	`5`	Guardado: extensión automática + lista «Plantillas».
`FILEOPEN_LINK_PREVIEW_IMAGE_TEMPLATE`	`6`	Apertura: inserción como enlace + previsualizar + plantilla.
`FILEOPEN_PLAY`	`7`	Apertura: reproducir.
`FILEOPEN_READONLY_VERSION`	`8`	Apertura: sólo lectura + versión.
`FILEOPEN_LINK_PREVIEW`	`9`	Apertura: inserción como enlace + previsualizar.

FILESAVE_AUTOEXTENSION	10	Guardado: extensión autom.
FILEOPEN_PREVIEW	11	Apertura: previsualizar.
FILEOPEN_LINK_PLAY	12	Apertura: inserción como enlace + reproducir.

Códigos de retorno

`com.sun.star.ui.dialogs.ExecutableDialogResults.XXX`

`CANCEL` `0` Cancelar `OK` `1` Aceptar

El objeto FolderPicker (u OfficeFolderPicker o SystemFolderPicker)

```
oFldrPicker = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
```

Description	Texto de ayuda (sobre el diálogo, en un OfficeFolderPicker, no se ve).
DisplayDirectory	Directorio inicial. (Si no se indica, posible Usuario> Documentos)
Execute	Transfiere el foco al diálogo y provee de un codigo de retorno.
Title	El título del diálogo.
Directory	Elegido por el usuario

Abrir un único archivo (FilePicker)

El ejemplo crea un `FilePicker` con el tipo predeterminado (`FILEOPEN_READONLY_VERSION`), inicia el diálogo (propiedades y métodos), lo ejecuta. Lee la respuesta del usuario mediante las propiedades `CurrentFilter`, `DisplayDirectory` y `Files` (array)

> ☞ > (`Files(0)` sólo contiene un valor.

```
Dim oFilePicker As Object, FPTipo(0) As Integer
Dim NombreArchivo As Variant, DirInicio As String
FPTipo(0) = 8 'Con opción solo lectura
  NombreArchivo = ""
  DirInicio = ""
'Iniciación del diálogo FilePicker
  oFilePicker = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
  oFilePicker.Initialize(FPTipo())
  oFilePicker.DisplayDirectory = DirInicio
  oFilePicker.appendFilter("Hojas de cálculo", "*.ods")
  oFilePicker.CurrentFilter = "Hojas de cálculo"
  oFilePicker.Title = "Elige la hoja de cálculo"
'Ejecución y verificación del código de retorno (OK)
  If oFilePicker.execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK Then
' También su puede usar el código retorno:  If oFilePicker.execute = 1 Then
    NombreArchivo = oFilePicker.Files(0) 'resultado (único archivo)
    Msgbox ("Ha elegido "& Chr(10) & NombreArchivo,0+64+0,"Elección")
  End If
```

Abrir varios archivos (FilePicker)

Como lo anterior, Iniciar (en modo múltiple `oFilePicker.MultiSelectionMode = True` y el resultado (un array) se obtiene con `oFilePicker.getSelectedFiles()` que devuelve la selección efectuada por el usuario.

Guardar un archivo (FilePicker)

Como lo anterior, Iniciar en modo guardar `FPTipo(0) = FILESAVE_XXX` y el retorno con un único archivo `oFilePicker.Files(0)`.

Elegir un directorio (FolderPicker)

Crear un `FolderPicker`, iniciarlo (propiedades y métodos indicados), el retorno (lectura de la elección del usuario) se da en `oFoldP.Directory`.

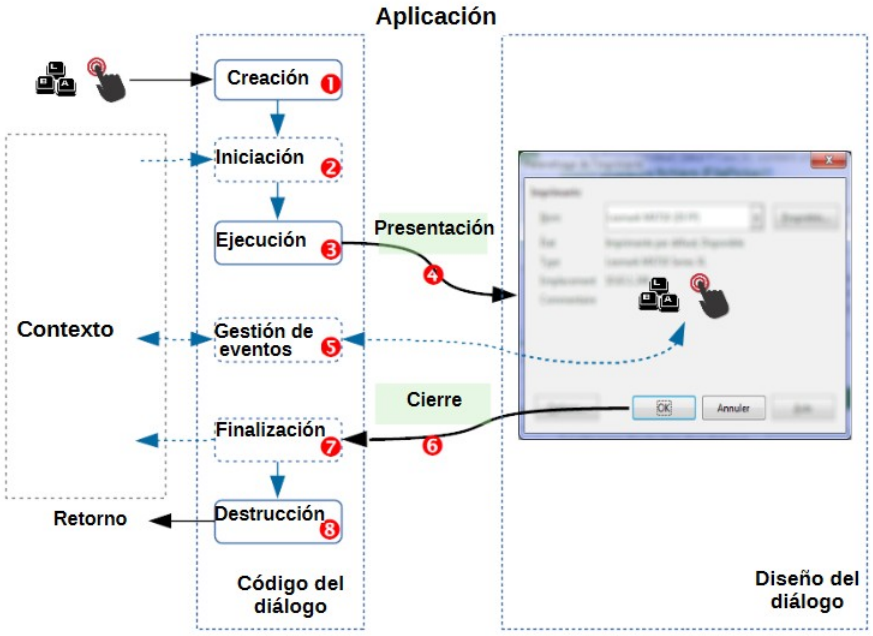
```
Dim oFoldP As Object, NombreDir As String
  NombreDir = ""
  oFoldP = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
  oFoldP.DisplayDirectory = ConvertToURL("C:\Ruta\al\Directorio")
  oFoldP.Description = "Seleccione un directorio"
  oFoldP.Title = "Elige el directorio de backup"
  If oFoldP.execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK Then
    NombreDir = oFoldP.Directory
    Msgbox "Directorio Destino = " & NombreDir
  End If
```

Diálogos personalizados - Principios

Diálogo Basic = un módulo **Diálogo** (diseño) + (al menos) un módulo de **código**

Secuencia de ejecución de un diálogo

Figura 10: Secuencia de ejecución de un diálogo



La secuencia de ejecución se muestra en el gráfico:

1. **Creación** del diálogo en respuesta a un suceso de la aplicación
2. **Iniciación** de los controles desde el contexto de aplicación
3. **Ejecución** el diálogo que recibe el flujo de la ejecución
4. Se muestra en pantalla,
5. Gestión de los **sucesos** de los controles del diálogo
6. Los sucesos provocan **el cierre** del diálogo (**Aceptar, cancelar**);
7. **Finalización** hacia el contexto aplicativo si es preciso (operaciones)
8. **Destrucción** del diálogo y **devuelve código** a la macro que hace la llamada

Creación, iniciación, ejecución, finalización por nuestro código.
Presentación, cierre: operaciones automáticas consecutivas a las precedentes

⚠ Prevenga las respuestas a los sucesos de los controles (Guía ref. n.o 4).

Carga de las bibliotecas de diálogos

☞ Si usa mucho código o varios diálogos, piense en utilizar bibliotecas dedicadas, una por diálogo.

⚠ Las bibliotecas de diálogos no se cargan nunca automáticamente.
Los nombres de las bibliotecas son sensibles a las Mayúsculas !

Modal frente a No modal

Modal	Un diálogo modal toma el control total del teclado, ratón y pantalla en espera de una acción del usuario, la aplicación permanece inaccesible ☞ Los diálogos son modales de manera predeterminada.
No modal	Un diálogo no modal no bloquea el acceso a la aplicación. Ej.: el diálogo Buscar y reemplazar de LibreOffice ⚠ Cuidado con las ejecuciones múltiples de diálogos no modales, ya que pueden bloquear la aplicación.

Diálogos personalizados comunes (Modales)

Tenemos un módulo de diálogo `MiDlg` y un módulo de código `MiCodigoDlg` en la biblioteca `MiLibDlg`. En una subrutina del módulo de código se crea una instancia de un objeto diálogo (`oDlg`) a partir del diálogo.

Creación/carga en memoria

```
DialogLibraries.loadLibrary("MilibDlg")
oLib = DialogLibraries.getByName("MiLibDlg")
oModulo = oLib.getByName("MiDlg")
oDlg = CreateUnoDialog(oModulo)
'y ahora se puede manipular oDlg
```

Sólo Llamada

```
oDlg.execute
```

☞ El flujo de ejecución se tranfiere al diálogo.

Llamada y comprobación del retono

```
If oDlg.execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK Then …
```

☞ El flujo del ejecución se transfiere al diálogo y el valor de retorno comprobado (¿Ha pulsado el usuario el botón **Aceptar?**).

Finalización / destrucción del diálogo

```
oDlg.dispose
```

Ejemplo recapitulativo (módulo de código)

```
Sub MostrarDialogo()
  Dim oLib As Object, oModulo As Object, oDlg As Object
  DialogLibraries.loadLibrary("MiLibDlg")
  oLib = DialogLibraries.getByName("MiLibDlg")
  oModulo = oLib.getByName("MiDlg")
  oDlg = CreateUnoDialog(oModulo)
  'IniciarDlg() 'initializar el contenido del diálogo
  If oDlg.execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK Then
```

```
    'FinalizarDlg() 'Hacer algo con los datos obtenidos
  End If
oDlg.dispose
End Sub
```

Diálogos personalizados no modales

Tenemos un módulo de diálogo `MiDlgNM` y un módulo de código `MiCodigoDlgNM` en la biblioteca `MiLibDlgNM`. En una subrutina del módulo de código se crea una instancia de un objeto diálogo (`oDlg`) a partir del diálogo.

Aplicar los mismos principios que antes con las sustituciones:

1. **La presentación** del diálogo se asegura con `oDlg.SetVisible(True)` en lugar de `oDlg.execute`,
2. Dos variables globales booleanas de control:
   - `gCorriendo` (impide las ejecuciones múltiples).
   - `gMostrar` controla la presencia del diálogo en la pantalla
3. **las respuestas a los eventos** (controles) ponen `gMostrar a Falso cierre diálogo.`

Presentación del diálogo

`oDlg.SetVisible(True)`

El diálogo se muestra.
El flujo de ejecución **NO** se transfiere al diálogo

Ejemplo recapitulativo (módulo de código)

```
'Control presentación del diálogo, declarado al inicio del módulo!
Dim gMostrar As Boolean
'control de ejecuciones múltiples no deseadas también al inicio
Dim gCorriendo As Boolean

Sub MostrarDialogoNoModal() 'gestiona creacción y presentación. del diálogo
  Dim oBibli As Object, oModulo As Object, oDlg As Object
  'évitar ejecuciones múltiples
  If Not gCorriendo Then
    gCorriendo = True : gMostrar = True
    DialogLibraries.loadLibrary("MiLibDlgNM")
    oBibli = DialogLibraries.getByName("MiLibDlgNM")
    oModulo = oLib.getByName("MiDlgNM")
    oDlg = CreateUnoDialog(oModulo)
'IniciarDlg() 'iniciar el contenido del diálogo
'Mostrar el diálogo mientras que gMostrar sea True
    Do While gMostrar = True
      Wait 20  'permite la ejecución de otros programas
      oDlg.SetVisible(True)  'mantenerlo en pantalla
    Loop
    'FinalizarDlg() 'hacer algo con los datos obtenidos si es preciso
    oDlg.dispose
    gCorriendo = False
  End If
End Sub 'MostrarDialogoNoModal

Sub AlPulsarAceptar(ByRef pEvt As Object)
' Instrucciones tras la Réspuesta a un clic en Aceptar
'realizar las acciones necesarias y   'Finalizar con :
  gMostrar = False  '=> fin del bucle while y por consiguiente cierre del diálogo
End Sub 'OnBtnOKClick
```

Asociar un suceso con una macro

Nuestro diálogo se comunica con la aplicación a través de los **sucesos** (**5** del esquema). Habrá que crear entonces las macros que respondan a los sucesos (extracto de Guía ref. n.o 4):

1. **Creamos la macro** a ejecutar siguiendo el modelo:

```
Sub NombreDeLaMacro()
End Sub
```

☞ El nombre de la macro se relaciona con el objeto, acción y el tipo de suceso.
ejemplo: `Sub AlPulsarAceptar()`

La Subrutina puede incluir un parámetro Ver abajo «Obtener informaciones...»,

2. **seleccionamos el objeto** que va a interceptar el evento.
3. Accedemos a su configuración (método variable según el objeto),
4. **elegimos el suceso** a interceptar,
5. **Asignamos la macro** a ejecutar en el desplegable del suceso (punto 1.).

☞ Más informaciones sobre los eventos en el Guía ref. n.° 4.

Obtener informaciones sobre el evento desencadenante

La macro de procesamiento puede consultar el parámetro que recibe para obtener las informaciones complementarias sobre el evento:

```
Sub RespuestaEvento(ByRef Evento As Object)
End Sub
```

La estructura y las propiedades del objeto `Evento` dependerán del tipo de evento que desencadene la llamada al procedimiento

Información de los controles

Para acceder a	Consulta
(Objeto) control que hace la llamada	`Event.Source`
(Objeto) **modelo** del control	`Event.Source.Model`
(Objeto) **diálogo** contenedor del control	`Event.Source.Context`

Iniciación y finalización

Iniciación

El diálogo necesita a menudo informaciones procedentes del contexto de ejecución (**2** en el esquema). La macro de iniciación configura el diálogo a partir de sus datos.

Finalización

El proceso de finalización(**7** en el esquema) implica realizar la operación inversa a la anterior: actualizar los datos contextuales a partir de los ingresados o elegidos en el diálogo

Gestión de los módulos de diálogo

LibreOffice gestiona los módulos de diálogo independientemente del código (Guía de referencia. n.o 1). Es posible copiar estos objetos de un documento a otro.

Copiar un módulo de diálogo de una biblioteca a otra

(en el mismo documento o entre documentos / contenedores)

En el IDE, abrimos los dos documentos/contenedores origen y destino.

Abrimos el **Organizador de macros** (botón ), pestaña **Diálogos**, arrastramos y soltamos desde el origen al destino.

> De manera predeterminada, los diálogos se mueven, para copiarlos pulse Ctrl al desplazarlos.

Guardar una copia de un módulo de diálogo

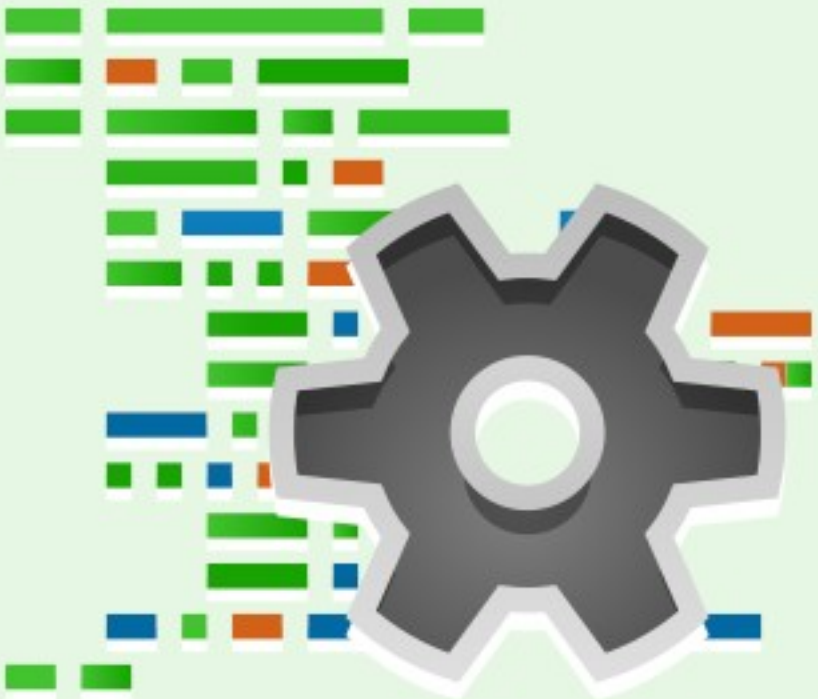
1. En el IDE, abrimos el módulo Diálogo que queremos guardar.
2. Pulsamos el botón  **Exportar diálogo** en la barra de herramientas.
3. Asignamos un nombre al archivo y lo guardamos.

El documento se guarda en formato XML y con la extensión `.xdl`.

> La importación (acción inversa) de hace con el botón  **Importar diálogo**

Ficha nº 8.
Archivos

Nivel: Avanzado



Manipulación de archivos y directorios

Mediante las instrucciones nativas

`Dir()`	Devuelve el nombre de un archivo o directorio o también de todos los archivos y directorios existentes en una unidad o directorio correspondiente a la ruta especificada.
`FileCopy()`	Copia un archivo.
`FileDateTime()`	Devuelve una cadena de caracteres que contiene la fecha y hora de creación o última modificación de un archivo.
`FileExists()`	Devuelve `True` si existe el archivo o directorio.
`FileLen()`	Devuelve el tamaño de un archivo (en bytes).
`GetAttr()`	Devuelve el tipo de archivo, unidad o directorio.
`GetPathSeparator()`	Devuelve el separador de ruta usado en el sistema operativo.
`Kill()`	Elimina un archivo de la unidad.
`MkDir()`	Crea un directorio nuevo.
`Name()`	Renombra un archivo o directorio existente
`RmDir()`	Elimina un directorio existente
`SetAttr()`	Define los atributos del archivo indicado.

Mediante un objeto SimpleFileAccess

Métodos del servicio `com.sun.star.ucb.SimpleFileAccess`:

```
oSFA = createUNOService("com.sun.star.ucb.SimpleFileAccess")
```

`copy`	Copia un archivo.
`createFolder`	Crea un directorio nuevo.
`exists`	Verifica si un archivo o directorio existe.
`getContentType`	Devuelve el tipo de contenido del archivo.
`getDateTimeModified`	Devuelve la fecha de la última modificación del archivo.
`getFolderContents`	Devuelve el contenido de un directorio.
`getSize`	Devuelve el tamaño de un archivo.
`isFolder`	Verifica si una URL es un directorio.
`isReadOnly`	Verifica si un archivo es de solo lectura.
`kill`	Elimina un archivo o directorio. Incluso elimina un directorio aunque no esté vacío.
`move`	Mueve un archivo.
`setInteractionHandler`	Establece un controlador de interacción para operaciones.
`setReadOnly`	Activa el indicador de sólo lectura del archivo (se necesitan los permisos adecuados).

Rutas de los archivos

Al ser multiplataforma, las rutas de los archivos se expresan en formato URL:
`file:///soporte/Ruta/al/archivo.ext`

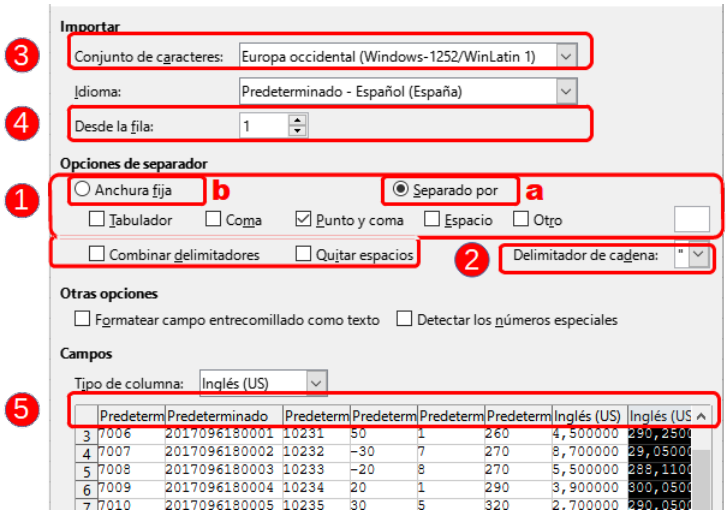
Funciones de conversión:

De nativo a URL	`NombreUrl = ConvertToURL(NombreArchivoNativo)`
De URL a nativo	`NombreOS  = ConvertFromURL(NombreArchivoURL)`

Importación de archivos de texto separados por comas (CSV)

Manualmente

Figura 11: Diálogo de importación de Calc



Elementos de los filtros CSV

Los filtros CSV necesitan 5 parámetros (Tome como referencia el diálogo mostrado).

Opciones de separador:

–*Formato variable (separado por)*

Código Ascii del separador (vea la tabla Ascii en Ficha 5). `"9"` (tabulador)

Si hay alternativas sepárelas mediante la barra inclinada (`/`) `"9/36"`

Si son varios caracteres fusionados agregue /MRG `"36/36/MRG"`

–*Formato fijo (Anchura fija)* `"FIX"`

Delimitador de cadena

Código Ascii del delimitador (vea tabla Ascii) `"34"` (`""` si no hay ninguno)

Conjunto de caracteres: (si se indica "", se asume UTF-8) los más usuales son:

Windows-1252/WinLatin1	`ANSI`	ISO-8859-1	`12`	ISO-8859-15/EURO	`22`	UTF-8	`76`

Lista actualizada de conjuntos: servicio `"com.sun.star.document.FilterFactory"`

Primera linea de datos del archivo a importar (desde la fila)

Número de la línea (empezando por 1) `"2"` (`"1"` o `""` si es la 1ª linea)

Formato de las columnas (tipo de columna)

–*Variable* Por cada columna secuencia de 4 caracteres: rango (base 1) / formato/ (vea la tabla Formatos)`"1/2/ 2/2/ 3/1/"`(`""` si utiliza valores predet.)

–*Fijo* Por cada columna secuencia de 4 caracteres: pos.1er carácter (base 0) / formato / (vea la tabla Formatos)`"0/1/ 8/2/ 13/1/"`

Puede intercalar espacios para hacer la secuencia más legible.
Puede especificar solamente las columnas útiles.

Filtro de importación CSV

El filtro está formado por la concatenación de sus 5 parámetros separados por comas:

```
          ① ② ③ ④      ⑤
Filtro = "9,34,76,1,1/2/2/2/3/1"
```

Según el contexto, determinados valores se pueden omitir (vea detalles de los parámetros):
`Filtro = "59,,76,,"`

Información adicional

Códigos Ascii más frecuentes (notación decimal)

9	Tabulador	32	Espacio	34	"	35	#	36	$	39	'	44	,	58	:	59	;

Formato de las columnas

1	Estándar (automático Calc)	5	YY/MM/DD
2	Texto	9	(descartar columna (ocultar)
3	MM/DD/YY	10	Formato US (sep. decimales y millares)
4	DD/MM/YY		

Importar un archivo CSV en una hoja de cálculo Calc

Se quiere copiar el contenido de un archivo CSV con nombre con nombre MiArchivo.`csv` en la hoja con nombre NombreHoja del libro en uso.

El proceso se efectua mediante el empleo de una memoria intermedia (buffer) y después la eliminación de la misma.

```
Dim CsvURL As String    'ruta del archivo .csv origen
Dim Filtro As String
Dim Hoja As Object      'hoja destino en el libro
Hoja = ThisComponent.Sheets.getByName("Nombre_de_la_Hoja")
CsvURL = ConvertToURL("C:\ruta\a\MiArchivo.csv")
'opciones de lectura del archivo .csv
Filtro = "9,34,ANSI,1,1/2/ 2/2/ 3/1/ 4/1/"
'importación mediante un buffer entre la hoja y el origen .csv
Hoja.link(CsvURL, "", "Text - txt - csv (StarCalc)", _
Filtro, com.sun.star.sheet.SheetLinkMode.VALUE)
'liberación del buffer para volver el libro autonomo
Hoja.setLinkMode(com.sun.star.sheet.SheetLinkMode.NONE)
```

⚠ El contenido anterior de la hoja se **elimina** sin previo aviso.

Crear un libro Calc partiendo de un archivo CSV

Se quiere crear un libro Cal a partir de un archivo CSV con nombre MiArchivo.`csv`.

```
Dim props1(1) As New com.sun.star.beans.PropertyValue
Dim props2()
Dim CsvURL As String    'ruta del archivo .csv origen
Dim DocURL As String    'ruta del archivo .ods destino
Dim oDoc As Object            'Libro destino
CsvURL = ConvertToURL("C:\ruta\a\MiArchivo.csv")
'opciones de lectura del archivo .csv
props1(0).Name = "FilterName"
props1(0).Value = "Text - txt - csv (StarCalc)"
props1(1).Name = "FilterOptions"
props1(1).Value = "9,34,ANSI,1,1/2/ 2/2/ 3/1/ 4/1/"
'carga del archivo origen .csv en la primera hoja
```

```
oDoc = StarDesktop.loadComponentFromURL(CsvURL, "_blank", 0, props1())
'guardado del libro en formato .ods
DocURL = ConvertToURL("C:\ruta\a\MiLibro.ods")
oDoc.storeAsURL(DocURL, props2())
```

El documento creado contendrá sólo una hoja que toma nombre del origen CSV.
En el ejemplo, el documento creado se muestra en pantalla. Para evitar esto:
– utilice la opción `Hidden` (valor `True`) en `props1()`
– agregue `oDoc.close(True)` al final del proceso.

Si existe un documento Calc con el mismo nombre se eliminará sin previo aviso.

Gestión de contenidos – Instrucciones nativas

Pasos del proceso

1. **Obtener** el n.º identificativo interno del archivo con (`FreeFile`),
2. **Abrir** el archivo (`Open`),
3. **Escribir** en archivo (`Print`, `Put` o `Write`) o **leerlo** (`Get`, `Line Input#` ou `Input#`),
4. **Cerrar** el archivo (`Close`)

Acceso al contenido de los archivos mediante su n.º identificativo (*handle)*

`Close`	Cierra un archivo previamente abierto con `Open`.
`Eof()`	Determina si el cursor del archivo ha llegado al final del archivo
`FileAttr()`	Devuelve el modo de acceso en que el archivo se ha abierto (*handle)*
`FreeFile`	Devuelve un n.º *handle* disponible antes de la apertura de un archivo.
`Get`	Lee un registro de archivo para insertarlo en una variable.
`Input`	Lee los datos de un archivo secuencial abierto.
`Line Input`	Lee una linea de un archivo.
`Loc()`	Devuelve la posición actual en un archivo abierto.
`Lof()`	Devuelve el tamaño de un archivo abierto (en bytes)
`Open`	Abre un canal de datos
`Print`	Escribe los datos en un archivo secuencial (linea no delimitada).
`Put`	Escribe un registro en un archivo.
`Reset`	Cierra todos los archivos abiertos y fuerza la escritura en disco del contenido de la memoria intermedia (buffers) de los archivos.
`Seek()`	Devuelve la posición de la siguiente escritura o lectura en un archivo abierto con la instrucción `Open For Random`.
`Write`	Escibe los datos en un archivo secuencial (linea, delimitador).

¿`Print` o `Write`?
`Print` graba el texto mientras que `Write` inserta el texto con delimitadores que caracterizan el tipo de información (texto: `"`, hora y booleano: `#`). Estos delimitadores se desprecian cuando se leen posteriormente con `Input`

Modos de apertura (archivos de texto)

Escritura secuencial `Open For Output`
Lectura secuencial `Open For Input`

Para otros tipos de acceso (acceso directo o binario), utilice el «flujo» de la API.

Ejemplo de lectura secuencial de un archivo de texto

Lectura de un archivo identificado por su dirección `URL`.

```
Dim Handle As Integer, Linea As String, URLArchivo as String
Handle = FreeFile 'obtiene el n.º identificativo del archivo abierto
Open URLArchivo For Input As #Handle
'bucle para lectura linea a linea
Do While Not Eof(Handle)
  Line Input #Handle, Line    'lectura de cada linea
Loop
Close #Handle
```

Ejemplo de escritura secuencial de un archivo de texto

Escritura en un archivo identificado por su dirección `URL`.

```
Dim Handle As Integer
Handle = FreeFile ' obtiene el n.º identificativo de archivo abierto
Open URLArchivo For Output As #Handle
'écritura linea a linea
Print #Handle, "Un Texto."
Print #Handle, "De nuevo otro texto..."
Print #Handle, "Para acabar."
Close #Handle
```

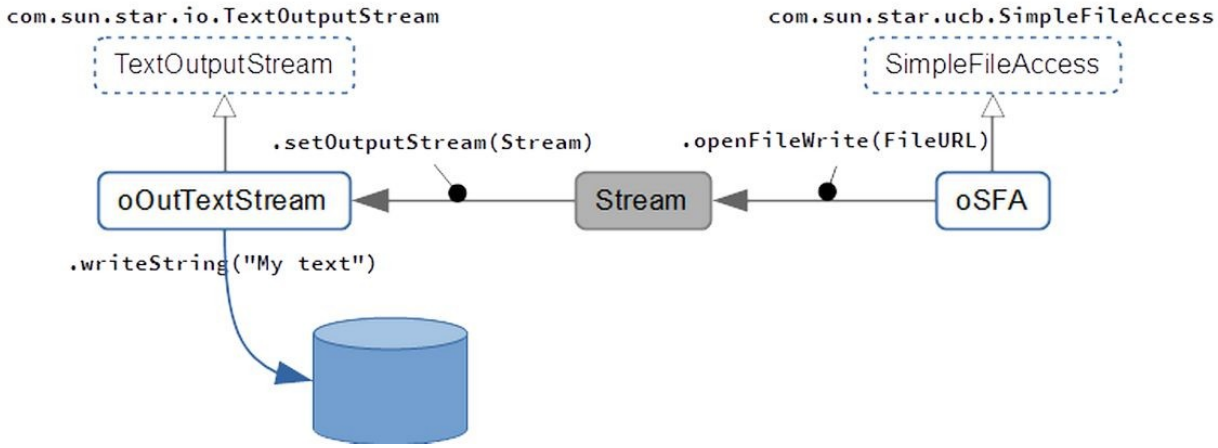
Gestión de contenidos – «flujo» de la API

Usa la llamada a los servicios `SimpleFileAccess` y `Stream` de la API de LibreOffice.

Principio

Ejemplo: (vea el código siguiente).

Figura 12: Escritura de un archivo de texto



Pasos del proceso

1. **Crear el objeto** de acceso a los archivos.
   `oSFA = createUNOService ("com.sun.star.ucb.SimpleFileAccess")`
2. **Conectar** el flujo correspondiente al tratamiento (según tipo de acceso).
3. **Escribir** en el archivo o **leerlo** (según el tipo de archivo).
4. En caso de escritura se debe **purgar** el flujo (`.flush`),
5. **Cerrar** el archivo (`.closeXxx`).

Acceso al contenido de los archivos

Servicio SimpleFileAccess (SFA)

`openFileRead`	Abre el archivo en modo lectura.
`openFileWrite`	Abre el archivo en modo escritura.
`openFileReadWrite`	Abre el archivo en modo lectura y escritura.

Servicios Flujo (InputStream, OutputStream y Stream)

Son los servicios «activos». Correspondencia entre los métodos del servicio `SFA` y los flujos:

Método SFA	Servicio flujo
`openFileRead`	`com.sun.star.io.InputStream` o `com.sun.star.io.TextInputStream`
`openFileWrite`	`com.sun.star.io.OutputStream` o `com.sun.star.io.TextOutputStream`
`openFileReadWrite`	`com.sun.star.io.Stream`

Servicio Stream

`getInputStream`	Devuelve la parte `InputStream` del flujo lectura/escritura. ⚠ El cierre de este flujo implica la del flujo `OutputStream`.
`getOutputStream`	Devuelve la parte `OutputStream` del flujo lectura/escritura. ⚠ El cierre de este flujo implica la del flujo `InputStream`.

Servicio InputStream

`readBytes`	Lee el número especificado de bytes.
`readSomeBytes`	Lee el número de bytes disponibles con el máximo especificado.
`skipBytes`	Salta el número especificado de bytes (valor positivo).
`available`	Indica el n.º de bytes que pueden ser leidos o saltados sin bloqueo
`closeInput`	Cierra el flujo.

Servicio TextInputStream

Heredado de `InputStream`.

`readLine`**:** Lee el texto hasta encontrar un salto de linea (`CR`, `LF`, ou `CRLF`) o `EOF` y devuelve la cadena correspondiente (omitiendo `CR` ou `LF`).

> ⚠ Los caracteres leídos se convierten según la codificación especificada por setEncoding. Si se alcanza `EOF` antes de la ejecución del método devuelve una cadena vacía.

`readString`: Lee el texto hasta encontrar uno de los delimitadores o `EOF`. Y devuelve la cadena correspondiente.

> ⚠ CRLF No es el delimitador predeterminado Si no se ha indicado ningún delimitador o no se encuentra, se supone que es hasta `EOF`.Los caracteres leidos se convierten según la codificación especificada por setEncoding. Si se alcanza `EOF` antes de la ejecución del método devuelve una cadena vacía`.`

`isEOF`: Devuelve el estado de `EOF`.

> ⚠ Este estado no se puede detectar al intentar leer una cadena vacía porque esta respuesta puede ser válida para readLine() (cuando la linea está vacía) y readString() (cuando se suceden dos delimitadores).

`setEncoding`: Establece la codificación de caracteres (`UTF-8 predeterminada`). Los nombres utilizados son los que se encuentran en el documento: http://www.iana.org/assignments/character-sets (columna `Name`). El conjunto de caracteres disponibles dependen de la implementación.

Servicio OutputStream

`writeBytes`	Escribe una secuencia completa en el flujo (llamada de bloqueo).
`flush`	Vacía la memoria intermedia (buffers) usada por el flujo
`closeOutput`	Se utiliza par indicar que se han escrito todos los datos.

Service TextOutputStream (Heredado de `OutputStream`).

`writeString:` Escribe una cadena en el flujo utilizando la codificación definida por setEncoding. Los saltos de linea y delimitadores necesarios se tienen que añadir manualmente a la cadena.

`setEncoding`: Establece la codificación de caracteres (`UTF-8 predeterminada`). Los nombres utilizados son los que se encuentran en la columna. `Name` del documento: http://www.iana.org/assignments/character-sets. El conjunto de caracteres disponibles dependen de la 'implementación.

Ejemplo: Lectura de un archivo de texto

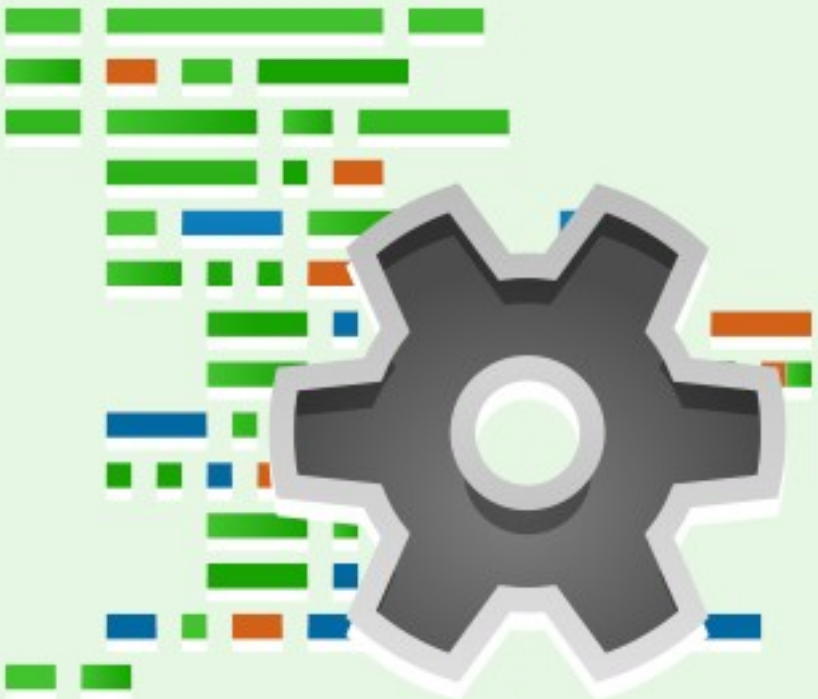
```
Dim oSFA As Object, oInText As Object
Dim URLArchivo As String, Linea As String
oSFA = createUNOService("com.sun.star.ucb.SimpleFileAccess")
URLArchivo = ConvertToURL("C:\ruta\a\MiArchivo.txt")
oInText = createUNOService("com.sun.star.io.TextInputStream")
oInText.setInputStream(oSFA.openFileRead(URLArchivo))
Linea = oInText.readLine()
oInText.closeInput()
```

Ejemplo: Escritura en un archivo de texto

```
Dim oSFA As Object, oOutText As Object
Dim URLArchivo As String
oSFA = createUNOService("com.sun.star.ucb.SimpleFileAccess")
URLArchivo = ConvertToURL("C:\ruta\a\MiArchivo.txt")
oOutText = createUNOService("com.sun.star.io.TextOutputStream")
oOutText.setOutputStream(oSFA.openFileWrite(URLArchivo))
'Escritura (se debe especificar el delimitador de linea CRLF)
'- [empleo de CRLF (Windows)]
oOutText.WriteString("Hola Mundo" & Chr(13) & Chr(10))
oOutText.WriteString("Linea 2" & Chr(13) & Chr(10))
'vaciado de los buffers y cierre
oOutText.flush
oOutText.closeOutput()
```

Ficha nº 9.
Parámetros de ejecución

Nivel: Avanzado



Rutas de acceso que utiliza LibreOffice

Rutas de los archivos de usuario

Son las rutas definidas en **Herramientas > Opciones > LibreOffice > Rutas** . Para acceder a estas, se puede usar el servicio `PathSettings`

```
Dim oPaths As Object, Dirs As String
oPaths = CreateUnoService("com.sun.star.util.PathSettings")
Dirs = oPaths.getPropertyValue("Xxx")
```

⚠ (N. del T. El servicio `PathSettings` está obsoleto desde la versión 4,3 de L. O.y se recomienda usar en su lugar el singleton: `com.sun.star.util.thePathSettings`)

```
Dim oContext as object, oPaths as object, Dirs as string
  oContext = getProcessServiceManager().DefaultContext
  oPaths = oContext.getValueByName("/singletons/com.sun.star.util.thePathSettings")
Dirs = oPaths.Xxx
```

Dirs obtiene una cadena con los directorios de Xxx separados por punto y coma. `Xxx` es la propiedad correspondiente al directorio según la tabla.

☞ En Algunas de las propiedades se pueden aislar sus variantes con prefijos `_internal`, `_user` y/o `_writable` (*Listing 168 OpenOffice Macros Explained)

La propiedad...	... Indica el directorio de:
`Addin`	Complementos de la antigua API.
`AutoCorrect`	Parámetros del diálogo de corrección automática.
`AutoText`	Textos automáticos
`Backup`	Las copias de seguridad.
`Basic`	Archivos Basic utilizados por las macros.
`Bitmap`	Iconos de las barras de herramientas.
`Config`	Archivos de configuración.
`Dictionary`	Diccionarios proporcionados con la instalción.
`Favorite`	Respaldo de los marcadores de las carpetas.
`Filter`	Filtros
`Gallery`	Archivos multimedia y la base de datos de la Galería.
`Graphic`	Se muestra por defecto cuando se abre o graba un gráfico
`Help`	Archivos de Ayuda
`Linguistic`	Archivos para la revisión ortográfica.
`Module`	Módulos.
`Palette`	Archivos de las paletas de colores y estilos (`.sob` y `.sof`).
`Plugin`	Complementos o plug-in.
`Storage`	Información de servidores de correo, noticias y FTP.
`Temp`	Archivos temporales (usado por LibreOffice).
`Template`	Plantillas.
`UIConfig`	Archivos de configuración.
`UserConfig`	Ajustes propios del usuario.
`Work`	Directorio de trabajo (puede estar modificado por el usuario)

La propiedad...	... Indica el directorio de:
BasePathShareLayer	Directorio de configuración compartido([prog_inst]./share/config)
BasePathUserLayer	Directorio de configuración de usuario en red ?(..[usuario]/config)

Ruta de instalación de una extensión o complemento

Use el «singleton» "/singletons/com.sun.star.deployment.PackageInformationProvider"

```
Dim oInfo As Object, Path As String
  oInfo =
GetDefaultContext.getByName("/singletons/com.sun.star.deployment.PackageInformation
Provider")
  If Not IsNull(oInfo) Then
    Path = oInfo.getPackageLocation(ExtID)
  End If
  If (Path <> "") Then Path = Path & "/"
```

ExtID es el identificador único de la extensión (ej: "com.company.NombreExtension")

☞ Path contiene el directorio (formato URL) o vacío si no se encuentra. También puede usar el expansor de cadenas que proporciona la macro UNO_USER_PACKAGES_CACHE

Parámetros de ejecución de LibreOffice

Se dispone de dos servicios complementarios: PathSubstitution y MacroExpander.

Uso del servicio PathSubstitution

```
Dim oSubst As Object, Resultado As String
oSubst = CreateUnoService("com.sun.star.util.PathSubstitution")
Resultado = oSubst.getSubstituteVariableValue("$(nombre_variable)")
```

☞ La variable es una cadena que se debe indicar en la forma: $(nombre_variable)
El resultado devuelto es en formato URL.

Variables sustituibles:

La variable...	... indica:
$(inst)	Directorio de instalación de LibreOffice.
$(prog)	Directorio de acceso al programa soffice.
$(user)	Directorio del usuario (usado por LibreOffice)
$(work)	Directorio de trabajo del usuario En Windows, será Mis Documentos. En Unix, posiblemente «home».
$(home)	Directorio del usuario (sistema operativo). En Unix es «home». En Windows «Documents and Settings\<username>\Documents».
$(temp)	Directorio de archivos temporales en uso.
$(path)	Indica el contenido de la variable de entono PATH.
$(username)	El nombre del usuario de la sesión en curso (en Windows es sin dominio)
$(langid)	El código del idioma utilizado por LibreOffice.Ej: 3082 para el español (España).
$(vlang)	El código del idioma utilizado por LibreOffice, en forma textual. (es = español).

Uso del singleton expansor de (cadenas) macros

Mediante el singletón MacroExpander se pueden obtener /utilizar variables específicas del entorno propias de LibreOffice.

Utilice `"/singletons/com.sun.star.util.theMacroExpander"` y emplee su método `ExpandMacros():`

```
 Dim oContext as Object     'objet contexto
 Dim oMacroExpand as Object     'macro expansor
 Dim Resultado As String
   oContext = getProcessServiceManager().DefaultContext
   oMacroExpand =
oContext.getValueByName("/singletons/com.sun.star.util.theMacroExpander")
  Resultado = oMacroExpand.ExpandMacros("$UNO_USER_PACKAGES_CACHE")
```

El nombre de una cadena macro debe empezar por `$` como en el ejemplo precedente.
La ruta a los directorios o archivos que devuelve están en formato URL.

Macros

Vea : https://wiki.documentfoundation.org/Development/Environment_variables Algunas variables son específicas del sistema operativo y /o parámetros de configuración de LibreOffice.

Las cuatro primeras se suelen utilizar en el archivo de arranque *(bootstrap.ini*:) cuando se utiliza LibreOffice en modo servidor y las siguientes proporcionan algo de información.

`ORIGIN`	Directorio de instalación de LibreOffice.
`SYSUSERCONFIG`	Directorio de la configuración del usuario en su sesión
`UNO_USER_PACKAGES_CACHE`	Directorio de almacenamiento de las extensiones.
`USERNAME`	Nombre de cuenta de usuario.
`OS`	Sistema operativo
`TMP`	Directorio temporal
`PYTHONPATH`	Directorio de Python
`PATH`	Lista de directorios incluidos en el path
`LANGUAGE`	Código completo del idioma (ej: `es_ES.UTF-8`)
`SYSTEMROOT`	Específico de Windows (ej: `C:\Windows`)

Parámetros de la linea de comandos LibreOffice

En la linea de comandos, los parámetros deben ir precedidos por la ruta al ejecutable programa ej. `C:\Program Files\LibreOffice\program\soffice.exe [parámetros].` Aquí sólo se describen las opciones más útiles para el uso de la linea de comandos en «macros».

Encontrará instrucciones completas en la siguiente página (EN) (verificado 04/2021):
https://dnimruoynepo.blogspot.fr/2016/12/command-line-arguments-in-libreoffice.html

Ayuda e información

`--version`	Muestra el número de la versión.
`--nstemporarydirectory`	(sólo en el entorno de pruebas (sandbox) de MacOS X) Devuelve la ruta del dir. temporal del usuario activo. Prioritario sobre todos los parámetros

Parámetros generales

`--quickstart[=no]`	[Dés]activa el arranque rápido. Sólo se admite el valor (`no`), para desactivar el servicio
`--nolockcheck`	Desactiva la verificación de instancias remotas durante la instalación.

`--infilter={filtre}`	Obliga la elección de un tipo de filtro si es posible. Si no es posible LibreOffice utiliza el filtro disponible para el documento.
	Ejemplo:
	`--infilter="Calc Office Open XML"`
	`--infilter="Text (encoded):UTF8,LF,,"`
	Los nombres de los filtros pueden cambiar. Los ejemplos muestran el uso de ciertos parámetros. Desafortunadamente no es facil saber la lista de filtros disponibles.

`--pidfile={archivo}`	Graba el `pid` de `soffice.bin` en `{archivo}`.
`--display {pantalla}`	Establece la variable de entorno al valor`{pantalla}`. (compatible solo en plataformas UNIX)

Control de la interfaz de usuario

`--nologo`	Desactiva el logotipo de presentación inicial.
`--minimized`	Se inicia en modo minimizado, y sin mostrar el logotipo inicial.
`--nodefault`	Se inicia sin ventana de inicio sólo muestra el logotipo
`--invisible`	Se inicia en modo invisible. No se muestra el logotipo ni la ventana del programa LibreOffice, los documentos y los diálogos se pueden controlar y abrir mediante las llamadas a la API. Cuando se utiliza este parámetro, LibreOffice sólo se puede cerrar usando el administrador de tareas (Windows) o el comando `kill` Windows/Unix). `--invisible` no se puede usar con --`quickstart`.
`--headless`	Se inicia en modo «headless» que permite usar la aplicación sin ningúna interfaz. Este peculiar modo se utiliza para controlar la aplicación desde clientes externos mediante llamadas a la API.

☞ `invisible` frente a `headless`.
`--invisible` no desactiva la interfaz gráfica: documentos y diálogos se muestran en la pantalla. `--headless` o «modo silencioso» cuando no se necesita una interfaz gráfica

`--norestore`	Desactiva el reinicio y la recuperación de archivos después de un error del sistema
`--safe-mode`	Arranque en modo seguro. Este modo permite utilizar un perfil nuevo y ayuda a restaurar una configuración corrupta.
`--accept={UNO-URL}`	Permite el uso de una cadena «UNO Accept String» al crear una cadena de «UNO Acceptor Threads» para que otros programas se puedan conectar para acceder a la API.`{UNO-URL}` se compone de `:uno:connection-type,params; protocol-name,params;ObjectName`. De acuerdo con el código LibreOffice `ObjectName` se desestima
`--unaccept={UNO-URL}`	Cierra un canal de aceptación creado por `--accept`. Utilice `--unaccept=all` para cerrar todos los canales

Parámetros para el desarrollador

`--terminate_after_init`	Sale después del inicio (sin carga de documento).
`--eventtesting`	Sale después de la carga de documentos.

Creación de documentos

Estas opciones permiten crear documentos vacíos del tipo especificado, no es posible usar más que un solo tipo por comando. Cuando se especifica un nombre de archivo, LibreOffice intenta abrir ese archivo mediante el módulo indicado. Si no es posible, el archivo se abre con el más conveniente

Las opciones siguientes crean documentos vacíos del tipo especificado:

`--writer`	`--calc`	`--draw`	`--impress`	`--base`	`--math`	`--global`	`--web`

Apertura de archivos

Estos argumentos definen como se tratan los nombres de archivos. Un tratamiento nuevo comienza después del argumento y finaliza con el siguiente argumento. El tratamiento por defecto es la apertura de los archivos para su edición,o la creación de nuevos archivos a partir de una plantilla.

`-n`	Trata los documentos especificados como plantillas para crear documentos nuevos.
`-o`	Abre los documentos especificados para su edición, se trate de plantillas o no.
`--pt {Impresora}`	Envía los documentos especificados a la impresora `{Impresora}`, y después los cierra.(el logotipo de inicio no se muestra). - Si se especifica `{Impresora}` varias veces, sólo se utiliza la última ocurrencia para todos los documentos. Tenga en cuenta que el parámetro `--printer-name` de la opción `--print-to-file` interfere con `{Impresora}`.
`-p`	Envia los archivos especificados a la impresora predeterminada. El logotipo de inicio no se muestra. Si el nombre de un archivo contiene espacios debe estar entrecomillado por comillas dobles.
`--view`	Abre los archivos especificados en modo visualización (sólo lectura).
`--show`	Abre e inicia la presentación inmediatamente para cada uno de los archivos especificados. Los archivos se cierran después de la presentación. Los archivos que no sean de Impress se abren en modo predeterminado, independientemente del modo anterior.

`--convert-to OutExt[:OutFilterName] [--outdir Directorio]`	Conversion de archivos por lotes (`--headless necesario`).
	`OutEx: extensión de los archivos destino`
	`OutFilterName`: filtro de conversion.
	- Si no se especifica `--outdir`, se utiliza el directorio de trabajo en uso como el directorio de salida.
	- Si se utiliza varias veces `--convert-to` Se usará el último valor de `OutExt[:OutFilterName].`
	- Si se utiliza varias veces `--outdir` sólo se toma en cuenta el último valor.
	Ejemplos:
	`-- convert-to pdf *.doc`
	`-- convert-to pdf:writer_pdf_Export --outdir /home/user *.doc`
	`-- convert-to "html:XHTML Writer File:UTF8" *.doc`
	`-- convert-to "txt:Text (encoded):UTF8" *.doc`
	Desafortunadamente no es facil saber la lista de filtros disponibles.
	El uso de esta opción es muy delicada, pese a su gran utilidad.

Una respuesta a la pregunta planteada en: Ask Libreoffice sobre el tema de los filtros enlaza con una lista de filtros disponibles en:
https://cgit.freedesktop.org/libreoffice/core/tree/filter/source/config/fragments/filters
(EN) (verificado 04/2021) o bien
https://github.com/LibreOffice/core/tree/master/filter/source/config/fragments/filters

`--cat`	Envía el contenido de texto de los archivos especificados a la ventana del procesador de comandos (se debe especificar la opción `--headless`). No se puede utilizar con `--convert-to`.

`--print-to-file [--printer-name Impresora_1] [--outdir Directorio_2]`	Impresión de archivos por lotes. - Si no se especifica `--outdir` se utiliza el directorio de trabajo en uso como el directorio de salida. - Si se utilizan varias veces `--printer-name` o `–outdir`, sólo se toma en cuenta el último valor de cada uno de ellos. - Si se espedifican `--printer-name` o `–outdir` varias veces, sólo se utiliza la última ocurrencia para todos los documentos. Tenga en cuenta que el parámetro {Impresora} de la opción} `--printer-name` de la opción `--pt` interfiere con `--printer-name`.
`-env:var[=valor]`	Establezca una variable de bootstrap. Por ejemplo para establecer una ruta de perfil de usuario: `-env:UserInstallation=file:///tmp/test` Tampoco es facil conocer la lista de variables disponibles

Llamada a una macro desde la linea de comandos

Sintáxis

☞ La opción `--headless` facilita la ejecución silenciosa (vea abajo).

Macro global

```
{soffice} "macro:///biblioteca/modulo/macro[(parametros)]"
```

Macro en un documento ODF

```
{soffice} ruta/a/doc.odf "macro://./biblioteca/modulo/macro[(parametros)]"
```

☞ `- {soffice}` (según sistema operativo):
Windows: `%programfiles%\libreoffice\program\soffice.exe`
GNU/Linux:`/opt/LibreOffice/program/soffice`

Instalar una macro... mediante una macro

Instalará la biblioteca que contiene la macro, (con la macro incluida).

Principio

Un archivo «contenedor» (Writer, Calc, etc.) contiene la macro que desea instalar y una macro de instalación.

- La macro de instalación debe estar en la biblioteca `Standard` del documento.
- la macro a instalar debe estar aislada de la *«instaladora»*, es decir, en su propia biblioteca: la biblioteca que se va a instalar.

☞ Salvo algún caso en particular, el tipo de archivo «contenedor» (Writer, Calc, etc.) no contiene ninguna información de la macro para instalar. Writer es un buen soporte puesto que permite **documentar** la instalación.

La macro que se quiere instalar

Colóquela en su propia biblioteca en el documento contenedor.

La macro instaladora

Su papel es copiar la biblioteca que se desea instalar al contenedor global `Mis Macros`. Se muestra un ejemplo del proceso de instalación de una biblioteca de tipo «código Basic»:

```
Dim oSrcLib As Object  'biblioteca origen (en el documento)
Dim oDestLib As Object  'biblioteca destino (en 'Mis Macros')
Dim i As Integer
```

```
Dim SrcModules() As String
'se crea la biblioteca destino si no existe
If Not GlobalScope.BasicLibraries.hasByName("bibl destino") Then
  GlobalScope.BasicLibraries.createLibrary("bibl destino")
End If
'se copian los módulos
If BasicLibraries.hasByName("bibl origen") Then
  BasicLibraries.loadLibrary("bibl origen")
  oSrcLib  = BasicLibraries.getByName("bibl origen")
  oDestLib = GlobalScope.BasicLibraries.getByName("bibli destino")
  SrcModules = oSrcLib.getElementNames()
  'instalación de módulos uno a uno
  i = LBound(SrcModules())
  Do While (i <= uBound(SrcModules()))
    If Not oDestLib.hasByName(SrcModules(i)) Then
      oDestLib.insertByName(SrcModules(i), oSrcLib.getByName(SrcModules(i)))
    End If
    i = i + 1
  Loop
End If
```

Para instalar una biblioteca de tipo «diálogo», remplace `BasicLibraries` por `DialogLibraries` y `GlobalScope.BasicLibraries` por `GlobalScope.DialogLibraries`.

Crear una extensión:

Consiste en empaquetar sus macro en forma de extensión para una mejor difusión.

Es una tarea bastante complicada. Consulte la página de la wiki Desarrollo de extensiones y El capítulo 4 de «Developers Guide» (en inglés)

LibreOffice

https://es.libreoffice.org/