RIB | Presto

Tutorial de programación de Presto

con Visual Basic Scripting VBS y OpenAI

Índice

Tutorial de programación de Presto 3
Puesta en marcha 3
Abrir una obra 4
Gestión de errores 4
Leer datos de la obra 5
Ventana de mensajes 6
Modificación de la obra 6
Guardar los cambios 7
Revisión final 8

Tutorial de programación de Presto

con Visual Basic Scripting VBS y OpenAI

Los complementos, plugins o macros son programas que pueden intervenir en una obra de Presto para:

- Extraer información y utilizarla de diferentes maneras
- Modificar el contenido de la propia obra

Pueden realizarse en muchos lenguajes de programación pero aquí nos vamos a centrar en el desarrollo con VBS.

Puesta en marcha

La única herramienta realmente necesaria es un editor de texto plano. Se podría usar el “Bloc de notas”, pero es mucho más cómodo un editor especializado en código, por las facilidades que da. Aquí vamos a utilizar “Notepad++”, disponible gratuitamente.
Aparece un archivo vacío. Si no es así usamos “Archivo: Nuevo”.
Para que los acentos y caracteres especiales se mantengan al pasar de un editor a otro elegimos:

“Codificación: Juego de caracteres: Europeo occidental: Windows-1252”.

Para que reconozca las funciones y variables y las muestre en color elegimos:

“Lenguaje: V: Visual Basic”.

Es conveniente escribir una primera línea con acentos para que al volver a abrir el complemento veamos enseguida si se ha reconocido bien.

```
' áéíóú
```

Empieza con la marca “’” de comentario, que indica que no es una instrucción propiamente dicha.

Para programar hay que tener claro el objetivo o la funcionalidad que queremos.

La siguiente línea de cabecera debe contener otro comentario con esta utilidad:

```
' Ejemplo de complemento para adaptar a otros usos
```

Ya podemos guardar el archivo con “Archivo: Guardar”, copiando ese mismo nombre. La extensión debe ser “.VBS”, que no es la de defecto “.vb”.
El directorio puede ser cualquiera en el que podamos escribir, pero para que Presto lo reconozca debe ser uno de los directorios de “Entorno de trabajo: Directorios”:

- “Complementos”. Si está accesible, es más cómodo, porque aparecerá directamente en la lista.
- “Complementos [Usuario]”. Aparecerá en “Complementos de usuario”

Presto lee la lista de complementos al abrirse. Si lo ha guardado después, cierre y abra Presto o acceda a “Entorno de trabajo” y seleccione el directorio adecuado de nuevo.

Abrir una obra

Abrimos la obra “Modelo Educativo de Revit”.
Creamos un objeto, o una clase, que nos permitirá trabajar con los datos de una obra. Esta obra puede ser la obra abierta, una referencia o cualquier obra guardada, pero aquí vamos a trabajar con la obra abierta. Para ello escribimos:

```
Set Obra = GetObject ("", "Presto.App.18")
```

Dependiendo del editor, veremos que reconoce y colorea distintos elementos, como “Set” o lo que esté entre comillas. Esta es una de las ventajas de usar un editor de textos especializado.
La sintaxis de esta instrucción “Set” no proviene de Presto, sino de Visual Basic, y puede ser distinta en otros lenguajes.

- “Obra” es el nombre que damos a esta obra exclusivamente para su uso en el complemento. Si abrimos varias obras cada una tendrá un nombre distinto.
- El parámetro “” indica que vamos a acceder a la obra abierta. Con otros valores se puede acceder a las referencias o cualquier obra accesible.
- "Presto.App.18" vale para todas las obras desde Presto 18 en adelante.

Guardamos el complemento, normalmente con [Ctrl][S].

Gestión de errores

Como siempre que se programa, ejecutaremos el complemento con cada cambio para comprobar que no hay ningún error.
Si aparece un mensaje de error nos muestra la línea y la columna, es decir, la posición en la que se detecta, y una breve indicación de lo que puede estar funcionando mal. Hay que revisar primero cuidadosamente esa línea y, si es necesario, todo el entorno o el programa completo, porque podemos cometer muchos tipos de errores distintos, de VBS, de Presto, de lógica, etc.
Una de las desventajas de VBS es que la única forma de saber los valores de las variables y de detectar errores es insertar mensajes en los puntos deseados del complemento, que irán a esa la ventana de mensajes. Ahora bien, por la forma en

que Presto gestiona esta ventana, hay que tener en cuenta que se escriben todos juntos, al final del proceso del complemento.

Leer datos de la obra

Todas las funciones disponibles están en el documento “API Funciones detalladas”.

Ahora queremos leer información de la obra. Puede ser útil generar o abrir previamente un informe que recorra la obra de la manera que buscamos, porque el sistema del complemento se parece mucho. Para abrir la primera tabla en un informe usaríamos una sección “Elemento 1”, que aquí es la función “Set Element”.

```
Obra.SetElement 1, "Conceptos","Conceptos.Código" , """*""", "Conceptos.Nat == 5 && Conceptos.Pres > 0"
```

- El primer parámetro es el mismo número de la sección “Elemento 1” de los informes, permite abrir y enlazar varias tablas.
- Tabla que queremos abrir
- La tabla “Conceptos” tiene dos campos clave, “Código” y “Nat”. La clave afecta al orden de recorrido y a la máscara.
- La máscara se aplica a la clave elegida y solo se recorren los registros que la cumplen. Sirve para filtrar por iniciales o para probar el complemento con una selección pequeña. Tiene que llegar a Presto incluyendo las comillas. Como a su vez tiene que ir encerrada entre comillas, se puede usar una comilla doble a cada lado, que representa una sola comilla, o bien escribir “chr(34) & “*” & chr (34)”. La máscara deseada se escribe a la izquierda del carácter “*”.
- El quinto parámetro es un filtro opcional. Se puede usar cualquier expresión válida de Presto para la tabla abierta y es muy útil crear el filtro antes en Presto y comprobarlo. Simplifica el complemento porque sólo van a leerse los conceptos que lo cumplan, aunque tiene que pasar por el concepto para verificarlo. En este ejemplo, leemos los conceptos de tipo partida (naturaleza “5”) y que tengan precio.

Esta instrucción dice que elementos queremos leer y de qué tabla pero no los lee todavía. Para ello usamos:

```
Obra.GetElement(1)
```

El número es el mismo que hemos usado antes para la tabla. La expresión valdrá “0” mientras queden elementos por recorrer. La tenemos que insertar en un bucle que leerá uno por uno todos los registros de la tabla. En este caso usamos un bucle “While”, que se realiza siempre una vez y en las siguientes se comprueba antes la condición.

```
While Obra.GetElement(1) = 0
…
Wend
```

En el medio escribimos lo que queremos hacer con cada registro. Aquí vamos a leer el código de cada elemento y asignarlo a una variable “Codigo” con:

```
Codigo = Obra.GetField “Conceptos.Código”
```

Observe que la variable del complemento no puede tener acentos, porque el lenguaje no lo permite, pero la variable de Presto sí debe llevarlo.

Ventana de mensajes

Vamos a mostrar en la ventana de mensajes los códigos leídos, junto con el resumen y el precio.

```
Obra.LogMsg Codigo, 0
```

El último parámetro puede ser también 1 (azul) y 2 (rojo).

La ventana de mensajes puede moverse a un marco independiente para consultarla con comodidad.

Es recomendable mostrar los resultados en esta ventana y modificar los registros sólo cuando estemos seguros de que el resultado es el deseado. Esta ventana se puede usar también para generar listas o para crear exportaciones, copiando y pegando el resultado en un editor de texto para guardarlo como archivo.

Modificación de la obra

Los complementos suelen realizar acciones que no se pueden hacer fácilmente con filtros y expresiones de Presto. Como ejemplo, vamos a poner en minúsculas todos los resúmenes, menos la primera letra y las palabras que tengan números. Abrimos la versión gratuita OpenAi 3.5 y preguntamos algo parecido a esto:

```
“En VBS quiero poner en minúsculas un texto, menos la primera letra y las palabras que tengan números”
```

Pegamos directamente el código obtenido en el complemento, después del bucle principal, porque se trata de dos funciones. No necesitamos entenderlas, solo enviar el texto que queremos convertir y recibirlo en una variable:

```
ResumenNuevo = ConvertirATextoMinusculo(Resumen)
```

Vamos a mostrarlo como mensaje.

```
Obra.LogMsg ResumenNuevo, 0
```

Vemos que convierte a minúsculas también la primera letra.

Las respuestas de OpenAI no son perfectas y varían con el contexto, puede dar un resultado diferente en otras ocasiones.

Tenemos dos opciones, preguntar de nuevo a OpenAI o corregir directamente el problema, que puede ser más fácil.

```
ResumenNuevo = UCase (Left(ResumenNuevo, 1)) & Mid (ResumenNuevo, 2, Len(Resumen))
```

Funciona bien, pero hay muchas abreviaturas que no se reconocen porque no tienen espacio en blanco detrás del punto. Podemos añadir este espacio:

```
ResumenNuevo = Replace (Resumen, ".", ". ")
```

Si se copia y pega este texto hay que comprobar que las comillas son las que espera el editor. Y enviamos esta variable en lugar de "Resumen".
El resultado es mejor y más legible, pero aun así podríamos revisar:

- Palabras que no deberían cambiar, como "UNE", "PVC"
- Cifras con un punto decimal, donde aparecería también un espacio.

Nos detenernos aquí, pero este es un buen ejemplo de cómo la programación, sin ser difícil, puede requerir mucho tiempo hasta obtener exactamente el resultado deseado.

Guardar los cambios

Como en este proceso no nos hemos movido del concepto leído no hay que buscarlo de nuevo. Basta con utilizar la expresión opuesta a "Getfield", añadiendo el nuevo valor:

```
Obra.SetField "Conceptos.Resumen", ResumenNuevo
```

Y actualizar el registro con:

```
Obra.UpdateRecord ("Conceptos")
```

Pero no vamos a ejecutarlo todavía, hasta añadir una mejora, activando el sistema de deshacer de Presto. Para ello hay que insertar dos funciones, una antes de empezar a modificar registros y la otra al final del código:

```
Obra.BeguinRedo
```

```
Obra.EndRedo
```

Si por lo que sea el complemento se interrumpe antes de ejecutarse por completo y se ha ejecutado "BeginRedo", pero no "EndRedo", el sistema deshacer de Presto

quedaría bloqueado. Para evitarlo usamos un pequeño truco de programación. Al principio del complemento ponemos:

```
PrimeraVez = True
```

Justo antes de modificar un registro por primera vez insertamos el inicio:

```
If PrimeraVez = True Then
    Obra.BeginRedo
    PrimeraVez = False
End If
```

Y al terminar el complemento insertamos el cierre:

```
If PrimeraVez = False Then
    Obra.EndRedo
End If
```

Así hay más seguridad de que o se ejecutan ambas funciones, o ninguna. Ya lo podemos ejecutar y comprobar.

Revisión final

Para terminar, es fundamental revisar el complemento y dejarlo de manera que cualquier otro, y nosotros mismos en el futuro, podamos entenderlo y modificarlo. Por ejemplo, añadimos un mensaje al principio para mostrar qué hace y que el usuario pueda aceptarlo o cancelarlo.

```
Descripcion = "Convierte los resúmenes de partidas a minúsculas respetando la inicial y palabras con números"
Respuesta = MsgBox (Descripcion, vbOKCancel )
If Respuesta <> vbCancel Then
```

Y añadimos el cierre de la condición al final del código.

```
End if
```