Programación con Objetos en 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 con Objetos: Introducción

Visual Basic Script es un lenguaje de programación de alto nivel no compilado para Windows, y eso, solo quiere decir una cosa: que como la mayoría de lenguajes para Windows de este tipo, tiene acceso al uso de **Objetos**, también llamados **ActiveX**.

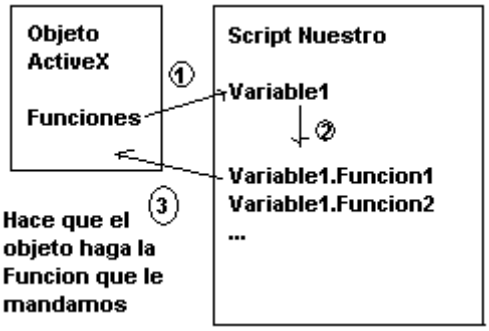
Antes que nada hay que comprender bien que es un Objeto y para que sirve. Estoy seguro que a todos más o menos os suena la palabra **Librería**. Son los archivos con extensión (que acaban en…) "**.dll**", y que sin ellos los programas no van. Pues bien, los Objetos, podríamos decir que son una derivación de las librerías, ampliadas y mejoradas. Normalmente **una librería tiene tan solo funciones**, a las que un programa llama. Es como si tuviéramos un script con un montón de funciones creadas por nosotros mediante la sentencia Function, y todos los scripts que creáramos en nuestra vida usaran esas funciones. Para que sea más fácil y ocupe menos, estas funciones se separar en una librería (dll) para que no se tengan que incluir en todos los programas. Sin embargo, como he dicho, los **Objetos** o **ActiveX** son un derivado de las librerías y pueden (y suelen) contener mucho mas que simples funciones. Los objetos **contienen trozos de programas enteros**. Pongamos un ejemplo, tú haces una Base de Datos para una empresa, y dicha base de datos tiene un sistema que hace que el programa se actualice (como lo tienen generalmente muchos programas); pues perfectamente el sistema de actualización entero (con sus ventanas y todo) puede ser un Objeto que este ya hecho y que tu solo utilices en tu programa. Como se puede ver es más sencillo que hacer tú todo el sistema de actualización.

Pues bien, Visual Basic 6 es un lenguaje basado en objetos que los usa al 100%, pero ya veremos en su momento lo que eso significa; lo que nos interesa ahora es que a diferencia de este, **Visual Basic Script solo nos permite utilizar las funciones** (y por supuesto no nos permite acceder a Librerias) de los Objetos, nada más. Esto se debe a que VBScript no soporta introducir elementos de programas discretos en un script que es secuencial, pero la razón no importa ahora, es así y nos tenemos que conformar. De esta manera pues, podríamos aprender a utilizar las muchísimas funciones que hay en los muchísimos objetos que existen (ya que de echo, con VB6 se pueden crear aún más Objetos), pero nosotros nos vamos a centrar en los Objetos simples del sistema; o lo que es lo mismo, en los objetos más comunes, estandarizados y que han sido diseñados exclusivamente pensando en Visual Basic Script. Estos son:

- **Scripting.FileSystemObject**
- **WScript.Shell**
- **Scripting.Dictionary**

Por supuesto existen miles de objetos más, pero esos tres nos van a dar casi el control total sobre el sistema, además de que son los más adecuados para VBScript. Ya cuando se vea VB6 veremos otros como **MSWinsock.Winsock** que nos permitirán crear aplicaciones que trabajen en red.

Bien, la forma de trabajar con Objetos es la de crear una instancia del objeto en una variable mediante un método especial (sería algo así como copiar la lista de funciones del objeto a la variable), y luego usar esa variable para utilizar las funciones del objeto:



Como podemos comprobar es el objeto quien hace la función, lo único que pasa es que al haberlo copiado nosotros en Variable1, es como si se tratara de una función de nuestro script. Más o menos esta es la teoría, pero vamos a ver como se hace en realidad esto (pongo todo el código para que lo veáis claro, pero a partir de ahora omitiré el proceso de creación de variables por motivos de brevedad):

```
Option Explicit
Dim fso, ws, dic
Set fso = CreateObject("Scripting.FileSystemObject")
Set ws = CreateObject("WScript.Shell")
Set dic = CreateObject("Scripting.Dictionary")
```

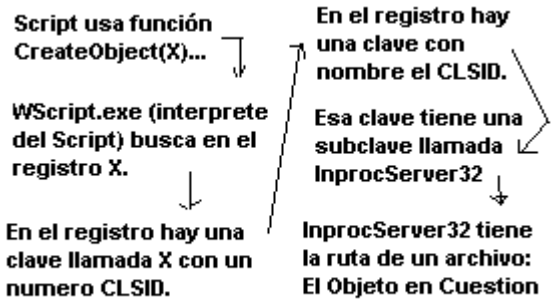
Este es el código necesario para realizar el Paso 1, es decir, crear una instancia del objeto en una variable de nuestro script. Analicemos la situación: Primero creamos tres variables con nombres "**fso, ws y dic**", hasta ahí normal. Luego, vemos que usamos la **Función CreateObject**, que tiene un argumento que se corresponde a un valor de cadena que indica el **nombre del objeto** que queremos utilizar. De esta forma, la variable fso esta lista para utilizar las funciones del objeto *Scripting.FileSystemObject,* la variable ws las del objeto *WScript.Shell* y dic las de *Scripting.Dictionary.* Hasta ahí tan solo es aprender una nueva función. Pero vemos algo más, vemos un **Set** por ahí que no sabemos muy bien que hace, pues hasta ahora nos servía sin el. La razón de poner ***Set*** *fso = CreateObject("Scripting.FileSystemObject")* y no *fso = CreateObject ("Scripting.FileSystemObject"),* como habíamos hecho hasta ahora reside en el tipo de valor que le damos a la variable. La variable utilizada para la función CreateObject, no recibe ningún valor conocido hasta ahora: recibe un valor binario, y para indicar eso debemos poner ese **Set** delante. Y eso es todo para copiar un objeto a una variable.

Ya sabemos realizar el primer paso, pero solo con eso no tenemos nada. Debemos aprender a utilizar los objetos. Y para eso, nada mejor que un ejemplo teórico:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set ws = CreateObject("WScript.Shell")
'
'Codigo del script
'
Variable = fso.función(...)
Variable2 = ws.función(...)
```

El hecho de que sea un ejemplo teórico es que aún no he enseñado ninguna de las funciones que estos objetos poseen, así que más vale verlo, por el momento así. Lo que hay es muy simple: al principio copiamos a dos variables dos objetos, luego tenemos nuestro script, hasta que en un momento determinado necesitamos usar esas funciones. Como a casi todas las funciones, se les **asigna una Variable** que recibe el valor resultante de la función; y **poseen X argumentos** que varían de unas a otras. Lo único que hay que destacar es que por ejemplo si la función se llama *pintar,* no pondremos *Variable = pintar(argumentos)*, sino *Variable = fso.pintar(argumentos),* pues como he explicado **es la variable fso** (o ws si se diera el caso) **quien tiene la función** (porque la ha copiado del objeto). Por supuesto cada Objeto tiene sus propias funciones que son diferentes entre si.

Siempre me gusta llegar hasta el fondo de la cuestión, así que voy a explicar como el script encuentra al Objeto (que nosotros no sabemos donde esta). En este momento recomiendo que si no sabes que es **el registro de Windows (regedit.exe)** , te informes antes de continuar.



En el esquema vemos el proceso más o menos simple que sigue, y yo especifico que en el paso 2 y 4 donde busca es en la rama *HKEY_CLASSES_ROOT*. Si echáis un vistazo veréis la barbaridad de objetos que existen, la mayoría de los cuales son privados de programas y no pueden utilizarse para programar. Por otro lado antes comente que los objetos son **archivos acabado es ".ocx"**, pero si hacéis la prueba con estos tres objetos veréis que son dll. Esto se debe a que realmente estos objetos son Librerias que han sido adaptadas a VBScript simulando que son objetos, aparte que fueron de las primeras que se hicieron y por tanto aún no se diferenciaban del todo. Aún así funcionan a la perfección y **vienen de serie** con todos los Windows hasta la fecha.

Bien pues, antes de dar por finalizada la introducción y pasar ya a la practica, he de introducir aquí una excepción con un objeto que **solo se puede usar en VBScript** (yno en VB6, es el único), y **no requiere copiarlo a una variable** pues se trata del objeto **WScript**, que es en definitiva, darle ordenes al interprete que usamos. Si bien, nos servirá para ver ejemplos sencillos de uso antes de empezar con lo fuerte, además de que interesantes de verdad este objeto solo tiene dos funciones.

La primera es la función **Sleep**. Esta función no requiere declarar ningún objeto simplemente porque el objeto que usa es el propio **WScript**. Esta función se encarga de realizar una pausa en la ejecución del script durante el tiempo en milisegundos que le indiquemos (parando obviamente al intérprete). De esa forma podemos hacer una pausa

entre dos MsgBox o darle tiempo a los comandos que lo requieren. Se utiliza como muestra el ejemplo siguiente, que pausaría la ejecución un segundo:

```
WScript.Sleep 1000
```

Como vemos utilizamos la variable autodefinida **Wscript** seguido de un punto y la función (su nombre) y a continuación un argumento numérico que expresa tiempo en milisegundos. Podemos ver como no se le ha asignado a la función ninguna variable, ya que no devuelve ningún valor, simplemente para el tiempo y ya. Es necesario hacer mención (brevemente) ya a la diferencia entre asignar una variable a una función y no hacerlo:

```
Msgbox "Me llamo Juanda"
Variable = Msgbox ("Me llamo Juanda")
```

Esta es la función MsgBox, en el primer caso sin que le asignemos una variable (en tal caso el valor se pierde), y en el segundo, asignándole la variable, en la que recogemos el valor dado. Como podemos ver, la diferencia radica en que **si no asignamos variable** debemos **suprimir los paréntesis**, nada más. Por otro lado, seguro que os habréis dado cuenta que muchas funciones (como Msgbox) pueden omitírseles varios argumentos. De hecho, en este ejemplo solo le ponemos uno. Para saber de que funciones que **campos son omisible**s consulta alguna Referencia de Funciones de VBScript. Pero en definitiva, no es nada importante, tan solo cuestión de detalles que pueden hacer que un código sea menos limpio y hecho más rápido.

Pero continuemos con lo nuestro, ***la segunda función*** es la función **Quit**. Esta función no requiere declarar ningún objeto simplemente porque el objeto que usa es WScript. Esta función se encarga de finalizar la ejecución del script enviando el código de error que le indiquemos (si el numero del error no existe no se enviara error). De esa forma podemos f**inalizar el script cuando se cumpla una condición** determinada sin preocuparnos si hay más código o no a continuación. La **funcion Quit** tiene la sintaxis siguiente:

```
WScript.Quit(1)
```

Y eso es todo en lo que a excepciones fuertes en objetos se refiere. Son en definitiva dos funciones que puede llegar a resultar bastante útiles, no lo olvidemos. Por otro lado, a partir de ahora he de decir que voy a mostrar los objetos **Scripting.FileSystemObject** y **WScript.Shell** mezclados, ya que lo que nos interesa son las cosas que podemos hacer con ellos y no una lista completa de sus funciones. Pero dejaré aparte el **Scripting.Dictionary**, al tratarse de un objeto un tanto diferente a los otros dos. Comentar también que no es necesario copiar cada vez que se quiera usar una función los objetos en las variables, basta con que lo copiemos una vez para poder usar todas las funciones que queramos. Y con esto, doy fin a esta pequeña introducción.

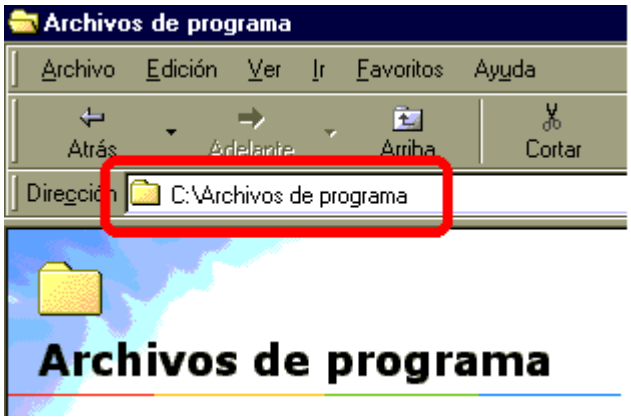
Programación con Objetos: Trabajando con Archivos

Bien. Ahora vamos a ver un grupo de funciones pertenecientes a ambos objetos mencionados en la introducción, que permiten realizar las **tareas básicas de tratamiento de archivos**. En otras palabras, aprenderemos a que nuestro script pueda borrar, copiar, mover, renombrar, etc cualquier tipo de archivo que tengamos. Para ello, veamos las diversas funciones que tenemos a disposición:

La primera es la función **DeleteFile**. Esta función borra el archivo que le indiquemos. El argumento **True** indica que si se eliminan los archivos con el atributo de solo lectura y **False** que no lo hace, por si no queremos borrar algo que sea solo lectura. La sintaxis "c:\Doc2.txt" indica la ruta y el archivo que se va a eliminar.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.DeleteFile "c:\Doc2.txt", True
```

Al ser esta la primera función vamos a darle una explicación especial, que se aplica de forma genérica a todas las demás funciones. En primer lugar, vemos como asignamos a la variable fso el objeto de esta función: **Scripting.FileSystemObject**. A partir de esto debéis ser capaces de **saber de que objeto es cada función**, pues en los ejemplos incluyo el proceso de copia del objeto a la variable con ese fin. Esto lo hemos visto bastante y esta claro. Luego, es la segunda línea tenemos a la función **DeleteFile** (que nace de la variable fso, pues ella quien la tiene ahora), con dos argumentos que se ha explicado que son y hacen cada uno. El primero de ellos, la **ruta del archivo**, si no sabes como colocarla (más vale que te asegures de aprenderlo bien), solo ves a la carpeta por el explorador y mira la barra de dirección:



Y luego mira el nombre del archivo con extensión incluida. Luego lo juntas todo, de forma que si tienes "*C:\Archivos de Programa*" y el archivo se llama "*Documento.doc*" (fíjate que ponemos también la extensión ".doc"), lo que debes poner es la unión: "*C:\Archivos de Programa\Documento.doc*". No será difícil cuando le cojas la práctica. Por otro lado, este es un ejemplo del uso de la función **sin recoger en una variable el resultado**, pues lo único que hace es borrarlo y punto (si no se pudiera provoca un error en tiempo de ejecución, así que es bueno tener el **On Error resume Next** a mano). Bueno, en realidad devuelve **True** si lo consigue y **False** si no, así que como practica dejo que pruebes tú a asignarle una variable y comprobar si lo que digo es verdad con el **Msgbox**. Esto también se aplica a la mayoría de las funciones siguientes.

La segunda es la función **CopyFile**. Esta función copia el archivo que le indiquemos. La sintaxis "*c:\Doc2.doc*", indica la ruta y el archivo que se va a copiar y la sintaxis **"***c:\Mis Documentos\a.doc"* indica la ruta y el nombre a donde se copia el archivo. Un total de dos rutas, una de origen (la primera) y la otra de destino. Lo demás es igual que en la anterior: se debe usar una variable que contenga el objeto con el método explicado.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CopyFile "c:\Doc2.doc", "c:\Mis Documentos\a.doc"
```

La tercera es la función **MoveFile**. Esta función mueve el archivo que le indiquemos. La sintaxis "*c:\Doc2.doc",* indica la ruta y el archivo que se va a mover y la sintaxis *"c:\Mis Documentos\a.doc"* indica la ruta y el nombre de destino. Esta función también se usa para cambiar el nombre de los archivos, moviéndolos a la misma carpeta pero utilizando un nombre diferente.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.MoveFile "c:\Doc2.doc", "c:\Mis Documentos\a.doc"
```

La cuarta es la función ***FileExists.*** Esta función comprueba si un archivo existe o no, devolviendo **True** si existe y **False** si no existe. Esta función se suele usar para comprobar la existencia de un archivo antes de borrarlo (para no causar errores), etc. Su único argumento es la ruta del archivo que queremos comprobar su existencia.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Variable = fso.FileExists ("c:\Doc2.doc")
```

Esta función es sumamente importante pues antes de realizar funciones de borrado se debe usar **esta con la sentencia If** para evitar mandar borrar algo que no existe (o crear algo que ya existe, o mover o copiar cosas que no existen…), **evitando así provocar errores,** que fastidiarían el funcionamiento de tu script.

La quinta es la función **Attributes**. Esta función cambia los atributos del archivo indicado. Vemos que en la segunda línea se indica la ruta y el archivo mediante "c:\Doc2.doc", y recogemos el valor en una variable mediante **Set**. Esto es así porque le damos a la variable de forma binaria una estructura de un archivo, y al ser un dato binario necesita el **Set**. Así que al usar la función **GetFile**, le damos a la variable arc mediante el **Set** una estructura del archivo. A continuación en la tercera línea usamos una **función de la estructura que le hemos dado a arc** para cambiar los atributos, mediante un número. Esto es común en diversas funciones, que requieren de la estructura de un archivo para funcionar. En el ejemplo siguiente lo vemos claro:

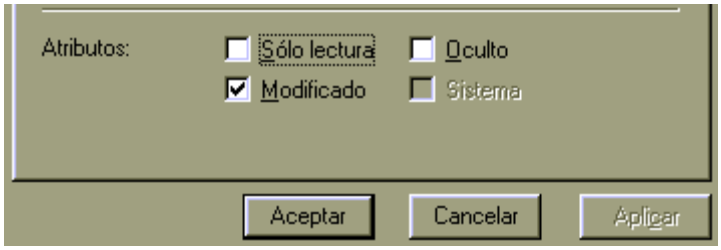
```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\Doc2.doc")
arc.Attributes = 3
```

En realidad no es difícil, basta copiarlo cada vez que necesitemos hacer esta función. En la tercera línea en la que se le dan los nuevos atributos, debemos hacer lo siguiente:

- Para quitarlos todos los atributos se coloca el numero 0.

- Para colocar uno o mas atributos se suman los siguientes valores según los atributos que queramos darle: Solo lectura (valor 1), Oculto (valor 2) y Sistema (valor 4)

En este ejemplo se le da el valor 3 lo cual significa que se le han dado los atributos de Solo lectura y Oculto ***(1+2=3).*** Vemos que le damos el valor con el símbolo del "=", esto se debe a que como he explicado, **se trata de una estructura de datos**, y en definitiva, no es más que una variable numérica que representa los atributos del archivo. Parece difícil, pero es sencillo si practicas un poco, solo debes pensarlo con lógica y **hacerlo tal y como se explica**. Por último, para ver los atributos de un archivo, haz clic en el botón derecho en el y elige propiedades, luego busca este recuadro, que esta abajo de todo:



La sexta función es la función **Size**. Esta función devuelve un valor que indica el tamaño **en bytes** de un archivo (variable numerica). Para que no aparezcan números muy grandes podemos, dividiéndolo entre 1020, pasarlo a Kbytes, y con la función Fix quitarle los decimales. La variable tamaño guarda los datos obtenidos en este ejemplo:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\Doc2.doc")
tamano = fix(arc.size/1020)
```

Como vemos, esta función también requiere darle a una variable la estructura de un archivo: el archivo que le indiquemos en la segunda línea, y la función que nos da el tamaño es **arc.Size**, que **técnicamente no es más que una variable de solo lectura de la estructura del archivo** que hemos creado en la línea dos, que contiene en ella un numero que representa el tamaño del archivo. Un ejemplo, para que se vea mejor podría ser el siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set pepe = fso.GetFile("c:\Doc2.doc")
tamano = fix(pepe.size/1020)
atributo = pepe.Attributes
```

En este ejemplo vemos las dos anteriores funciones. Vemos como en la línea 1 **creamos el objeto** en la variable fso, en la línea 2 **creamos una estructura** del archivo Doc2.doc que hay en el disco duro C, y en la tercera y cuarta línea usamos las funciones explicadas anteriormente para obtener respectivamente el tamaño del archivo y sus atributos actuales. No olvidemos que **cualquier variable en la que podamos escribir también la podremos leer** (no todas las que se pueden leer se pueden escribir, como es

el caso de **Size**), por lo tanto no hay problema en saber cuales son los atributos de un archivo en vez de ponérselos tu. Esto **se aplica también a muchas otras funciones**, estate atento, pues ese tipo de cosas debes ir sacándolas ya tu solo.

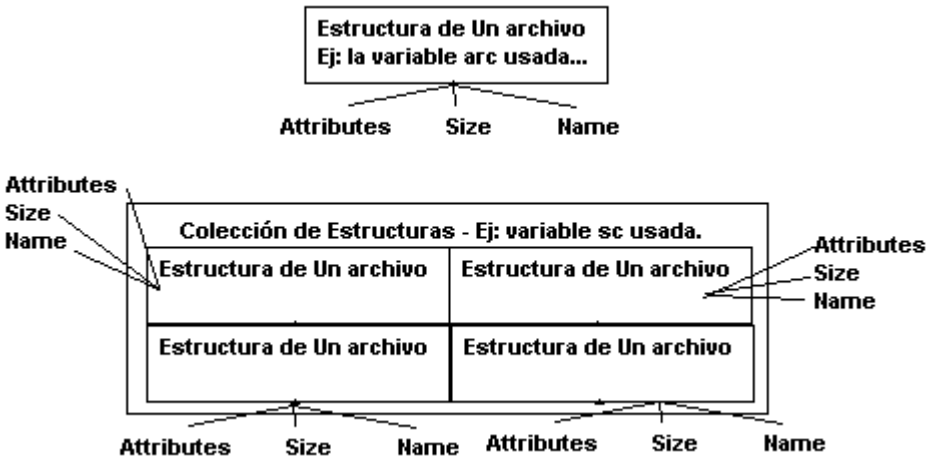
La septima es la función **Run**. Esta función abre el archivo que le indiquemos, de la forma que comúnmente se abre *haciendo doble clic encima*. Fíjate que esta usa el objeto **WScript.shell**. La sintaxis "c:\Doc2.txt" indica la ruta y el archivo que se va a ejecutar. Con esta función también podemos abrir **páginas Web**, colocando la URL en el argumento de la función, en vez de un archivo.

```
Set ws = CreateObject("WScript.Shell")
ws.Run "c:\Doc2.txt"
```

La octava es realmente una mezcla de varias que he echo yo para facilitar los datos. (En realidad es la función ***Files*** que indica todos los archivos de una carpeta en una colección. Yo lo que he echo es pasar esa colección a una variable normal.) Esta función crea una lista de archivos de la carpeta que le indiquemos y la guarda en la variable ListaArchivos. La sintaxi "c:\carpeta" indica la ruta y la carpeta de la que se va a sacar esa lista. Cada archivo va en una línea diferente. Así, se puede guardar esos datos en un archivo y luego ir leyéndolos línea por línea para operar en todas las subcarpetas. Este es el método que, combinado con la función SubFolders, utilizan algunos virus para copiarse en todas las carpetas del ordenador.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetFolder("c:\carpeta")
Set sc = car.Files
For Each car1 in sc
ListaArchivos = ListaArchivos&car1.name&chr(13)
Next
```

Bueno, en realidad así es muy sencillo, pero me quedaría mal si no explicará que es eso de una colección y que pinta ahí un bucle For tan raro. Bueno, en primer lugar, sabed que en la línea dos **GetFolder** hace lo mismo que **GetFile**, pero con una carpeta, lo cual quedo que queda bastante claro, y con un poco de ingenio os permitirá tocar algo de carpetas antes de que lo explique. La tercera línea lo que hace es **dar a la variable sc una colección**, que es otro tipo de datos binarios en una variable (por eso el **Set**). Entonces tenemos lo que se llama una colección en la variable sc, que por supuesto es **una estructura también**. Supongo que entenderíais que hace si os dijera que ese carl.name que hay en la línea 5 es en realidad sc.name. Si, lo que hace es **sacar el nombre del archivo** de la variable sc usando el mismo método que hemos usado en anteriores funciones. Pero porque entonces el For raro y el carl.name en vez de sc. Es simple, porque como he dicho antes es una colección. Una vez lo entendáis será sencillo. En realidad en las funciones de antes (src.Size por ejemplo) teníamos **una estructura de un archivo** y lo que hacíamos era sacar el dato que queríamos, nada más. Pero aquí no tenemos la estructura de un archivo, tenemos la **estructura de todos los archivos de la carpeta**: una colección. ¿Lo vais entendiendo ya? Pongo una imagen:



La estructura creo que queda clara. Pero si esto es verdad, como le decimos al script cual de las X estructuras queremos el **.Name**: Para eso esta ese **For** raro:

*For Each **car1** in sc*
* ListaArchivos = ListaArchivos&**car1**.name&chr(13)*
Next

Este **For** significa literalmente: Por *Cada carl En sc repetir el bucle*. En otras palabras, lo que hace es coger el primer elemento de la colección de **sc** y **asignárselo** a la variable **carl**, realizar las funciones del bucle (en este caso añadir el nombre del archivo *"car1.Name"* a una lista), y vuelve al principio del bucle. Coge el segundo elemento, se lo asigna a car1, y repite otra vez el bucle. Luego hace lo mismo con el tercero, **y así sucesivamente** hasta que **se terminen todos los elementos** de la colección. De esa forma, con la función que os he puesto antes listamos los archivos de una carpeta. Pero no solo se puede hacer eso. Podemos, por ejemplo borrar todos los archivos así:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetFolder("c:\carpeta")
Set sc = car.Files
For Each car1 in sc
fso.DeleteFile "c:\carpeta\" & carl.name, True
Next
```

O obtener el tamaño total sumando los de cada archive con la función carl.**Size**, o ejecutar todos los archivos con la función **Run**, etc. Ya cuando enseñe a trabajar con carpetas, deberás asegurarte antes de eso que la carpeta en cuestión existe, etc. Fíjate, que con cuatro o cinco funciones que he enseñado, y todo lo que tú sabías se te han multiplicado exponencialmente las posibilidades, y esto solo es el principio…

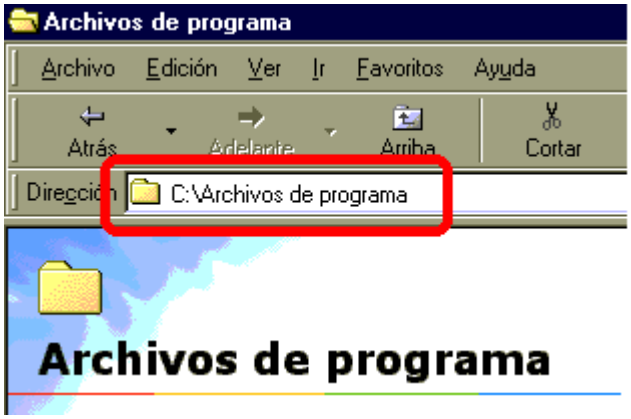
Bueno, pero continuemos, pues nos dejamos cosas importantes. La siguiente función necesita el objeto **Scripting.FileSystemObject** y permite crear un archivo y escribir en él. El tipo de archivos que se pueden crear son muy variados, algunos de ellos son: Archivos de Texto, Scripts de VBScript (¡atento a esto!) y JScript, Paginas Html... Lo que con esta función escribas en un archivo, equivaldría a haber editado dicho archivo con el Bloc de Notas, poniendo en él lo que le dices a la función. En otras palabras, te

permite crear archivos de texto plano (no binarios, que ya se verán mucho, mucho más adelante). La función se coloca de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.write("Este es el contenido del archivo")
var.close
```

Donde en la ***primera linea*** colocamos el objeto. En la ***segunda linea*** utilizando el objeto usamos la funcion **CreateTextFile (**que lleva delante unido por un punto (.) la variable (fso) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. El texto **True** significa que crea el archivo aunque exista, aunque también se puede poner **False** que hará que si ya existe no sobrescriba. Todo esto esta unido a una variable mediante **Set var**, esto guarda el la estructura del archivo en blanco que acabamos de crear (igual que las anteriores estructuras, de las que también se pueden usar sus funciones) para que pueda ser **editado posteriormente usando esta variable**.

Si no sabes como colocar la ruta solo ves a la carpeta por el explorador y mira la barra de dirección, y procede como se explicó anteriormente (no volveré a repetirlo):



En la ***tercera linea*** usamos la variable **var** seguido de un punto y la función **write**, que dentro de ella lleva lo que queramos escribir en el archivo. Aquí se pueden usar las *constantes de VBScript (menos la Chr(13)) y el uso de la concatenación (&), al igual que las usamos en las variables de cadena,* para escribir en el archivo:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
variable = InputBox("Escribe","Escribe","Escribe")
var.write("Este es el contenido del InputBox: "&variable)
var.close
```

La ***cuarta línea*** es la función **close**, que cierra el archivo **e impide volver a usar las funciones de escritura** como **Write**. Es **necesario cerrar siempre el archivo** para no causar errores, al finalizar de escribir en él.

Hay más funciones de escritura. Estas son: **WriteLine** y **WriteBlankLines**. La función **Write** escribe en una línea el texto que quieres, pero si usas dos, no cambia de línea sino que sigue en la misma, escribiendo a continuación de donde lo dejó:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.write("Este es el contenido")
var.write(" del archivo.")
var.close
```

Esto dará como resultado: ***Este es el contenido del archivo.*** En cambio, la función **WriteLine,** escribe el texto que introduces en una línea, pero si pones dos o mas, cada nueva que pongas se escribirá en una línea diferente.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.writeline("Este es el contenido")
var.writeline(" del archivo.")
var.close
```

Esto dará como resultado:
Este es el contenido
del archivo.
Por último la función **WriteBlankLines** escribe tantas líneas vacías como le indiques; donde 2, es el número de líneas que quieres que baje:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.CreateTextFile("c:\ArchivoPrueba.txt", True)
var.writeline("Este es el contenido")
var.writeblankline(2)
var.writeline(" del archivo.")
var.close
```

Esto, escribirá en el archivo:
Este es el contenido

del archivo.

En último lugar, en la ***cuarta línea*** se escribe **var.close** (como he dicho antes) para cerrar el archivo. No se debe omitir (Recuerda que **var** es una variable). Así finaliza la explicación de crear un archivo nuevo y escribir en él. Pero… ¿Y si lo que queremos es abrir uno que ya existe y escribir a partir del final (anexarle datos)? La siguiente función necesita el objeto **Scripting.FileSystemObject** y permite escribir en el final de un archivo, a continuación de lo que había. La diferencia entre esta función y la función **CreateTextFile** es que en esta no borras lo que había escrito a no ser que indiques lo contrario y en la otra si, además, esta función no crea un archivo y luego escribe en el, sino que abre un archivo que ya existe para escribir en él. Se coloca de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.Write "Texto que se escribe o se anexa en el archivo"
esc.Close
```

En la ***primera línea*** colocamos el objeto. En la ***segunda línea*** utilizando el objeto usamos la función **GetFile** (que lleva delante unido por un punto (.) la variable (fso) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. Esta función abrirá el archivo que le hayamos indicado. Todo esto esta unido a una variable mediante **Set arc**, esto guarda la estructura del archivo que hemos abierto en la variable **arc** (que se puede usar para funciones como Size).

En la ***tercera línea*** usamos la función **OpenAsTextStream** unido a arc para indicarle como queremos editar el archivo cuya estructura está en **arc**. Si colocamos 2, lo sobrescribiremos (borras y escribes desde el principio) y tendrá la misma función que **CreateTextFile,** en cambio, si colocamos 8, anexará al final del archivo el texto que queramos escribir. Esto, va unido a una variable llamada **esc** que guardará la estructura del archivo con las propiedades de escritura.

En la ***cuarta línea*** usamos la variable **esc** (que contiene los datos del archivo y el modo de edición) seguido de un punto y la función **write**, que dentro de ella lleva lo que queramos escribir en el archivo. Aquí se pueden usar las *constantes de VBScript (menos la Chr(13)) y el uso de la concatenación (&), al igual que el uso de variables,* para escribir en el archivo:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
variable = InputBox("Escribe","Escribe","Escribe")
esc.write("Este es el contenido del InputBox: "&variable)
esc.close
```

Este es un buen ejemplo de lo que se podría hacer en breves líneas. Ten en cuenta que estamos anexando al final de lo que ya hay escrito en el archivo, y que si este no existe provocará un error, así que ya sabes: usa **FileExists**.

Hay más funciones de escritura. Estas son: **WriteLine** y **WriteBlankLines**. La función **Write** escribe en una línea el texto que quieres, pero si usas dos, no cambia de línea sino que sigue en la misma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.Write "Texto que se escribe o"
esc.Write " se anexa en el archivo."
esc.Close
```

Esto dará como resultado: ***Texto que se escribe o se anexa en el archivo.*** En cambio, la función **WriteLine,** escribe el texto que introduces en una línea, pero si pones dos o mas, cada nueva que pongas se escribirá en una línea diferente.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.WriteLine "Texto que se escribe o"
esc.WriteLine " se anexa en el archivo."
esc.Close
```

Esto dará como resultado:
Texto que se escribe o
se anexa en el archivo.

Por último la función **WriteBlankLines** escribe tantas líneas vacías como le indiques, donde 2, es el número de líneas que quieres que baje:

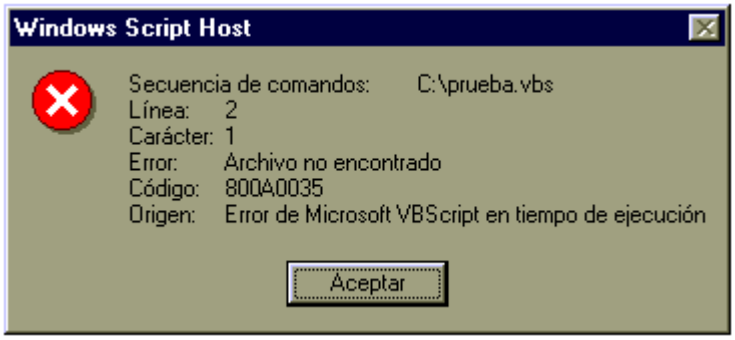
```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFile("c:\ArchivoPrueba.txt")
Set esc = arc.OpenAsTextStream(8)
esc.WriteLine "Texto que se escribe o"
esc.WriteBlankLines(2)
esc.WriteLine " se anexa en el archivo."
esc.Close
```

Esto, escribirá en el archivo:
Texto que se escribe o

se anexa en el archivo.

En último lugar, en la ***quinta linea*** se escribe esc.**close** para cerrar el archivo. No se debe omitir o lo dejarás abierto y no se podrá tocar (esto es una forma de proteger un archivo de que algunos programas lo borren).

Si el archivo que se intenta abrir para anexar no existe se produce el siguiente error:



Por eso, es mejor que te asegures antes con la sentencia If y **FileExists** o creándolo con la función **CreateTextFile** escribiendo False en vez de True, por ejemplo.

Bueno, mi recomendación es que practiques mucho, pues la **práctica hace maestros**. Hemos visto todo lo que se puede hacer con archivos, a estas alturas estoy seguro que ya sabes que es lo único que falta: leer un archivo. La siguiente función necesita el objeto **Scripting.FileSystemObject** y permite leer un archivo o alguna parte de el. El tipo de archivos que se pueden leer son muy variados, algunos de ellos son: Archivos de Texto, Scripts de VBScript y JScript, Paginas Htm. En definitiva: texto plano. (No se pueden leer documentos del Word) Se coloca de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
lectura = var.Read(5)
var.close
```

Donde en la ***primera línea*** colocamos el objeto. En la ***segunda línea*** utilizando el objeto usamos la función **OpenTextFile (**que lleva delante unido por un punto (.) la variable (fso) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. El numero **1 significa que es un archivo para leer**, y no se puede cambiar, pues el 2 es para sobrescribir y el 8 anexar, que son para escribir, no leer. Todo esto esta unido a una variable mediante **Set var**, esto guarda la estructura del archivo para que pueda ser leído posteriormente usando esta variable. En la ***tercera línea*** usamos la variable **var** seguido de un punto y la función **read**, que dentro de ella lleva el número de caracteres que se quieren leer. Esto va unido a la variable **lectura,** que guardará los datos leídos.

Hay más funciones de lectura aparte de **Read**. Estas son: **ReadLine, ReadAll**, **Skip** y **SkipLine**. La función **ReadLine** lee una línea del archivo y la guarda en una variable. Se considera una línea hasta llegar a un salto de línea (vbCrlf) vista con el bloc de notas.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
lectura = var.ReadLine
var.close
```

La función **ReadAll,** lee todo el archivo y lo guarda en una variable. Esta función tarda un poco porque suele realizarse.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
lectura = var.ReadAll
var.close
```

La función **Skip** se salta un número de caracteres determinado. Esto es útil si lo que te interesa es un trozo solo del archivo. Abajo un ejemplo de esta función combinado con la función read para leer desde el caracter cinco hasta el diez.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
var.Skip(5)
lectura = var.Read(5)
var.close
```

La función **SkipLine** se salta una línea para no leerla. Esto es útil si lo que te interesa es una línea determinada del archivo. Abajo un ejemplo de esta función combinado con la función readline para leer la segunda línea de un archivo.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
var.SkipLine
lectura = var.ReadLine
var.close
```

En último lugar, en la ***cuarta linea*** se escribe **var.close** para cerrar el archivo. No se debe omitir. (Recuerda que **var** y fso son variables y puedes llamarlas como quieras).

Se puede saber cuando se llega al final de línea y/o del archivo por medio de las propiedades **AtEndOfLine, AtEndOfStream y Line**. Si se llega al final de una línea, **AtEndOfLine** recibe el valor **True**, en cambio, si se llega a final del archivo, es **AtEndOfStream** quien recibe ese valor.

Esto, unido con el bucle do...loop permite hacer cosas como la que hace el siguiente script:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
Do
var.Skip(1)
lectura = lectura&var.Read(1)
Loop Until var.AtEndOfLine=True
var.close
```

Este pequeño script permite que se lea una letra si, una no hasta que se llega a final de línea. Al leer una letra, la guarda en lectura, al leer la siguiente, la guarda junto con la anterior en lectura y así sucesivamente hasta que **AtEndOfLine** recibe el valor True (significa que ya ha llegado al final de la línea), el bucle se rompe y deja de leer. El mismo uso se le podía dar a **AtEndOfStream** para que parara en vez de al final de línea, lo hiciera al final del archivo.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
Do
var.Skip(1)
lectura = lectura&var.Read(1)
Loop Until var.AtEndOfStream=True
```

```
var.close
```

Por último, **Line** nos indica cuando **empieza** una línea determinada. Si estamos en la línea 5, Line tendrá el valor 5. A continuación el ejemplo anterior, pero esta vez parando en la línea 4, es decir, cuando empiece la quinta línea.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set var = fso.OpenTextFile("c:\Doc2.txt", 1)
Do
var.Skip(1)
lectura = lectura&var.Read(1)
Loop Until var.Line=5
var.close
```

Como podemos ver, las posibilidades son muchas. Por último, y antes de cerrar este capitulo, hay que saber como podemos escribir a mitad del archivo. La cosa es sencilla, lees la primera parte, la guardas en una variable, lees la segunda, la guardas en otra, borras el archivo, escribes toda la primera parte, lo que querías añadir a mitad, y toda la segunda parte. Siento decepcionar, pero en eso consiste la programación: Tus barreras estarán allí donde tu ingenio no llegue. Bueno, y con esto insistir una vez más en practicar y decir que esto es todo con los archivos.

Programación con Objetos: Carpetas y Discos

Ahora vamos a ver un grupo de funciones, que permiten realizar las **tareas básicas de tratamiento de carpetas y discos.** En otras palabras, aprenderemos a que nuestro script pueda borrar, copiar, mover, renombrar, etc cualquier tipo de carpeta que tengamos; y analizar, listar y trabajar con todos los discos (extraíbles y duros) de nuestro PC. Para ello, veamos las diversas funciones que tenemos a disposición. Las siguientes funciones necesitan los objetos **Scripting.FileSystemObject** y **WScript.Shell.**

La primera es la función **DeleteFolder**. Esta función borra la carpeta que le indiquemos y todo su contenido. **True** indica que si se eliminan las carpetas con el atributo de solo lectura y **False** que no lo hace. La sintaxis "c:\carpeta" indica la ruta de la carpeta que se va a eliminar. Su uso es muy similar al de la función **DeleteFile**; de hecho, todas estas funciones se parecen mucho a las que trabajan con archivos:

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.DeleteFolder "c:\carpeta", True
```

La segunda es la función **CopyFolder**. Esta función copia la carpeta que le indiquemos **y todo su contenido**. La sintaxis *"c:\carpeta"*, indica la ruta, y la carpeta que se va a copiar y la sintaxis *"c:\Mis Documentos\carpeta"* indica la ruta y el nombre a donde se copia la carpeta y su contenido. Esta función también se utiliza para **renombrar carpetas**, moviéndolas al mismo sitio donde estaban pero cambiándoles el nombre.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CopyFolder "c:\carpeta", "c:\Mis Documentos\carpeta"
```

La tercera es la función **MoveFolder**. Esta función mueve la carpeta que le indiquemos y todo su contenido. La sintaxis "*c:\carpeta*", indica la ruta y la carpeta que se va a mover y la sintaxis **"***c:\Mis Documentos\carpeta***"** indica la ruta y el nombre de destino.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.MoveFolder "c:\carpeta", "c:\Mis Documentos\carpeta"
```

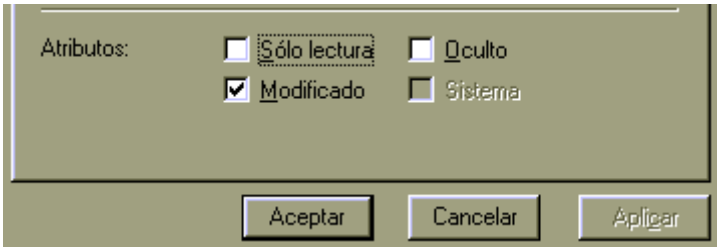
La cuarta es la función **Attributes**. Esta función cambia los atributos de la carpeta indicada, y se usa exactamente igual que la función que hace lo mismo con archivos, con la diferencia que aquí en la línea 2 usamos **GetFolder** y no **GetFile**, y que, por tanto, la estructura que le damos a **arc** es la de una carpeta. En la segunda línea se indica la ruta de la carpeta mediante "*c:\carpeta*". En la tercera se le dan los nuevos atributos:

- Para quitarlos todos se coloca el 0 solo.
- Para colocar uno o mas atributos se suman los siguientes valores según los atributos que queramos darle a la carpeta en cuestión: Solo lectura (valor 1), Oculto (valor 2) y Sistema (valor 4).

En este ejemplo se le da el valor 3 lo cual significa que se le han dado los atributos de Solo lectura y Oculto ***(1+2=3).*** Esta función también se puede usar para saber los atributos de una carpeta, asignándola a una variable (los valores que devuelve siguen los patrones explicados anteriormente).

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set arc = fso.GetFolder("c:\carpeta")
arc.Attributes = 3
```

Para ver los atributos de una carpeta, haz clic en el botón derecho en ella y elige propiedades, luego busca este recuadro, que esta abajo de todo:



La quinta es la función **Size**. Esta función devuelve un valor que indica el tamaño en bytes de una carpeta y todo su contenido. Para que no aparezcan números muy grandes, dividiéndolo entre 1020 lo pasamos a Kbytes y con la función Fix le quitamos los decimales. La variable tamano guarda los datos obtenidos. En definitiva, es exactamente igual que la que usamos con archivos, pero poniendo **GetFolder** en vez de **GetFile** en la línea 2, ya que lo que queremos es la estructura de una carpeta.

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

```
Set arc = fso.GetFolder("c:\carpeta")
tamano = fix(arc.size/1020)
```

La sexta es la función **Run**. Esta función abre la carpeta que le indiquemos. La sintáis "c:\carpeta" indica la ruta y la carpeta que se va a ejecutar. Con esta función también podemos **abrir páginas web**. Es exactamente igual que la usada con archivos.

```
Set ws = CreateObject("WScript.Shell")
ws.Run "c.\carpeta"
```

La septima es la función **CreateFolder**. Esta función crea la carpeta que le indiquemos. La sintaxis "c:\carpeta" indica la ruta y la carpeta que se va a crear. En este ejemplo se creara una **carpeta llamada carpeta** en el disco duro c. Es recomendable comprobar antes si existe o no la carpeta para no sobrescribirla.

```
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CreateFolder "c:\carpeta"
```

La octava es una mezcla de varias que he echo yo para facilitar los datos. (En realidad es la función ***SubFolders*** que indica todas las subcarpetas de una carpeta en una **colección**. Yo lo que he echo es pasar esa colección a una variable normal.) Esta función crea una lista de subcarpetas de la carpeta que le indiquemos y la guarda en la variable **ListaCarpetas**. La sintaxis "c:\carpeta" indica la ruta y la carpeta de la que se va a sacar esa lista. Cada subcarpeta va en una línea diferente. Así, se puede guardar esos datos en un archivo y luego ir leyéndolos línea por línea para operar en todas las subcarpetas. Este es el método que, combinado con la función Files, utilizan algunos virus para copiarse en todas las carpetas del ordenador. No olvidemos la explicación dada en el tratamiento de archivos para recordar de que otra manera se puede usar este bucle For, y lo que significa el siguiente código en realidad (y lo que se puede hacer):

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetFolder("c:\carpeta")
Set sc = car.SubFolders
For Each car1 in sc
ListaCarpetas = ListaCarpetas&car1.name&chr(13)
Next
```

La novena y es la función **GetSpecialFolder**. Esta función da la ruta de las carpetas especiales de Windows que le indiquemos. El valor 1 indica la carpeta que queremos que nos devuelva. Escribiremos **0** si queremos que nos devuelva el directorio de Windows, pondremos **1** si queremos que nos devuelva la carpeta de sistema y **2** si queremos la carpeta temporal.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetSpecialFolder(1)
```

Recuerdo que el resultado de la función (lo que recibe arc) es la estructura de dicha carpeta, y por lo tanto, es lo mismo (en este ejemplo) que poner ***Set car*** =

*fso**.GetFolder(**"C:\Windows\System32"**)*. La utilidad de esta función es que si han cambiado la ruta de instalación de Windows (p.ej. hay ordenadores con WinNT, Win98, WinXp y cosas por el estilo) o la carpeta temporal o la del sistema, esta función siempre te da la buena, y así no hay error posible. Si lo que queremos saber es la ruta completa debemos hacer lo siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set car = fso.GetSpecialFolder(1)
Variable = car.Path & "\" & car.Name
```

La función **Name** me parece que la he explicado, devuelve el **nombre del archivo o carpeta** en cuya estructura se haga la función (en este caso devolvería System o System32, por tratarse ese del nombre de la carpeta del sistema, pero con archivos puede devolver Doc2.txt o cosas así), y es tanto de **lectura como de escritura** (se puede obtener el nombre de la carpeta o archivo como en el ejemplo, o cambiarlo con *car.Name = "Pepe"*) . **Path**, por el contrarío es de **solo lectura** y devuelve la **ruta donde se encuentre archivo o carpeta** en cuya estructura se realice. De esta forma, **uniendo** la **ruta** donde esta la carpeta y el **nombre** de la misma, podemos obtener la **ruta completa** de la carpeta de sistema de Windows.

La décima función es **FolderExists.** Esta función comprueba si una carpeta existe o no, devolviendo **True** si existe y **False** si no existe. Esta función se suele usar para comprobar la existencia de una carpeta antes de borrarla (para no causar errores), etc. Su único argumento es la ruta de la carpeta de la que queremos comprobar su existencia. Es muy recomendable **usarla siempre** que no sepamos seguro si puede o no existir.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Variable = fso.FolderExists ("C:\Archivos de Programa\ScSWinter")
```

Creo que con este ejemplo queda bastante claro. Y esto es todo con las carpetas por ahora, y no es poco. Recomiendo de nuevo práctica para aprender esto. Si alguno se ha molestado en aprender de otras fuentes puede que vea mas funciones, o que se usan diferente; es normal puesto que esto es un tutorial para aprender lo más importante de Visual Basic Script, pero existen otras funciones que realizan lo mismo que estas, o que no tiene ninguna utilidad; así que, cuando estéis seguros de saber a la perfección todo lo que pone en estos tutoriales, no viene nunca mal mirar una referencia del lenguaje para verlo todo en conjunto y de forma ordenada (pero no explicada). Pero no me voy a ir más por las ramas, en el ultimo tutorial explicaré como hacer esto, por ahora continuemos con lo nuestro. Las siguientes funciones necesitan el objeto **Scripting.FileSystemObject.** Estas funciones se encargan de averiguar datos sobre **unidades de disco**, bien sean Discos Duros, unidades de Red, CD-Rom's, disquetera... Todas estas funciones se basan en una especial que les indica a las demás la unidad elegida. Esta función es **GetDrive.** A esta función solamente le tenemos que indicar le letra de la unidad con la que queremos operar según el siguiente ejemplo. A partir de ahí, la información sobre la unidad elegida se guarda en la variable **uni** en forma de estructura, esta vez de un disco, que es lo que usaremos en adelante.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
```

Si la unidad no existe nos dará un error: "El dispositivo no esta listo.", que podemos evitar fácilmente con la función **On Error Resume Next** (no recomendado) o usando la función **DriveExists** junto con la **sentencia If** para verificar si existe de la siguiente forma (esta función devuelve **true** si existe y **false** si no):

```
Set fso = CreateObject("Scripting.FileSystemObject")
If fso.DriveExists("C") Then
Set uni = fso.GetDrive("C")
End If
```

Este ejemplo muestra como se comprueba si existe la unidad C: y si existe, le aplica la función **GetDrive.** De esta forma evitamos que produzca un error. A partir de aquí podemos usar las siguientes funciones para operar con las unidades que existen en nuestro PC:

La primera función útil para trabajar con unidades es **DriveType**. Esta función da a variable un valor numérico según el **tipo de unidad** de que se trate. Devuelve el valor 0 si es un tipo de unidad desconocida, 1 si es extraíble (disquetera, disco zip...), 2 si es fijo (Disco Duro), 3 si es una unidad de red, 4 para los CD-rom y 5 para los discos RAM. De esa forma, por ejemplo podremos no escribir en una unidad CD-Rom para que no de error, mostrar información sobre una unidad o saber si escribimos sobre la RAM, porque al apagar el ordenador esta memoria se borra (un disco Ram es raro encontrarlo, es lo que usan los LiveCD de Linux: usar la RAM como un Disco Duro).

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = uni.DriveType
```

La segunda es la función **IsReady**. Esta función sirve para saber si la **unidad esta lista** para escribir en ella. Se suele utilizar en unidades de disco extraíble (disquetera) para saber si están llenas (es decir, si tienen un disquete dentro). Variable obtendrá el valor **True** si esta lista y **False** si no lo esta (es de solo lectura). De esa forma podemos mandar a escribir en la disquetera si esta llena:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("A")
variable = uni.IsReady
If variable = True Then
'(Aqui va lo que quieras que pase si esta la disquetera llena)
End If
```

La tercera función para trabajar con unidades es **FileSystem**. Esta función da a variable un valor que indica el **sistema de archivos** que esta utilizando la unidad. (Si no sabes lo que es, puedo decirte que de eso depende la capacidad de tu disco duro. Prueba a investigar en la red.) Normalmente los valores que devuelve son FAT para disquetes, FAT, FAT16, FAT32, FAT64 y NTFS para discos duros y CDFS para CD-Roms. Se utiliza de la siguiente forma:

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

```
Set uni = fso.GetDrive("C")
variable = uni.FileSystem
```

La cuarta función es **VolumeName**. Esta función da a variable un valor que indica el **nombre** que esta utilizando la unidad. Se utiliza segun el ejemplo siguiente, que da a variable el nombre de la unidad C:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = uni.VolumeName
```

También se puede usar para escribir un **nuevo nombre** a una unidad. Esto se realiza colocando un = y después el nuevo nombre entre comillas. Hazlo como en el ejemplo siguiente, que le da el nombre de Nuevo a la unidad C, ya que es de lectura y escritura:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
uni.VolumeName = "Nuevo"
```

La quinta función para trabajar con unidades es **TotalSize**. Esta función da a variable un valor numérico que indica el **espacio total** que tiene la unidad en Bytes. Si queremos que nos lo de en Kbytes hay que dividirlo por 1024, si lo queremos en Mbytes hay que dividirlo dos veces por 1024, y si lo queremos en GBytes lo dividiremos un total de tres veces. Para quitarle los decimales podemos usar la ***función Fix*** que se explica en **funciones con números**... Un ejemplo de uso de esta función es el siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = Fix(uni.TotalSize/1024/1024)
```

De esta forma variable recibe el valor en MBytes sin decimales de espacio total de la unidad C:, aunque las unidades las decides tu.

La sexta función es **FreeSpace**. Esta función da a variable un valor numérico que indica el **espacio libre** que tiene la unidad en Bytes. En general se usa igual que la función **TotalSize**, solo que esta devuelve el espacio libre y no el total. Un ejemplo de uso de esta función es el siguiente:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
variable = Fix(uni.FreeSpace/1024/1024)
```

A partir de estas dos, **podemos sacar el espacio utilizado** restando el espacio libre al espacio total de la unidad. El siguiente ejemplo muestra tres MsgBox que indican el espacio total, el espacio libre y el espacio usado de la unidad C:

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set uni = fso.GetDrive("C")
```

```
variable1 = Fix(uni.TotalSize/1024/1024)
variable2 = Fix(uni.FreeSpace/1024/1024)
variable3 = variable1 - variable2
mensaje = MsgBox("Espacio total: "&variable1&" MBytes")
mensaje = MsgBox("Espacio libre: "&variable2&" MBytes")
mensaje = MsgBox("Espacio usado: "&variable3&" MBytes")
```

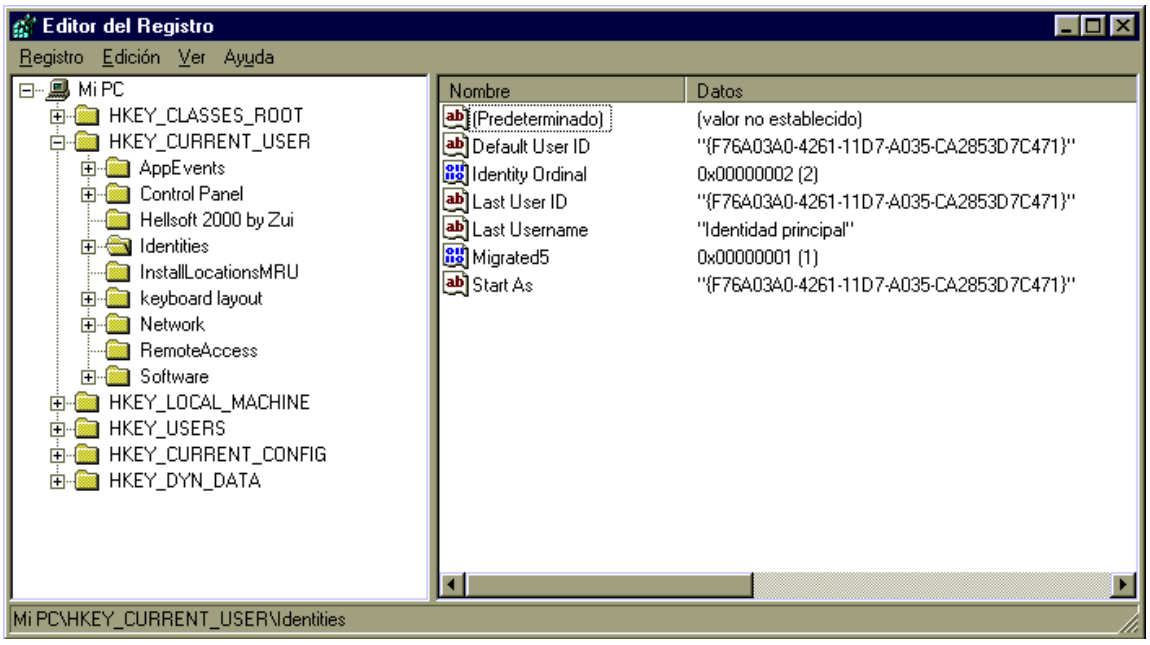
La séptima y última es una mezcla de varias que he echo yo para facilitar los datos. (En realidad es la función ***Drives*** que indica todos los discos en una colección. Yo lo que he echo es pasar esa colección a una variable normal.) Esta función crea una lista de todos los discos que tiene el PC y la guarda en la variable ListaDiscos. Cada disco va en una **línea diferente**. Así, se puede guardar esos datos en un archivo y luego ir leyéndolos línea por línea para operar en todos los discos. Recuerda usar la **función IsReady** para saber si están listos o no para escribir.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set sc = fso.Drives
For Each car1 in sc
ListaDiscos = ListaDiscos&car1.DriveLetter&chr(13)
Next
```

Creo que al ser esta la tercera vez no viene mal recordar que el bucle **For** se repite tantas veces como estructuras tiene la colección **sc**, y que cada vez es **car1** quien tiene la estructura de esa repetición del bucle. Por tanto, podemos usar aquí también las funciones **Name**, **IsReady**, **Size**, etc. Si quieres saber más vuelve un poco atrás a la explicación de la función Files.

Programación con Objetos: Otras Funciones Utiles

Bueno, aquí voy a tratar unas cuantas funciones muy útiles. Las primeras se relacionan con el Registro de Windows. El registro de Windows es, como su propio nombre indica, un grupo de datos que dan información a los programas y al sistema. Modificar, borrar o crear datos en él sin saber lo que se hace puede ocasionar el incorrecto funcionamiento de Windows o alguno de sus programas. Para verlo, en el menú inicio, hacemos clic en ejecutar y escribimos regedit. A continuación nos saldrá una pantalla más o menos así:



En la parte izquierda tenemos en forma de árbol unas carpetas que se llaman claves. Esto sirve para ordenar la información del registro. Al abrir una, nos aparece una lista de subclaves, y al abrir una de estas nos aparecerán más. Al mismo tiempo, una de esas claves puede tener una determinada información de cualquier tipo, que es lo que nos aparece a la derecha. Esta información puede hacer referencia a muchas cosas diferentes, como por ejemplo, el nombre de la papelera de reciclage o la versión de Windows que tienes. Las seis claves principales son:

HKEY_CLASSES_ROOT: Aquí tenemos registradas todas las extensiones, tipos de archivo.
HKEY_CURRENT_USER: Detallado de las configuraciones del usuario actual.
HKEY_LOCAL_MACHINE: Configuraciones de nuestro PC tales como dónde está nuestro software y dónde los drivers instalados.
HKEY_USERS: Las configuraciones de los usuarios de ese PC (urls visitadas, aplicaciones favoritas...).
HKEY_CURRENT_CONFIG: Una especie de especificación de LOCAL_MACHINE, más detalles de la configuración actual, pero la que esta ahora activa y no toda en general (la del usuario activo).
HKEY_DYN_DATA: La información "dinámica", se "forma" al encender el ordenador y desaparece al apagarlo.

Hay tres tipos de información que pueden guardar las claves:

Valor de la cadena: Guarda una cadena con valor alfanumérico. Tienen delante el icono con las letras rojas.

Valor binario: Guarda un valor binario. Es decir, una sucesión de 1 y 0, que significan algo. Tienen el símbolo con números azules. Es el único tipo de datos que es muy difícil, por no decir casi imposible, leer con VBScript.

Valor dword: Guarda un valor decimal o hexadecimal, es decir un numero. Tienen el símbolo con números azules, pero a diferencia del valor binario, el dato se guarda por

ejemplo como 0x00000001(1) lo que significa que tiene el valor uno (mira solo el numero del paréntesis).

No voy a hacer ahora un tutorial del registro de Windows, si lo queréis saber, investigad. Las funciones siguientes operan con el registro de Windows y lo modifican, y son lo que nos interesa ahora. Las siguientes funciones necesitan del objeto **WScript.Shell.** Estas funciones son las tres siguientes:

La primera es la función **RegDelete**. Esta función borra una clave del registro o uno de sus valores según lo que le indiquemos. En este ejemplo de abajo se borra un valor de una clave porque al final no hay una barra (\). **Si colocamos al final esa barra** estaremos **indicando que queremos borrar una clave**. Por ejemplo si queremos borrar toda la clave de Internet Explorer pondríamos:

"HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\".

Si solo queremos **borrar el valor** Versión lo haríamos como en el ejemplo, es decir **sin la barra final** porque no es una clave.

```
Set ws = CreateObject("WScript.Shell")
ws.RegDelete "HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Version"
```

La segunda es la función **RegRead**. Esta función lee el valor que le indiquemos. Delante de la función colocamos una variable que se encargará de guardar el **valor que lea en el registro la función**. Si el valor que lee la función en el registro es un valor de la cadena, la variable recibirá una cadena, si es un valor dword la variable recibirá un número y si es un valor binario normalmente dará error. En este ejemplo, la variable recibirá la versión actual de Internet Explorer.

```
Set ws = CreateObject("WScript.Shell")
variable = ws.RegRead ("HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Version")
```

La tercera es la función **RegWrite**. Esta función crea una clave nueva, agrega un valor a una clave ya existente o cambia un valor ya existente. La función tiene tres partes: La primera donde se indica la ruta donde queremos crear el valor y el valor que queremos crear. La segunda es el valor que le asignamos al valor creado. La tercera es el tipo de valor. Los tipos de valores son REG_SZ si queremos crear un valor de la cadena y REG_DWORD si queremos crear un valor dword. Si queremos crear una clave nueva es tan sencillo como **indicarlo como si existieran desde el principio** en la primera parte. Por ejemplo, en el siguiente ejemplo creamos el valor Web1, que es una cadena que contiene el valor www.google.es, dentro de la clave HKEY_LOCAL_MACHINE\ Software\Microsoft\Internet Explorer\Lista deWebs\. Y realmente lo que estamos haciendo aquí es ir a Internet Explorer, crear una clave llamada Lista de Webs y crear dentro el valor Web1. Es sencillo si entiendes el funcionamiento del Registro de Win.

```
Set ws = CreateObject("WScript.Shell")
ws.RegWrite "HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Lista
```

```
deWebs\Web1","www.google.es","REG_SZ"
```

Atención: Modificar datos en el registro de Windows puede ocasionar el mal funcionamiento de su SO o de alguno de sus programas. Si no esta seguro de lo que hace no lo intente y si lo intenta cree por lo menos una copia de seguridad del registro antes de realizar cambios en el para poder restaurarlo si algo ocurre mal. Recomendamos también que antes de modificar el registro con VBScript, lo haga manualmente hasta que sepa exactamente que es cada cosa, como se expresan las direcciones, las calves que no se deben tocar, que función realiza cada tipo de valor...

A continuación, las siguientes funciones que se van a enseñar sirven para **controlar aplicaciones**. Supongo que habréis visto alguna vez a algún amigo o familiar **usar algún programa solo con el teclado**. Lo primero que deberéis hacer es saber hacer eso, es decir saber **controlar cualquier aplicación con el teclado**. No es difícil, por ejemplo, normalmente en los Procesadores de Texto se cambia entre la zona de escritura y el menú con el botón Alt. Es cuestión de probar y conocer los de aquella aplicación que quieras controlar. Una cosa a **tener en cuenta es el tipo de aplicación a controlar**, por ejemplo, controlar la línea de comandos es una tontería, porque es mas sencillo crear un archivo bat y ejecutarlo; por otro lado, el paint no lo podemos controlar porque requiere el uso del ratón.

Tened en cuenta el **tipo de control** que vamos a realizar sobre la aplicación. Hay muchas formas de controlarlas, pero esta **es generalizada** y además, por decirlo de alguna manera espectacular. Con esto quiero decir que lo que haremos será ejecutar la aplicación, comprobar que tenga el foco y que no la cierren, y ir enviándole pulsaciones. Esto tiene el efecto claro a ojos de una persona normal de que alguien te esta controlando desde Internet al estilo de las películas.

En primer lugar debemos aprender la siguiente función: la función **SendKeys**. Esta función utiliza el objeto WScript.Shell. Esta función envía una serie de pulsaciones al área de texto activa que haya en ese momento. Es muy útil para ayudar la programación en otros lenguajes. También puede tener otras utilidades, que variaran según tu imaginación. Abajo hay un ejemplo de uso de esta función:

```
Set ws = CreateObject("WScript.Shell")
Variable = MsgBox("En 10 segundos escribirá automáticamente el código de registro.
Clique sobre la zona de escritura...",0,"Registro")
WScript.Sleep 10000
ws.SendKeys "an123rgb2"
```

Esto enviará al área de escritura la cadena an123rgb2, recomiendo probarlo antes de continuar y ver a que me refiero. Recuerda aquí las funciones **Sleep** y **Quit** enseñadas anteriormente. El método **SendKeys** utiliza algunos caracteres especiales como modificadores de los caracteres normales (en lugar de utilizar su valor impreso). Este conjunto de caracteres especiales incluye los *paréntesis, llaves, corchetes, el signo más "+", el acento circunflejo "^", el signo de porcentaje "%", y la tilde llana "~"*. Para enviar estos caracteres al área de escritura, **inclúyalos entre llaves "{}" por separado**, según el ejemplo siguiente:

```
Set ws = CreateObject("WScript.Shell")
ws.SendKeys "3{+}2=5 {(}Suma basica{)}"
```

También decir que algunas pulsaciones de **teclas no generan caracteres** (como Enter y Tab). Otras pulsaciones representan acciones (como Retroceso y Ctrl.). A continuación tenemos un ejemplo y la tabla de pulsaciones de **teclas no generan caracteres** (como Enter y Tab) y/o representan acciones (como Retroceso e Ctrl). En el siguiente ejemplo escribimos en el área de texto Para abrir la ayuda pulse F1 y luego la abrimos automáticamente enviando la pulsación del F1:

```
Set ws = CreateObject("WScript.Shell")
ws.SendKeys "Para abrir la ayuda pulse F1{F1}"
```

A continuación está la tabla con la lista de pulsaciones especiales de la **función SendKeys:**

Tecla	Hay que escribir...
RETROCESO	{BACKSPACE}, {BS} o {BKSP}
INTERRUMPIR	{BREAK}
BLOQ MAYÚS	{CAPSLOCK}
SUPR o SUPRIMIR	{DELETE} o {DEL}
FLECHA ABAJO	{DOWN}
FIN	{END}
ENTRAR	{ENTER} o ~
ESC	{ESC}
AYUDA	{HELP}
INICIO	{HOME}
INS o INSERTAR	{INSERT} o {INS}
FLECHA IZQUIERDA	{LEFT}
BLOQ NUM	{NUMLOCK}
AV PÁG	{PGDN}
RE PÁG	{PGUP}
IMPRIMIR PANTALLA	{PRTSC}
FLECHA DERECHA	{RIGHT}
BLOQ DESPL	{SCROLLLOCK}
TAB	{TAB}
FLECHA ARRIBA	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}

F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

Por ultimo decir sobre esta función que para enviar aquellos caracteres del teclado formados por la pulsación de una tecla normal combinada con Mayús, Ctrl o Alt debe colocar delante el caracter al que influyen uno de estos signos dependiendo de la siguiente tabla:

Tecla	Carácter especial
Mayúsculas	+
Ctrl	^
Alt	%

Por ejemplo, si queremos pulsar el Ctrl+Alt+Supr para abrir las tareas de Windows enviaremos "^(%{DEL})". Ten en cuenta que como puedes comprobar si unes dos teclas especiales tienes que **usar el paréntesis** para encerrar a la segunda y a la tercera pulsación enviada.

Pero bueno, creo que me he extendido demasiado con esta función, pero es la base de controlar las aplicaciones. Pero aún hay más que aprender. Para realizar un buen control necesitaremos las funciones **Sleep**, **SendKeys**, y las funciones nuevas **Exec**, **AppActivate**, **ProcessID**, **Terminate** y **Status**. Ahora explicare el uso de cada una y por ultimo pondré un ejemplo para controlar una aplicación sencilla.

Empezare con la función Exec, que tiene una función similar a la función **Run**, pero esta guarda la estructura de la aplicación en marcha en una variable con la que luego operaremos para controlarla a voluntad. Se usa utilizando el objeto **WScript.Shell**, de la siguiente forma (en este ejemplo abrimos el regedit, pero ten en cuenta que el directorio de instalación de Win **no siempre es el mismo** así que lo correcto sería usar **GetSpecialFolder**, sacar el nombre y luego usarlo aquí):

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("C:\Windows\regedit.exe")
```

Donde en la ***primera línea*** colocamos el objeto. En la ***segunda línea*** utilizando el objeto usamos la función **Exec** (que lleva delante unido por un punto (.) la variable (ws) que guarda el objeto). Dentro de la función colocamos primero la ruta, que en este caso es el disco duro c:, seguido de \ y el nombre del archivo, un punto y su extensión. Esta función **ejecutará el archivo** que le hayamos indicado. Todo esto esta unido a una variable mediante **Set** VarApplicaion, esto guarda la estructura de la aplicación

ejecutada en la variable para luego poder controlarla. El **efecto de esta función** es simple, se ejecutara el Editor del Registro de Windows.

Las siguientes **dos funciones** (AppActivate y ProcessID) se utilizan siempre juntas. Estas sirven para **darle el foco a la ventana que hemos abierto**. Dar el foco a una ventana significa solamente **activarla**, es decir, tiene el mismo efecto que hacer clic sobre ella. Esto es muy importante porque si no lo hacemos **las pulsaciones** pueden irse a cualquier sitio y **no provocan el efecto** deseado. Por eso siempre se usará antes de la función SendKeys. Abajo un ejemplo de uso de dicha función:

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("C:\Windows\regedit.exe")
Wscript.Sleep 10000
ws.AppActivate VarApplicaion.ProcessID
ws.SendKeys "{UP}"
```

Este pequeño código ejecuta el Registro, se espera diez segundos a que se abra (por si se utiliza un procesador lento), le concede el foco (por si se ha abierto en un procesador rápido y alguien cansado de esperar a abierto algo) y le envía una pulsación de la flecha arriba. En este Script **la cuarta línea es la que le concede el foco a la aplicación** abierta. Esto lo hace de la siguiente forma: ws.AppActivate **le da el foco a una ventana**. Esto lo hace **especificando el nombre** de la ventana (como valor de cadena), **o el número del proceso** (como valor numérico). VarApplicaion.ProcessID es una función que **devuelve el numero del proceso** de la aplicación abierta por VarApplicaion (solo lectura), y así, podemos **devolver el foco a la aplicación**. Otra forma de usar esta función es **indicando el nombre de la ventana**, pero no es recomendable porque normalmente suelen cambiar (estoy pensando en el Word), aunque en otras como la Calculadora serviría.

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("calc")
Wscript.Sleep 5000
ws.AppActivate "Claculadora"
```

En este caso, como el nombre de la ventana de la calculadora **es siempre el mismo**, siempre funciona.

Otra funcion util y necesaria es **Terminate**. Esta función **finaliza el programa** que se ha abierto con la función **Exec**. Es relativamente fácil de usar, y se coloca siempre cuando ya no vayamos a utilizar más el programa, para que no se quede abierto. Siguiendo el siguiente modelo, que ejecuta la calculadora y al cabo de cinco segundos la cierra sin hacer nada más:

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("calc")
Wscript.Sleep 5000
VarApplicaion.Terminate
```

La ultima función es Status, esta devuelve un valor numérico, que **se corresponde con 0, si la aplicación esta abierta, y se corresponde con 1, si esta cerrada**. Por tanto, esta función, unida a la *sentencia If,* nos puede ayudar a saber si el usuario del ordenador ha cerrado o no la aplicación que se intenta controlar. Un ejemplo de uso seria el siguiente:

```
Set ws = CreateObject("WScript.Shell")
Set VarApplicaion = ws.Exec("calc")
Wscript.Sleep 10000

If VarApplicaion.Status = 0 Then
Msgbox "No has cerrado la Calculadora"
VarApplicaion.Terminate
End If

If VarApplicaion.Status = 1 Then
Msgbox "Has cerrado la Calculadora"
End If
```

Aquí, abrimos la calculadora y tras esperarnos 10 segundos, **comprobamos si la calculadora esta abierta** con VarApplicaion.Status, si lo esta, se muestra un mensaje y se cierra; si no, se muestra otro mensaje diferente. Esta función también se puede emplear para **volver a iniciar todo el proceso de control sobre una aplicación** si el usuario la cierra, y no parar hasta conseguirlo.

Para controlar una aplicación lo único que hay que hacer es **unir ingeniosamente estas funciones con paciencia** hasta lograr lo que queremos hacer. Por poneros un ejemplo, dejo un Script que controla la Calculadora (se obvia la creación de variables):

```
Set ws = WScript.CreateObject("WScript.Shell")
Variable1=MsgBox("Vamos a calcular cuanto vale la siguiente operacion:
(43+21)x11",0,"Calculo")
Set VarApp = ws.Exec("calc")

WScript.Sleep 500
ws.AppActivate "Calculadora"
ws.SendKeys "43"

WScript.Sleep 500
ws.AppActivate VarApp.ProcessID
ws.SendKeys "{+}"

WScript.Sleep 500
ws.AppActivate VarApp.ProcessID
ws.SendKeys "21"

WScript.Sleep 500
ws.AppActivate "Calculadora"
ws.SendKeys "~"

WScript.Sleep 500
ws.AppActivate VarApp.ProcessID
ws.SendKeys "*11"
```

```
WScript.Sleep 500
ws.AppActivate "Calculadora"
ws.SendKeys "~"

WScript.Sleep 2500

If VarApp.Status = 0 Then
  Variable1=MsgBox("Ahora ya lo sabes: ¡La Calculadora no miente!",0,"Calculo")
  VarApp.Terminate
End If

If VarApp.Status = 1 Then
  Variable1=MsgBox("¡Has cerrado la Calculadora! Pues ahora te quedas sin saberlo, mi mente
no es capaz de hacerlo tan bien como ella...",0,"Calculo")
End If
```

Fíjate bien y verás que todo es conforme lo explicado y que la operación la hace bien. Por ultimo decir que hay ciertas aplicaciones como por ejemplo el Word o el Outlook que se pueden controlar de otras formas más efectivas, pero esta, aunque larga y cansina de programar, siempre **es la más espectacular y sirve para todo tipo de programas**. Además, no es necesario que uses estas funciones solo para controlar aplicaciones: sabes lo que hacen, úsalas según mejor te convenga o necesites.

Programación con Objetos: Objeto Dictionary

El **objeto Dictionary** es un objeto especial, ya que se encarga de crear una **forma nueva de administrar variables**. Las administra **por pares** (en Perl lo llaman matriz asociativa). Esto significa que en una variable crea una **colección de valores** y los **asocia a una clave** para identificarlos. Por ejemplo, si tenemos la variable VarDic, nos podemos encontrar con esto:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "a", "Atenas"
VarDic.Add "b", "Belgrado"
VarDic.Add "c", "El Cairo"
```

Luego explicare las funciones y su significado, ahora quedaros con lo esencial. Digo que crea una colección porque la variable VarDic contiene a la vez los valores Atenas, Belgrado y El Cairo. Para que el Script sepa cual dar cada vez, **se le asignan una clave** (como a, b, c...). De esa forma, VarDic funcionara como si tuviese el valor Atenas **cuando se le especifique el valor** a, funcionara como Belgrado cuando tenga b, y como El Cairo cuando c. Luego explicare como se especifica la clave y como se usa.

Este objeto tiene mucha utilidad cuando el Script es muy largo y tienes que utilizar muchas variables (con esto las puedes reducir drásticamente), o bien cuando quieres agruparlas porque tienen algo en común (al estilo de las estructuras de C++). Ahora voy a explicar un poco como se utiliza. Pongamos otro ejemplo:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
```

```
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
```

Primero tenemos la variable VarDic, a la que le **asignamos el objeto dictionary**, para luego poder usar esa variable como una colección. A diferencia de otros objetos este **se debe de crear una vez para cada variable que queramos emplear** de esta manera. Bien, ahora tenemos una colección en VarDic, vacia. Si queremos **asignarle valores utilizamos la función Add**. Esta se utiliza, poniendo el nombre de la variable seguido de un punto y la palabra **Add**, un espacio, la clave entre comillas, una coma, y el valor que puede estar entre comillas, si se trata de un valor de cadena, o sin ellas si es un valor numérico. En la línea dos hemos asignado a la clave amigo el valor Juan, y en la tres al valor novia Maria. Si luego queremos **ver el contenido de** VarDic cuando esta tiene la clave amigo, por ejemplo, **lo haremos así**:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
Msgbox VarDic.Item("amigo")
```

Utilizamos la **función Item** para obtener el valor de la clave amigo, y por lo tanto en el MsgBox se mostrara el valor Juan. La forma de usarlo es similar a la de la función Add: Utilizamos el nombre de la variable seguido de un punto y la palabra Item, un espacio y la clave entre comillas y paréntesis. Para que os aclaréis mas, podemos decir que todo el bloque VarDic.**Item**("amigo"), **se puede considerar como una variable**, y podéis utilizarlo como si se tratara como tal.

Es decir, realmente solo he utilizado la variable VarDic, pero en cambio, por decirlo de alguna manera ahora tengo tres, dependiendo de la clave que ponga en VarDic.**Item**("amigo"). Para que quede claro, podemos decir que los dos siguientes códigos son idénticos en cuanto a funcionalidad.

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
Msgbox VarDic.Item("amigo")
```

```
VarAmigo = "Juan"
VarNovia = "Juan"
VarPadre = "Pepe"
Msgbox VarAmigo
```

De esta forma, VarDic.**Item**("amigo") y **VarAmigo** son exactamente lo mismo y se pueden usar de la misma forma.

La funcionalidad de este objeto viene cuando por ejemplo quiero poner los nombres, edades, direcciones y teléfonos de veinte personas. Con este objeto, en vez de crear ochenta variables, con crear cuatro (uno para nombres, uno para direcciones, uno para teléfonos, y el ultimo para edades) sobra. Además, puedes hacer que se correspondan, y así tienes menos trabajo:

```
Set Nombre = CreateObject("Scripting.Dictionary")
Set Edad = CreateObject("Scripting.Dictionary")
```

```
Set Telefono = CreateObject("Scripting.Dictionary")
Set Direccion = CreateObject("Scripting.Dictionary")
Nombre.Add "amigo", "Juan"
Edad.Add "amigo", 15
Telefono.Add "amigo", 965330000
Direccion.Add "amigo", "Calle de Bartolo n13"
Msgbox Nombre.Item("amigo")&" Edad:"&Edad.Item("amigo")
```

Más o menos, esa seria la idea de uso del **objeto Dictionary**, que es **conveniente conocer**, pues en programas más complejos es la única salida de estructurar variables que Visual Basic Script nos permite. Las siguientes funciones necesitan el objeto **Scripting.Dictionary** y se encargan de operar con los pares de valores del objeto.

La primera es la función **Key**. Esta función **cambia la** clave del par que le indiquemos. Por ejemplo, en el siguiente ejemplo cambia la clave del valor Maria, de novia, que era lo primero que tenía asignado, a exnovia, que es el que le acabamos de poner. Si probamos de **ver el valor de la clave novia nos dara error**, ya que ahora su clave es exnovia:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarDic.Key("novia") = "exnovia"
```

La función se utiliza indicando la variable, luego un punto y la palabra **Key**, luego la clave que **queremos cambiar** entre comillas y paréntesis, y por ultimo la nueva clave después del igual, entre comillas.

La segunda es la función **Remove**. Esta función **elimina un** valor, indicándole la clave de este. Por ejemplo, en el siguiente ejemplo **borra** el valor de la clave amigo, de forma que si luego intentamos visualizarlo, no podremos porque lo hemos eliminado.

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarDic.Remove("amigo")
```

La función se utiliza indicando la variable, luego un punto y la palabra **Remove** y luego la clave que **queremos borrar** entre comillas y paréntesis. Con esto es como si nunca hubiéramos escrito la segunda línea. Esta función tiene interés cuando se necesita **cambiar el valor (no la clave) de un par**, ya que lo que haremos será borrarla y crearla de nuevo con el nuevo valor.

La tercera es la función **RemoveAll**. Esta función **vacía la variable del objeto** dejándola sin nada. Es decir, **borra todos los** valores **que tiene** la variable. Pongamos un ejemplo de uso de esta función:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarDic.RemoveAll
```

Este código de arriba equivaldría a uno en el que solo estuviera la primera línea. Es decir, **ha borrado los tres** valores **que le hemos dado**. Esto sirve para **reutilizar la variable** o para cambiar todos los valores de esta. Su uso es sencillo, solo hay que indicar la variable a **borrar**, y se usa como todas las de este tipo que hemos visto (no tiene ningún argumento).

La cuarta es la función **Exists**. Esta función **comprueba si existe o no un valor asignado a una clave** concreta. Devuelve el valor True si existe y el valor False si no existe. Por ejemplo, si queremos saber si realmente ha escrito algo en un InputBox y no ha apretado cancelar podemos realizar lo siguiente, utilizando la sentencia If:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarResp = InputBox("¿Que comida te gusta?","Comida","Introduce comida favorita")
If VarResp<>"" Then VarDic.Add "respuesta", VarResp 'Esto es la sentencia If abreviada
(solo 1 caso)
If VarDic.Exists("respuesta")=True Then
VarMsg = "El valor de la clave es: "&VarDic.Item("respuesta")
Else
VarMsg = "No existe ningun valor..."
End If
```

En este ejemplo si se ha escrito algo, se crea un valor para la clave respuesta con lo que haya escrito en el InputBox. Luego comprueba con el método **Exists** si realmente **se ha creado o no la clave y su valor**, y le asigna a la variable VarMsg un valor diferente según ocurra.

La quinta y ultima es la función **Count**. Esta función devuelve un valor numérico que **indica el número de claves que existen** en la colección. Es decir, simplemente nos indica cuantos valores hemos creado. Para utilizarla se coloca el nombre de la variable, un **punto y** la palabra **Count**. Abajo tenéis un ejemplo de uso, en el que la variable VarNum obtiene el valor 3:

```
Set VarDic = CreateObject("Scripting.Dictionary")
VarDic.Add "amigo", "Juan"
VarDic.Add "novia", "Maria"
VarDic.Add "padre", "Pepe"
VarNum = VarDic.Count
```

Esto se suele usar para hacer alguna operación en todos los pares de la variable mediante un bucle For. Hasta aquí la explicación del **objeto Dictionary**, el uso que cada uno le de dependerá de su imaginación o sus necesidades. Este objeto, como se ha visto, va más encarado a la **ordenación de información** que a alguna función en concreto, pero repito que es **muy útil**. En cuanto a los Objetos se refiere, esto es lo básico...