Comencemos a programar con

VBA - Access

Entrega 19

Trabajar con ficheros III

Eduardo Olaz

Exportar, importar y vincular ficheros de texto

Access posee una serie de herramientas muy potentes que posibilitan las operaciones de Exportación, Importación y Vinculación de ficheros de texto, suministrándonos además unos asistentes que nos facilitarán estas tareas.

No obstante, hay casos en los que somos nosotros los que deberemos tomar las riendas de todo el proceso.

En la entrega anterior vimos varias formas de leer y escribir en ficheros de texto.

En esta entrega vamos a ver la posibilidad que poseen tanto VBA como VB, de trabajar con ficheros de texto definidos con la ayuda de un fichero adicional que contiene un esquema de su estructura; estoy hablando del fichero **Schema.ini**.

También veremos cómo trabajar con ficheros especiales, tanto para datos de inicialización como de información; los ficheros **ini**, y su alternativa más actual, leer y escribir en el **Registro de Windows**, lo que de paso nos dará pie a introducirnos en el "apasionante mundo" del **API** de Windows.

Hay un tipo especial de ficheros de datos que aunque no vamos a abordarlos en esta entrega, serán objeto especial de un Apéndice: los ficheros de datos con formato **XML**.

Nota:

En las líneas de código se van a utilizar objetos que todavía no hemos visto, como las librerías DAO y ADO, y algunos procedimientos del API de Windows.

Ruego paciencia al lector si hay partes del código que no llega a entender.

Los temas aquí tratados y que resulten totalmente nuevos, se verán con más profundidad en posteriores entregas.

Fichero Schema.ini.

Este archivo es un archivo especial, que contiene una descripción de la estructura, de un fichero de texto organizado como una tabla de datos, proporcionando información sobre:

- El formato que tiene el archivo.
- Los nombres, la longitud y tipos los tipos de datos de los campos.
- El juego de caracteres que se ha utilizado.
- Información sobre las conversiones especiales de los tipos de datos.

El fichero **Schema.ini** debe encontrarse en la misma carpeta donde estará ubicado el fichero de datos.

La información contenida en un fichero **Schema.ini**, sobre los formatos de datos, tiene prioridad frente a la configuración predefinida en el registro de Windows.

Para que VBA pueda manejarlo de forma adecuada, el nombre de este fichero debe mantenerse, por lo que deberá usarse siempre como **Schema.ini**.

Como veremos, Schema.ini es un tipo especial de los ficheros **ini** de configuración.

El tipo genérico de ficheros **ini** será analizado en un punto más avanzado de esta misma entrega.

© Eduardo Olaz

Estructura de un fichero Schema.ini.

Como todos los ficheros ini, los datos contenidos en Schema.ini están en las denominadas **Secciones**.

Una sección se define por un nombre delimitado por paréntesis cuadrados (corchetes).

En el caso del fichero Schema.ini, la primera sección contiene el nombre del fichero de datos. Por ejemplo **`[Datos.txt]`**.

En una sección de un fichero ini, se puede incluir nombres de variables o claves, seguidas del signo igual **=** y a continuación el valor que queremos que tome esa variable o clave.

Podemos definir, ficheros con campos de longitud fija, o con caracteres delimitadores.

A continuación incluyo el contenido de un fichero **Schema.ini** y pasaremos a describir el contenido del mismo. El siguiente paso será analizar las diferentes posibilidades que puede presentar cada clave.

Primero vamos a definir el fichero Schema.ini que contendrá la información del fichero `Alumnos.txt`, con una serie de campos de longitud fija.

Para crearlo utilizaremos un simple editor de texto, como el **Bloc de Notas**.

```
[Alumnos.txt]
ColNameHeader=False
Format=FixedLength
MaxScanRows=25
CharacterSet=Oem
Col1= idAlumno Integer Width 6
Col2= Nombre Char Width 15
Col3= Apellido1 Char Width 15
Col4= Apellido2 Char Width 15
Col5= FechaNacimiento Date Width 11
Col6= Poblacion Char Width 15
Col7= idCurso Integer Width 5
```

Grabamos el fichero con el nombre **Schema.ini** en la misma carpeta que la base de datos.

Antes que nada veamos cómo se puede, por código, exportar datos de una tabla a un fichero de texto. Voy a usar los métodos de la librería DAO (que veremos más adelante).

Se supone que la tabla a exportar está en una base de datos llamada **Datos.mdb**, en la tabla **Alumnos** y que contiene los campos descritos.

Para poder usar DAO, si no está activada, activamos la referencia a su Librería. Para ello usaremos, desde el editor de código, la opción **Referencias…**del menú **Herramientas**.

Y en concreto la librería **Microsoft DAO N.M Object Library**. Tras esto escribimos

```
Public Sub ExportacionConDAO()
    Dim db As DAO.Database
    Dim strSQL As String
    Dim strcarpeta As String
    Dim strBaseDatos As String
```

```
    Dim strTabla As String
    Dim strFichero_txt As String
    Dim n As Long
     ' Nombres de los elementos _
      intervinientes en el proceso
    strcarpeta = CurrentProject.Path
    strBaseDatos = strcarpeta & "\Datos.mdb"
    strTabla = "Alumnos"
    strFichero_txt = strTabla & ".txt"

    ' Abrimos la base de datos
    Set db = OpenDatabase(strBaseDatos)

    ' Si existe la tabla, la borramos
    If Dir(strcarpeta & "\" & strFichero_txt) =
strFichero_txt Then
        Kill strcarpeta & "\" & strFichero_txt
    End If
    ' definimos la consulta de acción
    strSQL = "SELECT * " _
            & "INTO [TEXT;DATABASE=" _
            & strcarpeta _
            & "]." _
            & strFichero_txt _
            & " FROM " _
            & strTabla

     ' Hacemos que se ejecute la consulta de acción
     db.Execute strSQL
     ' Cerramos la Base de Datos
    db.Close
     ' Eliminamos la referencia a la Base de datos
    Set db = Nothing

End Sub
```

No se preocupe demasiado si no entiende ese código.

Lo que hace es exportar el contenido de la tabla **Alumnos** de la base de datos **Datos.mdb** a la tabla **Alumnos.txt**.

Veamos ahora qué significa cada una de las líneas del fichero **Schema.ini**.

© Eduardo Olaz

Ya hemos indicado que la primera línea, **[Alumnos.txt]** indica el nombre del fichero al que se va a exportar.

La segunda línea **ColNameHeader**=False indica que la tabla no va a contener los nombres de los campos.

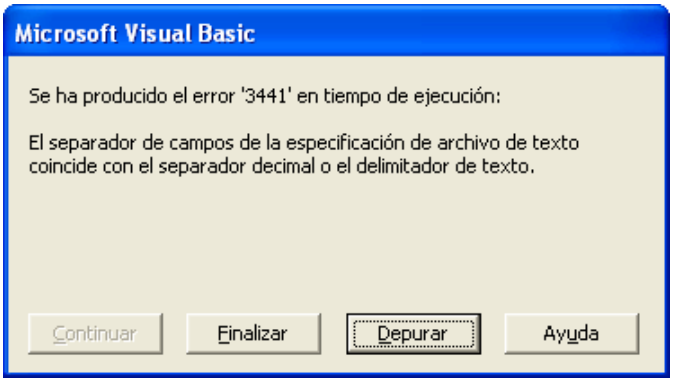
Si hubiéramos puesto: **ColNameHeader**=True, el resultado de exportar la tabla sería:

```
"idAlumno","Nombre","Apellido1","Apellido2","FechaNacimiento","Poblacion","idCurso"
1       Antonio         Martínez        Zúñiga          02/10/1984 Pamplona        35
2       Pedro           Iturralde       Etxegárate      03/01/1985 Tudela          14
3       Marta           Valduz          Goikoa          27/07/1985 Tafalla         10
4       Enrique         Viana           Vergara         03/09/1984 Pamplona        10
```

La tercera línea **Format**= FixedLength hace que los datos se exporten con un formato de longitud variable. Los posibles valores que puede tomar esta clave son:

TabDelimited	Los campos se delimitan mediante el carácter **Tab**
CSVDelimited	Utiliza comas como separador de campos
Delimited (*)	Hace que los campos se delimiten por asteriscos. Se puede utilizar cualquier carácter, excepto las comillas dobles **(")**. Por ejemplo **(;)** haría que el delimitador fuera un punto y coma.
FixedLength	Hace que los campos sean de longitud fija

En España, al usar en la configuración regional la coma como separador decimal, si también la usáramos como separador de campos, nos generaría el error 3441, por esa coincidencia entre el separador de campos y el separador decimal.



Por ello sería aconsejable, en su lugar, la utilización del punto y coma **Format**= **Delimited (;)** como delimitador de campos.

La cuarta línea **MaxScanRows**=25 le indica al motor de la base de datos, el número de filas que debe examinar en la tabla, para determinar el tipo de dato que contiene cada campo. Si ponemos **MaxScanRows**=0 le estamos pidiendo que examine toda la tabla.

La quinta línea **CharacterSet**=OEM define el conjunto de caracteres que se va a utilizar. Esto es importante, dependiendo de sobre qué plataforma se va a exportar.

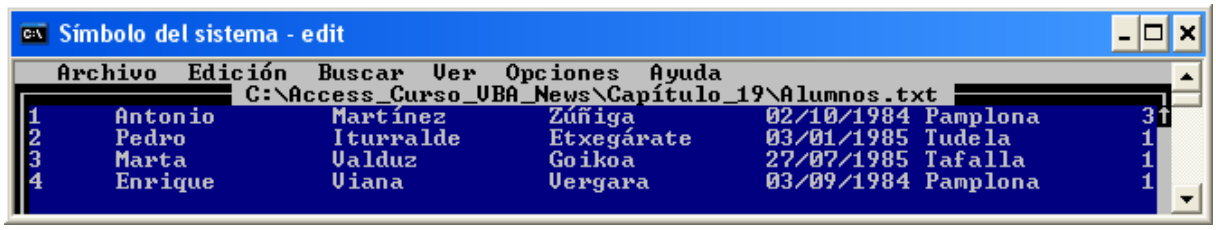
Los posibles valores para **CharacterSet** son **Ansi**, **Oem** y **Unicode**.

En lugar de las palabras **Ansi / Oem / Unicode**, se puede usar unos valores numéricos que las sustituyen. Estos valores son **1252** para el juego de caracteres **Ansi**, **1200** para **Unicode**, y **850** para el **Oem**.

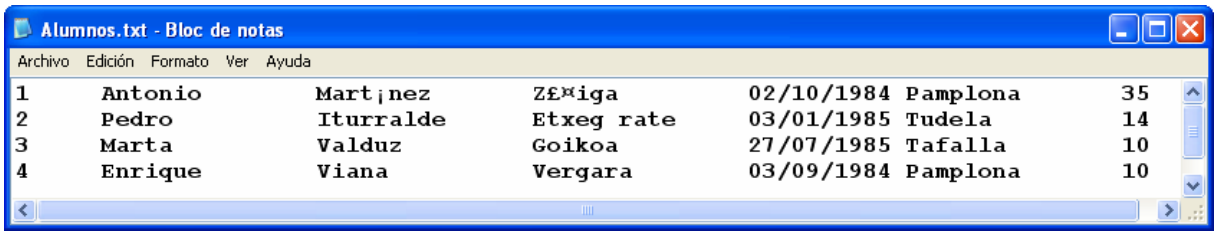
El valor predeterminado existente en el registro de Windows para una configuración regional de Español es el valor **1252** (**Ansi**)..

El usar la opción **Oem** nos permite exportar la tabla sin problemas a un fichero que vaya a ser trabajado con herramientas del tipo **MSDos**.

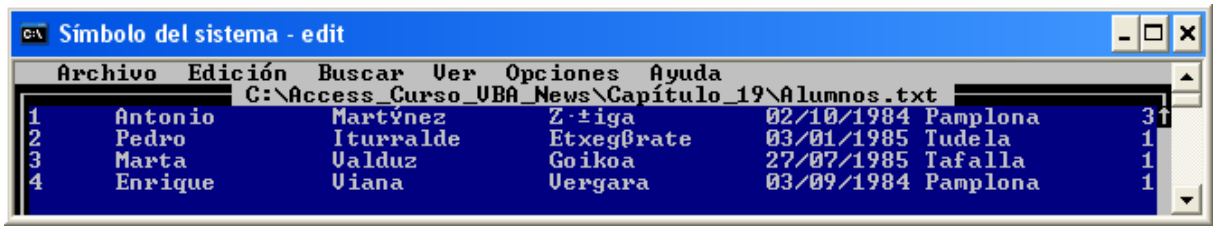
Por ejemplo, si abrimos el fichero desde el emulador de Dos, con el antiguo editor de textos **Edit** nos mostrará lo siguiente:



Si abriéramos el fichero con un editor Windows, por ejemplo con el Bloc de Notas, veríamos el fichero de forma incorrecta.



En cambio, si hubiéramos puesto **CharacterSet**=**Ansi**, el resultado con Edit sería el siguiente:



Vamos que las vocales acentuadas se ven de forma incorrecta, lo mismo que el carácter **ñ**.

Si abrimos ahora el mismo fichero con el Bloc de Notas, lo veríamos de forma correcta.

```
1      Antonio         Martínez        Zúñiga          02/10/1984Pamplona      35
2      Pedro           Iturralde       Etxegárate      03/01/1985Tudela        14
3      Marta           Valduz          Goikoa          27/07/1985Tafalla       10
4      Enrique         Viana           Vergara         03/09/1984Pamplona      10
```

La opción para **CharacterSet Unicode**, permitiría exportar tablas con caracteres internacionales, ubicados por encima del carácter ASCII Nº 255, por ejemplo los caracteres chinos.

© Eduardo Olaz

Desde la versión 2000 Access, puede trabajar con caracteres **Unicode**, que necesitan 2 Bytes para el almacenamiento de cada carácter. Esta es la razón por la que, si no se usa la opción Compresión Unicode en el diseño de las tablas, éstas llegan a ocupar casi el doble de espacio que con las tablas Access de la versión 97 ó anteriores.

Las siguientes líneas empiezan con la sílaba **Col** seguida de un número.

Col1 Col2 . . . **Col7**

En ellas se especifica qué campos van a ser exportados de la tabla, y en qué orden.

Además se indica el tipo de dato y la anchura de la columna en que se va a exportar.

Es importante que la anchura de la columna sea la misma que la del campo en la tabla, ya que en caso contrario se podrían cortar su contenido.

Indicar los campos y el orden en que se van a exportar, es opcional para los ficheros con separador de campos y obligatorio para los ficheros con campos de longitud fija.

El siguiente fichero **Schema.ini** exportará los campos de la tabla a **Alumnos.txt**

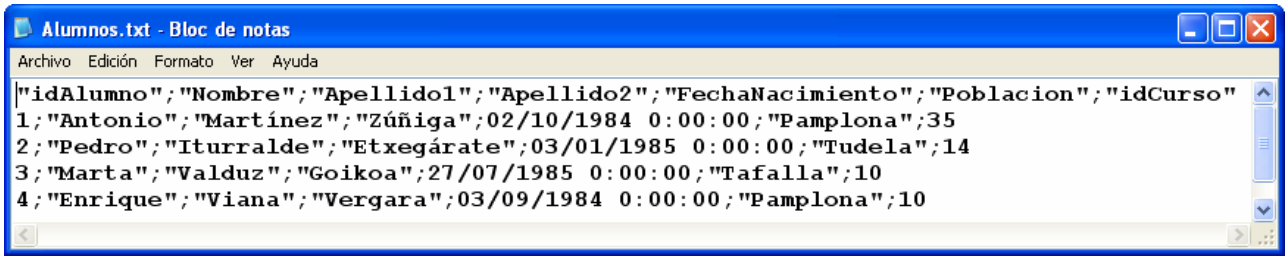
Incluirá una cabecera con los nombres de los campos

Como delimitador de campos usará el carácter punto y coma (**;**).

Hará una exploración previa de toda la tabla para averiguar los tipos de los campos.

En la exportación usara el juego de caracteres **ANSI**.

```
[Alumnos.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=0
CharacterSet=Ansi
```



Vemos que, además del delimitador de campos, los campos tipo texto están contenidos entre comillas.

En cambio ni los valores numéricos ni los del tipo Fecha/Hora, lo hacen.

Las posibilidades de los ficheros Eschema.ini, llegan mucho más lejos que lo expuesto hasta ahora.

Por ejemplo hay todo un juego de posibilidades para dar formato a los campos, por ejemplo para mostrar los números con decimales ó las fechas con un formato específico.

En la exportación anterior vemos que la fecha de nacimiento la ha exportado con el formato **" dd/mm/yyyy hh:nn:ss "**

Probablemente nos interesará que sólo aparezcan los datos de fecha.

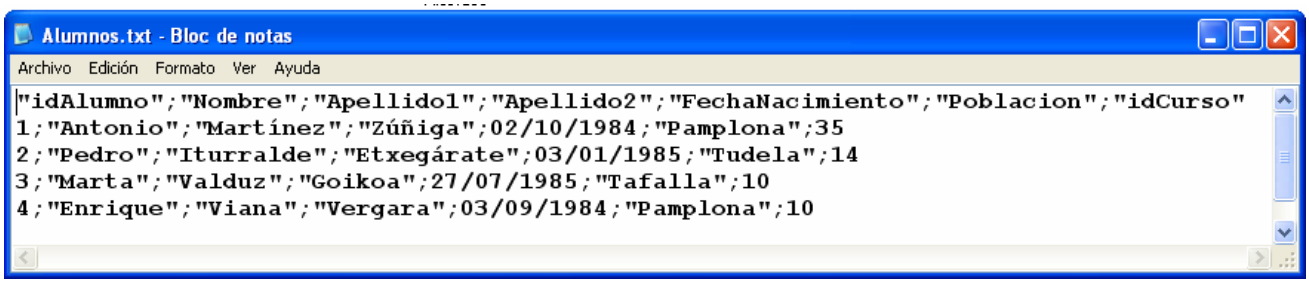
Esto se podría hacer con la siguiente opción de formato:

```
DateTimeFormat = dd/mm/yyyy
```

Si cambiamos el fichero anterior por

```
[Alumnos.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=0
CharacterSet=Ansi
DateTimeFormat = dd/mm/yyyy
```

El resultado sería



Vemos que ahora la fecha de nacimiento se muestra sin la hora.

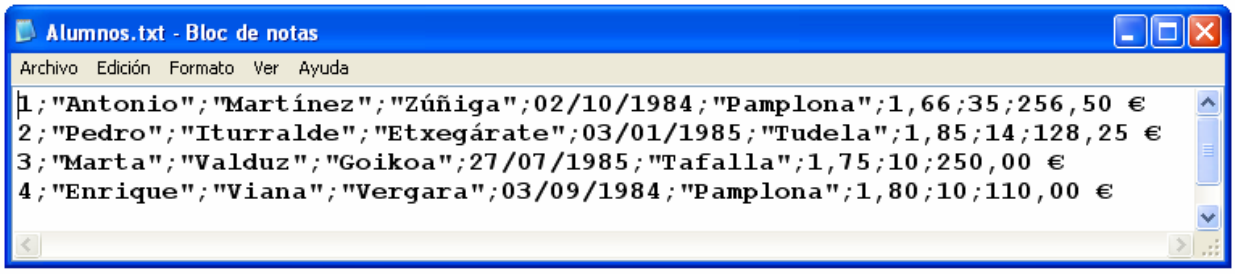
Para ver las opciones de formato que afectan al símbolo monetario y al formato de números decimales, voy a añadir a la tabla original dos nuevos campos:

Altura del tipo **`Single`** (Simple) e **ImportePagado** del tipo **`Currency`** (Moneda).

El nuevo fichero **`Schema.ini`** que vamos a usar será el siguiente:

```
[Alumnos.txt]
ColNameHeader=False
Format=Delimited(;)
MaxScanRows=0
CharacterSet=Ansi
DateTimeFormat=dd/mm/yyyy
CurrencySymbol=€
NumberDigits=2
```

El resultado es



© Eduardo Olaz

A la hora de definir las columnas, un fichero con campos de anchura fija, se debe especificar el tipo de campo y la anchura que debe mostrar. Los tipos de campo que podemos usar son:

Byte, Bit
Currency
DateTime, Date
Double, Float
Long
Memo, LongChar
Short, Integer
Single
Text, Char

Otros tipos de formato en ficheros Schema.ini

Además de los visto hasta ahora, podemos definir cómo se representarán las cantidades monetarias negativas.

Por ejemplo si tenemos el número **-128.60**, podemos representarlo de diferentes formas mediante los valores asignados a la clave **CurrencyNegFormat**, suponiendo que hemos definido como símbolo monetario el del **Euro**, mediante **CurrencySymbol=€**

Valor	Resultado	Valor	Resultado
0	-€128,60	**8**	-128,60 € (Predeterminado)
1	-€128,60	**9**	-€ 128,60
2	€-128,60	**10**	128,60 €-
3	€128,60-	**11**	€ 128,60-
4	(128,60€)	**12**	€ -128,60
5	-128,60€	**13**	128,60- €
6	128,60-€	**14**	(€ 128,6)
7	128,60€-	**15**	(128,60 €)

CurrencyNegFormat=8 devolvería el formato **-128,60 €**

También podemos definir el formato de las cantidades monetarias positivas, mediante la clave **CurrencyNegFormat**, que puede tomar uno de estos valores:

Valor	Resultado	Valor	Resultado
0	€128,60	**2**	€ 128,60
1	€128,60	**3**	128,60 €

CurrencyNegFormat=3 devolvería el formato **53,55 €**

Adicionalmente podríamos definir el separador de decimales, en las cantidades monetarias, con el carácter que nos interese, mediante la clave **CurrencyDecimalSymbol**.

CurrencyDecimalSymbol=. devolvería el formato **53.55 €**

Podemos definir el número de dígitos decimales en un valor monetario, mediante la clave **CurrencyDigits**.

CurrencyDigits=0 devolvería el formato **53 €**

Podemos definir el símbolo que va a usar como separador de miles en los valores monetarios mediante la clave **CurrencyThousandSymbol**.

CurrencyThousandSymbol=, devolvería el formato **1,253.35 €**

DecimalSymbol define el carácter para el separador de decimales.

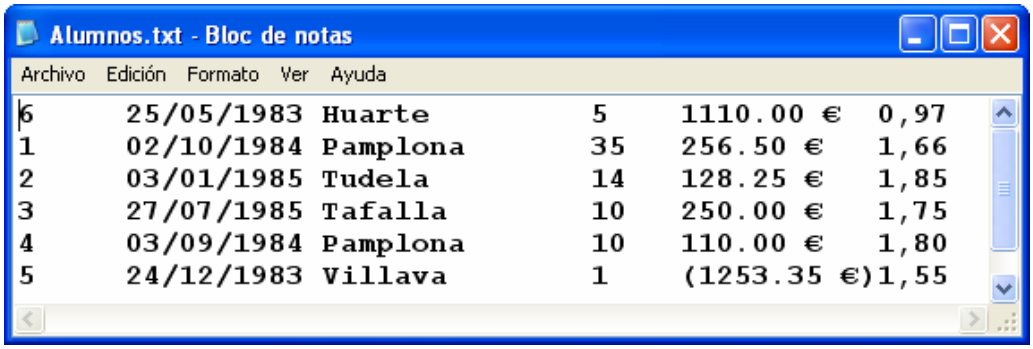
NumberLeadingZeros es un valor Boleano que define si los valores numéricos menores que 1 y mayores que -1, deben llevar un cero a la izquierda.

NumberLeadingZeros=False haría que **0,123** se representara como **,123**

Si definiéramos el fichero **Schema.ini** como

```
[Alumnos.txt]
ColNameHeader=False
Format=FixedLength
MaxScanRows=25
CharacterSet=1252
CurrencySymbol=€
CurrencyDecimalSymbol=.
CurrencyNegFormat=15
NumberDigits=2
Col1= idAlumno Integer Width 6
Col2= FechaNacimiento Date Width 11
Col3= Poblacion Char Width 15
Col4= idCurso Integer Width 5
Col5= ImportePagado Currency Width 11
Col6= Altura Single Width 5
```

Obtendremos este resultado



© Eduardo Olaz

He añadido nuevos datos a la tabla original para comprobar los efectos del formato.

Varios esquemas en un único archivo Schema.ini

Podemos incluir en un archivo de texto varias configuraciones distintas para la Importación / Exportación de diversos archivos.

Lógicamente se deberán separar por secciones, en las que el nombre de la sección, especificará el fichero al que exportar o del que importar.

```
[Alumnos.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=0
CharacterSet=Ansi

[Profesores.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=0
CharacterSet=Ansi
```

Abrir ficheros de texto como si fuera un Recordset

Una vez escrito un fichero de texto podemos hacer que lo cargue como un **Recordset** y pueda trabajar con él.

Para entender de forma simple qué es un **Recordset** diremos que es un conjunto de datos, por ejemplo leídos de una tabla o proveniente de una consulta.

Si existe un fichero **Schema.ini** tratará de aprovecharlo para facilitar la lectura de datos.

```
Sub AbrirFicheroDeTextoDAO()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strcarpeta As String
    Dim strBaseDatos As String
    Dim strTabla As String
    Dim m As Long, n As Long

    strcarpeta = CurrentProject.Path
    strBaseDatos = strcarpeta & "\Datos.mdb"
    strTabla = "Alumnos.txt"

    ' Hacemos que la carpeta donde está el fichero _
      la trate como una base de datos (DAO)
    Set db = OpenDatabase(strcarpeta, _
```

```
                                False, _
                                False, _
                                "Text;")
        ' Carga el recordset desde la tabla de texto
        Set rst = db.OpenRecordset(strTabla)
        With rst
            ' Mostramos los registros
            Do While Not .EOF
                For m = 0 To .Fields.Count - 1
                    ' Mostramos los campos
                    Debug.Print .Fields(m),
                Next m
                Debug.Print
                ' Vamos al siguiente registro
                .MoveNext
            Loop
        End With
        rst.Close
        Set rst = Nothing
        db.Close
        Set db = Nothing
    End Sub
```

En mi ordenador el código anterior imprime lo siguiente:

```
6           25/05/1983    Huarte        5           1110          0,97
1           02/10/1984    Pamplona      35          256,5         1,66
2           03/01/1985    Tudela        14          128,25        1,85
3           27/07/1985    Tafalla       10          250           1,75
4           03/09/1984    Pamplona      10          110           1,8
5           24/12/1983    Villava       1           -1253,35      1,55
```

Este código utiliza la librería de **DAO**. Os recuerdo lo comentado en la página 3 sobre activar la referencia a la librería correspondiente.

Podemos hacerlo de forma semejante usando la librería de **ADO** en vez de la de **DAO.**

```
    Sub AbrirFicheroDeTextoADO()
        Dim cnn As ADODB.Connection
        Dim rst As ADODB.Recordset
        Dim strcarpeta As String
        Dim strTabla As String
        Dim m As Long, n As Long

        strcarpeta = CurrentProject.Path
        strTabla = "Alumnos.txt"
```


© Eduardo Olaz

```
        ' Creamos una nueva conexión (ADO)
         Set cnn = New ADODB.Connection
        ' Asignamos los parámetros de la conexión _
          Incluida la carpeta donde está el fichero.
        With cnn
            .Provider = "Microsoft.Jet.OLEDB.4.0"
            .ConnectionString = "Data Source=" & strcarpeta
            .Properties("Extended Properties") = "TEXT;HDR=Yes"
            .Open
        End With

        ' Creamos un nuevo objeto Recordset
        Set rst = New ADODB.Recordset

        ' Abrimos el Recordset, _
         asignándole la tabla de la carpeta
        rst.Open strTabla, cnn, , , adCmdTable
        With rst
            ' Mostramos los registros
            Do While Not .EOF
           ' Recorremos la colección de campos
                For m = 0 To .Fields.Count - 1
                    ' Mostramos los campos
                    Debug.Print .Fields(m),
                Next m
                Debug.Print
                ' Vamos al siguiente registro
                .MoveNext
            Loop
        End With
        rst.Close
        Set rst = Nothing
        cnn.Close
        Set cnn = Nothing
    End Sub
```

El resultado es exactamente el mismo que con el procedimiento **DAO**.

Nota:

La utilización de la librería **DAO** y la más moderna **ADO**, con sus métodos y propiedades más usuales las veremos en sus correspondientes entregas posteriores.

No se preocupe si no termina de entender este código. En entregas posteriores lo veremos.

Tome el código como referencia cuando ya conozca estos objetos, o como plantilla si necesita cargar datos, por código, desde un fichero de texto.

Ficheros ini

En la época del Windows de 16 bits, (**Windows 3.x**) los programadores, para establecer una serie de valores de inicialización, utilizábamos ficheros **ini**.

Un fichero ini, del que el fichero **`Schema.ini`** es un caso particular, consta de una serie de secciones encabezadas por su nombre, entre paréntesis cuadrados (corchetes), y seguidas de una serie de claves, con sus valores asignados. Recordemos un fichero .

```
                [Alumnos.txt]              ← Sección
Clave →    ColNameHeader=True   ← Valor
Clave →    Format=Delimited(;)  ← Valor
```

Otro ejemplo:

```
[Inicio]
UltimoUsuario=Eduardo
FechaDesconexion=03/09/2005
[Datos]
idAlumno=123
idClase=25
```

Nos indica una serie de variables y los valores que han ido tomando.

Leer y escribir un fichero ini

En un fichero ini podría leerse o escribirse de forma directa, como en cualquier otro fichero de texto, por ejemplo utilizando las sentencias que vimos en la entrega anterior.

Realizar esta tarea “a mano” y por código, puede resultar una tarea ingrata, por decirlo de una forma suave…

Aunque VBA no posee unas herramientas específicas para grabar o leer los ficheros ini, es el propio Windows, con las llamadas **Funciones API**, el que nos brinda esta posibilidad.

Las funciones API las veremos en un capítulo específicamente dedicado a ellas.

En este punto utilizaremos algunas específicas.

No obstante podemos decir que no es algo excesivamente difícil de utilizar.

Las funciones API que vamos a utilizar son:

```
GetPrivateProfileString
GetPrivateProfileInt
GetPrivateProfileSection
WritePrivateProfileString
WritePrivateProfileSection
```

Introducción (necesaria) a la utilización de las funciones API.

Las funciones API son funciones definidas en el interior de unas librerías dll de Windows.

© Eduardo Olaz

Su nombre API viene de **Application Programming Interface** , lo que “en cristiano” viene a significar algo así como **Interfaz para la Programación de Aplicaciones**.

Estas funciones son externas a los lenguajes de programación, como **VBA**, **VB.net**, **C++**, **C#**, **Pascal**, etc… Devuelven un valor y admiten parámetros, que al pasarlos, permiten obtener una serie de valores.

Además pueden ejecutar diversas acciones, en nuestro caso leer o escribir ficheros ini.

Para poder usar una función API, primero debemos declararla

Como ejemplo, vamos a usar la clásica función API que nos devuelve el nombre de la carpeta donde está instalado Windows. Es la función **`GetWindowsDirectory`**

Para poder usar esta función API, desde VBA primero debemos declararla con la instrucción **`Declare`**.

En esa instrucción se debe poner

- El nombre de la función
- La librería de Windows donde está definida
- El nombre de su Alias, o nombre nativo de la función en la librería.
- Los nombres de los parámetros y su tipo
- El tipo de valor devuelto por ella

En nuestro caso la declaración de la función **`GetWindowsDirectory`** es la siguiente

```
Declare Function GetWindowsDirectory _
                        Lib "kernel32.dll" _
                        Alias "GetWindowsDirectoryA" ( _
                        ByVal lpBuffer As String, _
                        ByVal nSize As Long) _
                        As Long
```

Podemos cambiar a nuestra discreción tanto el nombre de la función como el de los parámetros que utiliza, siempre que respetemos el orden y el tipo de de los mismos y el nombre del Alias de la función. Por ejemplo, la siguiente declaración sería equivalente a la primera:

```
Declare Function DameNombreCarpetaWindows _
                        Lib "kernel32.dll" _
                        Alias "GetWindowsDirectoryA" ( _
                        ByVal ContenedorNombre As String, _
                        ByVal TamañoContenedor As Long) _
                        As Long
```

Aunque puede hacerse, y de hecho parece una declaración más clara, es algo que no se suele hacer. Microsoft tiene ya definidas una serie de declaraciones estándar para las funciones API, que tratan de respetar la mayoría de los programadores. Ya hablaremos de este tema en una entrega posterior.

En este caso, el valor devuelto por la función, será la longitud de la cadena que contiene el nombre de la ruta completa de la carpeta de Windows.

Una vez declarada la función podemos utilizarla sin problemas como si fuera una función desarrollada por nosotros mismos, o propia de VBA.

El nombre que utilizaremos para llamar a la función es el que aparece detrás de **`Declare`**

Veamos el código que utilizará la función.

He creado la función **DameCarpetaWindows** que devuelve una cadena con el nombre de la carpeta donde está instalado **Windows**

Este es su código:

```
Public Declare Function GetWindowsDirectory _
                        Lib "kernel32.dll" _
                        Alias "GetWindowsDirectoryA" ( _
                        ByVal lpBuffer As String, _
                        ByVal nSize As Long) _
                        As Long


Public Function DameCarpetaWindows() As String

   'Devuelve la carpeta de Windows

   Const conlngCadena As Long = 255

   Dim strDirectorio As String * conlngCadena
   Dim lngDirectorio As Long
   Dim lngPath As Long

   lngDirectorio = conlngCadena
   lngPath = GetWindowsDirectory( _
                   strDirectorio, _
                   lngDirectorio)
  DameCarpetaWindows = Left$( _
                   strDirectorio, _
                   lngPath)
End Function
```

A la función **GetWindowsDirectory** le pasamos la cadena **strDirectorio** y se almacenará en ella la ruta de la carpeta de **Windows**.

La variable **lngPath** recoge la longitud de la cadena obtenida, por lo que si hacemos

```
Left$(strDirectorio, lngPath)
```

Obtendremos la ruta completa de Windows.

Si en mi ordenador escribo `Debug.Print` **DameCarpetaWindows**`()`

Me mostrará **C:\WINDOWS**, que es donde efectivamente he instalado Windows en mi PC.

Información en Internet sobre las APIs de Windows

Además del MSDN de Microsoft, hay un gran número de páginas de Internet en donde podemos obtener información sobre la utilización de las APIs de Windows.

A continuación pongo los enlaces a algunas de esas páginas:
http://www.silared.com/usuarios/vbasic/api/index.htm

© Eduardo Olaz

http://www.ciberteca.net/articulos/programacion/win32apis/ejemplo.asp

http://www.mentalis.org/index2.shtml La muy recomendable página de AllApi.net.

http://mech.math.msu.su/~vfnik/WinApi/index.html Otra página clásica muy interesante.

http://custom.programming-in.net/

http://www.mvps.org/access/api/

http://vbnet.mvps.org/index.html?code/fileapi/

Declaración de las funciones API para el manejo de un fichero ini.

Creemos un módulo estándar con un nombre descriptivo, como **DeclaracionesAPI**

Lógicamente en él pondremos las declaraciones de las funciones que vamos a utilizar.

Los parámetros de cada función están descritos en los propios comentarios.

```
Option Explicit

    ' GetPrivateProfileString _
      Recupera una cadena en un fichero de inicialización
Declare Function GetPrivateProfileString _
                        Lib "kernel32.dll" _
                        Alias "GetPrivateProfileStringA" ( _
                        ByVal lpApplicationName As String, _
                        ByVal lpKeyName As String, _
                        ByVal lpDefault As String, _
                        ByVal lpReturnedString As String, _
                        ByVal nSize As Long, _
                        ByVal lpFileName As String _
                        ) As Long

    ' lpApplicationName _
      Sección en la que se va a buscar la Clave.
    ' lpKeyName _
      Nombre de la clave a buscar.
    ' lpDefault _
      Valor por defecto a devolver _
      si no se encuentra la clave.
    ' lpReturnedString _
      Cadena en la que escribirá el resultado. _
      Debe tener como mínimo la longitud de nSize.
    ' lpFileName _
      Nombre del archivo de inicialización.
```

```
    ' GetPrivateProfileInt _
      Recupera un entero de un fichero de inicialización
Declare Function GetPrivateProfileInt _
                    Lib "kernel32.dll" _
                    Alias "GetPrivateProfileIntA" ( _
                    ByVal lpApplicationName As String, _
                    ByVal lpKeyName As String, _
                    ByVal nDefault As Long, _
                    ByVal lpFileName As String _
                    ) As Long
    ' nDefault _
      Valor por defecto a devolver _
      si no se encuentra la clave.
    ' GetPrivateProfileSection _
      Recupera una lista con los nombres de todas las claves _
      de una Sección y sus valores.
Declare Function GetPrivateProfileSection _
                    Lib "kernel32" _
                    Alias "GetPrivateProfileSectionA" ( _
                    ByVal lpApplicationName As String, _
                    ByVal lpReturnedString As String, _
                    ByVal nSize As Long, _
                    ByVal lpFileName As String _
                    ) As Long
    ' lpReturnedString _
      Lista en la que se cargan las claves y sus valores. _
      Cada cadena está separada por un valor Nulo _
      Al final de la lista se encontrarán dos valores nulos.
    ' nSize _
      Tamaño asignado a lpReturnedString

Declare Function WritePrivateProfileString _
                    Lib "kernel32.dll" _
                    Alias "WritePrivateProfileStringA" ( _
                    ByVal lpApplicationName As String, _
                    ByVal lpKeyName As String, _
                    ByVal lpString As String, _
                    ByVal lpFileName As String _
                    ) As Long
    ' lpString _
```


© Eduardo Olaz

```
      Valor a escribir en una clave. _
      Si lo que se quiere es borrar la clave _
      Pasar el valor vbNullString

Declare Function WritePrivateProfileSection _
                  Lib "kernel32" _
                  Alias "WritePrivateProfileSectionA" ( _
                  ByVal lpApplicationName As String, _
                  ByVal lpString As String, _
                  ByVal lpFileName As String _
                  ) As Long
    ' lpApplicationName _
      Sección en la que vamos a escribir las Claves y Valores
    ' lpString _
      Lista en la que se pasan las claves y sus Valores. _
      Como en el caso de lpReturnedString _
      cada cadena está separada por un valor Nulo _
      Al final de la lista se pondrán dos valores nulos.
Declare Function GetPrivateProfileSectionNames _
                  Lib "kernel32" _
                  Alias "GetPrivateProfileSectionNamesA" ( _
                  ByVal lpReturnedString As String, _
                  ByVal nSize As Long, _
                  ByVal lpFileName As String _
                  ) As Long
    ' lpReturnedString contiene la lista de las Secciones _
      cada cadena está separada por un valor Nulo _
      Al final de la lista se pondrán dos valores nulos.
```

Escritura y lectura en un fichero ini

Para escribir los procedimientos, de escritura y lectura en un fichero ini vamos a abrir un nuevo módulo estándar al que vamos a llamar, por ejemplo, **`ProcedimientosINI`**.

Ya hemos dicho que un fichero ini posee un **Nombre**, con la **extensión ini**, una serie de **Secciones** con su nombre escrito entre corchetes (paréntesis cuadrados) y una serie de **Claves** con un **Valor**.

Vamos a crear un procedimiento que escriba un valor de una clave en una sección de un fichero ini. A este procedimiento lo vamos a llamar **`EscribeClaveINI`**.

Para ello utilizará la función API: **`WritePrivateProfileString`**.

```
    Public Sub EscribeClaveINI( _
                  ByVal FicheroINI As String, _
                  ByVal Seccion As String, _
```

```
                    ByVal Clave As String, _
                    ByVal Valor As String, _
                    Optional ByVal Carpeta As String)

    Dim lngRetornado As Long

    ' Se hace un pequeño control _
      La Carpeta pasada
    Carpeta = CarpetaValida(Carpeta)
    ' Si la carpeta no existiera se generaría un error
    ' Validamos también el fichero
    ' Si el fichero no existiera generaría un error
    Fichero = FicheroValido(Fichero, Carpeta)
    ' Llamamos ahora a la función API
    '      WritePrivateProfileString _
      que hemos declarado anteriormente. _
      El valor de retorno de la función _
      se lo asignamos a lngRetornado
    lngRetornado = WritePrivateProfileString( _
                            Seccion, _
                            Clave, _
                            Valor, _
                            Carpeta & FicheroINI)
End Sub
```

Este es el código de la función **CarpetaValida**, que vamos a usar en varios procedimientos, como una comprobación **muy primaria** de la carpeta pasada:

```
Public Function CarpetaValida( _
                Optional ByVal Carpeta As String = "" _
                ) As String
    On Error GoTo HayError
    ' Esta función comprueba la carpeta pasada como _
      Parámetro.
    Carpeta = Trim(Carpeta) ' Quita blancos de los extremos
    ' Si no se pasa nada, devuelve la de la aplicación.
    If Carpeta = "" Then
        Carpeta = CurrentProject.Path
    End If
    ' Si el nombre de la carpeta no acaba con "\" lo pone
```


© Eduardo Olaz

```
    If Right(Carpeta, 1) <> "\" Then
        Carpeta = Carpeta & "\"
    End If
    ' Si no existe la carpeta genera un error'
    If Len(Dir(Carpeta, vbDirectory)) = 0 Then
        Err.Raise 1000, _
                "CarpetaValida", _
                "No existe la Carpeta " _
                & Carpeta
    End If
Salida:
    CarpetaValida = Carpeta
    Exit Function
HayError:
    If Err.Number = 1000 Then
        MsgBox "No existe la carpeta " _
            & Carpeta, _
            vbCritical + vbOKOnly, _
            " Error en la función " & Err.Source
    Else
        MsgBox "Se ha producido el error " _
            & Format(Err.Number, "#,##0") _
            & vbCrLf _
            & Err.Description, _
            vbCritical + vbOKOnly, _
            " Error no controlado en la función " _
            & "CarpetaValida()"
    End If
    Resume Salida
End Function
```

De una forma similar a como hemos creado la función **`CarpetaValida`**, vamos a crear la función **`FicheroValido`**, que comprobará si existe el fichero.

```
Public Function FicheroValido( _
                ByVal Fichero As String, _
```

```
                Optional ByVal Carpeta As String = "" _
                ) As String
    On Error GoTo HayError
    ' Esta función comprueba la existencia del fichero _
      pasado como Parámetro.
    Carpeta = Trim(Carpeta) ' Quita blancos de los extremos
    ' Si no se pasa nada, devuelve la de la aplicación.
    If Carpeta = "" Then
        Carpeta = CurrentProject.Path
    End If
    ' Si el nombre de la carpeta no acaba con "\" lo pone
    If Right(Carpeta, 1) <> "\" Then
        Carpeta = Carpeta & "\"
    End If
    ' Si no existe el fichero genera un error'
    If Len(Dir(Carpeta & Fichero)) = 0 Then
        Err.Raise 1100, _
                "FicheroValido", _
                "No existe el fichero " _
                & Fichero
    End If
Salida:
    FicheroValido = Fichero
    Exit Function
HayError:
    If Err.Number = 1100 Then
        MsgBox "No existe el fichero " _
            & Carpeta & Fichero, _
            vbCritical + vbOKOnly, _
            " Error en la función " & Err.Source
    Else
        MsgBox "Se ha producido el error " _
            & Format(Err.Number, "#,##0") _
            & vbCrLf _
```

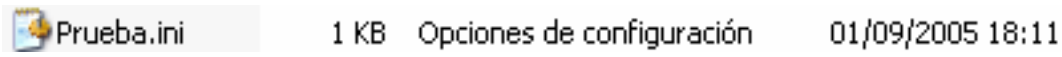

© Eduardo Olaz

```
            & Err.Description, _
            vbCritical + vbOKOnly, _
            " Error no controlado en la función " _
            & "FicheroValido()"
    End If
    Resume Salida
End Function
```

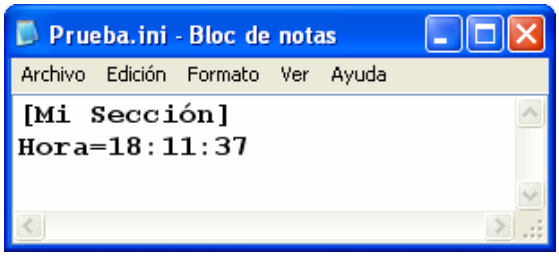
Vamos a probar el procedimiento, para lo que en la ventana de depuración escribimos:

```
EscribeClaveINI "Prueba.ini", "Mi Sección", "Hora", CStr(Time)
```

Si abrimos con el explorador de Windows la carpeta donde está ubicado el programa, veremos que tenemos un bonito fichero Prueba.ini



Si abrimos el fichero con el Bloc de notas veremos que contiene lo que en principio esperábamos:



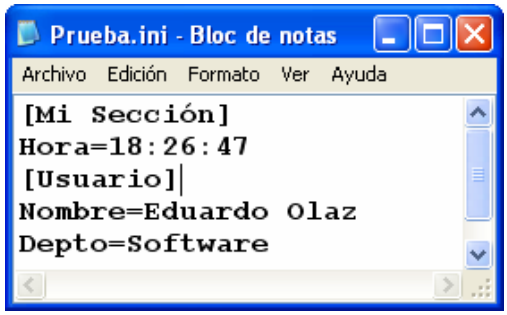
Si volvemos a efectuar la misma llamada, podremos comprobar que se actualiza la hora.

Vamos a llamarlo ahora con estos parámetros:

```
EscribeClaveINI "Prueba.ini", "Usuario", "Nombre", "Eduardo"
EscribeClaveINI "Prueba.ini", "Usuario", "Depto", "Software"
```

Si lo editamos con el bloc de notas, veremos que `Prueba.ini` ha cambiado.

De hecho se ha actualizado la hora, se ha añadido la sección Usuario que contiene las claves Nombre, con mi nombre, y Depto con el valor Software.



¿Cómo podríamos **borrar una clave por código**?

Muy sencillo, pasándole la cadena nula representada por la constante **vbNullString**

Si hacemos

```
EscribeClaveINI "Prueba.ini","Usuarios", "Depto", vbNullString
```

Podremos comprobar que ha desaparecido la clave **Depto** en **Prueba.ini**

Conclusiones respecto al código anterior:

- Si el fichero **ini** no existiera, lo crearía
- Si existiera una clave en la sección correspondiente, la actualizaría con el nuevo valor pasado como parámetro.
- Si no existe la clave, la crea en su sección
- Si no existiera la sección la crearía.
- Para borrar una clave le pasamos como valor **vbNullString**

Vamos a ver cómo podríamos leer La clave **Nombre** en la Sección **Usuario**.

Para ello vamos a utilizar la función API **GetPrivateProfileString**, que nos devolverá una cadena con el **valor** de la **clave**.

```
Public Function LeeCadenaClaveINI( _
                ByVal FicheroINI As String, _
                ByVal Seccion As String, _
                ByVal Clave As String, _
                Optional ByVal Carpeta As String, _
                Optional ByVal ValorDefecto As String _
                ) As String

    Const conLongitud As Long = 255
    Dim lngRetornado As Long
    Dim strDevuelto As String

    Carpeta = CarpetaValida(Carpeta)
    FicheroINI= FicheroValido(FicheroINI, Carpeta)

    ' Creamos la cadena  contenedora con 255 caracteres * _
      Esta cadena la pasaremos como cadena que contendrá _
      el valor leído.
    strDevuelto = String(conLongitud, "*")
    ' Llamamos ahora a la función API
    lngRetornado = GetPrivateProfileString( _
```


© Eduardo Olaz

```
                                    Seccion, _
                                    Clave, _
                                    ValorDefecto, _
                                    strDevuelto, _
                                    Len(strDevuelto), _
                                    Carpeta & FicheroINI)
        ' Devolvemos el dato
        ' El número de caracteres leídos estará en lngRetornado
        ' La cadena leída en strDevuelto
        LeeCadenaClaveINI = Left(strDevuelto, lngRetornado)
    End Function
```

Con el código anterior, si escribimos la sentencia

```
LeeCadenaClaveINI("Prueba.ini", "Usuario", "Nombre")
```

nos devolverá la cadena "**Eduardo**". Con lo que habremos leído una cadena de tipo String.

Si supiéramos que determinada clave contiene un valor **entero**, podríamos utilizar la función API , **GetPrivateProfileInt** que es más sencilla de manejar, ya que la propia función devolvería el entero solicitado en formato Long.

Antes que nada, vamos a grabar en nuestro fichero ini, un valor entero **Long**.

Para ello utilizaremos el procedimiento creado con anterioridad **EscribeClaveINI**.

Supongamos que queremos grabar el valor **123456**., como valor de la Clave **idUsuario** en la sección **Usuario** de nuestro fichero ini.

Llamaríamos a nuestro procedimiento **EscribeClaveINI** de la siguiente forma:

```
EscribeClaveINI "Prueba.ini", _
                "Usuario", _
                "idUsuario", _
                CStr(12345)
```

Tras esto, el fichero **Prueba.ini** contendrá lo siguiente:

```
[Mi Sección]
Hora=18:26:47
[Usuario]
Nombre=Eduardo
Depto=Software
idUsuario=12345
```

Vamos ahora a crearnos una función llamada **LeeEnteroClaveINI**, que nos devolverá un entero de tipo **Long**, llamando a la función API **GetPrivateProfileInt**.

```
Public Function LeeEnteroClaveINI( _
                ByVal FicheroINI As String, _
                ByVal Seccion As String, _
```

```
                    ByVal Clave As String, _
                    Optional ByVal Carpeta As String, _
                    Optional ByVal ValorDefecto As Long _
                    ) As Long

    Carpeta = CarpetaValida(Carpeta)
    FicheroINI= FicheroValido(FicheroINI, Carpeta)

    ' Invocamos directamente la función API _
         GetPrivateProfileInt
    LeeEnteroClaveINI = GetPrivateProfileInt( _
                            Seccion, _
                            Clave, _
                            ValorDefecto, _
                            Carpeta & FicheroINI)
End Function
```

Si llamamos a la función mediante:

LeeEnteroClaveINI("Prueba.ini", "Usuario", "idUsuario", , 0)

nos devolverá la última clave introducida, es decir **12345**

También podríamos haber extraído ese valor usando la función **LeeCadenaClaveINI**

LeeCadenaClaveINI("Prueba.ini", "Usuario", "idUsuario", , 0)

Pero en ese caso nos lo hubiera devuelto en forma de cadena, es decir **"12345"**.

Lista las Secciones de un fichero ini

La lista de las secciones de un fichero ini, se puede obtener mediante la función API **GetPrivateProfileSectionNames**.

Esta función devuelve una cadena con las secciones separadas por el valor ASCII **0**, y a continuación del nombre de la última sección, poniendo dos caracteres Nulos de ASCII **0**.

Vamos a crear una función a la que llamaremos **SeccionesINI**, que devolverá un **array** con los nombres de las secciones.

Veamos su código:

```
Public Function SeccionesINI( _
                    ByVal FicheroINI As String, _
                    Optional ByVal Carpeta As String _
                    ) As Variant
    On Error GoTo HayError
    Dim lngLongitud As Long
    Dim lngDevuelto As Long
```


© Eduardo Olaz

```
    Dim strContenedor As String
    Dim strSecciones As String
    Dim aSecciones As Variant
    On Error GoTo HayError

    ' lngLongitud será la longitud de strContenedor _
      que recogerá los nombres de las secciones
    lngLongitud = 2 ^ 12
    strContenedor = String(lngLongitud, "*")

    Carpeta = CarpetaValida(Carpeta)
    FicheroINI = FicheroValido(FicheroINI, Carpeta)

    lngDevuelto = GetPrivateProfileSectionNames( _
                            strContenedor, _
                            Len(strContenedor), _
                            Carpeta & FicheroINI)
    ' Si el valor devuelto fuera 0 sería porque _
      no ha encontrado secciones en el fichero _
      o no ha encontrado el fichero
    If Not CBool(lngDevuelto) Then
        ' Preparamos un array sin datos
        ReDim aSecciones(0 To 0)
        aSecciones(0) = ""
    Else
        strSecciones = Left(strContenedor, lngDevuelto - 1)
        ' El separador es el carácter Chr$(0)
        ' Split devuelve un Array _
          desde una cadena con separadores
        aSecciones = Split(strSecciones, Chr$(0))
    End If
    ' Devuelve el array
    SeccionesINI = aSecciones
Salida:
    SeccionesINI = aSecciones
    Exit Function
HayError:
        MsgBox "Se ha producido el error " _
            & Format(Err.Number, "#,##0") _
            & vbCrLf _
```

```
                & Err.Description, _
                vbCritical + vbOKOnly, _
                " Error no controlado en la función " _
                & "SeccionesINI()"
        ReDim aSecciones(0 To 0)
            aSecciones(0) = ""
        Resume Salida
    End Function
```

Si efectuamos una llamada a la función mediante **`SeccionesINI("Prueba.ini")(0)`**

en mi ordenador, considerando que antes hemos creado esas secciones, devuelve el nombre de la sección **`"Usuario"`**

Si Hacemos **`SeccionesINI("Prueba.ini")(1)`**, devuelve **`"Mi Sección"`**

Fijémonos en que la función **`SeccionesINI`** devuelve un **`Array`**, y por tanto se le puede llamar pasándole, además de los parámetros propios de la misma, el **`índice`** del Array devuelto.

Podemos generar un procedimiento que muestre los nombres de las secciones de un fichero del tipo **`ini`**.

```
    Public Sub MuestraSeccionesIni( _
                    ByVal FicheroINI As String, _
                    Optional ByVal Carpeta As String)
        Dim aSecciones As Variant
        Dim i As Long

        Carpeta = CarpetaValida(Carpeta)
        aSecciones = SeccionesINI(FicheroINI, Carpeta)

        Debug.Print "Secciones de " _
                    & Carpeta & FicheroINI
        For i = 0 To UBound(aSecciones)
            Debug.Print aSecciones(i)
        Next i
    End Sub
```

Si ejecuto este procedimiento en mi ordenador me muestra

`Secciones de C:\Access_Curso_VBA_News\Capítulo_19\Prueba.ini`
`Mi Sección`
`Usuario`

Así como tenemos posibilidad de leer, en un solo paso, todas las secciones de un fichero **`ini`**, podemos también leer ó escribir todas las **`claves`**, y sus **`valores`** correspondientes, dentro de una **`Sección`**.

© Eduardo Olaz

Lectura y escritura en bloque de toda una sección en un fichero ini

Como he comentado, además de leer y escribir cada clave, en un procedimiento ini, podemos también leer o escribir todos los elementos de una sección de un solo paso.

Para poder leer una sección completa, usaremos la función API

`GetPrivateProfileSection`.

Recordemos que la declaración de esta función es

```
Declare Function GetPrivateProfileSection _
                 Lib "kernel32" _
                 Alias "GetPrivateProfileSectionA" ( _
                 ByVal lpApplicationName As String, _
                 ByVal lpReturnedString As String, _
                 ByVal nSize As Long, _
                 ByVal lpFileName As String _
                 ) As Long
```

`lpApplicationName` representa el nombre de la sección.

`lpReturnedString` es la cadena que recogerá los datos de la sección, con las claves y valores separados por el carácter `ASCII 0`.

`nSize` es el tamaño con el que se ha pasado la cadena.

El valor Long devuelto por la función es el número de caracteres cargados en la variable `lpReturnedString`.

El nombre que vamos a poner a la función que va a devolvernos los datos de una sección va a ser **`LeeSeccionINI`**.

```
Public Function LeeSeccionINI( _
                ByVal FicheroINI As String, _
                ByVal Seccion As String, _
                Optional ByVal Carpeta As String _
                ) As Variant
    ' Devuelve los datos de una sección en un array _
      de 2 dimensiones, de tipo String
    '  aDatos(1 to N, 0 to 1) As String
    ' Para ello utilizará la función API _
      GetPrivateProfileSection

    Dim lngLongitud As Long
    Dim lngDevuelto As Long
    Dim strContenedor As String
    Dim strSecciones As String
```

```
        ' hacemos que el tamaño de la cadena contenedora _
          sea "razonablemente grande"
        lngLongitud = 2 ^ 15 - 1
        strContenedor = String(lngLongitud, "*")

        Carpeta = CarpetaValida(Carpeta)
        FicheroINI = FicheroValido(FicheroINI)

        lngDevuelto = GetPrivateProfileSection( _
                        Seccion, _
                        strContenedor, _
                        lngLongitud, _
                        Carpeta & FicheroINI)
        ' La cadena con los datos separados por Chr$(0) _
          está en los primeros lngDevuelto - 1 Caracteres _
          de strContenedor
        strContenedor = Left(strContenedor, lngDevuelto - 1)
        ' Llamamos a la función que extrae los datos _
          desde la cadena
        LeeSeccionINI = DescomponerCadenaConNulos( _
                    strContenedor)
    End Function
```

La función auxiliar **DescomponerCadenaConNulos** descompone los datos de una cadena del tipo **Clave1=Dato1 Clave2=Dato2 Clave3=Dato3 . . . ClaveN=DatoN**, en la que los sucesivos pares de **Clave_i=Dato_i**, están separados por el carácter **ASCII** de índice = **0** y los pasa a un array de 2 dimensiones.

Veamos cómo podemos hacer que esta función devuelva un **array** más manejable.

```
    Public Function DescomponerCadenaConNulos( _
                            ByVal Cadena As String _
                            ) As Variant
        ' Esta función recibe una cadena _
          con los datos separados por nulos _
          y devuelve un array String del tipo _
          aDatos(0 to n-1, 0 to 1)
        Dim lngLongitud As Long
        Dim lngDatos As Long
        Dim strNulo As String
        Dim aCadenas As Variant
        Dim aResultado() As String
        Dim aLinea As Variant
```


© Eduardo Olaz

```
    Dim i As Long

    strNulo = Chr(0)
    lngLongitud = Len(Cadena)

            ' Devolverá una matriz bidimensional _
              si existe algún separador
            If InStr(Cadena, strNulo) > 0 Then
                aCadenas = Split(Cadena, strNulo)
            Else
                ' Si no hay datos devuelve _
                  un array con caracteres vacíos
                ReDim aResultado(0 To 0, 0 To 1)
                aResultado(0, 0) = ""
                aResultado(0, 1) = ""
                DescomponerCadenaConNulos = aResultado
                Exit Function
            End If

            lngDatos = (UBound(aCadenas) + 1)
            ReDim aResultado(0 To lngDatos, 0 To 1)

            For i = 0 To lngDatos - 1
                aLinea = Split(aCadenas(i), "=")
                aResultado(i, 0) = aLinea(0)
                aResultado(i, 1) = aLinea(1)
            Next i
    DescomponerCadenaConNulos = aResultado

End Function
```

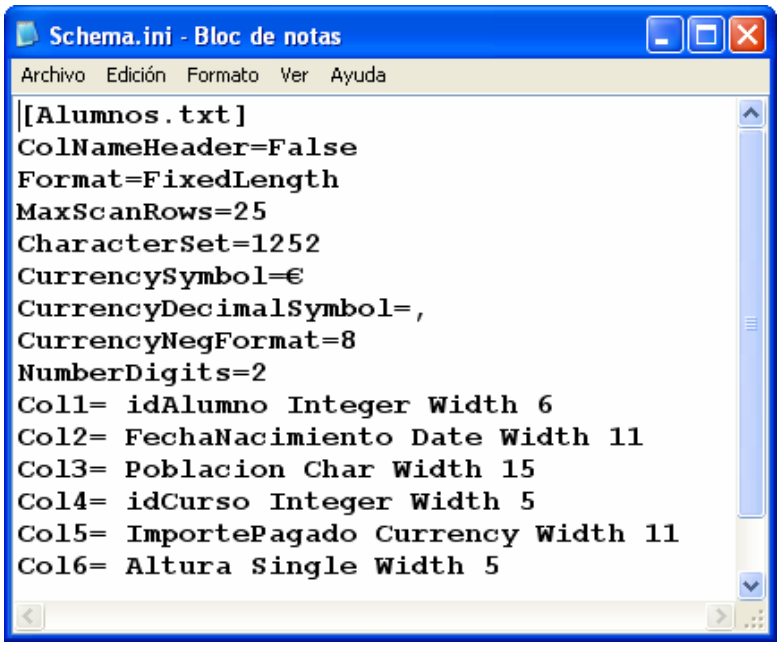
Lectura de una sección

Vamos a probar estas 2 últimas funciones.

Para ello utilizaremos el fichero **`Schema.ini`** que habíamos construido para usarlo en la exportación – importación de datos, en un punto anterior de esta entrega.

No debemos olvidar que sólo es un caso muy especial de fichero **`ini`**.

En este momento, en la carpeta de mi ordenador, el fichero **`Schema.ini`** contiene las siguientes líneas:



Vemos que sólo tiene una sección llamada **`Alumnos.txt`**

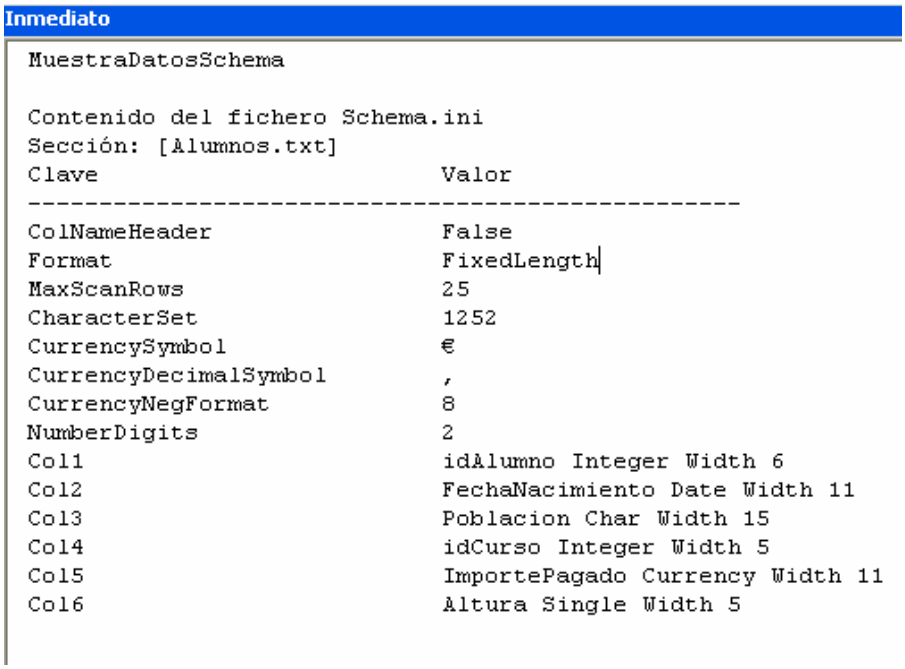
Crearemos un procedimiento que muestre los datos de esa sección en el fichero Schema.ini.

```
Public Sub MuestraDatosSchema()
    Const conTabulacion As Long = 30
    Dim aDatos As Variant
    Dim i As Long
    Dim strSeccion As String

    strSeccion = "Alumnos.txt"
    ' Llamamos a la función LeeSeccionINI
    aDatos = LeeSeccionINI("Schema.ini", strSeccion)
    ' Muestra los datos en la ventana Inmediato
    Debug.Print
    Debug.Print "Contenido del fichero Schema.ini"
    Debug.Print "Sección: [" & strSeccion & "]"
    Debug.Print "Clave", Tab(conTabulacion); "Valor"
    Debug.Print String(conTabulacion + 20, "-")
    For i = 0 To UBound(aDatos, 1)
        Debug.Print aDatos(i, 0),
        Debug.Print Tab(conTabulacion); aDatos(i, 1)
    Next i

End Sub
```

Tras ejecutar este procedimiento nos mostrará lo siguiente en la ventana de depuración o Inmediato.



Que, como podemos comprobar, efectivamente se corresponde con los datos contenidos en el fichero **`Schema.ini`**.

Escritura de una sección completa

Para poder escribir una sección completa, usaremos la función API

`WritePrivateProfileSection`.

Recordemos la declaración de esta función:

```
Declare Function WritePrivateProfileSection _
                 Lib "kernel32" _
                 Alias "WritePrivateProfileSectionA" ( _
                 ByVal lpApplicationName As String, _
                 ByVal lpString As String, _
                 ByVal lpFileName As String _
                 ) As Long
```

`lpApplicationName` Sección en la que vamos a escribir las **Claves** y **Valores**

`lpString` Lista que contiene las claves y sus Valores.

`lpFileName` es el fichero en el que se va a escribir los datos.

Al procedimiento a crear lo vamos a llamar **`EscribeSeccionINI`**.

El parámetro **`Datos`**, contendrá un **`array`** semejante al devuelto por la función anterior **`LeeSeccionINI`**. Por tanto será de dos dimensiones conteniendo datos de tipo **`String`**.

Crearemos también la función **`ConstruyeCadenaConNulos`** que hará la función inversa a la que hace la función anterior **`DescomponerCadenaConNulos`**; es decir **`ConstruyeCadenaConNulos`** extraerá los datos del **`array Datos`** y los añadirá a una cadena del tipo **`Clave_i=Valor_i`**, teniendo como separador el carácter **`ASCII 0`**.

Al final de la cadena pondrá **2** caracteres **`Ascii 0`**.

```
Public Sub EscribeSeccionINI( _
                ByVal FicheroINI As String, _
                ByVal Seccion As String, _
                ByVal Datos As Variant, _
                Optional ByVal Carpeta As String)
    ' Presuponemos que se le pasa un Array de _
      2 dimensiones, de tipo String
    ' aDatos(1 to N, 0 to 1) As String
    ' Para grabar los datos vamos a usar la función API _
        WritePrivateProfileSection
    Dim strCadenaDatos As String
    Dim lngResultado As Long

    ' Como en los casos anteriores _
      efectuamos una cierta validación
    Carpeta = CarpetaValida(Carpeta)
    FicheroINI = FicheroValido(FicheroINI)

    ' Pasamos los datos de Datos a strCadenaDatos
    strCadenaDatos = ConstruyeCadenaConNulos(Datos)

    ' Llamamos a la función API WritePrivateProfileSection
    lngResultado = WritePrivateProfileSection( _
                            Seccion, _
                            strCadenaDatos, _
                            Cadena & FicheroINI)
End Sub
```

Si la sección no existiera en el fichero, la crearía.

Si la sección ya existiera en el fichero, sustituirá todos los valores anteriores por los nuevos.

La función **ConstruyeCadenaConNulos** que convierte un **array** bidimensional a una cadena con separadores nulos, es la siguiente

```
Public Function ConstruyeCadenaConNulos( _
                    Datos As Variant _
                    ) As String
    Dim strDatos As String
    Dim i As Long
    Dim lngIndicemayor As Long
    Dim strNulo As String

    strNulo = Chr(0)
```


© Eduardo Olaz

```
        ' Vamos añadiendo los sucesivos eslabones
        For i = 0 To UBound(Datos)
            strDatos = strDatos _
                    & Datos(i, 0) & "=" & Datos(i, 1) _
                    & strNulo
        Next i
        ' Añadimos un nulo adicional al final de la cadena
        ConstruyeCadenaConNulos = strDatos & strNulo
    End Function
```

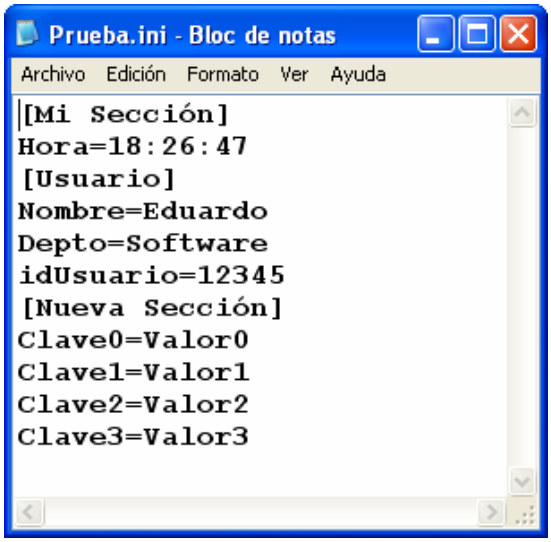
Este procedimiento pone a prueba **`EscribeSeccionINI`**.

```
    Public Sub PruebaGrabacionSeccion()
        Const conDatos As Long = 5
        Dim aDatos(conDatos - 1, 1) As String
        Dim strSeccion As String
        Dim i As Long

        strSeccion = "Nueva Sección"
        For i = 0 To conDatos - 1
            aDatos(i, 0) = "Clave" & CStr(i)
            aDatos(i, 1) = "Valor" & CStr(i)
        Next i

        Call EscribeSeccionINI( _
                "Prueba.ini", strSeccion, aDatos)
    End Sub
```

El resultado será



El Registro de Windows.

Hasta la aparición de Windows 95, la forma habitual de almacenar los parámetros de configuración de una aplicación, era la utilización de ficheros **ini**, tal y como hemos visto en los puntos anteriores de esta entrega.

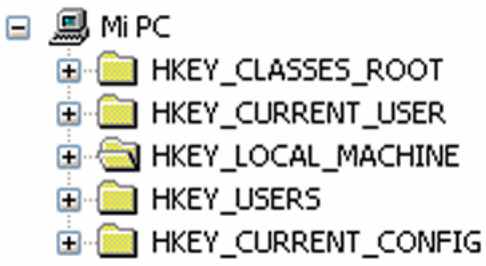
Desde la versión de Windows 95, tenemos a nuestro alcance una herramienta mucho más potente, que evita además la proliferación de ficheros ini, por todas las carpetas de las aplicaciones instaladas. Esta herramienta es el fichero **Regedit.exe**.

Este programa nos permite editar el **Registro de Configuraciones de Windows**, conocido también por su nombre reducido "**El registro**".

Normalmente suele hallarse en la carpeta **C:\Windows**.

El registro de Windows se divide en varios bloques principales.

Mi versión actual de Windows (XP Profesional - Service Pack 2) tiene los siguientes bloques:



Cada uno de los bloques almacena los parámetros de las diversas aplicaciones instaladas, y por supuesto de muchos de los virus, troyanos o programas SpyWare que hayan podido infectar nuestro ordenador.

Para poder leer y escribir en esas secciones debemos utilizar una serie de procedimientos API que nos suministra el propio Windows.

La utilización de estos procedimientos API la veremos en una sección de esta publicación en la que profundizaremos en esta y otras posibilidades no contempladas por el propio VBA.

Escritura y lectura del registro con VBA

Aunque VBA, por sí mismo, tiene limitado el acceso al registro, sí que existe una sección específica del mismo en la que podemos almacenar y leer nuestros parámetros.

Esta sección es

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
```

En esta carpeta del registro podremos almacenar y leer los datos que necesitemos, en unas ramas que partirán de ella con el nombre que le demos a nuestra aplicación.

Por ejemplo, supongamos que tenemos una aplicación llamada Gestión de Socios, y queremos que cada vez que arranquemos el programa averiguar el código del último socio creado, y la última fecha en la que se ha utilizado el programa.

Podríamos crear una sub-Carpeta, que cuelgue de la principal, llamada **`Gestión De Socios`**, creando una sección llamada **`Valores Finales`**, y en ella poner las claves **`Último Socio`** y **Fecha Utilización**, asignando su correspondiente valor.

El procedimiento a usar para grabar los datos en las claves es el **`SaveSetting`**.

Para Leer los datos podemos usar las funciones **`GetSetting`** y **`GetAllSetting`**.

© Eduardo Olaz

Para borrar las claves y sus datos deberemos utilizar **DeleteSetting**.

Me permito sugeriros que hagáis **copia de seguridad del registro** antes de manipularlo, más que nada por el desastre que podría ocurrir tras un manejo inadecuado.

Instrucción SaveSetting

Actualiza o crea una entrada para una aplicación en el registro de configuración de Windows.

Su sintaxis es

```
SaveSetting NombreAplicación, sección, clave, valor
```

NombreAplicación es el nombre de la subcarpeta que queremos poner colgando de

VB and VBA Program Settings y que normalmente tiene como nombre el de la propia aplicación.

sección es el nombre de la sección donde queremos grabar el valor de la clave.

valor es el valor, o la expresión que devuelve un valor a grabar.

Supongamos que en nuestro programa **Gestión De Socios** queremos almacenar dentro del registro de Windows los valores **1011** como último socio, y **3 de septiembre de 2005** como fecha de última utilización.

Para ello podemos crear el procedimiento **GrabarDatosDeSalida** al que llamaremos inmediatamente antes de cerrar el programa, pasándole los parámetros adecuados.

Podría ser algo tan simple como esto:

```
Public Sub GrabarDatosDeSalida( _
                        ByVal Clave As String, _
                        ByVal Valor As Variant)

    SaveSetting "Gestión De Socios", _
                    "Valores finales", _
                    Clave, _
                    Valor
End Sub
```

Inmediatamente antes de cerrar la aplicación, en alguna parte del código, se hará la siguiente llamada:

```
GrabarDatosDeSalida "Último Socio", 1011
GrabarDatosDeSalida " Fecha Utilización", #9/3/2005#
```

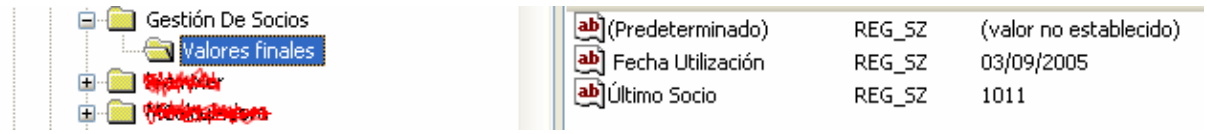
Normalmente en vez de introducir los valores concretos se efectuaría utilizando las variables que los contienen, como parámetros.

Supongamos que el número de socio lo hemos guardado, inmediatamente después de crearlo, en la variable **lngNumeroSocio**, y la fecha de ejecución del programa en la variable **datEjecucion**.

```
GrabarDatosDeSalida "Último Socio", lngNumeroSocio
```

```
GrabarDatosDeSalida " Fecha Utilización", datEjecucion
```

Tras ejecutar estas líneas, si editáramos el registro, podríamos comprobar que se han creado las siguientes entradas:



Función GetSetting

Devuelve el valor de una clave del registro.

Su sintaxis es

```
GetSetting NombreAplicación, sección, clave[,por defecto]
```

Los parámetros **NombreAplicación**, **sección** y **clave**, son los mismos que los explicados en la Instrucción **SaveSetting**.

El parámetro opcional por defecto, permite definir el valor que devolvería la función, si no se encontrara la clave en el registro. Si se omitiera equivaldría a poner una cadena de longitud cero.

Una vez grabados los datos en el punto anterior, al arrancar la aplicación podríamos ejecutar un procedimiento semejante al siguiente, para recuperar los valores del registro.

```
Public Sub CargarDatosDeEntrada()
    Dim datUltimaFecha As Date
    Dim strUltimaFecha As String
    Dim lngSocio As Long

  lngSocio = CLng(GetSetting("Gestión De Socios", _
                  "Valores finales", _
                  "Último Socio", _
                  "0"))
  datUltimaFecha = CDate(GetSetting("Gestión De Socios", _
                  "Valores finales", _
                  "Fecha Utilización", _
                  CStr(Date)))
    Debug.Print "Último socio grabado " _
                  & CStr(lngSocio)
    Debug.Print "Última fecha de utilización " _
                  & CStr(Date)
End Sub
```

Función GetAllSettings

Devuelve el valor de todas las claves de una sección del registro.

Su sintaxis es

```
GetSetting NombreAplicación, sección
```

Los parámetros **NombreAplicación**, **sección** y **clave**, son los mismos que los explicados en la Instrucción **SaveSetting**.

Las claves son guardadas en una variable de tipo **Array** de dos dimensiones basada en cero, declarada inicialmente como del tipo **Variant**.

Una vez cargados los datos en el array, para poder acceder a sus valores, podremos utilizar un bucle que vaya desde el índice cero, hasta el valor del índice mayor (función **UBound**).

Usando como valor el **0** en el segundo índice podremos obtener los nombres de las claves.

Si utilizamos como para el segundo índice el **1**, obtendremos los valores de las claves.

Veamos un ejemplo que aclarará este galimatías.

```
Public Sub MostrarDatosRegistro()

    Dim aDatos As Variant
    Dim i As Long
    aDatos = GetAllSettings("Gestión De Socios", _
                            "Valores finales")
    For i = 0 To UBound(aDatos, 1)
        Debug.Print "Clave " & CStr(i) _
                    & " - " _
                    & aDatos(i, 0) & ": " & aDatos(i, 1)
    Next i
End Sub
```

La línea cabecera del bucle, en vez de basarla directamente en el índice **0** podríamos usar la función **LBound** que nos devuelve el índice más bajo del **array**.

```
    For i = LBound(aDatos, 1) To UBound(aDatos, 1)
```

Tras ejecutarse este procedimiento se mostrará en la ventana inmediato, lo siguiente:

```
Clave 0 - Último Socio: 1011
Clave 1 -  Fecha Utilización: 03/09/2005
```

Instrucción DeleteSetting

Elimina una sección completa o una clave en el registro de configuración de Windows.

Su sintaxis es

```
DeleteSetting NombreAplicación, sección[, clave]
```

Los parámetros son los mismos que los de los procedimientos anteriores.

El nombre de la clave es opcional, si no se pusiera se borraría la sección completa.

Por ejemplo

```
DeleteSetting "Gestión De Socios", _
                    "Valores finales", _
```

```
                        "Fecha Utilización"
```

Borraría la clave "Fecha Utilización".

Si hubiéramos ejecutado

```
DeleteSetting "Gestión De Socios", _
                        "Valores finales"
```

Se borraría por completo la sección "Valores finales".

Grabar, leer y borrar cualquier Sección – Clave del registro

Como ya hemos indicado, los procedimientos anteriores son capaces de trabajar con secciones y claves ubicadas a partir de la rama del registro

HKEY_CURRENT_USER\Software\VB and VBA Program Settings

Para poder escribir en cualquier otra sección hay que utilizar APIS de Windows, como son:

RegOpenKeyEx	CreateRegistryKey
RegCloseKey	RegDeleteKey
RegCreateKeyEx	Etc...

Estos procedimientos los veremos más adelante en un capítulo específicamente dedicado al API de Windows.

Notas sobre este capítulo:

No quisiera desmoralizar a un posible lector, por el tipo de funciones que hemos analizado.

En principio, la utilización de las funciones API y las librerías de Acceso a Datos **DAO** y **ADO**, las veremos más adelante. Cuando abordemos esos temas la compresión de lo aquí expuesto en principio debería ser inmediata.

© Eduardo Olaz