UNIDAD 4: Archivos en Visual Basic

CONTENIDOS	
1. Conocer los tipos de archivos utilizados en Visual Basic;	1
2. Conocer las operaciones sobre archivos: apertura, lectura/escritura, cierre;	1
3. Variables definidas por el usuario para crear backup de tablas de una base de datos relacional. Recuperación de datos a través de archivos planos;	2
4. El tratamiento de Color e imágenes en Visual Basic. Representación hexadecimal de los colores, acceso a los colores del sistema, función RGB, paleta de colores, formatos gráficos, controles gráficos: Image, PictureBox, ImageList.	3

ACTIVIDADES	
Unidad 4	Tareas
1/2	Lectura, trabajos, prácticas propuestas en el Website de la Materia, Sección Actividades.

BIBLIOGRAFÍA para el Alumno

1. Contenidos Unidad 4
2. JALÓN, RODRÍGUEZ, BASÁLEZ, **Aprenda Visual Basic 6.0 como si estuviera en primero** (Universidad de Navarra) (*)
3. RAMÍREZ R., José Felipe, **aprenda VISUAL BASIC practicando** (Prentice-Hall, México, 1ª Edición, 2001) (**)

(*): Los trabajos se encuentran almacenados en el CD en formato PDF y en el Website de la materia.

(**): Libro impreso.

CONTENIDOS

1. Conocer los tipos de archivos utilizados en Visual Basic.

Hace muchos años atrás las Bases de Datos no eran tan populares porque utilizaban lo que se conoce como archivos ***secuenciales*** o también llamados archivos ***planos***. Este tipo de archivos están organizados de forma tal que la separación entre campos utiliza delimitadores (caracteres especiales) Están organizados de forma tal que la separación entre campos utiliza delimitadores (caracteres especiales) o bien considerando posiciones fijas, y en este caso podemos hablar de un registro.

Este tipo de archivos hoy en día se siguen utilizando debido a la transferencia de datos. En muchos sistemas relacionales de hoy en día se puede cargar una base datos o actualizar la misma haciendo uso de scripts que justamente se basan en el formato texto. Es una forma bastante veloz, que por ejemplo utiliza DB2 para realizar cargas y actualizaciones masivas.

Existen en Visual Basic 6.0 dos tipos de archivos:

- Ficheros ASCII o de texto: Están codificados en código ASCII y pueden leerse con cualquier editor de texto. Generalmente tienen extensión *.txt o *.bat. También ficheros con extensión *.c, *.cpp o *.java que son los fuentes de programas escritos en C, C++ o Java. En las bases de datos son utilizados, reiteramos para recuperar o realizar cargas masivas de datos.
- Ficheros binarios: son ficheros imagen de los datos o de programas tal como están en la memoria. No son legibles directamente. La ventaja es que no hay que pasarlos de código ASCII a lenguaje entendible por el ordenador y por ende se ahorra tiempo y espacio ya que ocupan menos lugar que los ASCII. Son comúnmente utilizadas las extensiones *.dat y *.bin.

Otro tema importante es el tipo de acceso a los archivos. Fundamentalmente existen tres formas:

- Acceso secuencial: se leen y escriben los datos en forma secuencial, un registro después de otro. Si se quiere acceder a un dato que está hacia la mitad de un fichero, habrá que leer primero los datos. Los ficheros de texto tienen acceso secuencial.
- Acceso aleatorio (random): posibilitan acceso directo a un dato sin tener que pasar por todos los anteriores y además el acceso es sin seguir un orden específico. La limitación está dada en que los datos están almacenados en unidades de información o bloques conocidos como registros. Esto hace a la organización del archivo y todos los registros deben tener el mismo tamaño. Estos ficheros son binarios.
- Acceso binario: similares a los anteriores pero el acceso no se ha por registros sino por bytes.

***Sólo abarcaremos* las operaciones de apertura, escritura/lectura y cierre sobre archivos planos o de texto** *debido a la utilidad que tiene para cargar y recuperar datos masivos de Bases de Datos. Los Archivos de Acceso Aleatorio excede el nivel del curso y por cierto, las Bases de Datos y las operaciones básicas sobre ellas que hemos visto en la Unidad 2 justamente un Sistema de Bases de Datos es de por sí aleatorio.*

2. Conocer las operaciones sobre archivos: apertura, lectura/escritura, cierre.

Para escribir o leer en un archivo primero hay que abrirlo y luego de utilizarlo hay que cerrarlo. Por ello veremos la sintaxis correspondiente de las operaciones fundamentales sobre archivos.

- **Open**

  Sintaxis: **Open Archivo For [Append|Input|Output] As #NumeroCanal**

  Archivo es el nombre del archivo y Numero de canal es el canal de comunicaciones o puerto asignado en forma exclusiva para el archivo, por lo tanto, cada archivo debe tener un número de canal distinto.

  Las ***opciones de modo de apertura***: Append, Input y Output nos permiten escribir al final del fichero (Append significa agregar), leer y escribir al comienzo de un fichero existente.

La lógica nos indica qué sucederá en estos casos:

- Si se intenta leer en una archivo que no existe (Input) tendremos un error como resultado, porque el archivo no existe;
- Si se intenta abrir un archivo que existe (tiene contenido) con Output se empezará a escribir al comienzo y se borrará el contenido anterior;
- Si se intenta abrir un archivo que no existe con Output, se creará el archivo sin contenido (longitud de 0 bytes);
- Si se intenta abrir un archivo que existe con Append se respetará el contenido anterior del archivo;
- Si se intenta abrir un archivo que no existe con Append se creará el archivo y como no tiene contenido (longitud de 0 bytes) comenzará a escribir desde el principio del archivo.

Como dos archivos no pueden tener el mismo puerto se puede utilizar la palabra reservada ***FreeFile*** para obtener el primer número de canal disponible del sistema.

En la sección ***Actividades*** en el Proyecto ***Archivos*** aparecen varios ejemplos de manipulación de archivos planos, a cuya teoría nos remitimos.

- **Close**

  Sintaxis: **Close #NumeroCanal**

- **Funciones Input y LineInput**
  - Sintaxis: **varString = Line Input #NumeroCanal**

    Esta instrucción lee una línea completa. La ventaja de esta función es que no incluye los caracteres que se usaron como separador de líneas. Si se quieren leer más de una línea hay que hacer un bucle o utilizar la siguiente función.

  - Sintaxis: **varString = Input(NumeroCaracteres, #NumeroCanal)**

    Esta instrucción lee del archivo (especificamos el puerto) la cantidad de caracteres especificados. Tiene como inconveniente que contiene todos los caracteres incluidos los Intro y retornos de carro si se han utilizado como separadores.

- **Funciones Print y Write**
  - Sintaxis: **Print #NumeroCanal, variable1, variable2, variable3, ...,variableN**

    Variable1, variable2, etc pueden ser variables, expresiones que dan un resultado numérico o alfanumérico, o cadenas de caracteres entre dobles comillas.

  - Sintaxis: **Write #NumeroCanal, campo1, campo2, campo3, ..., campoN**

    Los ficheros escritos con ***Write*** tienen la ventaja de que agregan el indicador de retorno de carro, cosa que no hace Print. A su vez son siempre visibles o leíbles con Input, cosa que no ocurres siempre con Print.

3. Variables definidas por el usuario.

Es recomendable utilizar tipos de datos definidos por el usuario. ***Un tipo de datos definido por el usuario es un conjunto de datos que se agrupan bajo un mismo nombre***; es decir, se trata de una estructura de datos, y actúan como una unidad.

¿Por qué? Simplemente porque nos interesa recuperar y guardar datos provenientes o dirigidos a Bases de Datos y las bases de datos relacionales. Se necesitan registros y un tipo de estructura

nos permite diseñar un registro básicamente. Es muy común que los tipos de datos definidos por el usuario se utilicen en archivos de acceso aleatorio por cuanto se puede localizar un registro en particular, sin embargo los utilizaremos en archivos planos, al menos por ahora.

Para definir un tipo de datos definido por el usuario se utiliza la instrucción ***Type*** que debe emplearse en el área de declaraciones generales. Se debe por lo tanto conocer la estructura de la base de datos, los tipos de datos con que se ha diseñado cada atributo de cada tabla y la longitud de ser necesario como en el caso de las Strings. A continuación mostramos un ejemplo de cómo se declara:

```
Private Type DatosCargos
IdCargo As Integer
NbreCargo As String * 12
MinSal As Single
MaxSal As Single
End Type
```

Se ha utilizado el tipo de datos Single, que permite decimales, para no utilizar Currency que ocupa más memoria porque los datos a representar son monetarios pero no representan grandes sumas de dinero ni exigen una gran precisión.

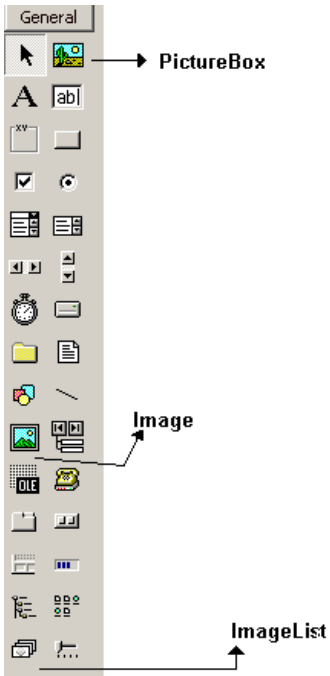
*En las prácticas utilizaremos las funciones **Input**, **Write**, por ser las más flexibles para el manejo de datos de Bases de Datos, juntamente con funciones definidas por el usuario.*

Igualmente aprenderemos a hacer más amigable a la aplicación utilizando controles CommonDialog lo que nos permitirá manejar los Cuadros de Diálogos Abrir Archivo y Guardar Como.

4. Tratamiento de Color e imágenes en Visual Basic. Representación hexadecimal de los colores, acceso a los colores del sistema, función RGB, paleta de colores, formatos gráficos, controles gráficos: Image, PictureBox, ImageList.

Para el tratamiento del color Visual Basic proporciona dos funciones: RGB y QBColor. ***RGB*** es el acrónimo de ***Red, Green, Blue (rojo, verde, azul)***. ***QBColor*** es una herencia de Quick Basic, la antigua plataforma de desarrollo Basic de Microsoft que se distribuía con MS Dos.

Con la función RGB, cuya sintaxis es: RGB(Red,Green,Blue), la gestión del color se realiza a través de la combinación de los colores primarios Rojo, Verde y Azul. Cada componente (rojo, verde o azul) es representado por un byte (8 dígitos == 0/255 (notación decimal) == 00/FF (notación hexadecimal). La variación y combinación de estos colores nos permite tener una paleta de colores que parte por lo tanto de la mezcla de los colores primarios y su intensidad.



Además, como ya se indicó, los colores se pueden representar en forma hexadecimal. Sin embargo, Visual Basic no guarda sólo 3 bytes sino 4 bytes (32 bits). El byte adicional es el que configura los colores del sistema.

Con la Función QBColor, que es más limitada que la anterior, ya que sólo posee 16 colores predefinidos (combinaciones básicas). Su sintaxis es QBColor(Color). El argumento Color puede variar entre 0 - 15.

En el proyecto Gráficos en el formulario Colores se presenta un ejemplo del uso de colores tanto con la función RGB como con la función QBColor.

Formatos Gráficos

Visual Basic permite insertar imágenes y gráficos en las aplicaciones.
Pueden ser de tipo Bitmap como de tipo vectoriales. Los tipos de formatos de ficheros gráficos que soporta son:

- de tipo bitmap:
  - bmp
  - ico
- de tipo vectorial:
  - wmf (Windows Meta File)
  - emf (Enhanced Meta File)
  - jpg (JPEG o Joint Photographic Experts Group)
  - gif (Graphic Interface Format)

CONTROLES GRÁFICOS

Los controles gráficos que veremos sí son los siguientes: Image, PictureBox e ImageList. En el proyecto ***Graficos*** hay ejemplos de uso de los siguientes controles.

Image

Permite mostrar imágenes de los formatos antes indicados. La propiedad esencial es ***Picture*** porque da la posibilidad de especificar el tipo de gráfico. Asimismo son también importantes ***BorderStyle,Stretch y ToolTipText***.

Hay dos formas de asignar un gráfico a la aplicación, una es a través de la ventana Propiedades, esto es

en ***tiempo de diseño*** es fácil pero no lo es tanto en ***tiempo de ejecución***. En este caso debemos utilizar la función ***LoadPicture***. La sintaxis de esta función es: LoadPicture("Archivo.ext") y puede incluirse o no la ruta de acceso del archivo.

En el proyecto ***Graficos*** en el formulario **Graficos1** se presenta un ejemplo de este control donde las imágenes se cargan con la función LoadPicture.

PictureBox

Es un control en apariencia muy parecido al anterior porque permite gestionar la misma variedad de archivos gráficos. Se diferencia básicamente en que puede actuar como objeto contenedor de objetos, como por ejemplo ***CommandButton y OptionButton***. Sin embargo la diferencia más notable está en que permite dibujar y por ello es un control muy útil para quienes desarrollan aplicaciones de dibujo.

La propiedad esencial de este control también es ***Picture***. También puede usarse la función LoadPicture.

A diferencia del control anterior, presenta métodos que sirven para dibujar: ***Circle, Cls, Line y Pset***

ImageList

En los casos anteriores con la función LoadPicture podíamos cargar un gráfico en tiempo de ejecución. Sin embargo estas son referencias explícitas y por ende debe existir el archivo gráfico o bien debe estar correctamente especificada la ruta de acceso al archivo, de otro modo tendremos por resultado un error.

Este control permite que las imágenes queden almacenadas en el formulario que las utilizará de modo que siempre estén disponibles. Este control pertenece al componente ***Microsoft Windows Common Controls 6.0*** que si no está agregado al proyecto habrá que hacerlo.

En resumidas palabras ImageList almacena imágenes en un tamaño estándar y las subordina a un módulo determinado. Visual Basic crea un archivo de extensión frx que forma parte del formulario y se distribuye con él.

En el proyecto ***Graficos*** en el formulario **Graficos3** se presenta un ejemplo completo de este control.

Fuentes Bibliográficas consultadas:

- BIRNIOS, BALTAZAR, BIRNIOS, MARIANO**, SMicorosft Visual Basic 6.0 Manual de Referencia** (MP Ediciones, Buenos Aires, 2000)
- RAMÍREZ R., José Felipe, **aprenda VISUAL BASIC practicando** (Prentice-Hall, México, 1ª Edición, 2001)