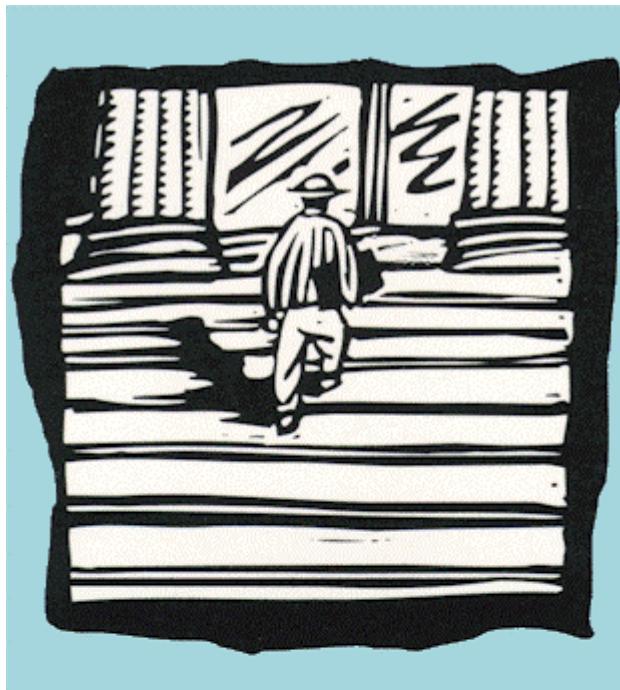


El texto enfoca el aprendizaje a la manera clásica de la asignatura de Fundamentos y Metodología de Programación, tal como se imparte en la enseñanza reglada. La única diferencia con este enfoque es que se sustituyen lenguajes como Pascal o C, con los que habitualmente se imparte la asignatura, por Visual Basic, por un entorno de desarrollo mucho más actualizado y que servirá a quienes lean este manual en mucha mayor medida que esos otros lenguajes pueden hacerlo.

Este es un texto ideal para los no programadores que deseen adentrarse en el mundo de la programación.



Fundamentos

FUNDAMENTOS DE PROGRAMACIÓN CON VISUAL BASIC 6

LUIS MIGUEL BLANCO



ADVERTENCIA LEGAL

Todos los derechos de esta obra están reservados a Grupo EIDOS Consultoría y Documentación Informática, S.L.

El editor prohíbe cualquier tipo de fijación, reproducción, transformación, distribución, ya sea mediante venta y/o alquiler y/o préstamo y/o cualquier otra forma de cesión de uso, y/o comunicación pública de la misma, total o parcialmente, por cualquier sistema o en cualquier soporte, ya sea por fotocopia, medio mecánico o electrónico, incluido el tratamiento informático de la misma, en cualquier lugar del universo.

El almacenamiento o archivo de esta obra en un ordenador diferente al inicial está expresamente prohibido, así como cualquier otra forma de descarga (downloading), transmisión o puesta a disposición (aún en sistema streaming).

La vulneración de cualesquiera de estos derechos podrá ser considerada como una actividad penal tipificada en los artículos 270 y siguientes del Código Penal.

La protección de esta obra se extiende al universo, de acuerdo con las leyes y convenios internacionales.

Esta obra está destinada exclusivamente para el uso particular del usuario, quedando expresamente prohibido su uso profesional en empresas, centros docentes o cualquier otro, incluyendo a sus empleados de cualquier tipo, colaboradores y/o alumnos.

Si Vd. desea autorización para el uso profesional, puede obtenerla enviando un e-mail fmarin@eidos.es o al fax (34)-91-5017824.

Si piensa o tiene alguna duda sobre la legalidad de la autorización de la obra, o que la misma ha llegado hasta Vd. vulnerando lo anterior, le agradeceremos que nos lo comunique al e-mail fmarin@eidos.es o al fax (34)-91-5012824). Esta comunicación será absolutamente confidencial.

Colabore contra el fraude. Si usted piensa que esta obra le ha sido de utilidad, pero no se han abonado los derechos correspondientes, no podremos hacer más obras como ésta.

© Luis Miguel Blanco, 2000

© Grupo EIDOS Consultoría y Documentación Informática, S.L., 2000

ISBN 84-88457-07-3

Fundamentos de programación con Visual Basic 6

Luis Miguel Blanco

Responsable editorial

Paco Marín (fmarin@eidos.es)

Coordinación de la edición

Antonio Quirós (aquiros@eidos.es)

Autoedición

Magdalena Marín (mmarin@eidos.es)
Luis Miguel Blanco (lmblanco@eidos.es)

Grupo EIDOS

C/ Téllez 30 Oficina 2
28007-Madrid (España)
Tel: 91 5013234 Fax: 91 (34) 5017824
www.grupoeidos.com/www.eidos.es
www.LaLibreriaDigital.com



Índice

ÍNDICE.....	5
INTRODUCCIÓN.....	12
LENGUAJES DE PROGRAMACIÓN Y PROGRAMAS.....	12
DIFERENCIAS ENTRE LENGUAJES.....	13
NIVELES DE LENGUAJES.....	13
CREACIÓN DE EJECUTABLES.....	14
DESARROLLO DE PROGRAMAS.....	15
ALGORITMOS	17
ESTUDIO DEL PROBLEMA	17
ALGORITMOS	18
DIAGRAMAS DE FLUJO	18
PSEUDOCÓDIGO	24
DIAGRAMAS NASSI-SHNEIDERMAN.....	26
PROGRAMAS	29
¿QUÉ ES UN PROGRAMA?	29
FASES DE EJECUCIÓN	29
COMPONENTES DE UN PROGRAMA.....	30
PROGRAMAS EN VISUAL BASIC	31
ELEMENTOS DEL LENGUAJE	37
INTRODUCCIÓN	37
DATOS.....	37
<i>Simples</i>	37

Numérico	38
Carácter.....	38
Lógico	38
Compuestos.....	39
Definidos por el usuario	39
IDENTIFICADORES	39
PALABRAS RESERVADAS.....	39
ORGANIZACIÓN DEL CÓDIGO	39
EL MÓDULO DE CÓDIGO	43
COMENTARIOS	45
VARIABLES	45
Declaración	45
Denominación.....	46
Tipificación.....	46
Asignación	49
Valor inicial.....	51
Declaración obligatoria	51
CONTINUACIÓN DE LÍNEA DE CÓDIGO	54
GRABACIÓN DEL PROYECTO	54
SEGUIMIENTO DE LA EJECUCIÓN.....	55
CONSTANTES	57
ENTRADAS Y SALIDAS DE INFORMACIÓN.....	58
MSGBOX()	58
INPUTBOX()	61
OPERADORES	62
Aritméticos.....	62
^ Potencia.....	62
* Multiplicación.....	62
/ División real	63
\ División entera.....	63
Mod.....	64
+ Suma.....	64
Resta	65
Comparación	65
Is	67
Like	68
Concatenación	69
Lógicos.....	69
And	69
Eqv.....	71
Imp.....	71
Not	73
Or.....	74
Xor.....	75
PRIORIDAD DE OPERADORES	76
MODELOS DE PROGRAMACIÓN	77
Programación lineal.....	77
Programación estructurada.....	77
Programación basada en eventos.....	78
PROCEDIMIENTOS	79
Sub.....	79
Function.....	82
Paso de parámetros por valor y por referencia.....	84
ByVal (Por valor)	84
ByRef (Por referencia).....	85

<i>Static</i>	86
<i>Asistente para la declaración de procedimientos</i>	87
ÁMBITO DE ELEMENTOS DEL LENGUAJE	88
<i>Procedimientos</i>	88
<i>Private</i>	88
<i>Public</i>	89
<i>Variables</i>	89
<i>Dim</i>	90
<i>Static</i>	90
<i>Private</i>	90
<i>Public</i>	92
<i>Constantes</i>	93
CONVENCIONES DE NOTACIÓN	93
<i>Variables</i>	93
<i>Constantes</i>	95
INDENTACIÓN DEL CÓDIGO	95
ESTRUCTURAS DE CONTROL	96
<i>Selección</i>	96
<i>If...End If</i>	97
<i>Select Case...End Select</i>	99
<i>Repetición</i>	100
<i>Do...Loop</i>	100
<i>While...Wend</i>	102
<i>For...Next</i>	102
<i>GoTo</i>	104
ANIDACIÓN DE ESTRUCTURAS	105
ARRAYS	106
<i>Declaración</i>	107
<i>Asignación</i>	107
<i>Establecer los límites en los índices</i>	108
<i>Option Base</i>	108
<i>To</i>	108
<i>Recorrer los elementos de un array</i>	109
<i>Arrays multidimensionales</i>	110
<i>Arrays dinámicos</i>	112
<i>Array()</i>	113
TIPOS DEFINIDOS POR EL USUARIO	113
EL TIPO ENUMERADO	115
COMPROBACIÓN DE TIPOS	117
FUNCIONES PROPIAS DEL LENGUAJE	119
<i>Numéricas</i>	119
<i>Cadenas de caracteres</i>	121
<i>Fecha y hora</i>	134
TÉCNICAS	141
RECURSIVIDAD	141
BÚSQUEDA DE DATOS	142
<i>Lineal</i>	142
<i>Binaria</i>	143
ORDENACIÓN DE DATOS	144
MANEJO DE ERRORES	144
<i>Err</i>	145
<i>On Error</i>	146
<i>On Error GoTo Etiqueta</i>	146
<i>On Error Resume Next</i>	147

On Error GoTo 0	148
PROGRAMACIÓN ORIENTADA A OBJETO (OOP)	149
¿QUÉ VENTAJAS APORTA LA PROGRAMACIÓN ORIENTADA A OBJETOS?	149
OBJETOS	150
CLASES	150
CLASES ABSTRACTAS	151
RELACIONES ENTRE OBJETOS	152
<i>Herencia</i>	152
<i>Pertenencia</i>	152
<i>Utilización</i>	153
ELEMENTOS BÁSICOS DE UN SISTEMA OOP	153
<i>Encapsulación</i>	153
<i>Polimorfismo</i>	153
<i>Herencia</i>	153
REUTILIZACIÓN	154
CREACIÓN DE CLASES	154
<i>Crear un módulo de clase</i>	154
<i>Definir las propiedades de la clase</i>	155
<i>¿Qué son los procedimientos Property?</i>	155
<i>Tipos de procedimientos Property</i>	155
<i>Ventajas de los procedimientos Property</i>	157
<i>Establecer una propiedad como predeterminada</i>	159
<i>Crear los métodos de la clase</i>	161
<i>¿Cuándo crear un procedimiento Property o un método?</i>	161
INSTANCIAR OBJETOS DE UNA CLASE	162
<i>Variables de objeto y tipos de enlace</i>	162
<i>Instanciar un objeto</i>	163
<i>Manipulación de propiedades y métodos</i>	164
<i>Eliminar el objeto</i>	165
LA PALABRA CLAVE ME	165
EMPLEAR VALORES CONSTANTES EN UNA CLASE	167
EVENTOS PREDEFINIDOS PARA UNA CLASE	169
UTILIDADES Y HERRAMIENTAS PARA EL MANEJO DE CLASES	170
<i>Generador de clases (Class Wizard)</i>	170
<i>Examinador de objetos (Object Browser)</i>	174
OBJETOS DEL SISTEMA	175
DESARROLLO DE UN PROGRAMA	177
CREAR UN NUEVO PROYECTO	177
CONFIGURAR EL ENTORNO AL INICIO	178
ACOPLE DE VENTANAS	179
DISEÑO DEL FORMULARIO	180
LA VENTANA DE PROPIEDADES	181
PROPIEDADES DEL FORMULARIO	182
VENTANA DE POSICIÓN DEL FORMULARIO	183
ASIGNAR UN NOMBRE AL PROYECTO	184
CONTROLES	184
LABEL	185
<i>Propiedades</i>	186
<i>Propiedades</i>	186
UBICACIÓN PRECISA DE CONTROLES	188
OPERACIONES CON GRUPOS DE CONTROLES	189
COPIAR Y PEGAR CONTROLES	190
CONCEPTOS DE FOCO DE LA APLICACIÓN Y CAMBIO DE FOCO	191

COMMANDBUTTON.....	191
<i>Propiedades</i>	191
CHECKBOX	192
<i>Propiedades</i>	192
LA VENTANA DE CÓDIGO	194
<i>Configuración del editor de código</i>	194
<i>Modos de visualización del código</i>	196
<i>División del área de edición</i>	196
<i>Búsquedas de código</i>	197
<i>Asistentes durante la escritura</i>	198
<i>Marcadores</i>	199
BARRAS DE HERRAMIENTAS	199
CODIFICACIÓN DE EVENTOS	202
GENERAR EL EJECUTABLE DEL PROYECTO	203
INICIAR EL FORMULARIO DESDE CÓDIGO.....	205
<i>Variables globales de formulario</i>	205
<i>Variables declaradas de formulario</i>	206
ASISTENTE PARA CREAR PROCEDIMIENTOS	207
IMAGE	207
ARRAYS DE CONTROLES	209
CREACIÓN DE CONTROLES EN TIEMPO DE EJECUCIÓN	211
PROPIEDADES ACCESIBLES EN TIEMPO DE DISEÑO Y EN TIEMPO DE EJECUCIÓN	213
EL DEPURADOR	215
<i>¿QUÉ ES UN DEPURADOR?</i>	215
ACCIONES BÁSICAS.....	216
MODO DE INTERRUPCIÓN.....	216
PUNTOS DE INTERRUPCIÓN	217
EJECUCIÓN PASO A PASO.....	218
INSPECCIONES	219
<i>La ventana de Inspección</i>	220
<i>Inspección rápida</i>	221
LA VENTANA INMEDIATO	221
LA VENTANA LOCALES	222
PILA DE LLAMADAS	223
ESTILOS DE FORMULARIOS	227
APLICACIONES SDI Y MDI	227
FORMULARIOS MODALES Y NO MODALES.....	230
<i>Modal</i>	230
<i>Modal a la aplicación</i>	230
<i>Modal al sistema</i>	232
<i>No Modal</i>	232
OTROS CONTROLES BÁSICOS.....	233
OPTIONBUTTON	233
<i>Propiedades</i>	234
FRAME.....	235
LISTBOX	237
<i>Propiedades</i>	237
COMBOBOX	239
<i>Propiedades</i>	239
TIMER.....	240
<i>Propiedades</i>	241
DRIVELISTBOX, DIRLISTBOX Y FILELISTBOX	242

<i>DriveListBox</i>	242
Propiedades.....	242
<i>DirListBox</i>	242
Propiedades.....	242
<i>FileListBox</i>	243
Propiedades.....	243
VALIDACIÓN DE CONTROLES	244
DISEÑO DE MENÚS.....	247
DESCRIPCIÓN DE UN MENÚ	247
CARACTERÍSTICAS DE UN MENÚ	247
PROPIEDADES DE UN CONTROL MENÚ.....	248
EL EDITOR DE MENÚS	248
AÑADIR CÓDIGO A UNA OPCIÓN DE MENÚ	250
CREACIÓN DE UN MENÚ	250
MENÚS EN FORMULARIOS MDI	255
MANIPULACIÓN DE FICHEROS.....	259
¿POR QUÉ USAR FICHEROS?	259
APERTURA DE UN FICHERO	260
CIERRE DE UN FICHERO.....	261
MODOS DE ACCESO A FICHEROS	261
ACCESO SECUENCIAL	262
<i>Apertura</i>	262
<i>Lectura</i>	262
<i>Escritura</i>	264
ACCESO ALEATORIO.....	267
<i>Apertura</i>	267
<i>Escritura</i>	267
<i>Lectura</i>	268
ACCESO BINARIO	269
<i>Apertura</i>	270
<i>Escritura</i>	270
<i>Lectura</i>	270
OTRAS INSTRUCCIONES PARA MANEJO DE FICHEROS	271
OBJETOS PARA EL MANEJO DE FICHEROS.....	275
<i>Drive</i>	275
Propiedades.....	275
<i>Folder</i>	277
Propiedades.....	277
Métodos	278
<i>File</i>	280
<i>FileSystemObject</i>	282
Propiedades.....	282
Métodos	282
<i>TextStream</i>	288
Propiedades.....	288
Métodos	288
MEJORAR LA FUNCIONALIDAD DEL INTERFAZ VISUAL DEL PROGRAMA	291
LA IMPORTANCIA DE DISEÑAR UN INTERFAZ ATRACTIVO	291
COMMONDIALOG.....	291
<i>Color</i>	293
<i>Tipo de letra</i>	294
<i>Impresión</i>	296

<i>Ayuda</i>	298
<i>Apertura de ficheros</i>	300
<i>Grabación de ficheros</i>	303
AGREGAR CONTROLES COMPLEMENTARIOS.....	303
IMAGELIST.....	305
TOOLBAR	306
<i>Propiedades</i>	306
STATUSBAR	311
CREAR UNA VENTANA SPLASH O DE INICIO	315
<i>Creación</i>	315
<i>Configuración de propiedades</i>	317
<i>Escritura de código y visualización</i>	317
TRATAMIENTO DE DATOS CON ACTIVEX DATA OBJECTS (ADO).....	321
INTRODUCCIÓN A LA PROGRAMACIÓN CON BASES DE DATOS	321
DISEÑO DE LA BASE DE DATOS	322
<i>Establecer el modelo de la aplicación</i>	322
<i>Analizar que información necesita la aplicación</i>	322
<i>Establecer la información en tablas</i>	322
<i>Normalización de la base de datos</i>	323
<i>Relacionar las tablas</i>	324
<i>Establecer la integridad referencial</i>	326
<i>Definir índices para las tablas</i>	326
<i>Definir validaciones en la inserción de datos</i>	326
ACCESO A INFORMACIÓN BAJO CUALQUIER FORMATO	327
ACCESO UNIVERSAL A DATOS (UDA)	327
OLE DB	328
ACTIVEX DATA OBJECTS (ADO)	329
<i>Connection</i>	331
<i>Propiedades</i>	331
<i>Métodos</i>	334
<i>Command</i>	339
<i>Propiedades</i>	339
<i>Métodos</i>	340
<i>Parameter</i>	341
<i>Propiedades</i>	341
<i>Métodos</i>	344
<i>Recordset</i>	345
<i>Propiedades</i>	345
<i>Métodos</i>	350
<i>Field</i>	356
<i>Propiedades</i>	356
<i>Métodos</i>	358
<i>Property</i>	358
<i>Propiedades</i>	358
<i>Error</i>	359
<i>Propiedades</i>	359
EL ADO DATA CONTROL	359
VALIDACIONES A NIVEL DE MOTOR	369
EL CONTROL DATAGRID.....	370
ASISTENTE PARA CREAR APLICACIONES.....	374
GLOSARIO DE TÉRMINOS	385
<i>NORMATIVA DE CODIFICACIÓN</i>	385

1

Introducción

En las referencias a la sintaxis de los elementos del lenguaje se emplean las siguientes convenciones en el texto:

- El texto contenido entre corchetes -[]-, indican que se trata de un elemento del lenguaje opcional, no es obligatorio su uso.
- Cuando encontramos diferentes elementos del lenguaje separados por una barra -|-, se indica que se puede utilizar sólo uno de dichos elementos.

Lenguajes de programación y programas

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes.

Un programa por lo tanto, es un conjunto de instrucciones que siguen una serie de reglas dictadas por el lenguaje de programación en el que se haya escrito. A ese grupo de instrucciones escritas por el programador para obtener el programa, se le denomina código fuente, y tras una fase de compilación, se convierten en el lenguaje máquina que ejecutará el ordenador en el que se haya instalado dicho programa.

El lenguaje máquina consiste en el código en formato binario de las instrucciones escritas en el lenguaje de programación, y es el único tipo de código a ejecutar que entiende el ordenador. En ese caso ¿no sería mejor dedicarnos a programar directamente en lenguaje máquina?.

La respuesta a esta pregunta se encuentra en que el desarrollo de un programa directamente en lenguaje máquina es excesivamente complejo, ya que dicho código se basa en las instrucciones en formato binario que son entendibles directamente por el ordenador. Debido a esto, la fase de creación del programa se realiza en un lenguaje que sea más comprensible por el programador, y una vez finalizada esta fase, se convierte a código máquina en la llamada fase de compilación.



Figura 1. Pasos para la conversión a código binario.

Diferencias entre lenguajes

Una vez que hemos aclarado el motivo de usar un lenguaje de programación, cabría hacernos una nueva pregunta, ya que debemos usar un lenguaje comprensible para desarrollar el programa ¿no sería más fácil que existiera un único lenguaje con el que desarrollar todos los programas?. ¿Por qué existe tal profusión de lenguajes de programación?

En este caso podríamos establecer una semejanza entre varios oficios que utilicen herramientas manuales. Cada oficio tiene una tarea concreta que resolver, y para ello el trabajador emplea un conjunto de herramientas especializadas que le ayudan en dicha labor.

Pues algo parecido podríamos aplicar en el caso informático. No es lo mismo crear un programa para realizar la facturación de una empresa, que tener que realizar una serie de operaciones que afecten directamente a los componentes físicos del equipo, o realizar cálculos científicos avanzados. Por este motivo, para cada uno de estos problemas encontraremos un lenguaje que esté especializado en alguna de estas áreas: gestión, matemático, hardware, etc.; y ese será el lenguaje que debamos emplear, ya que estará dotado de un conjunto de utilidades que ayuden al programador en ese aspecto en concreto que deba resolver.

Niveles de lenguajes

Dentro del tipo de problema a resolver, los lenguajes se pueden clasificar en una serie de niveles:

- Bajo nivel. En este tipo de lenguajes no existen estructuras de control complejas. El acceso a los dispositivos físicos del ordenador está muy optimizado y el código compilado que se genera es el que está más próximo al lenguaje máquina, ya que por cada instrucción en el lenguaje se genera una instrucción en lenguaje máquina. Por otra parte, al ser un lenguaje muy próximo al hardware de la máquina es más difícil de programar. El ejemplo más claro de este lenguaje es el Ensamblador.
- Medio nivel. Aquí ya se dispone de estructuras de control complejas y tipos de datos. También existe una alta capacidad de acceso a los dispositivos hardware, sin embargo, su complejidad queda en un punto intermedio entre un lenguaje de bajo nivel y uno de alto: no es tan complicado como el primero, pero exige una mayor curva de aprendizaje que el segundo. El lenguaje "C" es el ejemplo de este tipo.

- Alto nivel. Son los que suelen aportar un mayor número de estructuras de control y tipos de datos. Igualmente dispone de una sintaxis en lenguaje más natural y un amplio conjunto de funciones internas que ayudan al programador en diversas situaciones, así como un número determinado de utilidades y asistentes que ahorran tiempo y trabajo al programador. Visual Basic es uno de los ejemplos de este tipo de lenguaje.

Creación de ejecutables

Un programa finalizado y listo para utilizar, es básicamente un fichero con extensión EXE o ejecutable. Dependiendo del sistema operativo en el que ese programa deba funcionar y su finalidad, estará acompañado por una serie de ficheros adicionales como puedan ser ficheros de librerías, bases de datos, etc.

En teoría, un fichero EXE es autosuficiente, es decir, que no necesita de ningún otro fichero para ejecutarse, pero debido a lo comentado sobre ficheros adicionales, la falta de cierta librería, o la base de datos sobre la que se realiza un mantenimiento de información, puede hacer que nuestro programa no funcione al detectar la ausencia de alguno de estos elementos. Debido a esto, el concepto de programa, no puede circunscribirse en la mayoría de las ocasiones únicamente al ejecutable, entendiendo como programa al compendio formado por el ejecutable, librerías de código, bases de datos, controles que se instalan junto al programa, etc.

A pesar de todo, el punto de apoyo principal de un programa es el fichero EXE, que es creado por el programador y forma el elemento de unión entre el resto de componentes de un programa. A grandes rasgos, las fases de creación de un ejecutable son las siguientes, una vez que el programador ha finalizado la escritura del código fuente del programa, las vemos también en la Figura 2

- Compilación. Consiste en convertir el código escrito por el programador en el llamado código objeto, previo a la fase de obtención del ejecutable. Dependiendo del lenguaje utilizado, el código objeto se almacenará en forma de fichero o será pasado directamente a la siguiente fase de creación del ejecutable. Para transformar el código fuente a código objeto se emplea el compilador, que es un programa o utilidad encargada de esta tarea.
- Enlace. En esta fase se toman el código objeto generado por el compilador y mediante un programa o utilidad denominado enlazador, se une para obtener el ejecutable final (fichero EXE).

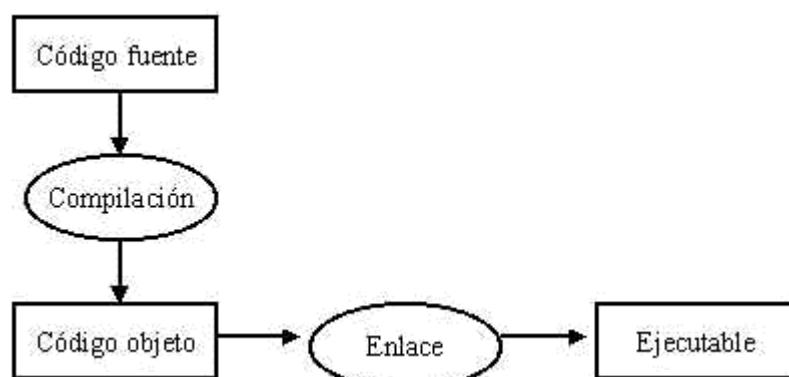


Figura 2. Fases de generación de ejecutables.

En antiguos lenguajes de programación, las labores antes descritas, debían ser realizadas por el programador de forma manual, ya que todos los elementos utilizados para la creación del programa, se encontraban separados. Disponíamos por un lado de un programa que consistía en un editor de código fuente, por otro lado el compilador, enlazador, etc. Sin embargo, con el cada vez mayor auge de los entornos de desarrollo integrados, que incorporan en un sólo programa todos los elementos necesarios para el desarrollo de programas. Estas tareas se han automatizado hasta tal nivel, que el programador sólo necesita escribir el código del programa, mientras que los pasos de compilación, enlazado y generación del ejecutable las realiza internamente el propio entorno.

Desarrollo de programas

No existen una serie de normas absolutas a la hora de escribir un programa, ya que se trata de un proceso creativo en el que se mezclan por un lado los requerimientos del problema a afrontar, las capacidades que nos proporciona el lenguaje empleado por otro y la pericia e inventiva del programador para resolver dicho problema.

Sin embargo, sí podemos establecer unas pautas generales a seguir, en lo que se puede llamar ciclo de desarrollo del programa, y que podemos dividir en dos grandes etapas:

- Analítica del problema. En esta etapa debemos encargarnos de estudiar en qué consiste el problema y desarrollar los procesos para resolverlo. Se subdivide en tres pasos:
 - Estudio del problema.
 - Desarrollo del algoritmo.
 - Comprobación del algoritmo.
- Implementación del programa. En esta etapa se procede a aplicar en el ordenador el resultado de la fase anterior. Los pasos de esta etapa son los siguientes:
 - Escritura del programa basándose en los resultados del algoritmo
 - Ejecución del programa.
 - Depuración del programa.
 - Documentación del programa.



Algoritmos

Estudio del problema

Antes de proceder a la creación del algoritmo de un proceso o programa, debemos estudiar cuidadosamente el problema planteado e identificar la información de que disponemos para resolverlo o información de entrada, y el resultado a obtener o información de salida. Pongamos el siguiente ejemplo:

"Una empresa de desarrollo de software recibe el encargo de realizar un programa. Dicha empresa cobra la hora de programación a 3.600 ptas., empleando 45 horas en desarrollar el programa. Al importe total de horas trabajadas se aplicará un IVA del 16%. Averiguar el importe total a pagar antes y después de impuestos y la cantidad de impuesto aplicado".

La información reconocida tras este enunciado sería la siguiente:

Entrada:

- Importe/hora: 3.600 ptas.
- Horas trabajadas: 45.
- Impuesto a aplicar: 16%.

Salida:

- Total antes de impuestos: 162.000 ptas.

- Impuestos: 25.920 ptas.
- Total después de impuestos: 187.920 ptas.

Es muy importante disponer de la suficiente información para resolver el problema, ya que sin ella, no podremos obtener los datos que supondrán el resultado del planteamiento. Supongamos este otro problema:

"Una oficina compra cartuchos para sus impresoras. Obtener el importe total a pagar y el porcentaje de IVA correspondiente".

En este caso no podemos resolver el problema, ya que tras analizarlo, falta la información de entrada como el número de cartuchos comprados, importe por cartucho y el IVA a aplicar, por lo que resulta imposible obtener los datos de salida.

Algoritmos

Un algoritmo se puede definir como el conjunto de acciones a realizar para resolver un determinado problema.

El modo de afrontar la creación de un algoritmo, pasa por descomponer el problema planteado en problemas más pequeños y fáciles de resolver independientemente. Una vez resueltos los subproblemas por separado, se unirán obteniendo de esta forma el correspondiente algoritmo.

El proceso indicado por un algoritmo debe ser claro y tener sus pasos bien definidos, de forma que si realizamos dicho proceso varias veces, empleando siempre los mismos valores en el algoritmo, deberemos obtener el mismo resultado.

Cuando finalicemos la escritura de un algoritmo, es muy conveniente realizar una ejecución de prueba para el mismo, empleando datos reales para comprobar que el resultado es el adecuado. En el caso de que obtengamos resultados no esperados, o bien, consideremos que es posible optimizar el proceso de ejecución del algoritmo, modificaremos las partes que consideremos necesarias para mejorarlo; este proceso se denomina depuración.

No existe una técnica única para la escritura de algoritmos, si bien disponemos de algunas que dadas sus características, nos facilitan dicha tarea, por lo que son mayormente utilizadas. Entre los sistemas de creación de algoritmos, tenemos los diagramas de flujo y el pseudocódigo.

Diagramas de flujo

Un diagrama de flujo consiste en una representación gráfica basándose en símbolos de los pasos que debe realizar un algoritmo. Estos símbolos pueden clasificarse de mayor a menor importancia, en:

- Básicos.
 - Terminador. Indica el principio o fin de un algoritmo, o bien una pausa. Figura 3



Figura 3. Terminador

- Datos. Contiene información de entrada o salida que será utilizada por el algoritmo para obtener un resultado. Figura 4

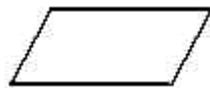


Figura 4. Datos.

- Proceso. Indica una o más operaciones a realizar durante la ejecución del algoritmo. Figura 5.

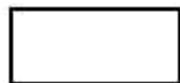


Figura 5. Proceso.

- Decisión. Contiene una operación que da como resultado un valor lógico, en función de la cual, el flujo del algoritmo se bifurcará en una determinada dirección. Figura 6.

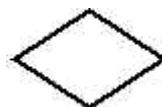


Figura 6. Decisión.

- Dirección. Indica el camino seguido por el flujo del algoritmo. También suelen utilizarse líneas simples para conectar símbolos en el diagrama. Figura 7.



Figura 7. Dirección.

- Principales.

- Decisión múltiple. Variante del símbolo de decisión, en la que el resultado de la operación que contiene puede ser uno de varios posibles, en lugar del simple verdadero o falso de la decisión sencilla. Figura 8.

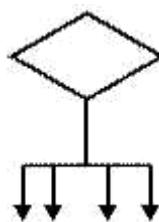


Figura 8. Decisión múltiple.

- Conectores. Unen dos puntos de un diagrama. El círculo indica una conexión dentro de la misma página, y el conector de dirección entre páginas diferentes del diagrama. Figura 9.

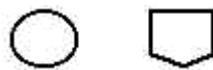


Figura 9. Conectores.

- Rutina. Indica una llamada a un procedimiento externo al algoritmo actual. Una vez procesado dicho procedimiento, el flujo del proceso volverá a este punto y continuará con el siguiente paso del algoritmo. Figura 10.



Figura 10. Rutina.

- Complementarios.
- Teclado. Indica una acción de entrada de datos en el algoritmo. Figura 11.

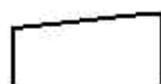


Figura 11. Teclado.

- Pantalla. Indica una acción de salida de datos en el algoritmo. Figura 12.

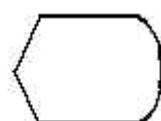


Figura 12. Pantalla.

- Impresora. Indica una acción de salida de datos en el algoritmo. Figura 13.



Figura 13. Impresora.

En el proceso de diseño de un diagrama de flujo, se indicarán las operaciones mediante el símbolo correspondiente, introduciendo dentro del símbolo si es necesario, o este lo requiere, una nota con la operación que va a realizarse.

Un algoritmo no sirve única y exclusivamente para ser aplicado en la resolución de problemas informáticos. Es posible emplear algoritmos para resolver variados problemas, incluso tareas simples.

Por este motivo, en la Figura 14, vamos a desarrollar un ejemplo de algoritmo resuelto mediante un diagrama de flujo, aplicándolo a un problema tan natural como abrir una puerta con una llave.

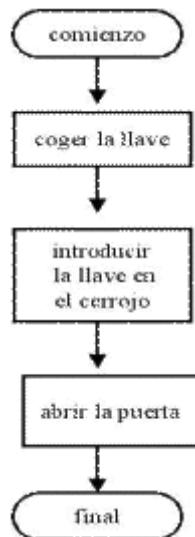


Figura 14. Diagrama para abrir una puerta.

Anteriormente hemos mencionado la depuración como la técnica que nos permite corregir o mejorar el proceso desarrollado por un algoritmo o programa; pues bien, dicho sistema puede ser aplicado a este algoritmo que acabamos de representar en forma de optimización. El anterior ejemplo sólo contempla la posibilidad de una llave para abrir la puerta, pero normalmente suele suceder que tengamos varias llaves entre las que tenemos que elegir la correcta. Teniendo este aspecto en cuenta, depuremos el anterior algoritmo tal y como se muestra en la Figura 15, de manera que nos sirva para introducir de paso un elemento de decisión.

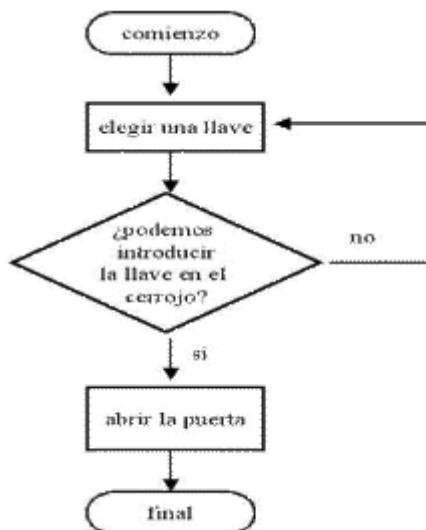


Figura 15. Diagrama para abrir una puerta con depuración

- Operadores, identificadores.

Dentro de los elementos de un diagrama de flujo es frecuente la necesidad de realizar operaciones para obtener diversos resultados. En función de la acción o partícula de la acción a representar, emplearemos un conjunto de caracteres en mayor o menor medida generalizados, de los cuales pasamos a describir los más comunes.

Cuando necesitemos indicar una operación aritmética, emplearemos los siguientes caracteres:

- + Suma.
- Resta.
- * Multiplicación.
- / División.
- \ División entera.
- ^ Potenciación.

Para establecer una comparación utilizaremos estos símbolos:

- > Mayor que.
- >= Mayor o igual que.
- < Menor que.
- <= Menor o igual que.
- = Igual que.

Al realizar una operación, el resultado necesitaremos guardarlo en algún sitio. Para ello disponemos de los llamados identificadores, que son elementos que representan un valor que utilizaremos a lo largo del desarrollo del algoritmo, y que pueden variar dicho valor según las operaciones que realicemos con ellos. A un identificador le daremos un nombre cualquiera, aunque es recomendable que guarde cierta relación con el valor que contiene. En el próximo ejemplo se utilizarán identificadores, de forma que el lector pueda comprobar como se lleva a cabo su uso.

El símbolo igual se utiliza también para asignar el resultado de una operación a un identificador, situando el identificador al que vamos a asignar el valor a la izquierda del símbolo y la operación a realizar en la derecha.

En el caso de necesitar representar una entrada o salida de datos, podemos usar las palabras Lectura y Escritura, seguidas de los identificadores a los que se asignará un valor o de los que se mostrará su valor.

En la Figura 16 crearemos un algoritmo mediante un diagrama de flujo basado en el siguiente planteamiento: calcular el importe de una factura de una empresa de consumibles que vende cartuchos de impresora. Cada cartucho tiene un precio de 2.100 Ptas., y el usuario debe de introducir la cantidad de cartuchos vendidos y calcular el total, mostrando finalmente dicho total. Para simplificar el algoritmo, no se tendrá en cuenta el importe de IVA en este caso. Este algoritmo nos servirá además para comprobar la representación de las entradas y salidas de información en un diagrama.

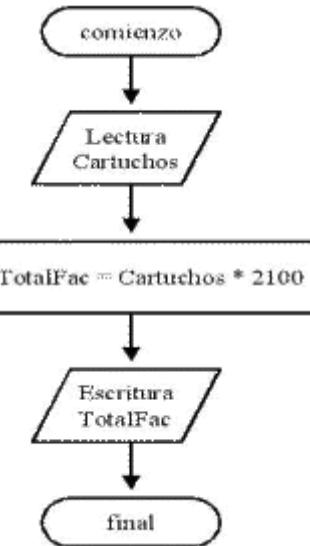


Figura 16. Algoritmo para calcular importe de cartuchos de impresora.

Suponiendo que el diagrama de la Figura 16 se basa en una primera versión del algoritmo, la Figura 17 muestra el proceso de depuración y optimización aplicado a dicho algoritmo para realizar el cálculo de un IVA del 16% sobre el importe de los cartuchos facturados y como afecta al total de la factura.

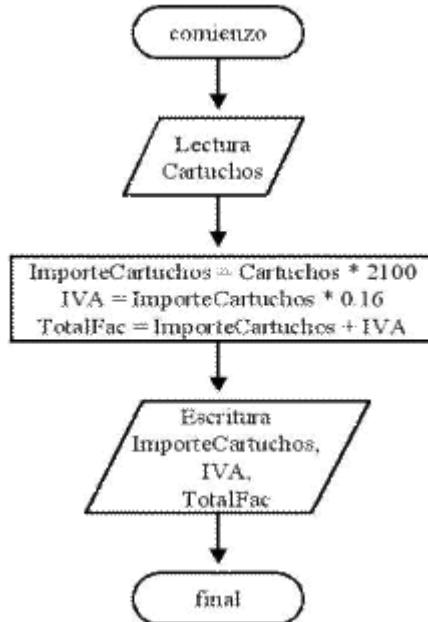


Figura 17. Algoritmo depurado del cálculo de cartuchos de impresora.

Los algoritmos representados hasta el momento se basan en un proceso simple de toma de datos, cálculo de operaciones y muestra de resultados. Pero una de las ventajas de la programación consiste en la posibilidad de repetir un número determinado de veces un mismo proceso en una estructura iterativa o bucle.

Otra particularidad relacionada con las estructuras repetitivas, es que normalmente se necesita acumular un valor en un identificador a cada iteración de la estructura. Para ello, sólo debemos hacer

algo tan simple como poner dicho identificador además de en la parte izquierda de la asignación, también en la parte derecha.

El siguiente caso, muestra las situaciones antes mencionadas en un problema algo más complejo y completo que los mostrados hasta el momento. El algoritmo a desarrollar se basa en un programa de venta de artículos; cada cliente puede comprar un número de artículos variable, por lo que el usuario del programa debe de ir introduciendo el precio de cada artículo. Una vez hecho esto, se comprobará el precio del artículo, y si sobrepasa el importe de 5.000 ptas., se le aplicará un descuento del 8%. En el caso de que el importe sea menor, se le otorgará al cliente un punto de bonificación sobre el artículo para la próxima compra. Cuando se hayan procesado todos los artículos, deberemos obtener el importe total de la compra y la cantidad de puntos de bonificación, finalizando de esta manera el algoritmo. El diagrama de flujo correspondiente quedaría como muestra la Figura 18.

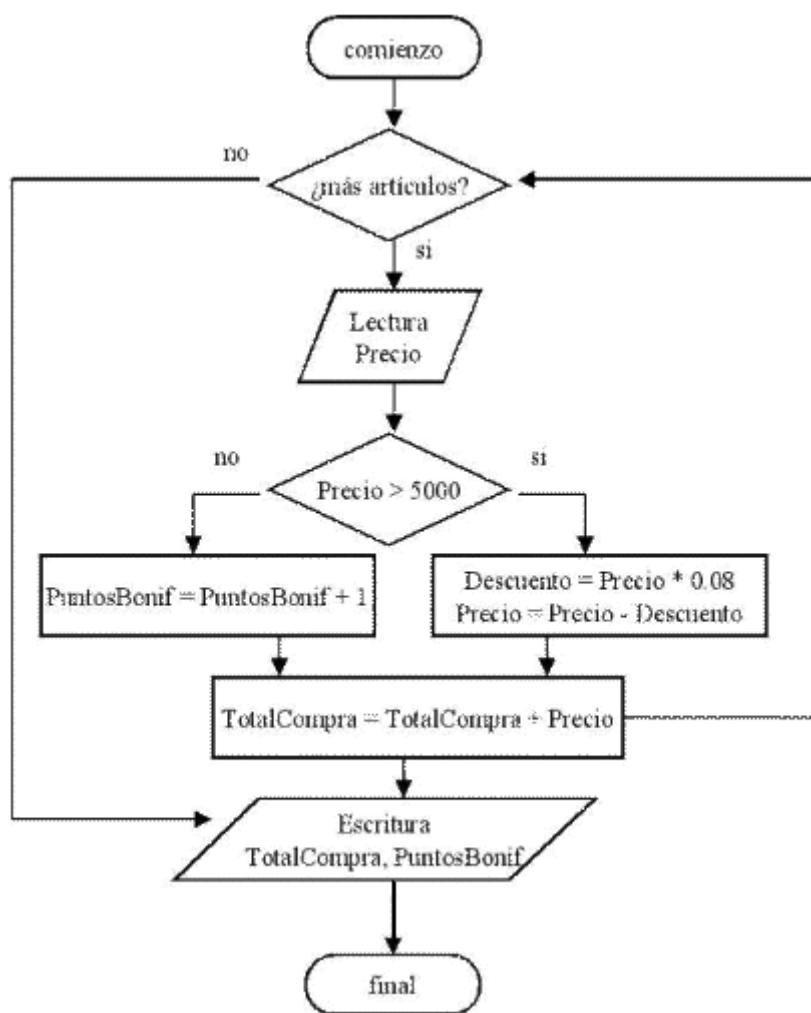


Figura 18. Diagrama de compra de artículos.

Pseudocódigo

El sistema de representación de algoritmos mediante diagramas de flujo, si bien es un medio gráfico y fácil de comprender, presenta el inconveniente de la lentitud en su proceso de creación y dificultad de mantenimiento, ya que un cambio de planteamiento en un diagrama largo, puede significar el rediseño de gran parte del mismo.

Como solución a este problema se hacía necesario una técnica de diseño de algoritmos que permitiera una creación más ágil del mismo mediante un lenguaje sencillo de escribir y mantener.

La respuesta la encontramos en el pseudocódigo, que consiste en una forma de expresar el desarrollo de un proceso empleando palabras de nuestro lenguaje cotidiano junto con símbolos y palabras clave genéricas de la mayoría de lenguajes de programación, es decir, un sistema que se encuentra a medio camino entre el lenguaje natural y el de programación.

No hay un conjunto de normas establecidas o sintaxis para la escritura de pseudocódigo, existiendo una serie de recomendaciones para el desarrollo de algoritmos mediante esta técnica, quedando el resto en manos de cada programador, que lo aplicará según sus necesidades o experiencia.

Partiendo de lo anterior, cuando en pseudocódigo necesitemos realizar operaciones que impliquen un cálculo matemático o comparación entre elementos, emplearemos los operadores explicados en el anterior apartado dedicado a los diagramas de flujo, cuya finalidad es equivalente.

En las indicaciones de principio, fin del algoritmo, lectura y escritura de datos, usaremos también las mismas palabras clave que empleamos en los diagramas de flujo.

Para estructuras repetitivas o bloques de instrucciones que se ejecuten basados en una condición, podemos utilizar las siguientes palabras: Si...FinSi, Para...Siguiente, Mientras...FinMientras, Seleccionar Caso...FinCasos.

Como medio de facilitar la legibilidad de las líneas del algoritmo, indentaremos aquellas que formen parte de un bloque o estructura.

Si debemos escribir comentarios aclaratorios en las operaciones del algoritmo, podemos emplear dos barras invertidas //, comilla ', llaves { }, etc.

Comencemos con un ejemplo sencillo de pseudocódigo, basado en el problema anterior de la apertura de una puerta después de elegir la llave correcta, que mediante este sistema, quedaría de la siguiente forma.

```

Apertura
Comienzo
    elegir una llave
    Si podemos introducir llave en cerrojo
        abrir la puerta
    FinSi
Final

```

Pasemos a continuación, a otro de los ejemplos anteriores realizados para un diagrama, de esta forma podremos establecer las comparaciones necesarias entre una y otra técnica. En este caso, escribiremos el pseudocódigo para el algoritmo de compra de cartuchos de impresora que incluye cálculo de impuestos.

```

CompraCartuchos
Comienzo
    Lectura Cartuchos
    ImporteCartuchos = Cartuchos * 2100
    IVA = ImporteCartuchos * 0.16
    TotalFac = ImporteCartuchos + IVA
    Escritura ImporteCartuchos, IVA, TotalFac
Final

```

Finalmente, convertiremos a pseudocódigo el diagrama del ejemplo sobre compra de artículos

```

ComprArticulos
Comienzo
  Mientras haya artículos
    Lectura Precio
    Si Precio > 5000
      Descuento = Precio * 0.08
      Precio = Precio - Descuento
    Sino
      PuntosBonif = PuntosBonif + 1
    FinSi
    TotalCompra = TotalCompra + Precio
  FinMientras
  Escritura TotalCompra, PuntosBonif
final

```

Diagramas Nassi-Shneiderman

Este tipo de diagramas se basa en la representación de los pasos del algoritmo mediante recuadros, en los que cada paso irá a su vez encerrado en otro rectángulo.

En el caso más simple, ejecución de varias instrucciones sin ningún tipo de condición o desvío en el flujo del programa, como es el ejemplo de apertura de llave sin selección, un diagrama de este tipo para dicho algoritmo quedaría de cómo se muestra en la Figura 19.

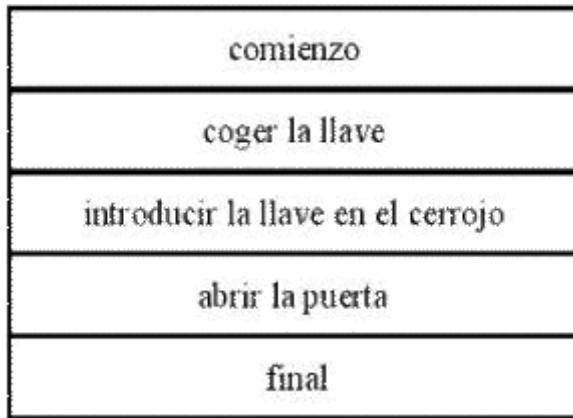


Figura 19. Diagrama N-S de apertura de puerta.

Este mismo algoritmo con selección de llave, y una pequeña variación, se representaría según la Figura 20.

En este tipo de diagramas, como observará el lector, los bloques de instrucciones que están a un mayor nivel de profundidad o que en pseudocódigo deberíamos indentar, se representan en recuadros interiores al cuadro principal del diagrama. Veamos en la Figura 21, como quedaría el algoritmo de compra de artículos en un diagrama de este tipo, pero eliminando la condición de que el artículo tenga descuento o bonificación para no recargar en exceso el diagrama.

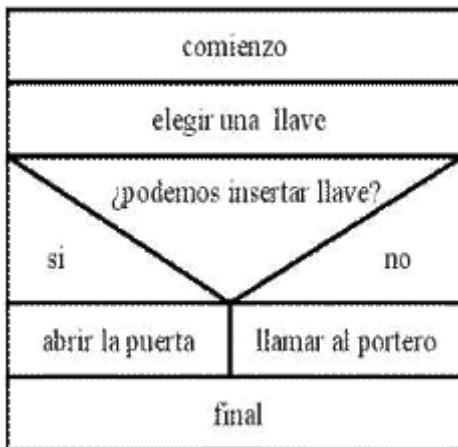


Figura 20. Diagrama N-S de apertura de puerta con selección de llave.

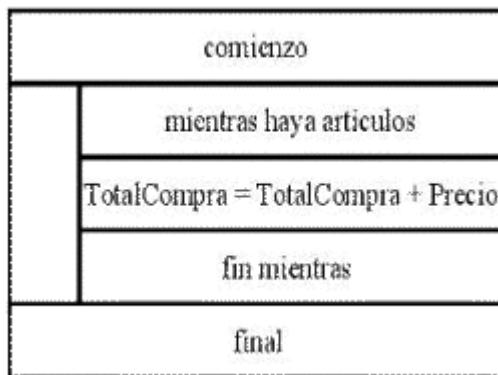


Figura 21. Diagrama N-S de compra de artículos.



Programas

¿Qué es un programa?

Como se definió en un tema anterior, un programa se compone de instrucciones o líneas de código fuente, que se obtienen al trasladar las indicaciones de un diagrama de flujo o pseudocódigo al lenguaje de programación elegido para desarrollar el programa. Una vez escrito el programa en un lenguaje determinado, podremos ejecutarlo para resolver los problemas para los que ha sido diseñado.

Fases de ejecución

De forma muy genérica, las fases o pasos que se llevan a cabo durante el periodo de funcionamiento (ejecución) de un programa son tres: captura de datos, proceso de información, muestra de resultados. Las podemos ver representadas en la Figura 22.

La captura de datos consiste en la solicitud al usuario de información, que posteriormente utilizarán los procesos escritos en el programa para resolver los problemas planteados.

El proceso de información es la etapa que se encarga de ejecutar el algoritmo/s planteado en el programa utilizando los datos introducidos por el usuario en la fase de captura.

La muestra de resultados es la fase final en la ejecución de un programa, en la que se obtiene la información resultante de ejecutar el problema del algoritmo, mostrándola al usuario.

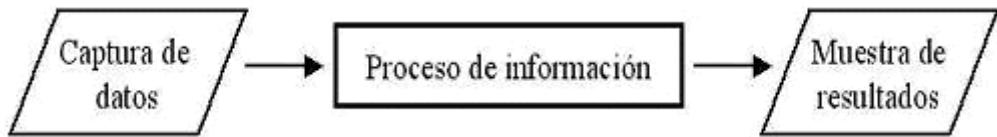


Figura 22. Fases de ejecución de un programa.

Componentes de un programa

Los componentes son todos aquellos elementos del lenguaje que utilizaremos para escribir el programa. A continuación se muestran de forma esquemática los principales componentes, que serán explicados en el próximo tema del curso.

- Datos.
 - Simples.
 - Compuestos.
 - Definidos por el programador.
- Identificadores.
 - Variables.
 - Constantes.
 - Subrutinas.
- Palabras reservadas.
 - Comandos (instrucciones).
 - Funciones propias del lenguaje.
 - Objetos propios del lenguaje.
- Operadores.
 - Aritméticos.
 - Relacionales.
 - Lógicos.
- Expresiones.
- Estructuras de control.
 - Selección.
 - Repetición.

- Contadores.
- Acumuladores.
- Interruptores.

Programas en Visual Basic

Este curso está diseñado para explicar los conceptos de común aplicación en todos los lenguajes de programación, de manera que una vez finalizado, el lector pueda comenzar el aprendizaje de cualquier herramienta de programación, aparte de Visual Basic, que es el lenguaje impartido aquí.

Sin embargo, ya que los ejemplos a exponer requieren estar realizados en un lenguaje determinado, y en este caso dicho lenguaje es Visual Basic, vamos a introducirnos en este apartado en las características principales de su entorno de desarrollo. Por el momento nos limitaremos a los elementos que implican la escritura de código; más adelante, trataremos los aspectos de la programación con formularios y controles.

En los antiguos lenguajes basados en modo texto, el programador disponía de un conjunto de utilidades separadas para la creación de programas: un editor de texto para escribir el código fuente de los programas, el compilador y el enlazador formaban básicamente los utensilios de trabajo. Adicionalmente se podía disponer de un programa para crear y manipular bases de datos, otro para la creación de librerías de código, etc., pero esta forma de desarrollo era bastante artesanal.

Posteriormente, con el auge de Windows y el aumento de aspectos a tratar en un programa desarrollado para este sistema operativo, comenzaron a proliferar herramientas de programación que incluían todos los elementos antes mencionados: editor de código, compilador, etc., más todas las utilidades necesarias para un programa que debía funcionar bajo una interfaz gráfica: editor de ventanas, controles, etc., aglutinados en lo que se conoce como IDE (Entorno integrado de desarrollo).

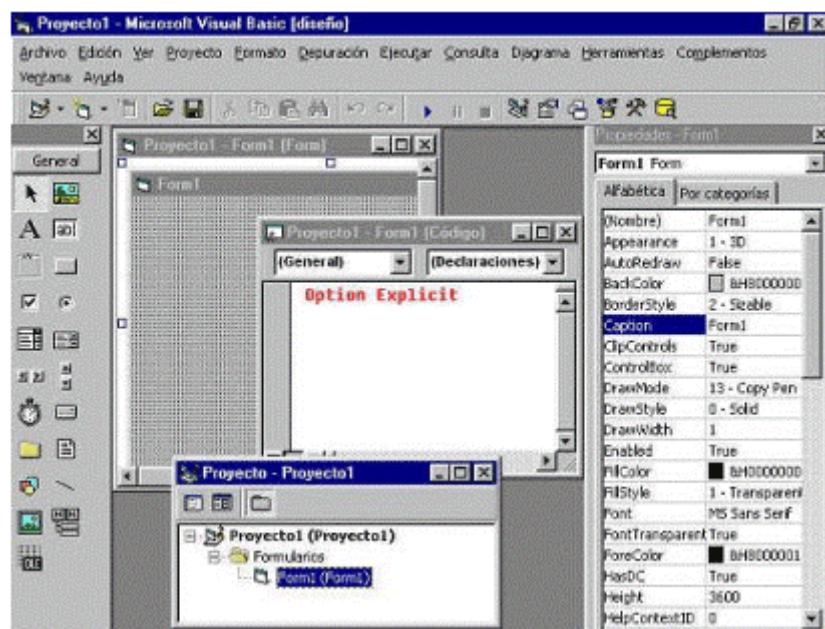


Figura 23. Entorno de desarrollo de Visual Basic.

Visual Basic o VB como también lo llamaremos a partir de ahora, es una de estas herramientas de programación, que ofrece un entorno de desarrollo con todos los componentes necesarios para la creación de programas. La Figura 23 muestra el aspecto típico de una sesión de trabajo en VB.

Quizá el elevado número de elementos a nuestra disposición, pueda a primera vista asustar al lector, pero no se preocupe, con un poco de práctica rápidamente nos adaptaremos a este modo de desarrollo y a desenvolvernos con soltura.

Entre las ventanas del entorno que formarán parte de nuestro trabajo habitual, destacamos las siguientes:

- Explorador de proyectos (Figura 24). Esta ventana muestra todos los elementos que forman parte del programa (proyecto, como se denomina en Visual Basic) que estamos desarrollando: formularios, código, etc.

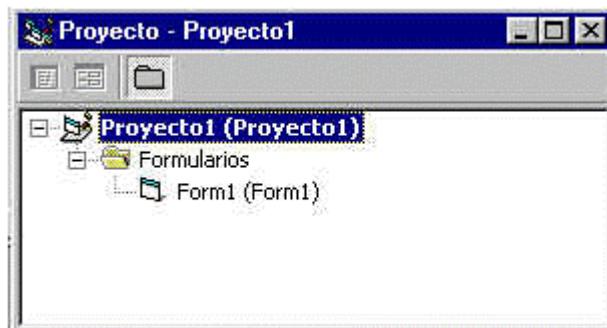


Figura 24. Explorador de proyectos.

Aparece por defecto al crear un nuevo proyecto o abrir uno existente, pero podemos mostrarla también mediante la opción del menú de VB Ver+Explorador de proyectos, la combinación de teclado Ctrl+R o el botón de la barra de herramientas(Figura 25).



Figura 25. Botón del Explorador de proyectos en la barra de herramientas.

- Código. En esta ventana escribiremos todo el código fuente que precisemos para nuestro programa. Dependiendo de si estamos escribiendo el código de un formulario o instrucciones que no están directamente relacionadas con un formulario, tendremos varias ventanas como la que se muestra en la Figura 27, a nuestra disposición.

Podemos visualizar esta ventana mediante la opción del menú de VB Ver+Código, y el botón de la barra de herramientas del Explorador de proyectos que vemos en la Figura 26



Figura 26. Botón de la ventana de código en la barra de herramientas del Explorador de proyectos.

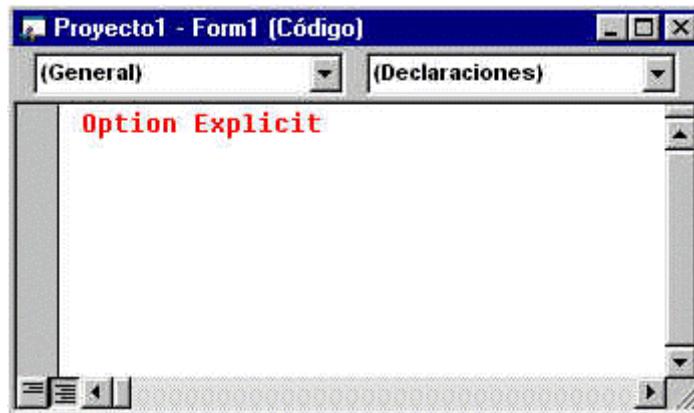


Figura 27. Ventana de código.

- Formulario (Figura 28). Nos permite la creación de una ventana de las que formarán parte del programa. Ya que VB emplea el término formulario para referirse a las ventanas de Windows, también será el utilizado en este curso.

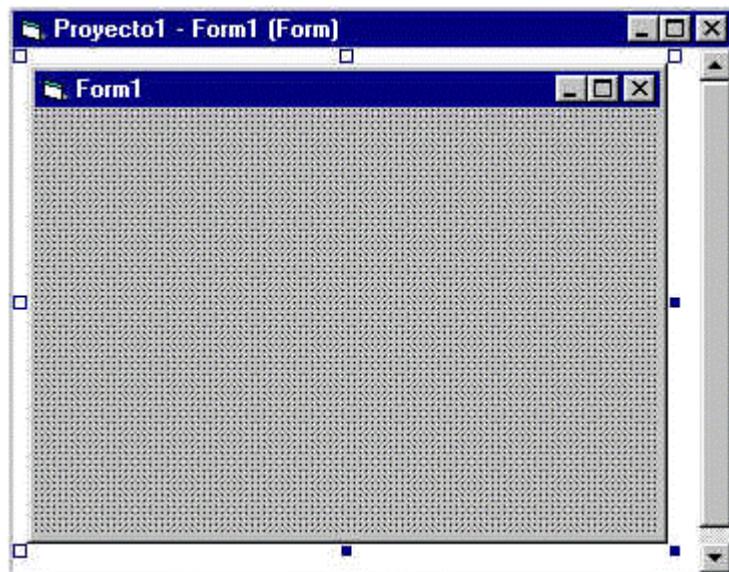


Figura 28. Ventana de formulario.

Esta ventana se muestra por defecto al abrir un proyecto, pudiendo ser abierta también con la opción Ver+Objeto del menú de Visual Basic, la combinación de teclado Mayús+F7 y el botón de la barra de herramientas del Explorador de proyectos, que aparece en la Figura 29.



Figura 29. Botón de la ventana de formulario en la barra de herramientas del Explorador de proyectos.

- Propiedades (Figura 30). Nos permite la edición de las propiedades del formulario o control que estemos creando o modificando actualmente. También permite manipular las propiedades de cualquier otro elemento del proyecto que tengamos seleccionado.

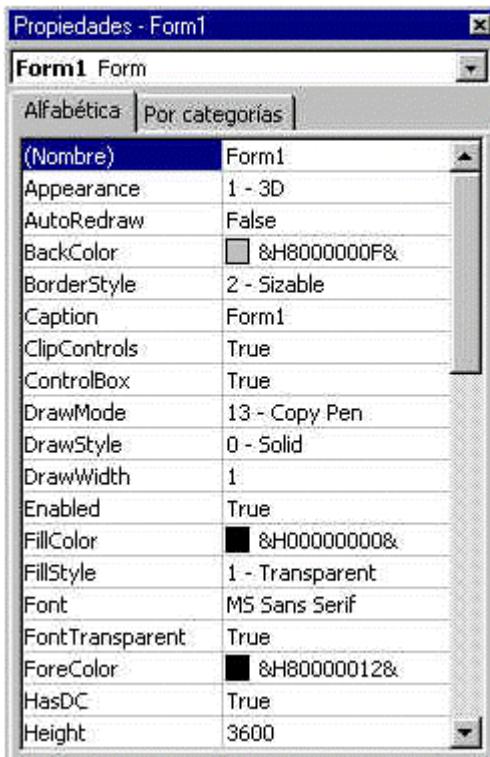


Figura 30. Ventana de propiedades.

Normalmente, esta ventana se encuentra abierta, pero podemos acceder a ella en el caso de que no esté visible con la opción Ver+Ventana Propiedades del menú de VB, la tecla F4 y el botón de la barra de herramientas de VB (Figura 31).



Figura 31. Botón de la ventana de Propiedades en la barra de herramientas.

- Cuadro de herramientas (Figura 32). Muestra el conjunto de controles que se encuentran disponibles en el proyecto para poder ser insertados en un formulario.



Figura 32. Cuadro de herramientas.

Aparece por defecto en el entorno y también podemos visualizarla con la opción Ver+Cuadro de herramientas del menú de VB y el botón de la barra de herramientas de VB que aparece en la Figura 33



Figura 33. Botón del Cuadro de herramientas en la barra de herramientas.

Elementos del lenguaje

Introducción

Un lenguaje de programación se compone de una serie de elementos que se utilizan para escribir las diferentes instrucciones y expresiones de que consta un programa. Además de los mencionados elementos, existen una serie de reglas de construcción de expresiones, que son las encargadas de decidir si determinada línea o líneas de código están bien escritas, de forma que nuestro programa se ejecute correctamente. Al conjunto de normas establecidas para la correcta escritura del código en un lenguaje de programación se le denomina sintaxis.

Datos

Los datos es la información que utiliza un programa para poder realizar operaciones. En función de la información que contengan se pueden clasificar en varios grupos: simples, compuestos y definidos por el programador.

Simples

Un dato simple está formado por una sola unidad de información; dependiendo del contenido de esa información, los datos simples se dividen en:

Numérico

Como su nombre indica, este tipo de dato contiene un número, que pueden ser entero (sin parte decimal) o real (con parte decimal).

Como ejemplos de número enteros podemos mostrar los siguientes:

8 1000 415 -80

Como ejemplos de número reales tenemos estos otros:

8.35 120.55 -28.42

Debido a la naturaleza de los ordenadores, hay que tener presente que para representar un número real en un programa, tal y como acabamos de observar, debemos utilizar como separador decimal el punto en lugar de la coma.

Carácter

Un dato de tipo carácter está formado por un único elemento, que puede ser cualquier letra, símbolo o número, teniendo en cuenta que los números que son representados como caracteres no se pueden utilizar para realizar operaciones aritméticas.

Para representar un dato de este tipo en un programa, se utilizan los denominados delimitadores, entre los que se encierra el carácter correspondiente, suelen ser habitualmente y para la mayoría de los lenguajes de programación, los símbolos de comilla simple o doble, como se muestra a continuación en los siguientes ejemplos:

"a" "?" "4" "+"

Cuando se desarrollan programas, es poco frecuente o útil el uso de un tipo carácter, siendo mucho más habitual agrupar varios datos de este tipo en lo que en programación se conoce como cadena de caracteres. Este aspecto puede resultar un tanto confuso, ya que una cadena es un dato compuesto y no simple, pero existen ciertos lenguajes que tratan las cadenas como datos de tipo simple, Visual Basic es uno de ellos, por lo que en este curso al referirnos a una cadena de caracteres la consideraremos un dato simple. Las siguientes líneas muestran ejemplos de datos de tipo cadena.

"el reloj marca las cinco"

"apagar el monitor antes de salir"

"el importe del ticket es: 5.000 ptas."

Algo similar ocurre con las fechas, aunque combinan números, letras y símbolos para su representación, son tratadas por el lenguaje como una unidad de información.

Lógico

Un dato lógico es aquel que representa uno de dos valores posibles: verdadero o falso, indicándose estos valores en Visual Basic como True y False respectivamente.

Compuestos

Un dato compuesto está formado por varios datos de tipo simple. Como ya hemos mencionado, una cadena es un dato compuesto (por datos simples de carácter) aunque algunos lenguajes lo traten como simple.

Como dato compuesto disponemos de los arrays o matrices, que son tablas de datos con un número fijo o variable de elementos.

Definidos por el usuario

En esta categoría tenemos el llamado tipo de dato definido por el usuario (Type) y el tipo enumerado (Enum), que permiten combinar un conjunto de datos de forma más compleja que los tipos compuestos.

Tanto los tipos compuestos como los definidos por el usuario serán tratados con mayor profundidad posteriormente, ya que es recomendable conocer algunos aspectos previos sobre el lenguaje antes de ahondar en este tipo de datos.

Identificadores

Un identificador es un nombre que el programador asigna a ciertos elementos del programa, como variables, constantes, subrutinas, etc., para poder reconocerlos durante el desarrollo del mismo. Dicho nombre debe comenzar por una letra y no puede ser una palabra reservada del lenguaje, ni contener caracteres especiales como operadores aritméticos, de relación, etc.

Aunque podemos utilizar cualquiera que se nos ocurra, es muy recomendable que el nombre del identificador esté relacionado con su contenido, de forma que durante la revisión del código del programa, sepamos rápidamente lo que contiene con sólo ver el nombre.

Palabras reservadas

Son todos aquellos nombres que forman parte integrante del lenguaje de programación, como puedan ser instrucciones, estructuras de código, funciones propias del lenguaje, objetos, etc. Este tipo de palabras no las podemos utilizar como identificadores, debido a que son empleadas internamente por el lenguaje.

Algunas palabras reservadas en Visual Basic son: Dim, Sub, Function, Optional, For, Integer.

Organización del código

El código de un programa se organiza en subrutinas, en las cuales escribimos las instrucciones necesarias para resolver un determinado problema. Dependiendo de la complejidad y la cantidad de situaciones a resolver, un programa tendrá un mayor o menor número de subrutinas o procedimientos.

Los procedimientos serán tratados con profundidad en un próximo apartado, por lo que de momento nos bastará saber que para declarar un procedimiento, utilizaremos la palabra clave Sub, seguida del

nombre del procedimiento, escribiendo a continuación las instrucciones necesarias y finalizando el procedimiento con End Sub, como vemos en el Código fuente 1.

```
Sub Calcular()
    instrucción1
    instrucción2
    instrucción3
    instrucción4
    instrucción5
    .....
    .....
    .....
    instrucciónN
End Sub
```

Código fuente 1

Todo programa debe tener un punto de entrada o subrutina inicial, en la que, si el programa a ejecutar es de pequeño tamaño, contendrá todo el código necesario. Sin embargo esta situación no es muy corriente, ya que lo normal es que los programas tengan una cantidad de código considerable, que obligue a separar en diferentes subrutinas. Por este motivo el procedimiento inicial, que en VB se denomina Main(), contiene el código para inicializar valores en el programa y hacer llamadas a otros procedimientos del programa entre los que está repartido el resto del código.

Los ejemplos mostrados a partir de ahora, estarán todos basados en programas sencillos y sólo precisarán el uso de un procedimiento, que será Main().

El motivo es que por el momento nos limitaremos a los aspectos del lenguaje, sin entrar en cuestiones relacionadas con formularios ni demás elementos visuales de un programa Windows.

Para ello debemos preparar el entorno de programación siguiendo los siguientes pasos:

- Ejecutar el entorno de Visual Basic 6. Desde el menú de inicio de Windows seleccionaremos las siguientes opciones: Programas+Microsoft Visual Studio 6.0+Microsoft Visual Basic 6.0.

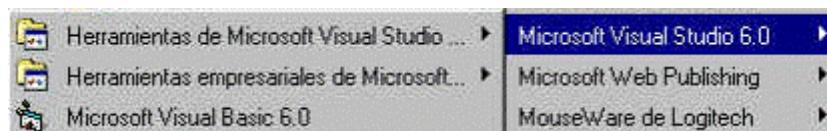


Figura 34. Menú de selección de Visual Basic.

- Seleccionar el tipo de proyecto. En la ventana de selección del tipo de proyecto elegir EXE Estándar.

Cuando a lo largo del curso se indique la creación de un nuevo proyecto, siempre nos estaremos refiriendo a este tipo.

También es posible crear un nuevo proyecto estando ya en VB, si seleccionamos la opción de menú Archivo+Nuevo proyecto o la combinación de teclado Ctrl+N.



Figura 35. Ventana inicial de Visual Basic.

- Agregar un módulo de código al proyecto. Un módulo de código es un elemento de un proyecto diseñado para escribir los procedimientos y declaraciones integrantes del programa. Podemos incluir más de un módulo al programa, tantos como necesitemos para organizar nuestro código. Los formularios tienen su propio módulo de código.

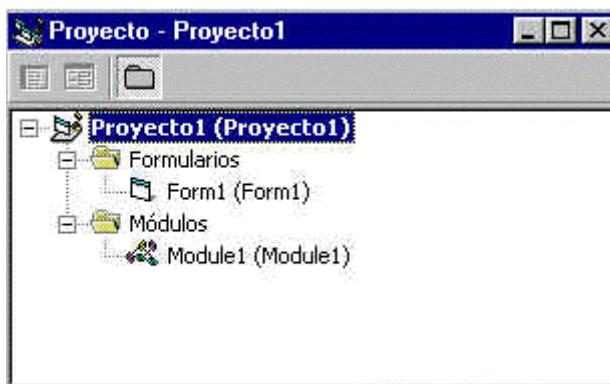


Figura 36. Ventana de proyecto con un nuevo módulo de código.

Utilizando la opción del menú de VB Proyecto+Agregar módulo, o bien desde la ventana de proyecto haremos clic con el botón derecho del ratón para abrir el menú contextual de esta ventana, eligiendo la opción Agregar+Módulo. Aparecerá una ventana que nos permitirá crear un nuevo módulo o abrir uno existente. Seleccionaremos la opción de crear uno nuevo, que se reflejará inmediatamente en la ventana de proyecto.

- Escribir la declaración de Main(). En este paso haremos doble clic sobre el nombre del nuevo módulo creado, lo que abrirá su ventana de código permitiéndonos escribir la declaración de la subrutina Main(), es decir, sólo el nombre del procedimiento, sin incluir instrucciones, como se muestra en la Figura 37.

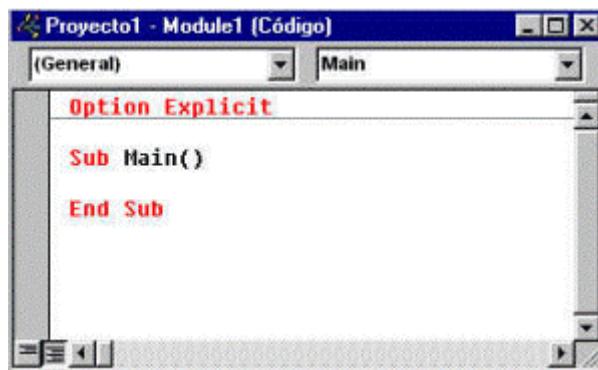


Figura 37. Ventana de código con la declaración de Main().

También podemos acceder a esta ventana con la opción Ver+Código de VB o el menú contextual de la ventana de proyecto, opción Ver código.

- Establecer el objeto inicial del proyecto. Por defecto el proyecto comenzará ejecutando el formulario que incluye. Para hacer que se inicie por un procedimiento Main(), debemos seleccionar la opción del menú de VB Proyecto+Propiedades de <NombreProyecto>..., que nos mostrará la Figura 38.

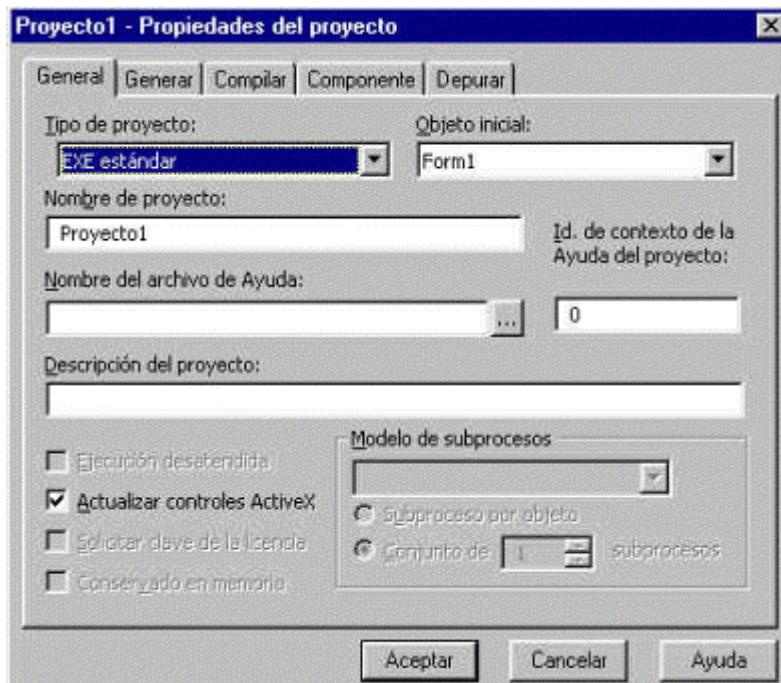


Figura 38. Ventana de propiedades del proyecto.

En Objeto inicial, debemos desplegar la lista que contiene y seleccionar Sub Main, esto hará que la ejecución del programa se inicie por un procedimiento de este tipo.

También podemos acceder a esta ventana desde la ventana de proyecto, haciendo clic sobre el nombre del proyecto y después clic con el botón derecho para abrir el menú contextual. Finalmente elegiremos la opción Propiedades de <NombreProyecto>...

- Asignar un nombre al módulo. Cada vez que añadimos un nuevo elemento a un proyecto, ya sea un módulo o un formulario, VB le asigna un nombre por defecto, pero dicho nombre, dista de ser lo bastante reconocible una vez que el proyecto crece en elementos. Para asignar un nombre que nos sea más identificativo, podemos realizar cualquiera de las siguientes acciones: abrir la ventana de proyecto, hacer clic sobre el elemento a nombrar y abrir el menú contextual de esta ventana, seleccionando la opción Propiedades; pulsar F4; el botón de la barra de herramientas correspondiente a las propiedades; o la opción de menú de VB Ver+Ventana propiedades. Cualquiera de estas acciones mostrará dicha ventana, que en lo referente a un módulo de código, nos permitirán darle un nuevo nombre en la propiedad Nombre, como podemos ver en la Figura 39, en la que se cambia el nombre del módulo por defecto por Datos.



Figura 39. Cambio de nombre a un módulo desde la ventana de propiedades.

Realizados los siguientes pasos, ya estamos en disposición de comenzar con los diferentes elementos del lenguaje. Todos los ejemplos mostrados a continuación se deberán realizar en un procedimiento Main(), y cada nuevo proyecto que creemos se deberá configurar según los pasos que acabamos de mostrar.

Cuando escribamos instrucciones en Main(), podremos ejecutarlas desde el propio entorno de VB pulsando F5 o el botón de la barra de herramientas que aparece en la Figura 40



Figura 40. Botón de la barra de herramientas para ejecutar un programa.

El módulo de código

Como se ha explicado anteriormente, un módulo es aquella parte de un proyecto que contiene código del programa. Se compone de declaraciones y procedimientos como podemos apreciar en la Figura 41.

La zona de declaraciones es la parte del módulo reservada a la escritura de instrucciones y declaraciones que afecten a todo el código del módulo; podemos localizarla en la cabecera o comienzo del módulo.

La zona de procedimientos corresponde al resto del módulo, y en ella se escribirán todos los procedimientos o subrutinas que sean necesarios.

Para acceder a los diferentes elementos del módulo en la ventana de código, disponemos de la lista desplegable Procedimiento, situada en la parte superior derecha de la ventana, como podemos ver en

la Figura 42. Si seleccionamos el elemento Declaraciones nos situaremos en dicha zona del módulo. El resto de elementos corresponde a los procedimientos escritos en el módulo; al seleccionar uno de ellos, nos situaremos al comienzo del mismo.

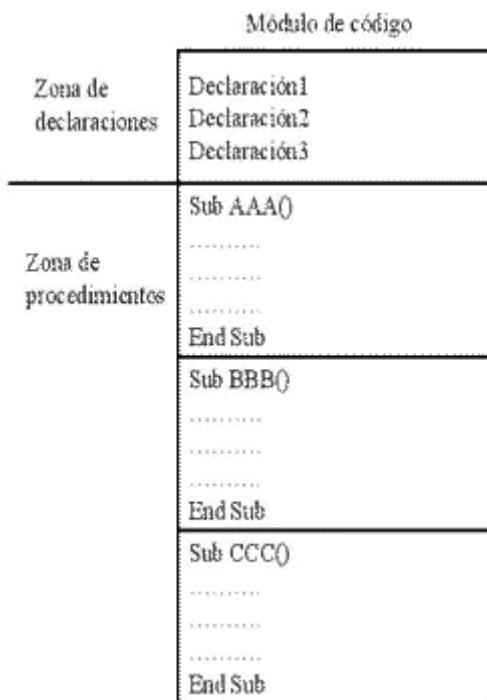


Figura 41. Esquema de distribución de código en un módulo de VB.

En la Figura 42 podemos observar como además de las declaraciones, aparecen los nombres de los procedimientos `Calcular()` y `Main()`. En la zona de declaraciones tenemos la instrucción `Option Explicit` que será explicada posteriormente.

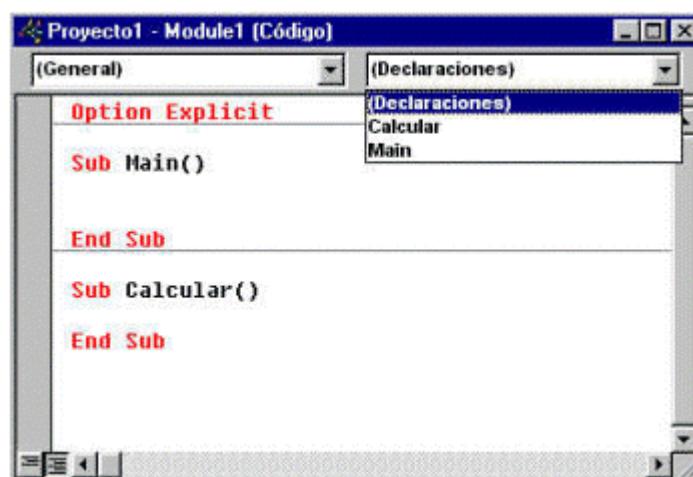


Figura 42. Ventana de código mostrando la lista de procedimientos.

La otra lista desplegable, Objeto, situada en la parte superior izquierda de la ventana, será tratada en el tema dedicado al desarrollo de programas, en la codificación de eventos de formularios.

Comentarios

Es una buena costumbre incluir comentarios aclaratorios en el código del programa, que sirvan como documentación posterior a la hora de revisar el código por una posible corrección debido a un mal funcionamiento o porque tengamos que ampliar la funcionalidad del programa.

En VB se utiliza el carácter de comilla simple (') para poder escribir anotaciones dentro del código del programa. Una vez incluida la comilla, todo lo que vaya a continuación será tenido en cuenta como comentario y no como instrucciones. Veamos un ejemplo en el Código fuente 2.

```
Sub Main()
    ' vamos a comenzar a ver el código
    instrucción1
    instrucción2 ' este comentario va a continuación de una línea
    ' esta es la última instrucción
    instrucción3
End Sub
```

Código fuente 2

Variables

Una variable es aquel elemento dentro de un programa que guarda un valor; dicho valor podrá cambiar durante el funcionamiento del programa (de ahí el nombre de variable). El contenido de la variable no es otra cosa que una posición en la memoria del ordenador que almacena el valor, y debido a que la memoria es un medio de almacenamiento no permanente, al finalizar el programa el valor se perderá, por lo que el uso de variables se limita al tiempo de ejecución del programa. Para manejar y asignar nombre a una variable, emplearemos un identificador.

Declaración

La declaración de una variable es la operación que se encarga de crear e inicializar una nueva variable en el programa. En VB se utiliza la instrucción Dim como podemos ver en el Código fuente 3

```
Dim Valor
```

Código fuente 3

El lugar de declaración de una variable depende del lenguaje de programación utilizado, en Visual Basic podemos declarar una variable en cualquier punto de un procedimiento, pero lo más recomendable es realizar la declaración de todas las variables del procedimiento al comienzo del mismo, a continuación del nombre, lo que nos ayudará a conseguir un código fuente más organizado, como se puede apreciar en el Código fuente 4.

```
Sub Main()
    ' declaración de variables al comienzo
    Dim Valor
```

```

Dim Nuevo
.....
.....
.....
' declaración de variables en la mitad
' de la subrutina
Dim Avanzar
.....
.....
.....
End Sub

```

Código fuente 4

Denominación

Como se comentaba en el apartado sobre identificadores, el nombre que podemos asignar a una variable debe seguir ciertas reglas, como no ser una palabra reservada ni incluir caracteres especiales del lenguaje. El Código fuente 5 nos muestra algunos ejemplos de declaraciones.

```

Sub Main()

' declaraciones correctas
Dim B
Dim Hola
Dim ABCD
Dim Nuevo10
Dim Es_otro

' declaraciones incorrectas
Dim 7Nuevo
Dim Avan+zar
Dim Va-bien

End Sub

```

Código fuente 5

Tipificación

La tipificación o asignación del tipo de dato a una variable nos permite establecer que clase de información va a contener la variable, el rango de valores permitidos y optimizar el código de cara a que la ejecución del programa sea más rápida.

En VB el tipo de dato se asigna al declarar la variable mediante la partícula As, como podemos ver en el Código fuente 6.

```

Sub Main()
Dim Propio As String    ' cadena de caracteres
Dim Valor As Integer    ' número
Dim Ahora As Date       ' fecha
End Sub

```

Código fuente 6

La Tabla 1 muestra los tipos de datos disponibles en este lenguaje.

Tipo de dato	Rango de valores disponibles	Espacio a reservar
Byte	0 a 255	1 byte
Boolean	True o False	2 bytes
Integer (entero)	-32.768 a 32.767	2 bytes
Long (entero largo)	-2.147.483.648 a 2.147.483.647	4 bytes
Single (coma flotante de precisión simple)	Valores positivos: 1,401298E-45 a 3,402823E38 Valores negativos: -3,402823E38 a -1,401298E-45	4 bytes
Double (coma flotante de precisión doble)	Valores positivos: 4,94065645841247E-324 a 1,79769313486232E308 Valores negativos: -1,79769313486232E308 a -4,94065645841247E-324	8 bytes
Currency (entero a escala –valores monetarios)	-922.337.203.685.477,5808 a 922.337.203.685.477,5807	8 bytes
Decimal	+/- 79.228.162.514.264.337.593.543.950 .335 (a) +/- 7,9228162514264337593543950335 (b) +/- 0,00000000000000000000000000000001 (c)	14 bytes
Date (fecha)	1 de Enero de 100 a 31 de Diciembre de 9999	8 bytes
Object	Cualquier tipo de objeto	4 bytes
String (cadena de longitud variable)	De 0 a 2.000 millones	10 bytes más la longitud de

		la cadena
String (cadena de longitud fija)	De 1 a 65.400 aprox.	Longitud de la cadena
Variant (usando números)	Cualquier número hasta intervalo de un Double	16 bytes
Variant (usando caracteres)	Igual que para un String de longitud variable	22 bytes más la longitud de la cadena

- (a) Sin punto decimal.
- (b) Con 28 posiciones a la derecha del signo decimal.
- (c) Número más pequeño distinto de cero.

Tabla 1. Tipos de datos en Visual Basic.

Si al declarar una variable no indicamos el tipo, por defecto tomará Variant, ya que es un tipo genérico que admite cualquier valor.

Según la información que acabamos de ver, si declaramos una variable de tipo Integer e intentamos asignarle el valor 42444 se va a producir un error, ya que no se encuentra en el intervalo de valores permitidos para esa variable. Esto puede llevar a preguntarnos ¿por qué no utilizar siempre Variant y poder usar cualquier valor?, o mejor ¿para qué necesitamos asignar tipo a las variables?.

El motivo de tipificar las variables reside en que cuando realizamos una declaración, el compilador debe reservar espacio en la memoria para los valores que pueda tomar la variable, como puede ver el lector en la tabla anterior, no requiere el mismo espacio en memoria una variable Integer que una Date. Si declaramos todas las variables como Variant los gastos de recursos del sistema será mayor que si establecemos el tipo adecuado para cada una, ello redundará en un mejor aprovechamiento de las capacidades del sistema y un código más veloz en ejecución. Cuantos más programas se diseñen optimizando en este sentido, el sistema operativo ganará en rendimiento beneficiándose el conjunto de aplicaciones que estén en ejecución.

Visual Basic dispone de una utilidad al asignar el tipo a una variable, que nos muestra la lista de tipos disponibles para poder seleccionar uno sin tener que escribir nosotros el nombre. Al terminar de escribir As, aparecerá dicha lista, en la que pulsando las primeras letras del tipo a buscar, se irá situando en los más parecidos. Una vez encontrado, pulsaremos la tecla Enter o Tab para tomarlo.

Si el lector no tiene esta característica disponible, puede habilitarla, seleccionando la opción del menú de VB Herramientas+Opciones, que mostrará la ventana de opciones. Una vez aquí, sólo debe marcar la opción Lista de miembros automática.

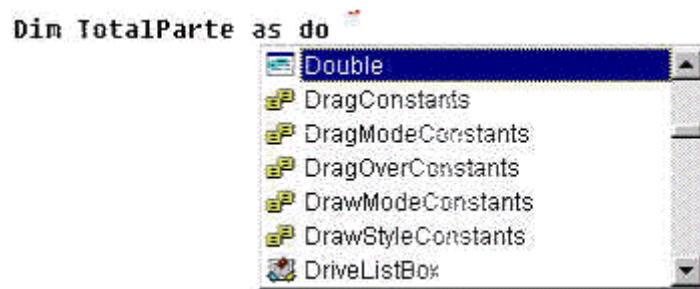


Figura 43. Lista de tipos disponibles al declarar una variable.

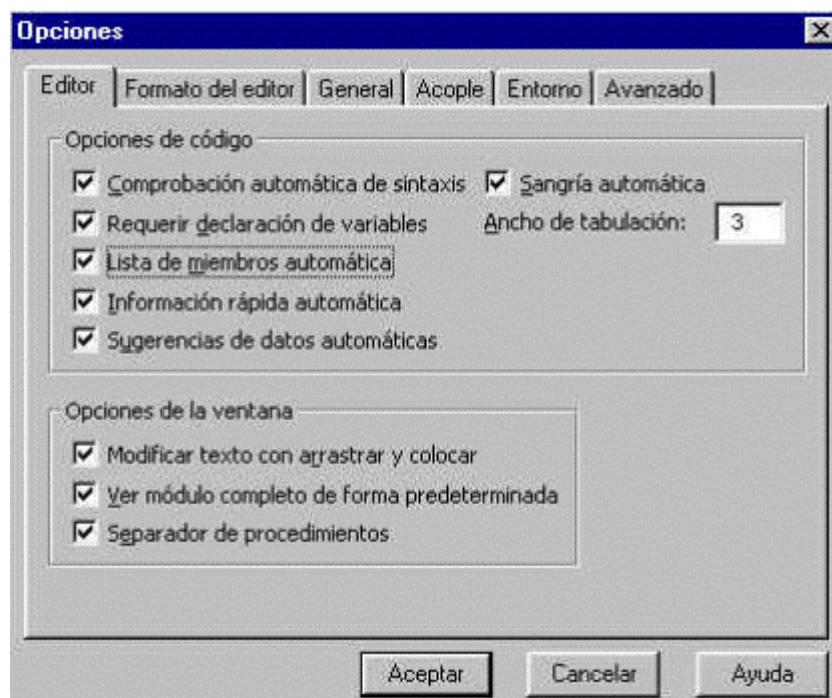


Figura 44. Ventana de opciones de Visual Basic.

Asignación

Para establecer o asignar un valor a una variable, utilizaremos el carácter igual (=), situando en la parte izquierda de la asignación la variable y en la derecha el valor a asignar, como se muestra en el Código fuente 7.

```
ValorNum = 520
```

Código fuente 7

Según el valor a asignar, necesitaremos encerrarlo entre delimitadores, que en el caso de cadenas serán comillas dobles(") y para las fechas el símbolo de almohadilla (#). Los números no necesitan delimitador. Vemos unos ejemplos en el Código fuente 8.

```
Sub Main()
Dim Propio As String    ' cadena de caracteres
Dim Valor As Integer    ' número

Propio = "mesa"
Valor = 12144

End Sub
```

Código fuente 8

Una variable de tipo cadena puede contener cualquier número de caracteres. Sin embargo, si queremos limitar la cantidad de caracteres que puede guardar una variable, emplearemos el símbolo de asterisco seguido del número de caracteres, como se muestra en el Código fuente 9.

```
' esta cadena podrá contener hasta un
'máximo de 8 caracteres
Dim Nombre As String * 8
```

Código fuente 9

A pesar de que la asignación de fechas permite el uso de comillas dobles como delimitador, se recomienda usar el símbolo #, ya que facilita la lectura del código.

El formato de asignación de fecha a una variable será mes/día/año cuando utilicemos el delimitador #, sin embargo en el caso de usar comillas dobles como delimitador, podemos asignar la fecha como día/mes/año.

En el Código fuente 10 se muestran varias asignaciones a una variable de tipo fecha, el comentario junto a cada asignación muestra el contenido de la variable después de haber asignado el valor. Como puede comprobar el lector, las variables de tipo fecha permiten además especificar valores horarios.

```
Sub Main()
Dim Ahora As Date ' fecha
Ahora = #10/5/1999#  ' 5/10/99
Ahora = "10-5-99"  ' 10/05/99
Ahora = #12/31/1992 6:20:40 PM#  ' 31/12/92 18:20:40
Ahora = #12/31/1992 6:20:40 AM#  ' 31/12/92 6:20:40
Ahora = #1:20:40 PM#  ' 13:20:40
End Sub
```

Código fuente 10

Si necesitamos asignar el valor de una variable a otra, pondremos la variable de la que vamos a tomar su valor en la parte derecha de la asignación, lo vemos en el Código fuente 11.

```
Sub Main()
Dim primero As String
Dim nuevo As String
primero = "silla"
```

```
' ahora las dos variables contienen
' el mismo valor
nuevo = primero
End Sub
```

Código fuente 11

En variables Variant es posible establecer valores no válidos asignando la palabra clave Null (Nulo).

```
Sub Main()
Dim prueba As Variant
prueba = 989
prueba = Null
End Sub
```

Código fuente 12

Valor inicial

Toda variable recién declarada contiene un valor inicial que dependerá del tipo de dato de la variable. A continuación se muestran algunos de los valores iniciales según sea el tipo de variable declarada.

- Numérico. Cero (0).
- Cadena. Cadena vacía ("").
- Fecha. 0:00:00
- Variant. Vacío (Empty).

Declaración obligatoria

La obligatoriedad de declarar una variable depende del lenguaje. Visual Basic puede ser configurado para forzar la declaración de variables o permitir emplear variables sin declararlas previamente.

Las ventajas de usar variables declaradas con el tipo de dato adecuado ya han sido comentadas en el punto dedicado a la tipificación; si no declaramos ninguna variable, todas serán asignadas internamente por VB como Variant, con la consiguiente penalización en el rendimiento de la aplicación.

La recomendación general es que siempre se declaren todas las variables utilizadas en el programa, y para asegurarnos de ello, situaremos en la zona de declaraciones del módulo de código la instrucción Option Explicit. Esta instrucción revisa antes de ejecutar una subrutina si existen variables sin declarar, provocando un error en caso afirmativo.

La Figura 45 muestra el uso de esta sentencia.

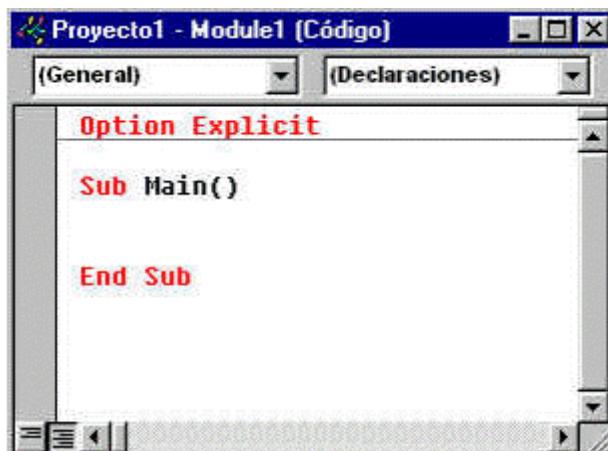


Figura 45. Instrucción Option Explicit situada en la zona de declaraciones.

Después de insertar esta instrucción, escribiremos una línea de código en la que asignaremos un valor a una variable sin declarar, como se muestra en el Código fuente 13.

```
Sub Main()
    codigo = 850
End Sub
```

Código fuente 13

Al ejecutar el programa se producirá el error que aparece en la Figura 46.

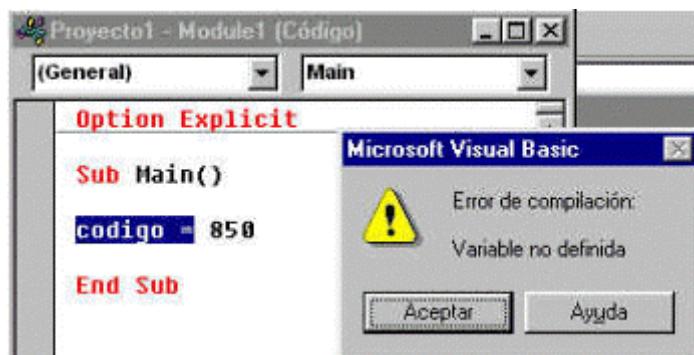


Figura 46. Error en variable no declarada.

Esta instrucción debe incluirse en todos los módulos de un proyecto en los que se necesite declarar las variables utilizadas. Si no queremos preocuparnos por dicha labor, podemos hacer que cada vez que agreguemos un nuevo módulo al proyecto, este incluya automáticamente dicha declaración. Para ello debemos seleccionar la opción del menú de VB Herramientas+Opciones..., que nos mostrará la ventana de opciones del proyecto (Figura 47)

En el apartado Opciones de código marcaremos Requerir declaración de variables, con lo que cada vez que creemos un nuevo módulo, se añadirá la instrucción antes mencionada.

Si no incluimos Option Explicit, podremos utilizar variables directamente en el código sin necesidad de haberlas declarado previamente.

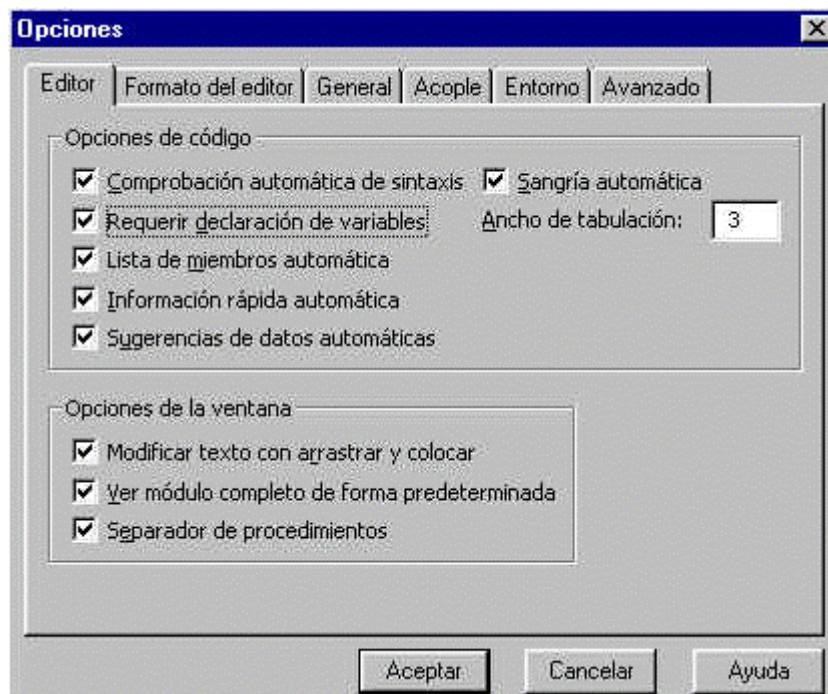


Figura 47. Ventana de Opciones del proyecto.

El Código fuente 13 no provocaría en ese caso un error. Sin embargo corremos el riesgo de que al escribir el nombre de una variable que ya hemos utilizado en el código, si dicho nombre no lo ponemos correctamente, Visual Basic lo tome como una nueva variable, no provocando un error pero originando resultados inesperados en la ejecución del programa, como muestra el Código fuente 14

```
Sub Main()
' asignar a la variable Util
' una cadena de caracteres
Util = "tuerca"

' ahora asignamos a la variable valor
' el contenido de la variable Util,
' pero nos equivocamos al escribir
' el nombre de la variable por Utik,
' por lo que VB la interpreta como una
' nueva variable no funcionando el programa
' como se espera
valor = Utik

End Sub
```

Código fuente 14

Continuación de línea de código

Puede sucedernos que al escribir una línea compleja de código, esta no quepa en la parte visible de la pantalla. En estos casos la recomendación es no escribir superando el margen derecho de la pantalla, puesto que es muy incómodo revisar el código de esta manera, sino poner un carácter de guión bajo (_), que en VB indica continuación de línea y seguir en la línea siguiente. El compilador lo interpretará como si fuera una única línea. El Código fuente 15 nos lo muestra.

```
Sub Main()
Dim Nombre As String
Dim PriApellido As String
Dim SegApellido As String
Dim NomCompleto As String
Nombre = "Pedro"
PriApellido = "Mesa"
SegApellido = "Gutierrez"
' si la línea es demasiado larga
' terminamos en la siguiente
NomCompleto = Nombre & PriApellido & _
SegApellido
End Sub
```

Código fuente 15

Grabación del proyecto

Durante el desarrollo de un programa, es conveniente que cada cierto tiempo grabemos a disco el proyecto sobre el que estamos trabajando. Cada componente del proyecto se grabará en un fichero, de forma que podremos hacer una pausa en nuestra sesión de trabajo, para volver a cargar el proyecto en VB posteriormente, o llevarnos el programa a otro ordenador (que disponga de la misma versión de VB) si fuera necesario.

Cuando hayamos decidido grabar el proyecto, elegiremos la opción del menú de VB Archivo+Guardar proyecto, que nos mostrará una serie de ventanas de diálogo para grabar en distintos tipos de fichero cada uno de los componentes de nuestro proyecto. La Figura 48 corresponde a la grabación del módulo de código, en donde debemos especificar una carpeta de disco y el nombre a asignar al fichero.

En los programas realizados durante este curso manejaremos módulos de código, que se graban en ficheros con la extensión .BAS; formularios, que se graban en ficheros con la extensión .FRM; y el proyecto, que se graba en un fichero con la extensión .VBP. Adicionalmente, cuando tratemos el tema dedicado a clases y objetos, los módulos de clase se grabarán en ficheros con extensión .CLS.

Al iniciar una sesión de trabajo en VB, en el cuadro de diálogo inicial de apertura de proyecto, visto en el apartado Organización del código de este tema (pestaña Existente o Recientes), o mediante la opción Archivo+Abrir proyecto, seleccionaremos el fichero .VBP que contiene el proyecto a abrir. Este fichero almacena toda la información sobre el resto de ficheros que componen el programa, por lo que no precisaremos de indicar la apertura de cada uno, sólo el del proyecto.

También es posible abrir un proyecto sin tener cargado el entorno de VB, sólo es preciso hacer doble clic en el fichero de proyecto sobre el que deseamos trabajar y esto hará que se ponga el IDE de Visual Basic en ejecución y cargue el proyecto seleccionado.

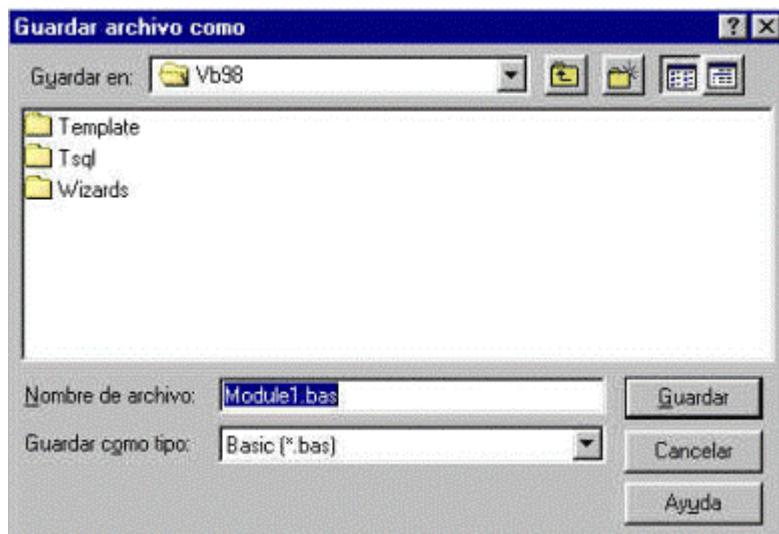


Figura 48. Grabación de un módulo de código del programa.

Seguimiento de la ejecución

En todos los ejemplos realizados hasta el momento, el lector al ejecutarlos sólo ha podido comprobar si el funcionamiento era correcto o no en el caso de que apareciera un mensaje de error. Sin embargo esto no es suficiente información para el programador a la hora de desarrollar una aplicación.

Un lenguaje de programación ha de proporcionar al programador utilidades que le permitan comprobar cómo se ejecuta el código de la aplicación línea por línea, información sobre el contenido de las variables, expresiones, interrupciones en la ejecución, etc. Al conjunto de todas estas técnicas o utilidades que debe aportar un lenguaje se le denomina depuración.

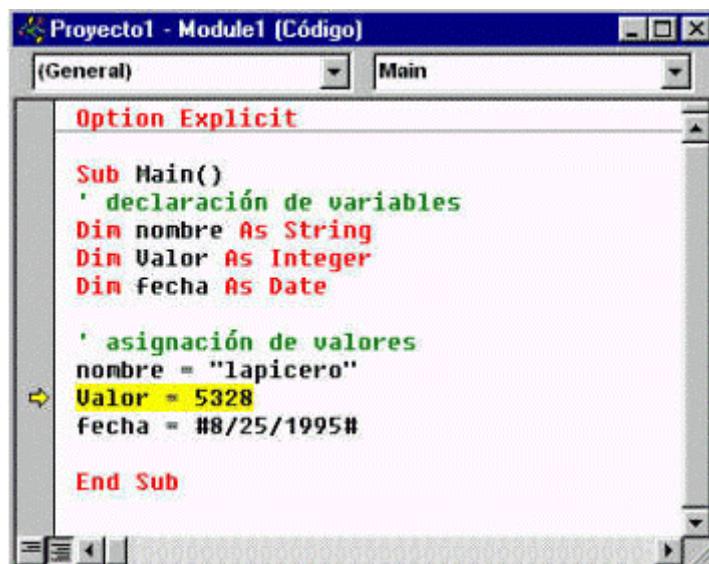
Visual Basic aporta un depurador muy completo del que vamos a utilizar algunas de sus funcionalidades básicas, dado que al ser este un curso enfocado sobre todo a los aspectos del lenguaje y no al desarrollo de elementos visuales, no necesitaremos emplear todos los elementos de depuración disponibles.

A efectos de comprobar como funciona el depurador, crearemos un nuevo proyecto, configurándolo de la forma explicada en anteriores apartados. Una vez agregado el módulo de código al proyecto escribiremos una subrutina Main() con las líneas de código del Código fuente 16.

```
Sub Main()
' declaración de variables
Dim nombre As String
Dim Valor As Integer
Dim fecha As Date
' asignación de valores
nombre = "lapicero"
Valor = 5328
fecha = #8/25/1995#
End Sub
```

Código fuente 16

Grabaremos el proyecto y a continuación elegiremos la opción del menú de VB Depuración+Paso a paso por instrucciones, o pulsaremos F8. Esto hará que comience a ejecutarse el programa en modo de depuración. En dicho modo el programa se sitúa en la primera línea de código, resaltándola con un color especial (por defecto amarillo) y espera a que el programador decida ejecutarla pulsando F8. Al pulsar dicha tecla, se pasa a la siguiente línea esperando también que se ejecute y así sucesivamente. La Figura 49 muestra como el depurador está situado en una línea de código en la que se va a realizar la asignación de valor a una variable.



```

Option Explicit

Sub Main()
    ' declaración de variables
    Dim nombre As String
    Dim Valor As Integer
    Dim Fecha As Date

    ' asignación de valores
    nombre = "lapicero"
    Valor = 5328
    Fecha = #8/25/1995#

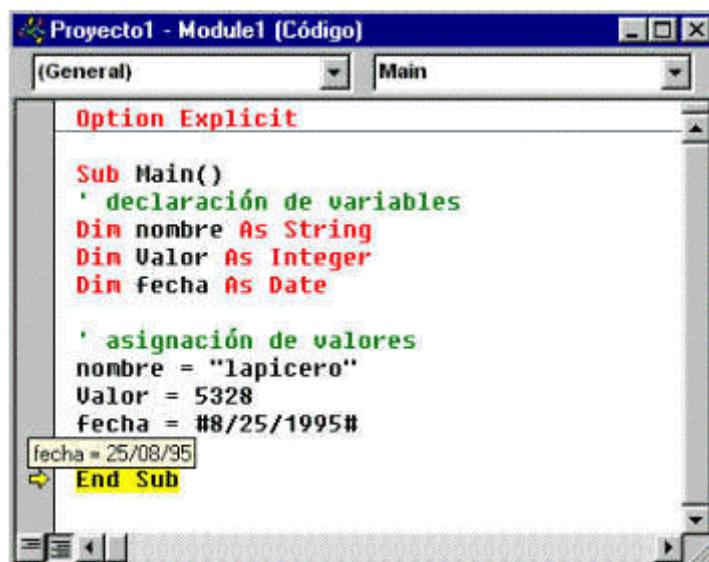
End Sub

```

Figura 49. Programa ejecutándose en modo de depuración.

De este modo podemos controlar paso a paso la ejecución de cada instrucción del programa en el caso de que se produzcan errores o no proporcione los resultados esperados.

Si necesitamos comprobar el valor de una variable o expresión sólo tenemos que situar el cursor del ratón sobre la misma y se mostrará su contenido en una viñeta flotante, como vemos en la Figura 50.



```

Option Explicit

Sub Main()
    ' declaración de variables
    Dim nombre As String
    Dim Valor As Integer
    Dim Fecha As Date

    ' asignación de valores
    nombre = "lapicero"
    Valor = 5328
    Fecha = #8/25/1995#
    fecha = 25/08/95
End Sub

```

Figura 50. Visualización del valor de una variable en una viñeta flotante durante la ejecución.

Cuando el programa sea muy largo y hayamos realizado todas las comprobaciones necesarias, podemos dejar que siga ejecutándose normalmente sin emplear el depurador, seleccionando la opción de menú de VB Ejecutar+Continuar, pulsando la tecla F5 o el botón de la barra de herramientas.



Figura 51. Botón de la barra de herramientas para iniciar o continuar la ejecución del programa.

Para comenzar a ejecutar la aplicación desde el comienzo una vez que ya estamos en mitad de la ejecución, seleccionaremos la opción Ejecutar+Reiniciar del menú de VB o la combinación de teclas Mayús+F5. Si queremos cancelar la ejecución del programa durante el proceso de depuración, disponemos de la opción Ejecutar+Terminar en el menú de VB o el botón de la barra de herramientas.



Figura 52. Botón de la barra de herramientas terminar la ejecución del programa.

Constantes

Al igual que ocurre con las variables, una constante es un elemento del programa que guarda un valor, sin embargo se diferencia de una variable en que dicho valor no podrá cambiar, permaneciendo igual durante la ejecución del programa. Para nombrar una constante emplearemos un identificador. También al igual en las variables, es recomendable declarar una constante antes de utilizarla, no siendo esta una operación obligatoria, pero si queremos que VB exija dicha declaración, situaremos la instrucción Option Explicit en la zona de declaraciones del módulo. Emplearemos la instrucción Const para declarar una constante. En el momento de la declaración deberemos asignarle el valor que va a contener. La forma más simple de declaración es la que muestra el Código fuente 17.

```
Const ARTICULO = "silla"
```

Código fuente 17

Sin embargo, este tipo de declaración crea una constante de tipo Variant. En el caso de que necesitemos especificar el tipo de dato de la constante, deberemos hacerlo como se muestra en el Código fuente 18, después del nombre.

```
Const ARTICULO As String = "silla"
```

Código fuente 18

Al intentar asignar un nuevo valor a una constante se producirá un error, como podemos ver en la Figura 53.

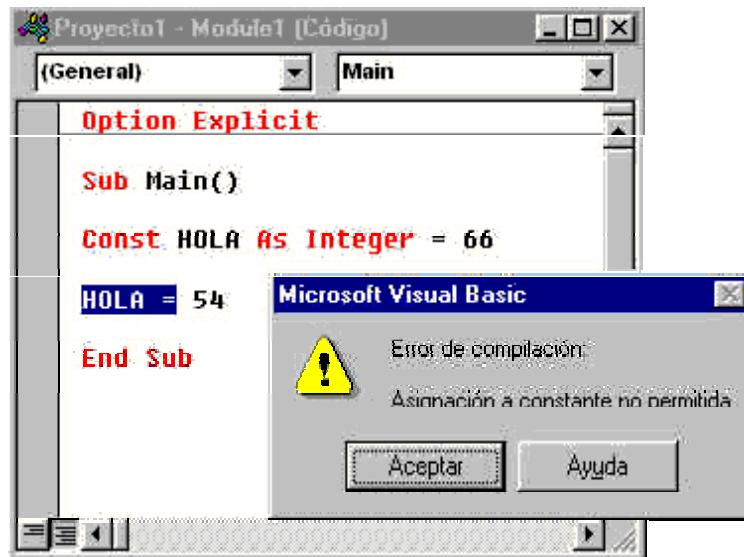


Figura 53. Error de asignación a constante.

Entradas y salidas de información

Todo lenguaje de programación debe disponer de un medio de establecer comunicación con el usuario, lo que comúnmente se denomina interfaz entre el programa y usuario.

En los lenguajes tradicionales para modo texto, las entradas y visualización de datos son muy simples, limitándose a un cursor indicativo de introducción de datos y cadenas de caracteres mostradas en pantalla a modo de mensaje. El trabajo de creación de una interfaz elegante y más funcional corre por cuenta del programador.

Cuando programamos, sin embargo, para un sistema operativo gráfico, las cosas cambian radicalmente, ya que es el propio sistema el que aporta la interfaz, ahorrando al programador esa fase del desarrollo.

En el caso de VB, disponemos de dos funciones del lenguaje que se ocupan de estas tareas: una es MsgBox() para mostrar mensajes al usuario, y la otra es InputBox(), para realizar la captura de información.

A pesar de que nos referimos a estas funciones como básicas, no se lleve el lector a engaño, aportan una gran funcionalidad y flexible interfaz, para el poco código que precisan. Claro está, siempre que necesitemos una interfaz más elaborada, deberemos programarla por nuestra cuenta.

MsgBox()

Esta función visualiza una caja de diálogo con un mensaje informativo, a la que el usuario deberá responder pulsando en uno de los botones que aparezcan en dicha caja. Podemos adicionalmente, recoger el valor de retorno de esta función, que indica mediante un Integer el botón pulsado.

Sintaxis:

MsgBox(Mensaje, Botones, Título, FichAyuda, Contexto) ---> Botón

- **Mensaje.** Cadena de caracteres con el texto informativo para el usuario, si queremos organizar el texto en varias líneas, podemos incluir varias cadenas con un retorno de carro y avance de línea utilizando la constante de VB vbCrLf entre cada cadena.
- **Botones.** Número o suma de números que identifican a los botones e icono que aparecen en la caja. En lugar de los números, podemos utilizar un grupo de constantes propias de VB, con lo que el código ganará en claridad. Lo vemos en la Tabla 2.

Constante	Tipo de botón, icono y mensaje
vbOKOnly	Botón Aceptar.
vbOKCancel	Botones Aceptar y Cancelar.
vbAbortRetryIgnore	Botones Anular, Reintentar e Ignorar.
vbYesNoCancel	Botones Sí, No y Cancelar.
vbYesNo	Botones Sí y No.
vbRetryCancel	Botones Reintentar y Cancelar.
vbCritical	Icono de mensaje crítico.
vbQuestion	Icono de pregunta.
vbExclamation	Icono de advertencia.
VbInformation	Icono de información.
vbDefaultButton1	Primer botón es el predeterminado.
vbDefaultButton2	Segundo botón es el predeterminado.
vbDefaultButton3	Tercer botón es el predeterminado.
vbDefaultButton4	Cuarto botón es el predeterminado.
VbApplicationModal	Mensaje de aplicación modal; el usuario debe responder al mensaje antes de poder continuar con la aplicación.
vbSystemModal	Mensaje modal de sistema; el usuario debe responder al mensaje antes de poder continuar con cualquier aplicación.

Tabla 2. Constantes disponibles para los botones de la función MsgBox().

- **Título.** Cadena de caracteres con el título del mensaje, si se omite se utiliza el nombre de la aplicación.
- **FichAyuda.** Cadena de caracteres con el nombre del fichero de ayuda para el mensaje.
- **Contexto.** Número que contiene el contexto de ayuda en el fichero especificado en FichAyuda.
- **Botón.** Número que contiene el valor de retorno del botón pulsado en la caja de mensaje. Los valores están disponibles en forma de constantes que podemos ver en la Tabla 3.

Constante	Botón pulsado
VbOK	Aceptar
vbCancel	Cancelar
VbAbort	Anular
VbRetry	Reintentar
vbIgnore	Ignorar
VbYes	Sí
VbNo	No

Tabla 3. Valores devueltos por MsgBox().

El modo de llamar dentro del código de un programa a esta o cualquier otra subrutina o función, pasa por escribir su nombre seguido de los valores que necesitemos en los diferentes parámetros. Si no necesitamos recuperar el valor que devuelve la función, correspondiente al botón de la caja pulsado, utilizaremos MsgBox() como indica el Código fuente 19.

```
Sub Main()
MsgBox "Introduzca un valor", vbOKOnly, "Atención"
End Sub
```

Código fuente 19

El resultado lo muestra la Figura 54.

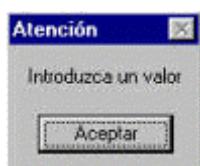


Figura 54. Resultado de la función MsgBox().

En el caso de mostrar varios botones y comprobar si se ha pulsado uno en concreto, lo haremos como se indica en el Código fuente 20.

```
Sub Main()
    ' declaramos una variable para
    ' guardar el valor del botón pulsado
    Dim Respuesta As Integer
    ' asignamos a la variable el resultado de MsgBox()
    Respuesta = MsgBox("Introduzca un valor", vbYesNoCancel + vbCritical, _ "Atención")
    ' en función del valor de respuesta de MsgBox()
    ' actuamos de una forma u otra
    If Respuesta = vbCancel Then
        MsgBox "Se ha pulsado la tecla 'Cancelar'", , "Información"
    End If
End Sub
```

Código fuente 20

InputBox()

Esta función muestra una caja de diálogo en la que existe un TextBox para que el usuario introduzca un valor, devolviendo una cadena con el resultado la acción tomada por el usuario: si pulsa el botón *Aceptar* o *Enter*, el valor de retorno será el contenido del TextBox de la caja de diálogo. Si pulsa el botón *Cancelar* o *Escape*, se devolverá una cadena vacía.

```
InputBox(Texto, Titulo, TextoDefecto, XPos, YPos, FicheroAyuda,
ContextoAyuda) ---> Cadena
```

- **Texto.** Cadena de caracteres con un texto informativo para el usuario. Sigue las mismas normas que para la función MsgBox().
- **Titulo.** Cadena de caracteres con el título para la ventana.
- **TextoDefecto.** Cadena con el texto que aparecerá en el TextBox. Si no se proporciona, el control aparecerá vacío.
- **XPos, YPos.** Coordenadas de situación de la ventana.
- **FicheroAyuda.** Nombre del fichero de ayuda asociado a esta ventana.
- **ContextoAyuda.** Número con el identificador del tema de ayuda.
- **Cadena.** Cadena de caracteres escrita por el usuario o vacía, según el modo de salida de la ventana.

Veamos un ejemplo de uso de esta función en el Código fuente 21.

```
Sub Main()
Dim Valor As String
```

```
Valor = InputBox("Introduzca nombre completo", "Toma de datos")
End Sub
```

Código fuente 21

El resultado, en la Figura 55.



Figura 55. Función InputBox() en ejecución.

Operadores

Los operadores son aquellos elementos del lenguaje que permiten combinar varios identificadores o elementos de código, como números y cadenas de caracteres (operando), para producir un resultado. Al resultado obtenido de tal combinación se le denomina expresión. Los operadores se clasifican en varias categorías en función del tipo de expresión a construir.

Aritméticos

Realizan el conjunto habitual de operaciones matemáticas. Como norma general, si uno de los operandos es Null, el resultado de la operación será Null. También en la mayor parte de los casos, si una expresión es Empty, se evaluará como 0.

^ Potencia

Para realizar una potenciación, hemos de situar a la izquierda el operando que actúe como base y a la derecha el exponente. El resultado puede ser negativo si el exponente es un entero. Podemos realizar varios cálculos de potencia en una sola instrucción, siendo evaluados de izquierda a derecha. El tipo de dato devuelto por la operación es Double.

```
Resultado = 15 ^ 3      ' 3375
```

Código fuente 22

* Multiplicación

El tipo de dato devuelto por una multiplicación se adapta al que sea más aproximado al resultado, con las siguientes variaciones:

- Si la multiplicación es entre un Single y un Long, el resultado es un Double.
- Cuando el resultado es un Variant con valor Long, Single o Date, que se sale de rango, se convierte a Variant con valor Double.
- Cuando el resultado es un Variant con valor Byte, que se sale de rango, se convierte a Variant con valor Integer.
- Cuando el resultado es un Variant con valor Integer, que se sale de rango, se convierte a Variant con valor Long.

```
Resultado = 44 * 5      ' 220
```

Código fuente 23

/ División real

Divide dos números y devuelve un resultado con precisión decimal.

```
Resultado = 428 / 17      ' 25,1764705882353
```

Código fuente 24

El tipo de dato resultante es Double excepto en los siguientes casos:

- Si ambos operandos son Byte, Integer o Single, resulta un Single, provocándose un error si excede su rango válido.
- Si ambos operandos son Variant con valor Byte, Integer o Single, resulta un Single. En el caso de que exceda su rango válido será un Variant con valor Double.
- Cuando sea un Decimal y otro tipo, el resultante será Decimal.

\ División entera

Divide dos números y devuelve un resultado con valor entero.

```
Resultado = 428 \ 17      ' 25
```

Código fuente 25

Antes de hacer la división, se redondean los números para convertirlos a Byte, Integer o Long, estos mismos tipos serán los resultantes de la operación, o un Variant con el valor de esos tipos. La parte fraccionaria de los números se trunca.

Mod

Devuelve el resto de la división de dos números.

```
Resultado = 428 Mod 17      ' 3
```

Código fuente 26

El resultado es un tipo Byte, Integer o Long; o un Variant con el valor de esos tipos.

+ Suma

Aparte de efectuar una suma de números, permite incrementar una fecha en una cantidad determinada de días o concatenar cadenas de caracteres. Para esto último se recomienda usar el operador **&**, de manera que en un rápido vistazo al código sepamos si estamos sumando o concatenando.

```
' con números:  
Resultado = 2533 + 431      ' 2964  
' con fechas:  
Fecha = #3/15/97#      ' 15/03/97  
Fecha = Fecha + 5      ' 20/03/97
```

Código fuente 27

Cuando una de las expresiones a sumar no es Variant, se producen los siguientes resultados:

- La suma se produce cuando ambas partes son números, o una es Variant.
- La concatenación se produce cuando ambas partes son String, o una es Variant.
- Si un operando es Variant Empty, devuelve el resto de la expresión sin modificar.
- Si un operando es numérico y el otro String se produce un error.
- Cuando las expresiones a sumar son Variant, se aplica lo siguiente:
- Se produce una suma cuando ambos operandos contienen números, o uno es numérico y el otro cadena de caracteres.

Por norma general, en la suma de números, el tipo del resultado es el del operando con mayor precisión, siendo este orden de precisión: Byte, Integer, Long, Single, Double, Currency y Decimal. En este punto se dan una serie de excepciones:

- El resultado es Double al sumar un Single y un Long.
- Se produce una conversión a Variant con valor Double si el resultado es Long, Single o Date con valor Variant que excede su rango.

- Se produce una conversión a Variant con valor Integer si el resultado es Byte con valor Variant que excede su rango.
- Se produce una conversión a Variant con valor Long si el resultado es Integer con valor Variant que excede su rango.

Al sumar a un tipo Date siempre se obtiene un Date.

Resta

Realiza una resta entre dos números o la negación de un número.

```
' resta normal
Resultado = 100 - 25      ' 75
' negación
Num1 = 84
Num2 = -Num1      ' -84
```

Código fuente 28

Los tipos de datos resultantes siguen las mismas reglas que en la suma, dándose el caso adicional de que al restar dos tipos Date, el resultado es Double.

Comparación

Estos operadores permiten comprobar el nivel de igualdad o diferencia existente entre los operandos de una expresión. El resultado obtenido será un valor lógico, True (Verdadero) o False (Falso). Si alguno de los elementos de la comparación es Null, el resultado será Null.

Cuando realicemos comparaciones entre cadenas de caracteres, podemos configurar el modo en que se realiza internamente empleando el Código fuente 29 en la zona de declaraciones del módulo.

```
Option Compare <TipoComparacion>
```

Código fuente 29

El valor de TipoComparacion puede ser uno de los siguientes.

- **Binary.** Valor por defecto. Realiza una comparación basada en los valores binarios de los caracteres. En este caso, una expresión como "a" = "A" resultaría falsa.
- **Text.** Realiza una comparación en base a los propios caracteres sin distinguir entre mayúsculas y minúsculas. Ahora la expresión "a" = "A" sería verdadera.
- **Database.** Sólo se puede usar con Access. Las comparaciones estarán basadas en el orden de los datos.

La Tabla 4 muestra los operadores de este tipo disponibles y su resultado al ser aplicados en expresiones.

Operador	Resultado True	Resultado False
< (Menor que)	Operando1 < Operando2	Operando1 >= Operando2
<= (Menor o igual que)	Operando1 <= Operando2	Operando1 > Operando2
> (Mayor que)	Operando1 > Operando2	Operando1 <= Operando2
>= (Mayor o igual que)	Operando1 >= Operando2	Operando1 < Operando2
= (Igual a)	Operando1 = Operando2	Operando1 <> Operando2
<> (Distinto de)	Operando1 <> Operando2	Operando1 = Operando2

Tabla 4. Operadores de comparación.

En el Código fuente 30 se muestran varios ejemplos de comparaciones.

```

Sub Main()
Dim Resultado As Boolean

Resultado = "hola" < "otra cadena"      ' Verdadero
Resultado = 550 <= 300      ' Falso
Resultado = 5 > 2      ' Verdadero
Resultado = 87 >= 22      ' Verdadero
Resultado = "larga" = "corta"      ' Falso
Resultado = 342 <> 65      ' Verdadero
End Sub
' -----
Sub Main()
' definimos una variable variant y otra numérica
' y hacemos dos pruebas de comparación
Dim Resultado As Boolean
Dim Numero As Variant
Dim Num As Integer
' si el Variant se puede convertir a numérico...
Numero = "520"
Num = 2380
Resultado = Num > Numero      ' Verdadero
' si el Variant no se puede convertir a numérico...
Numero = "hola"
Num = 2380
Resultado = Num > Numero      ' aquí se produce un error
End Sub

```

Código fuente 30

El modo de comparación entre valores que no son de tipo Variant son los siguientes:

- Se realiza una comparación numérica si las dos expresiones son de tipo numérico; Byte, Integer, Double, etc., o una es un Variant con valor numérico.
- Se realiza una comparación de cadenas si las dos expresiones son String, o una es un Variant.
- Se produce un error de tipos cuando una expresión es numérica y la otra es Variant de cadena de caracteres que no puede convertirse a número.
- Con una expresión numérica y otra Empty, se realiza una comparación numérica dando a Empty el valor 0.
- Con un String y un Empty, se realiza una comparación de cadenas dando a Empty el valor de cadena de longitud cero.

Si ambas expresiones son Variant, dependerá del tipo interno de la expresión, quedando las comparaciones de la siguiente manera:

- Hay comparación numérica si ambas expresiones son numéricas.
- Hay comparación de cadenas si ambas expresiones son cadenas de caracteres.
- Cuando una expresión es numérica y la otra una cadena, la expresión numérica es menor que la de cadena.
- Si una expresión es Empty y la otra numérica, se realiza comparación numérica dando a Empty el valor 0.
- Si una expresión es Empty y la otra una cadena, se realiza comparación de cadenas dando a Empty el valor de cadena de longitud cero.
- Cuando las dos expresiones son Empty el resultado es igual.

En comparaciones numéricas: entre un Single y un Double, se redondea a Single. Entre un Currency y un Single o Double, se convierte el último a Currency. Al comparar un Decimal con un Single o Double, el último se convierte a Decimal.

Hasta aquí hemos visto los operadores de comparación habituales, existen además dos operadores adicionales de comparación que poseen algunas variantes que veremos a continuación.

Is

Compara dos variables que contienen un objeto, si las dos hacen referencia al mismo objeto se devuelve True, en caso contrario se devuelve False.

```
Resultado = ObjetoA Is ObjetoB
```

Código fuente 31

Like

Compara dos cadenas de caracteres basándose en un patrón. Si la cadena en la que se efectúa la búsqueda coincide con el patrón se devuelve True, en caso contrario se devuelve False.

```
Resultado = CadenaDeBusqueda Like Patron
```

Los resultados de este operador dependen en gran medida de la instrucción Option Compare vista anteriormente. El patrón de búsqueda puede estar compuesto por caracteres normales y los caracteres especiales de la Tabla 5.

Carácter del patrón	Coincidencia en cadena a buscar
?	Cualquier carácter
*	Ninguno o varios caracteres
#	Un dígito (0-9)
[ListaCaracteres]	Cualquier carácter de la lista
[!ListaCaracteres]	Cualquier carácter que no esté en la lista

Tabla 5. Caracteres usados en el patrón de Like.

Si queremos realizar una búsqueda de los propios caracteres especiales, hay que incluirlos dentro de corchetes. Para buscar un carácter de corchete de cierre es necesario incluirlo sólo en el patrón. Podemos establecer un rango de búsqueda en la lista de caracteres, incluyendo un guión "-" entre los valores inferior y superior; este rango ha de incluirse en orden ascendente, de menor a mayor. Para buscar el propio carácter de exclamación "!", no hay que incluirlo entre corchetes. Para buscar el carácter de guión "-", se debe poner al principio o final de la lista de caracteres. Veamos en el Código fuente 32 un procedimiento de ejemplo con varias líneas de código que incluyen este tipo de operación.

```
Sub Main()
Dim Resultado As Boolean
Resultado = "B" Like "[A-M]" ' Verdadero
Resultado = "D" Like "[!A-M]" ' Falso
Resultado = "Z" Like "[!A-M]" ' Verdadero
Resultado = "hola amigos" Like "h*s" ' Verdadero
Resultado = "Es25s" Like "Es##s" ' Verdadero
Resultado = "Elefante" Like "Elef?nte" ' Verdadero
Resultado = "Ella tiene 2 hojas" Like "*# [f-i]*" ' Verdadero
Resultado = "SUPERIOR" Like "S?E*" ' Falso
End Sub
```

Código fuente 32

Concatenación

La concatenación es la operación por la cual dos cadenas de caracteres se unen formando una única cadena. Los operadores utilizados son "&" y "+", pero se recomienda utilizar exclusivamente "&" para evitar ambigüedades a la hora de revisar el código.

```
CadenaUnida = CadenaA & CadenaB
```

Si una de las dos cadenas no es un tipo String, se hace una conversión a Variant con valor String dando también como resultado Variant con String. Si alguna de las expresiones es Null o Empty se considera cadena de longitud cero. En el Código fuente 33 podemos ver algunas de las posibilidades de concatenación de valores.

```
Sub Main()
Dim Resultado As String
Dim Num As Integer
Dim Cambia As Variant
' vamos a concatenar cadenas sencillas
' atención al espacio en blanco entre cadenas,
' es importante tenerlo en cuenta
Resultado = "hola" & "amigos" ' "holaamigos"
Resultado = "hola " & "amigos" ' "hola amigos"
' vamos a concatenar una cadena y un número
Num = 457
Resultado = "el número " & Num ' "el número 457"
' el contenido de Cambia es Empty
Resultado = "el valor " & Cambia ' "el valor "
Cambia = Null ' asignamos Null a Cambia
Resultado = "el valor " & Cambia ' "el valor "
End Sub
```

Código fuente 33

Lógicos

Una expresión lógica es aquella que retorna un valor de tipo Boolean (True o False), en función de una condición establecida entre los operandos que forman parte de la expresión. Si la expresión se realiza entre elementos numéricos, es posible establecer una comparación a nivel de bit, es decir, se comparan los bits de las mismas posiciones de los dos números que componen los operandos de la expresión, obteniendo un valor numérico como resultado de la expresión.

And

Realiza una conjunción entre dos expresiones. La Tabla 6 muestra los valores que pueden tomar las expresiones y el resultante de la operación.

```
Resultado = ExpresiónY And ExpresiónZ
```

ExpresiónY	ExpresiónZ	Resultado
True	True	True

True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

Tabla 6. Valores resultantes del operador And.

En el Código fuente 34 vemos unos ejemplos.

```
Sub Main()
Dim Resultado As Boolean
Resultado = 2 > 1 And "hola" = "hola"      ' True
Resultado = 2 = 1 And "hola" = "hola"      ' False
Resultado = 2 < 1 And "hola" > "hola"      ' False

End Sub
```

Código fuente 34

En comparaciones a nivel de bit, And devuelve los resultados que aparecen en la Tabla 7.

Bit de Expresión Y	Bit de Expresión Z	Resultado
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 7. Valores resultantes del operador And a nivel de bit.

Eqv

Efectúa una equivalencia entre dos expresiones.

```
Resultado = ExpresionY Eqv ExpresionZ
```

La Tabla 8 muestra los valores que pueden tomar las expresiones y el resultante de la operación, en el Código fuente 35 vemos unos ejemplos.

ExpresiónY	ExpresiónZ	Resultado
True	True	True
True	False	False
False	True	False
False	False	True

Tabla 8. Valores resultantes del operador Eqv.

```
Sub Main()
Dim Resultado As Boolean
Resultado = 75 > 21 Eqv 63 > 59      ' True
Resultado = 21 > 75 Eqv 63 > 59      ' False
End Sub
```

Código fuente 35

En comparaciones a nivel de bit, Eqv devuelve los resultados que vemos en la Tabla 9.

Bit de ExpresiónY	Bit de ExpresiónZ	Resultado
0	0	1
0	1	0
1	0	0
1	1	1

Tabla 9. Valores resultantes del operador Eqv a nivel de bit.

Imp

Realiza una implicación entre dos expresiones.

Resultado = ExpresiónY Imp ExpresiónZ

La Tabla 10 muestra los valores que pueden tomar las expresiones y el resultante de la operación.

ExpresiónY	ExpresiónZ	Resultado
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Tabla 10. Valores resultantes del operador Imp.

En el Código fuente 36 podemos ver unos ejemplos.

```
Sub Main()
Dim Resultado As Boolean
Resultado = 50 > 30 Imp 64 > 56      ' True
Resultado = 22 > 88 Imp 42 > 90      ' True
Resultado = 31 > 20 Imp 26 > 85      ' False
End Sub
```

Código fuente 36

En comparaciones a nivel de bit, Imp devuelve los resultados que aparecen en la Tabla 11

Bit de ExpresiónY	Bit de ExpresiónZ	Resultado
0	0	1
0	1	1

1	0	0
1	1	1

Tabla 11. Valores resultantes del operador Imp a nivel de bit.

Not

Realiza una negación sobre una expresión.

Resultado = Not Expresión

La Tabla 12 muestra los valores de la expresión y el resultante de la operación.

Expresión	Resultado
True	False
False	True
Null	Null

Tabla 12. Valores resultantes del operador Not.

Vemos un ejemplo en el Código fuente 37.

```

Sub Main()
Dim Resultado As Boolean
Dim ValorOperacion As Boolean

' evaluar una expresión con resultado verdadero
ValorOperacion = 35 < 70      ' True

' negar el resultado de la anterior operación,
' con lo que verdadero pasará a falso
Resultado = Not ValorOperacion  ' False

' evaluar una expresión con resultado falso
ValorOperacion = 35 > 70      ' False

' negar el resultado de la anterior operación,
' con lo que falso pasará a verdadero
Resultado = Not ValorOperacion  ' True

End Sub

```

Código fuente 37

En comparaciones a nivel de bit, Not devuelve los resultados que vemos en la Tabla 13

Bit de Expresión	Resultado
0	1
1	0

Tabla 13. Valores resultantes del operador Not a nivel de bit.

Or

Realiza una disyunción entre dos expresiones.

```
Resultado = ExpresionY Or ExpresionZ
```

La Tabla 14 muestra los valores que pueden tomar las expresiones y el resultante de la operación.

ExpresiónY	ExpresiónZ	Resultado
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

Tabla 14. Valores resultantes del operador Or.

En el Código fuente 38 podemos ver un ejemplo

```
Sub Main()
Dim Resultado As Boolean
Resultado = 21 = 21 Or 65 < 12      ' True
Resultado = 21 <> 21 Or 65 < 12    ' False
Resultado = 21 > 21 Or 65 > 12    ' True
End Sub
```

Código fuente 38

En comparaciones a nivel de bit, Or devuelve los resultados que aparecen la Tabla 1

Bit de ExpresiónY	Bit de ExpresiónZ	Resultado
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 15. Valores resultantes del operador Or a nivel de bit.

Xor

Realiza una exclusión de dos expresiones.

Resultado = ExpresiónY Xor ExpresiónZ

Cuando una de las expresiones es Null, el resultado es Null. Para el resto de casos el resultado se ajusta a la Tabla 16 .

ExpresiónY	ExpresiónZ	Resultado
True	True	False
True	False	True
False	True	True
False	False	False

Tabla 16. Valores resultantes del operador Xor.

Vemos unos ejemplos en el Código fuente 39.

```
Sub Main()
Dim Resultado As Boolean
Resultado = 48 > 20 Xor 50 > 17      ' False
Resultado = 48 > 20 Xor 50 < 17      ' True
Resultado = 48 < 20 Xor 50 < 17      ' False
End Sub
```

Código fuente 39

En comparaciones a nivel de bit, Xor devuelve los resultados que aparecen en la Tabla 17.

Bit de Expresión Y	Bit de Expresión Z	Resultado
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 17. Valores resultantes del operador Xor a nivel de bit.

Prioridad de operadores

En una línea de código que contenga varias operaciones, estas se resolverán en un orden determinado conocido como prioridad de operadores.

Cada grupo de operadores tiene su propia prioridad, mientras que entre diferentes categorías de operadores, también existen niveles de prioridad, de tal forma que en una expresión que tenga operadores de distinta categoría, se evaluarán en primer lugar los operadores aritméticos, seguirán los de comparación y finalmente los lógicos.

Es posible alterar el orden natural de prioridades empleando paréntesis, que separan los elementos que deseamos sean resueltos en primer lugar, pero debemos tener en cuenta que dentro de los paréntesis se seguirá manteniendo la prioridad de la forma antes explicada.

Los operadores de comparación mantienen igual prioridad, resolviéndose en orden de aparición, de izquierda a derecha.

Los operadores aritméticos y lógicos se ajustan a la prioridad indicada en la Tabla 18.

Aritméticos	Lógicos
Potenciación (^)	Not
Negación (-)	And
Multiplicación y división (*, /)	Or
División entera (\)	Xor
Resto de división (Mod)	Eqv

Suma y resta (+, -)	Imp
---------------------	-----

Tabla 18. Prioridad de operadores.

El operador de concatenación (**&**) es el que menor prioridad tiene. El Código fuente 40 muestra algunos ejemplos de prioridad normal y alterada mediante el uso de paréntesis.

```

Sub Main()
Dim TotalOp As Integer
Dim Resultado As Boolean

' prioridad aritmética
TotalOp = 5 * 2 + 3    '13
TotalOp = 5 * (2 + 3)   ' 25

' prioridad aritmética y comparación
' en esta línea el resultado es verdadero ya que
' se efectúa en primer lugar la operación aritmética
' y después la comparación
Resultado = 5 * 7 > 10

' en esta línea el resultado es falso porque
' el paréntesis obliga a realizar la comparación antes
' de la operación aritmética
Resultado = 5 * (7 > 10)

End Sub

```

Código fuente 40

Modelos de programación

Podemos definir un modelo de programación, como la técnica empleada para ejecutar el código de un programa. Existen diversos modelos de programación que se describen a continuación.

Programación lineal

Este es el modelo más tradicional y ya está en desuso. El código se organiza linealmente, y se ejecuta desde la primera línea hasta la última sin existir bifurcaciones.

Programación estructurada

Las técnicas de programación estructurada, se basan en el análisis del problema a resolver, separándolo en partes más pequeñas que puedan ser resueltas más fácilmente y programadas en fragmentos de código conocidos como subrutinas o procedimientos. A este tipo de programación también se le conoce como programación procedural.

Un programa de este tipo debe disponer de un procedimiento inicial o de arranque, desde el cual se realicen llamadas al resto de procedimientos de la aplicación. De igual forma, dentro de un procedimiento se podrá llamar a otro procedimiento existente y así sucesivamente.

Finalizado un procedimiento, el flujo de la ejecución volverá al procedimiento que hizo la llamada, continuando en la línea de código siguiente a dicha llamada. En este tipo de programas, la ejecución del código se realiza en el orden establecido por las llamadas entre los procedimientos de la aplicación.

La ventaja de separar en subrutinas independientes la resolución de los problemas del programa, nos proporciona una mejor organización del código y evita igualmente el código redundante.

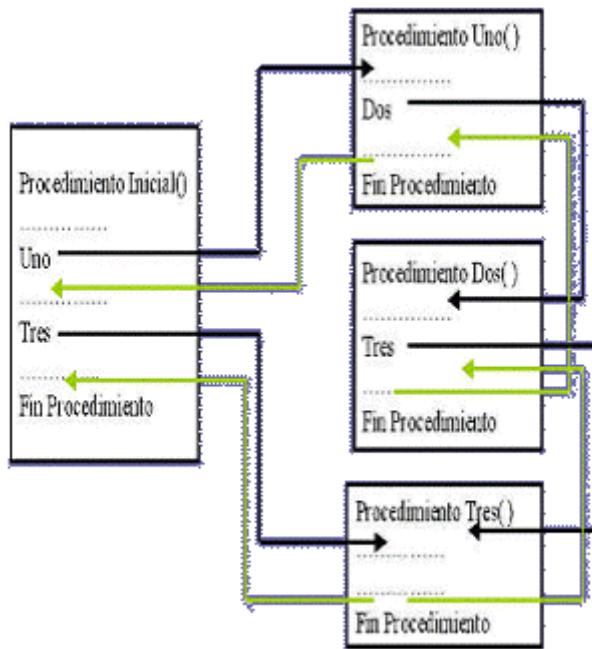


Figura 56. Sistema de llamadas entre procedimientos.

Programación basada en eventos

La programación basada en eventos o conducida por eventos (event-driven), se fundamenta en la ejecución del código en función de las acciones que tome el usuario de la aplicación; de mensajes enviados por el sistema al programa; de mensajes enviados por otros programas al nuestro o por la misma aplicación notificándose sus propios mensajes. Esto quiere decir que el orden de ejecución del código no sigue una pauta fija, por lo que cada vez que se ejecute el programa, el código podrá ejecutarse en una secuencia distinta.

En un sistema basado en ventanas, la programación basada en eventos es fundamental, puesto que el usuario no es probable que ejecute la aplicación de forma idéntica cada vez, ni tome las mismas decisiones en cada ejecución. De igual forma, hay un gran número de sucesos que pueden ocurrir en el transcurso de un programa, y para buena parte de ellos no necesitaremos proporcionar código. La programación por eventos nos permite escribir el código para los eventos que necesiten nuestra respuesta al usuario y dejar el comportamiento por defecto para los que no necesitemos que realicen ninguna tarea. De esta forma podemos escribir el código para la pulsación de un botón, la selección de una opción de menú, etc.

Tanto en programación procedural como por eventos, las estructuras de control son otro elemento clave del lenguaje, y nos van a permitir decidir que partes de código ejecutar según el cumplimiento o no de ciertas condiciones.

A continuación veremos en que consisten los procedimientos y los diferentes tipos existentes; las estructuras de control se tratarán en un apartado posterior.

Procedimientos

Como hemos podido comprobar por los ejemplos realizados, el código de una aplicación se incluye en entidades denominadas procedimientos. Un procedimiento es un conjunto de líneas de código, que reconoceremos del resto de los existentes en el programa mediante un nombre o identificador. El comienzo y fin del mismo se indica mediante una palabra reservada del lenguaje. A las instrucciones que componen el procedimiento se les denomina cuerpo del procedimiento. Veamos a continuación, los principales tipos de procedimiento.

Sub.

Sintaxis:

```
[Private|Public] [Static] Sub NombreProc([ListaParámetros])
[código]
[Exit Sub]
[código]
End Sub
```

- Sub...End Sub. Indican el comienzo y final del procedimiento respectivamente.
- NombreProc. Identificador utilizado para reconocer el procedimiento en el código del programa.
- ListaParámetros. Una o más variables, separadas por comas, que representan la información que recibe el procedimiento al ser llamado. Cada variable pasada como parámetro tiene la siguiente sintaxis:

```
[Optional] [ByVal|ByRef] [ParamArray] NombreVar [()] [As Tipo]
[= ValorPredet]
```

- Optional. Indica que no es necesario que el parámetro tenga valor. Una vez utilizado, el resto de parámetros también deberán ser opcionales.
- ByVal. El parámetro es pasado por valor.
- ByRef. Predeterminado. El parámetro es pasado por referencia. Las cuestiones de paso por valor y referencia se explicarán más adelante.
- ParamArray. Se debe usar como último parámetro de la lista, indica que el parámetro es un array de tipo Variant, lo que permite pasar un número variable de datos al procedimiento. Esta opción no puede utilizarse junto con las tres anteriores. El tratamiento de arrays se verá en un próximo apartado.
- NombreVar. Nombre de la variable que actúa como parámetro.
- Tipo. Tipo de dato del parámetro.

- ValorPredet. Si el parámetro se establece como Optional, es posible establecer un valor por defecto para el caso de que no se le pase valor.
- Exit Sub. Provoca la salida del procedimiento sin llegar a End Sub. El flujo del programa continuará en la línea siguiente a la que llamó a este procedimiento. Es posible incluir más de una instrucción de este tipo en el cuerpo del procedimiento.

Las palabras reservadas Private y Public serán explicadas cuando tratemos el ámbito de procedimientos.

Para ejecutar un procedimiento en una aplicación, sólo hemos de escribir el nombre de dicho procedimiento y los valores a pasar como parámetros en el caso de que tenga. Al llegar el flujo del programa al nombre del procedimiento, saltará a él y comenzará a ejecutarlo. Una vez finalizada la ejecución del procedimiento llamado, se devolverá el control del programa al procedimiento llamador y este continuará la ejecución a partir de la línea siguiente a la que llamó al procedimiento. Veamos un ejemplo en el Código fuente 41.

```
Sub Main()
' declarar variables
Dim Uno As Integer
Dim Dos As Integer
Dim Resultado As Integer
' asignar valores
Uno = 8
Dos = 445
' llamar al procedimiento
Prueba
' después de la llamada
' realizar una operación
' con las variables
Resultado = Uno + Dos
End Sub
' -----
Sub Prueba()
' al llamar a este procedimiento
' se ejecuta la siguiente línea
MsgBox "Estamos situados en el procedimiento Prueba()", , ""
End Sub
```

Código fuente 41

Un parámetro consiste en una información, que puede ser una variable, un valor o una expresión, que se envía a un procedimiento. En la declaración del procedimiento, deberá existir el nombre de una variable en la cual se copiará el dato enviado, utilizándose dicha variable para realizar las operaciones necesarias. Aunque no es obligatorio, si es muy conveniente especificar el tipo de dato del parámetro en la declaración del procedimiento, a efectos de conseguir un mejor rendimiento en ejecución.

Si necesitamos utilizar más de un parámetro, en la declaración del procedimiento especificaremos los diferentes parámetros separados por comas, y en la llamada al procedimiento se escribirán los valores a pasar también separados por comas. Cada valor en la llamada al procedimiento, se copiará en el correspondiente parámetro según el orden que ocupe en la lista.

En el Código fuente 42, veremos un procedimiento en el que se han definido parámetros y cómo son pasados desde el procedimiento llamador.

```

Sub Main()

    ' declarar variables
    Dim Uno As Integer
    Dim Dos As Integer

    ' asignar valores
    Uno = 8
    Dos = 445

    ' llamar al procedimiento
    ' podemos pasar como parámetro una
    ' variable o un valor directamente
    Calcular Dos, 25

End Sub

' -----
Sub Calcular(NumPrincipal As Integer, NumRestar As Integer)

    Dim Valor As Integer

    ' realizar operación con los parámetros
    ' obtener un resultado
    Valor = NumPrincipal - NumRestar

End Sub

```

Código fuente 42

En este fuente, al hacer la llamada al procedimiento `Calcular()`, se copian los valores pasados desde el procedimiento llamador a los parámetros del procedimiento llamado. Una vez que se comienza a ejecutar `Calcular()`, los parámetros se manipulan como una variable normal.

Al terminar de escribir el nombre de un procedimiento, si este tiene parámetros, será mostrada al programador una viñeta flotante informativa, con el formato del procedimiento, como vemos en la Figura 57.

Calcular
Calcular(NumPrincipal As Integer, NumRestar As Integer)

Figura 57. Información del formato de procedimiento.

Esta característica puede ser configurada abriendo la ventana Opciones desde el menú de VB (Herramientas+Opciones), en la pestaña Editor marcaremos o desmarcaremos el elemento Información rápida automática.

La escritura de procedimientos no puede anidarse, es decir, no puede escribirse, como se muestra en el Código fuente 43, un procedimiento dentro de otro. El código de los procedimientos debe ser independiente.

```

Sub Totalizar()

    Dim Valor As Integer
    Dim Primero As Integer

```

```

Valor = 342

' -----
' las siguientes líneas serían erróneas
Sub Cantidad()
Dim CuadreCaja As Integer
CuadreCaja =5000
End Sub
' -----
Primero = Valor +500

End Sub

```

Código fuente 43

Function

Sintaxis:

```

[Private|Public] [Static] Function NombreFunc([ListaParámetros]) As
Tipo
[código]
[NombreFunc = valor]
[Exit Function]
[código]
[NombreFunc = valor]
End Function

```

- Function...End Function. Indican el comienzo y final del procedimiento respectivamente.
- NombreFunc. Identificador utilizado para reconocer el procedimiento en el código del programa. Podemos asignarle un valor en el cuerpo del procedimiento, que será devuelto al punto en que fue llamado.
- Tipo. Tipo del dato que será devuelto por el procedimiento.
- Exit Function. Provoca la salida del procedimiento sin llegar a End Function. El flujo del programa continuará en la línea siguiente a la que llamó a este procedimiento. Es posible incluir más de una instrucción de este tipo en el cuerpo del procedimiento.

El resto de elementos del procedimiento, puede consultarlos el lector en la definición de Sub, ya que son equivalentes.

Un procedimiento Function (función) se diferencia de un Sub en dos aspectos.

- Devuelve un valor a la línea de código que lo ha llamado. Un procedimiento Sub no puede retornar valores.
- Un Function puede formar parte de una expresión, mientras que un Sub no.

El modo de devolver un valor en un Function, como hemos visto en la descripción de la sintaxis, pasa por asignar dicho valor al propio nombre del procedimiento, siendo posible hacer varias asignaciones de este tipo en el cuerpo del procedimiento. El tipo de dato devuelto deberá corresponderse con el tipo definido en la declaración del procedimiento. Por su parte, en la línea de código que efectúa la llamada

a la función, se deberá asignar a una variable el resultado de la llamada, encerrándose entre paréntesis los parámetros a enviar a la función.

La captura del valor devuelto por una función no es obligatoria, si no lo recogemos no se producirá un error. Veamos un ejemplo en el Código fuente 44.

```
Sub Main()

    ' declarar variables
    Dim Resultado As Integer

    ' llamar a la función:
    ' - para recoger el valor devuelto por la
    ' función, haremos una asignación a una variable
    ' - los parámetros en este tipo de llamada deben
    ' ser pasados entre paréntesis
    Resultado = Operacion(44)

    ' - la siguiente línea vuelve a llamar a la función
    ' pero sin recoger el valor devuelto
    ' - en este tipo de llamada no es necesario
    ' que los parámetros estén entre paréntesis
    Operacion 55

End Sub
' -----
' este procedimiento se ha declarado como Integer
' ya que devolverá un valor de ese tipo
Function Operacion(ValCalculo As Integer) As Integer

    Dim TotalOp As Integer

    TotalOp = 500 + ValCalculo

    ' asignamos al nombre de la función
    ' el valor que se devolverá al
    ' procedimiento que lo ha llamado
    Operacion = TotalOp

End Function
```

Código fuente 44

En lo referente a incluir una llamada a un Function como parte de una expresión, si partimos de una expresión como la siguiente.

```
Importe = 1000 + 380
```

Podemos hacer que uno de los elementos de la operación sea la llamada a una función que devuelva un resultado, dicho resultado será utilizado en la operación, como muestra el Código fuente 45.

```
Sub Main()

    ' declarar variables
    Dim Importe As Integer

    ' en esta expresión se realiza una
    ' suma y se asigna el resultado a una
```

```

' variable, pero uno de los elementos
' a sumar es el valor devuelto por
' la llamada a una función
Importe = 1000 + Mitad(50)

End Sub
' -----
Function Mitad(Valor As Integer) As Integer

Mitad = Valor / 2

End Function

```

Código fuente 45

En este ejemplo el lector puede comprobar cómo antes de realizar la suma, se llama al Function Mitad(), que devuelve un valor. El valor devuelto es el que se utilizará como uno de los operandos de la suma.

Paso de parámetros por valor y por referencia

Cuando pasamos variables como parámetros a un procedimiento, existen dos modos de realizar dicho envío, que afectarán a la manera en que es tratado el contenido del parámetro dentro del procedimiento. El modo de paso se especificará con las siguientes palabras clave junto al nombre del parámetro en la declaración del procedimiento.

ByVal (Por valor)

Al pasar una variable por valor a un procedimiento, se realiza una copia del valor de dicha variable al parámetro que está en el procedimiento llamado, por lo que si el procedimiento modifica el valor del parámetro, como se trata de una copia independiente, al retornar el control del programa a la línea de llamada, la variable no habrá sufrido ningún cambio, como comprobamos en el Código fuente 46.

```

Sub Main()
' declarar variables
Dim Nombre As String
' asignar valor
Nombre = "puerta"
' llamamos a un procedimiento
' pasándole como parámetro la
' variable Nombre
VerValor Nombre
' después de ejecutar el procedimiento
' mostramos el valor de la variable,
' que al haber sido pasada por valor
' no ha cambiado en este procedimiento
MsgBox "Valor de Nombre: " & Nombre, , ""
End Sub
' -----
' en este procedimiento, se copia el
' valor de la variable Nombre que pasamos
' desde Main() en el parámetro Contenido
Sub VerValor(ByVal Contenido As String)
' mostrar valor original
' del parámetro
MsgBox "Valor original del parámetro: " & Contenido, , ""

```

```

' cambiamos el valor del parámetro
' y volvemos a mostrarlo
Contenido = "mesa"
MsgBox "Valor cambiado del parámetro: " & Contenido, , ""
End Sub

```

Código fuente 46

Si tuviéramos que describir este proceso como interno de gestión de memoria y variables, se podría decir que el programa reserva al iniciarse, una determinada cantidad de memoria para almacenar entre otra información la referente a las variables. Cuando se declara la variable Nombre en Main(), se reserva una porción de esa memoria para el valor de la variable, y al llamar al procedimiento VerValor(), se reserva una nueva porción de espacio para el parámetro Contenido y aquí se copia el valor de la variable Nombre.

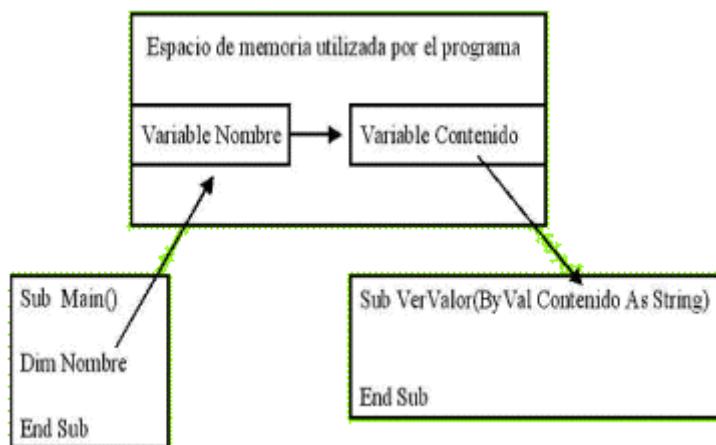


Figura 58. Representación del paso de parámetros por valor como interno.

ByRef (Por referencia)

Al pasar una variable por referencia a un procedimiento, lo que realmente se pasa es la dirección correspondiente a la porción de memoria en la que está ubicada dicha variable. En este caso, el procedimiento llamado no maneja una copia de la variable, sino la misma variable que está en el procedimiento llamador, aunque se utilice para ella otro nombre. Los cambios realizados en la variable serán permanentes, es decir, permanecerán al volver el control del programa al punto de llamada al procedimiento.

El lector podrá comprobar este aspecto en el Código fuente 47.

```

Sub Main()
' declarar variables
Dim Articulo As String
' asignar valor
Articulo = "televisor"
' llamamos a un procedimiento
' pasándole como parámetro la
' variable Articulo
Comprobar Articulo
' después de ejecutar el procedimiento
' mostramos el valor de la variable,

```

```

' que ha sido cambiado
MsgBox "Valor de Artículo: " & Artículo, , ""
End Sub
' -----
Sub Comprobar(ByRef Indica As String)
' cambiamos el contenido del parámetro
' que al pasarse por referencia,
' permanecerá después de finalizar
' el procedimiento
Indica = "monitor"
End Sub

```

Código fuente 47

Describiendo este proceso como interno de gestión de memoria y variables, al igual que en el caso anterior, el programa reserva al iniciarse, una determinada cantidad de memoria para almacenar la información de variables. Al declarar la variable Artículo en Main(), se reserva parte de esa memoria para la variable, y al llamar al procedimiento Comprobar(), se envía la propia variable Artículo al parámetro Indica. Esto quiere decir que realmente estamos trabajando con la variable Artículo, pero con otro nombre.

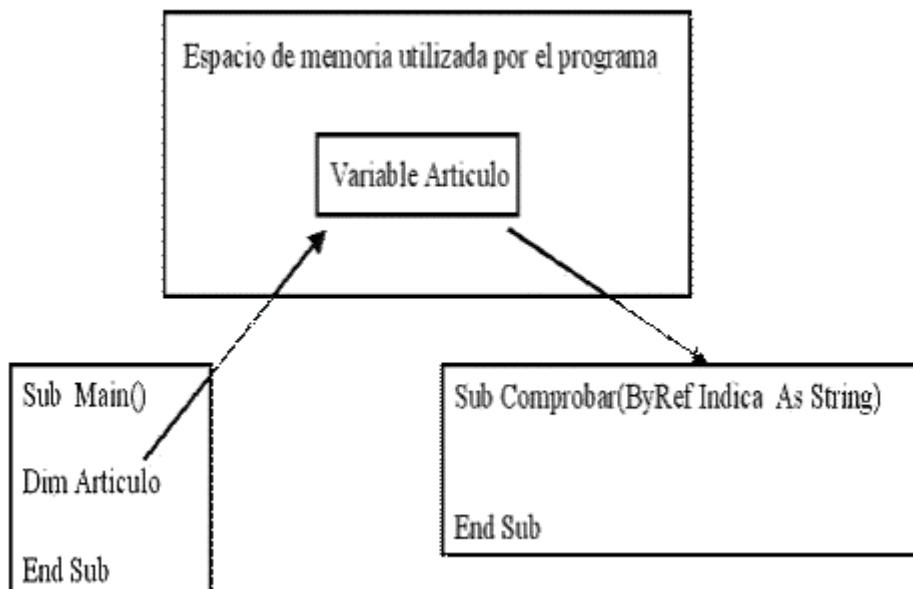


Figura 59. Representación del paso de parámetros por referencia en el ámbito interno.

El paso por referencia es el modo de envío de parámetros por defecto a un procedimiento, es decir, si no especificamos el tipo de paso en la declaración de un parámetro, este se pasará por referencia.

Como conclusión sobre el paso de parámetros, se recomienda al lector que siempre que sea posible, realice el paso de parámetros por valor, ya que se obtiene una mayor velocidad de ejecución. Sólo en el caso de que en el procedimiento llamado se necesite modificar el contenido del parámetro pasado, y que dicha modificación deba ser permanente, se justifica el paso por referencia.

Static

Cuando declaramos un procedimiento como Static, las variables declaradas en dicho procedimiento conservan su valor entre las llamadas que se realicen, es decir, no se inicializan cada vez que se

ejecute el procedimiento, sino que lo hacen la primera vez, manteniéndose el valor que tuviera desde la anterior ejecución.

```
Sub Main()
    ' llamamos dos veces al mismo
    ' procedimiento
    Comprobar
    Comprobar

    End Sub

    ' -----
    ' declaramos el procedimiento Static
    Static Sub Comprobar()
        ' la siguiente variable se inicializa
        ' sólo la primera vez que se llama
        ' al procedimiento
        Dim Articulo As String

        ' la primera vez que se ejecuta este procedimiento
        ' la siguiente línea muestra la variable vacía,
        ' pero en las demás, muestra un contenido

        MsgBox "Contenido de la variable Articulo: " & Articulo, , ""

        Articulo = "mesa"

    End Sub
```

Código fuente 48

En el Código fuente 48, al llamar por primera vez al procedimiento Comprobar(), se crea e inicializa la variable Articulo, asignándole un valor. En la siguiente llamada, la variable contiene el valor de la anterior ejecución. Por lo que podemos mostrar su contenido con un MsgBox() antes de llegar a la línea de asignación de valor.

Asistente para la declaración de procedimientos

Para facilitar la escritura del procedimiento, VB dispone de una utilidad que nos permite crear la declaración de un procedimiento.

Al seleccionar la opción de menú de VB Herramientas+Agregar procedimiento, aparecerá la siguiente ventana en la que escribiremos el nombre del procedimiento a crear, tipo y ámbito, lo que nos creará un procedimiento vacío.

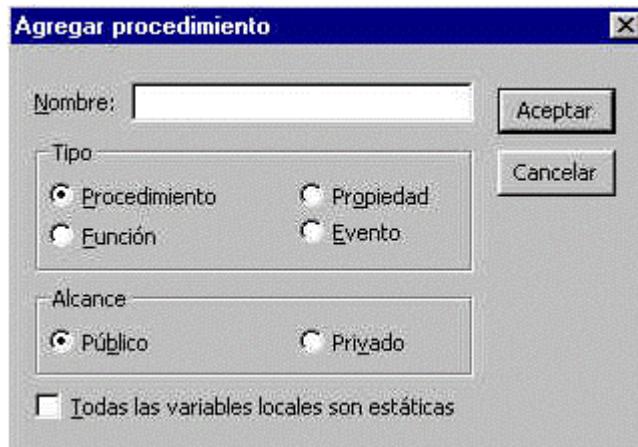


Figura 60. Ventana para añadir un procedimiento al programa.

Ámbito de elementos del lenguaje

Entendemos como **ámbito**, el nivel de visibilidad o accesibilidad de un procedimiento o variable dentro del código de un programa, es decir, los lugares de dicho programa desde los cuales vamos a poder hacer una llamada a un procedimiento o manipular una variable, según hayan sido declarados.

Procedimientos

Los modos de declaración de procedimientos, en lo que respecta al **ámbito** de los mismos, se describen a continuación. Las definiciones comentadas sirven tanto para Sub como para Function.

Private

Un procedimiento declarado con **Private**, sólo es visible (puede ser llamado), desde el módulo de código en el que se ha declarado.

Para comprobar este tipo de **ámbito**, recomendamos al lector la creación de un nuevo proyecto, agregando al mismo dos módulos de código, que podemos llamar Uno y Dos respectivamente.

En el módulo Uno del Código fuente 49, escribiremos el procedimiento **Main()** del proyecto.

```
' módulo Uno

Sub Main()
MsgBox "Vamos a llamar al procedimiento Prueba() del módulo Dos", , ""
Prueba

End Sub
```

Código fuente 49

En el módulo Dos del Código fuente 50, escribiremos este otro procedimiento, declarándolo privado.

```
' módulo Dos
Private Sub Prueba()
MsgBox "Estamos ejecutando el procedimiento Prueba()", , ""
End Sub
```

Código fuente 50

Lo que sucederá al intentar ejecutar el programa, es que se producirá un error, porque desde el procedimiento Main() del módulo Uno no se puede acceder al procedimiento Prueba() del módulo Dos, al haberse declarado este último procedimiento como Private.



Figura 61. Error al intentar ejecutar un procedimiento no visible.

Public

Un procedimiento declarado con Public es accesible, o puede ser llamado desde todos los procedimientos del proyecto, tanto del módulo en el que reside, como del resto de módulos.

Efectuemos por tanto, un pequeño cambio en el ejemplo anterior, y declaremos el procedimiento Prueba() como Public, lo vemos en el Código fuente 51.

```
' módulo Dos
Public Sub Prueba()
MsgBox "Estamos ejecutando el procedimiento Prueba()", , ""
End Sub
```

Código fuente 51

Al volver a ejecutar esta vez el programa, comprobaremos como sí se accede a dicho procedimiento desde un módulo diferente.

Cuando declaremos un procedimiento, si no especificamos el ámbito, por defecto será Public.

Variables

Según el ámbito o visibilidad que queramos dar a una variable, existen varios tipos de declaración que se relacionan seguidamente.

Dim

Este es el tipo de declaración ya conocida por el lector, y que venimos empleando en los ejemplos. Una variable declarada con Dim es creada al iniciarse la ejecución del procedimiento que la contiene, y se destruye al salir de él, por lo que sólo puede ser manejada dentro de dicho procedimiento. A este tipo de ámbito se le denomina ámbito local.

```
Sub Main()
    ' crear variable local
    Dim Nombre As String
    ' asignarle valor
    Nombre = "coche"
    ' al finalizar el procedimiento
    ' la variable se destruye
End Sub
```

Código fuente 52

Static

Una variable declarada como Static, mantiene su valor entre las diferentes llamadas que se realicen al procedimiento. No se destruye hasta que no finaliza el programa, pero su ámbito al igual que Dim es local, no se puede acceder a ella desde un procedimiento distinto al que fue declarada.

```
Sub Main()
    ' crear variable estática
    ' esta línea en la que se crea
    ' la variable, se ejecuta sólo la
    ' primera vez que se llama a este
    ' procedimiento
    Static Valor As Integer
    ' - se asigna una valor a la variable
    ' que se irá acumulando en las
    ' sucesivas llamadas
    ' - como en el resto de llamadas,
    ' la variable no se vuelve a inicializar
    ' sino que conserva el valor asignado
    ' en la última ejecución, pues dicho valor
    ' irá aumentando al ejecutar la siguiente línea
    Valor = Valor + 1
End Sub
```

Código fuente 53

Private

Las variables Private se deben declarar a nivel de módulo en la zona de declaraciones. Este tipo de variables son visibles sólo por los procedimientos que se encuentran en su mismo módulo, siendo inaccesibles para los procedimientos de los demás módulos.

```
' módulo Uno
' zona de declaraciones
```

```

Option Explicit
Private Articulo As String
' ----

' zona de procedimientos
Sub Main()
Articulo = "mesa"
Prueba
End Sub
' ----

' módulo Dos
Public Sub Prueba()
Articulo = "silla"
End Sub

```

Código fuente 54

Al ejecutar el anterior código, cuando el control del programa llegue al procedimiento Prueba() se producirá el error que aparece en la Figura 62.



Figura 62. Error de ejecución por variable fuera de ámbito.

Cuando una variable no es accesible desde un punto determinado del programa debido a la forma en que ha sido declarada, se dice que esa variable está fuera de ámbito.

Es posible, sin embargo, declarar en dos módulos diferentes, dos variables Private con el mismo nombre. A pesar de tener el mismo nombre, no existirán interferencias entre los valores, puesto que VB sabrá en cada momento sobre qué variable trabaja, en función del módulo que esté en ejecución en ese momento.

En el Código fuente 55, disponemos de dos módulos, en los que se declara una variable Private con el mismo nombre (Articulo). Cuando la ejecución llegue al procedimiento Prueba() en el módulo Dos, VB sabe que la variable Articulo de ese procedimiento, es independiente de la variable con el mismo nombre en el procedimiento Main().

```

' módulo Uno
' zona de declaraciones
Option Explicit
Private Articulo As String
' ----

' zona de procedimientos
Sub Main()

```

```

Articulo = "mesa"
Prueba
End Sub
' -----
' módulo Dos
' zona de declaraciones
Option Explicit
Private Articulo As String
Public Sub Prueba()
Articulo = "silla"
End Sub

```

Código fuente 55

Public

Estas variables también deben declararse en la zona de declaraciones del módulo. Sin embargo, a diferencia de las anteriores, una variable pública es visible por todos los procedimientos de todos los módulos del programa.

Para comprobarlo, tomemos uno de los ejemplos anteriores, en el que se declaraba una variable en el módulo Uno, intentando acceder a ella desde el módulo Dos y cambiémosle la declaración como indica el Código fuente 56

```

' módulo Uno
' zona de declaraciones
Option Explicit
Public Articulo As String

```

Código fuente 56

Esta vez no se producirán errores de ejecución, siendo la variable Articulo accesible desde el procedimiento Prueba() que está en un módulo diferente.

Debido al hecho de que es posible declarar dos variables con ámbito Public en dos módulos diferentes, se nos plantea el siguiente problema: ¿cómo podemos indicar a VB que la variable a manipular es la del otro módulo y no el actual?. La respuesta es, empleando la notación de objetos, nombre de módulo, operador envío y nombre de la variable.

En el Código fuente 57, tenemos dos módulos con la variable Valor declarada en ambos con ámbito público. Desde uno de los procedimientos se realiza un acceso a la variable del otro, de la forma que se muestra en dicho código

```

Module1
Option Explicit
Public Valor As String
' -----
Public Sub Main()
Valor = "estoy en módulo de main"
Calcular
End Sub
' ****
Module2

```

```

Option Explicit
Public Valor As String
' -----
Public Sub Calcular()
Valor = "estoy en módulo de calcular"
Module1.Valor = "información cambiada"
End Sub

```

Código fuente 57

Constantes

Las reglas de ámbito que se acaban de describir para las variables, rigen igualmente para las constantes.

Convenciones de notación

Por convenciones de notación, podemos entender un conjunto de reglas que utilizaremos en el desarrollo de una aplicación para denominar a los diferentes elementos que la componen: variables, constantes, controles, objetos, etc. Si bien esto no es inicialmente necesario, ni la herramienta de programación obliga a ello, en la práctica se ha demostrado que una serie de normas a la hora de escribir el código redundan en una mayor velocidad de desarrollo y facilidad de mantenimiento de la aplicación. Siendo útil no sólo en grupos de trabajo, sino también a programadores independientes.

En el Apéndice I se describen una serie de normas de codificación, que no son en absoluto obligatorias a la hora de escribir el código de la aplicación, pero si pretenden concienciar al lector de la necesidad de seguir unas pautas comunes a la hora de escribir dicho código, de manera que al compartirlo entre programadores, o cuando tengamos que revisar una aplicación desarrollada varios meses atrás, empleemos el menor tiempo posible en descifrar lo que tal o cual variable significa en el contexto de una rutina o módulo.

A continuación se describen dichas normas para variables y constantes, que emplearemos a partir de ahora en los ejemplos.

Variables

La estructura para denominar variables será la siguiente:

<ámbito><tipo><cuerpo>

- Valores para ámbito (Tabla 19).

G	Global
L	Local
S	Static
M	Módulo

V	Variable pasada por valor
R	Variable pasada por referencia

Tabla 19

- Valores para tipo (Tabla 20).

b	Boolean
by	Byte
m	Currency 64 bit
d	Double 64 bit
db	Database
Dt	Date+Time
F	Float/Single 32 bit
H	Handle
l	Long
n	Integer
c	String
u	Unsigned
ul	Unsigned Long
vnt	Variant
w	Word
a	Array
O	Objeto

Tabla 20

Para el cuerpo de la variable se utilizará WordMixing, que consiste en una identificación de la variable, en la que si necesitamos emplear más de una palabra, irán todas seguidas, sin separar por un guión bajo.

Ejemplos:

- Variable local, integer, para guardar un número de referencia.

`lnNumReferencia`

- Variable global, cadena de caracteres, que contiene el código de usuario de la aplicación.

GCUsuarioApp

- Variable a nivel de módulo, cadena de caracteres, que contiene el nombre de la aplicación.

mcAppNombre

Constantes

Seguirán las mismas reglas que las variables para el ámbito y tipo. El cuerpo de la constante deberá ir en mayúsculas, y en vez de utilizar WordMixing, se utilizará el guión bajo "_" para separar los distintos componentes del cuerpo.

Ejemplo:

- Constante NUMERO_LINEA, global, de tipo integer.

gnNUMERO_LINEA

Indentación del código

La indentación es una técnica para escribir el código de una aplicación, consistente en situar las líneas de código a diferentes niveles de tabulación o profundidad, de forma que se facilite su lectura al ser revisado.

Su uso está muy ligado a las estructuras de control, que veremos a continuación, ya que nos permite ver rápidamente las instrucciones contenidas dentro de estas estructuras y situarnos en las líneas necesarias.

En el Código fuente 58 se muestran unas líneas en pseudocódigo que contienen estructuras y código dentro de las mismas, de forma que pueda comprobar el lector el modo en que se indenta el código.

```

comienzo
' líneas con indentación al mismo nivel
instrucción1
instrucción2
' comienzo de estructura
EstructuraA <expresión>
    ' líneas indentadas a un nivel
    ' de profundidad
    instrucciónA1
    instrucciónA2
    ' comienza estructura anidada
    EstructuraB <expresión>
        ' líneas de nuevo indentadas
        ' a un segundo nivel de profundidad
        instrucciónB1
        instrucciónB2
    FinEstructuraB
    ' según salimos de las estructuras
    ' volvemos a subir niveles de indentación

```

```

instrucciónA3
instrucciónA4
FinEstructuraA
instrucción3
instrucción4
final

```

Código fuente 58

Cuando queramos indentar una línea, simplemente hemos de pulsar la tecla Tab, que nos situará en el siguiente tabulador de dicha línea. Para configurar la cantidad de espacios a indentar, debemos abrir la ventana de opciones de VB (opción de menú Herramientas+Opciones), y en el campo Ancho de tabulación, establecer la cantidad de espacios deseada.

Igualmente, si queremos que desde una línea indentada, al pulsar Enter, el cursor se sitúe al mismo nivel de indentación, marcaremos la opción Sangría automática, como vemos en la Figura 63.

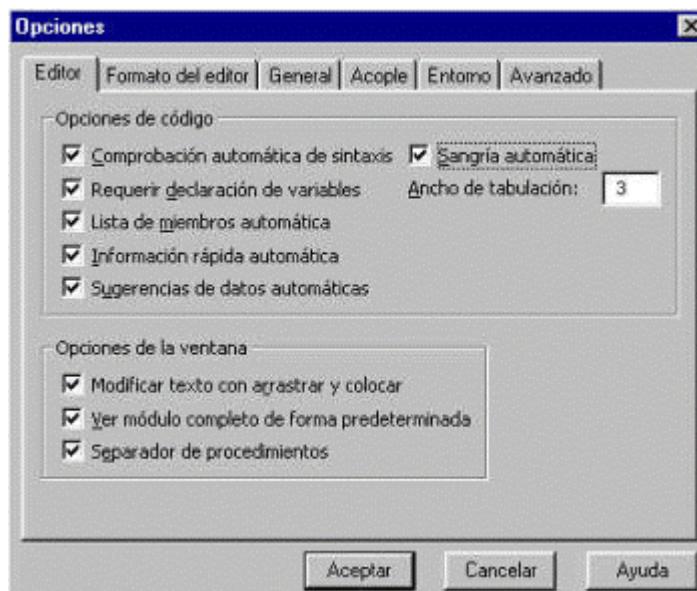


Figura 63. Ventana de opciones de Visual Basic.

Estructuras de control

Las estructuras de control junto a los procedimientos, forman los pilares de la programación estructurada, permitiéndonos decidir que partes de código ejecutar según el cumplimiento o no de ciertas condiciones. Al proceso por el cual se desvía el flujo de la ejecución de un programa en función de una condición se le denomina bifurcación.

Según el modo en que ejecuten el código contenido en su interior, las estructuras de control pueden clasificarse en dos categorías: selección y repetición.

Selección

Las estructuras de selección permiten ejecutar un bloque o conjunto de líneas de código entre varios disponibles, según el resultado de la evaluación de una expresión al comienzo de la estructura.

If...End If

En función del nivel de complejidad de la decisión a resolver, esta estructura puede emplearse de diversas formas:

- Decisión simple.

Sintaxis:

```
If <Expresión> Then
    ' código
    .....
    .....
End If
```

Si al resolver el contenido de Expresión, se obtiene Verdadero como resultado, se ejecutarán las líneas comprendidas entre If y End If. Si obtenemos Falso, se desviará el flujo del programa a la siguiente línea después de End If.

```
Public Sub Main()
Dim lcValor As String
lcValor = InputBox("Introducir valor", " ")
If lcValor = "mesa" Then
    MsgBox "Se está ejecutando la estructura If", , ""
End If
End Sub
```

Código fuente 59

- Decisión doble.

Sintaxis:

```
If <Expresión> Then
    ' código If
    .....
    .....
Else
    ' código Else
    .....
    .....
End If
```

Si al resolver el contenido de Expresión, se obtiene Verdadero como resultado, se ejecutará el código comprendido entre If y Else; si el resultado es Falso, se ejecutará el código que hay entre Else y End If.

```
Public Sub Main()
Dim lcValor As String
lcValor = InputBox("Introducir valor", " ")
If lcValor = "mesa" Then
    lcValor = lcValor & "azul"
Else
```

```

    MsgBox "La variable Valor contiene: " & lcValor
End If
End Sub

```

Código fuente 60

- Decisión múltiple.

Sintaxis:

```

If <ExpresiónA> Then
    ' código If
    .....
    .....
ElseIf <ExpresiónB> Then
    ' código ElseIf
    .....
    .....
[ElseIf <ExpresiónX> Then]
    [' código ElseIf
    .....
    .....
Else
    ' código Else
    .....
    .....
End If

```

Si al resolver el contenido de ExpresiónA, se obtiene Verdadero como resultado, se ejecutará el código comprendido entre If y ElseIf; si el resultado es Falso, se evaluará la expresión de ElseIf, y si esta resultara Verdadero, se ejecutarán las líneas entre ElseIf y el siguiente ElseIf (podemos especificar varios bloques ElseIf), o bien Else o End If.

En el caso de que el primer ElseIf resultara Falso, se comprueba si existe otro ElseIf a continuación, procediendo de la forma ya comentada.

Si no se cumple ninguna condición en las expresiones If o ElseIf definidas, y existe un Else (que siempre irá al final de la estructura), se ejecutarán las instrucciones entre el Else y End If.

```

Public Sub Main()
Dim lcValor As String
Dim lcControl As String
lcValor = InputBox("Introducir valor", " ")
lcControl = InputBox("Introducir número", " ")
If lcValor = "mesa" Then
    MsgBox "El contenido de Valor es correcto", , ""
ElseIf lcControl > 1000 Then
    MsgBox "El número es mayor de 1000", , ""
Else
    MsgBox "Ninguna de las comprobaciones ha sido correcta", , ""
End If
End Sub

```

Código fuente 61

En cualquier caso, una vez finalizada esta estructura, el flujo del programa continuará en la siguiente línea a End If.

Select Case...End Select.

Sintaxis:

```
Select Case <Expresión>
Case ListaValores
    ' código Case
    .....
    .....
[Case ListaValores]
    [' código Case
    .....
    .....
Case Else
    ' código Else
    .....
    .....
End Select
```

El funcionamiento de esta estructura es similar a una de tipo If...End If con varios ElseIf. En primer lugar, se evalúa el contenido de Expresión, y posteriormente se comprueba si su resultado coincide con los valores especificados en ListaValores de alguno de los Case siguientes (podemos poner varios), en cuyo caso se ejecutará el código que hay entre ese Case y el siguiente, o bien Case Else o End Select.

Si ninguno de los Case coincide con el valor a buscar, se ejecutará el código de Case Else si se ha proporcionado.

Los valores de ListaValores estarán separados por comas y podrán especificarse de la siguiente forma:

- Valor.
- ValorMenor To ValorMayor.
- Is OperadorComparación Valor

Si hay varios Case que cumplen la expresión, se ejecutará el código del que se encuentre en primer lugar.

Finalizada la ejecución de esta estructura, la ejecución del programa continuará en la siguiente línea a End Select.

```
Public Sub Main()
    ' inicializar variable
    Dim lnNum As Integer
    ' dar valor
    lnNum = 1250
    ' evaluar el valor de la variable y
    ' buscar algún Case que coincida
    Select Case lnNum
        Case 4100
```

```

' comprobar un sólo valor
MsgBox "La variable contiene " & lnNum
Case 2000, 1250, 800
    ' aquí se especifica una lista de valores
    ' como este es el primero que se cumple,
    ' es el que se ejecuta
    MsgBox "Estamos en Case de varios valores"
Case Is < "1000"
    ' usamos la palabra Is junto a
    ' una expresión de comparación
    MsgBox "Estoy en Case < '1000' "
Case 500 To 1300
    ' aquí también se cumpliría la condición
    ' por el intervalo de valores, pero no
    ' se ejecuta este código porque ya hay un
    ' Case anterior que se cumple, por lo cual
    ' no pasa por aquí
    MsgBox "Estoy en Case 500 To 1300"
Case Else
    MsgBox "No se cumple ninguna condición"
End Select
End Sub

```

Código fuente 62

Repetición

Las estructuras de repetición, se basan en la ejecución de un mismo conjunto de líneas de código, un número determinado/indeterminado de veces mientras se cumpla la condición incluida en una expresión al comienzo o final de la estructura. Al proceso por el cual se ejecuta una vez el código encerrado entre el principio y fin de una de estas estructuras se le conoce como iteración.

A este tipo de estructura se le denomina también bucle.

Do...Loop.

Ejecuta el código contenido entre estas dos palabras clave, existiendo tres variantes de uso.

```

Do While | Until <Expresión>
    ' código
    .....
    .....
    [Exit Do]
    ' código
    .....
    .....
Loop

```

Se comprueba Expresión al comienzo del bucle, si es correcta se comienzan a ejecutar las líneas contenidas en la estructura mientras que la expresión devuelva Verdadero o hasta que se cumpla la expresión.

Si necesitamos salir del bucle sin que se haya cumplido la condición de salida, emplearemos la instrucción Exit Do, que bifurcará el control del programa hasta la línea siguiente a Loop.

La diferencia entre usar While o Until para evaluar la expresión de esta estructura, estriba en que con While, el bucle se estará ejecutando mientras que la expresión devuelva Verdadero; con Until, el bucle se ejecutará mientras la expresión devuelva Falso. Veamos un ejemplo de cada uno en el Código fuente 63.

```
Public Sub Main()
    ' declarar variable y asignar valor
    Dim lnNum As Integer
    lnNum = 10
    ' mientras que la expresión del bucle
    ' sea True, ejecutarlo
    Do While lnNum < 20
        lnNum = lnNum + 1
    Loop
    ' -----
    ' reasignar valor
    lnNum = 10
    ' ahora usaremos Until
    ' hasta que la variable no tenga
    ' un determinado valor, ejecutar el bucle
    Do Until lnNum = 30
        lnNum = lnNum + 1
    Loop
    ' -----
    ' reasignar valor
    lnNum = 10
    ' ahora saldremos del bucle
    ' antes de cumplir la salida
    ' mientras que la expresión del bucle
    ' sea True, ejecutarlo
    Do While lnNum < 20
        lnNum = lnNum + 1

        ' cuando la variable tenga
        ' un determinado valor, forzamos
        ' la salida
        If lnNum = 15 Then
            Exit Do
        End If

    Loop
End Sub
```

Código fuente 63

El siguiente modo de uso de esta estructura, tiene como característica el que el código se ejecutará al menos una vez, ya que la condición de salida se encuentra al final. Por lo demás, el modo de funcionamiento es igual que en el caso anterior.

```
Do
    ' código
    .....
    .....
    [Exit Do]
    ' código
    .....
    .....
Loop While | Until <Expresión>
```

Finalmente, tenemos el modo más simple de uso, pero también el más peligroso, ya que no se especifica la condición de salida al principio ni final. El único modo de salir es mediante la instrucción Exit Do, por lo que debemos escribir una condición que permita ejecutar dicha instrucción o entraremos en un bucle infinito.

```
Public Sub Main()
    ' declarar variable y asignar valor
    Dim lnNum As Integer
    lnNum = 10
    ' entrar directamente en el bucle
    Do
        lnNum = lnNum + 1

        ' si no especificamos una condición
        ' de salida, permaneceremos todo
        ' el tiempo ejecutando la estructura
        If lnNum >= 25 Then
            Exit Do
        End If

    Loop
End Sub
```

Código fuente 64

Cuando diseñamos bucles, debemos poner sumo cuidado en proporcionar una salida del mismo, bien sea cuando deje de cumplirse la condición impuesta por el bucle o desde el interior del mismo, mediante una instrucción de salida.

Si no tenemos este factor en cuenta, puede darse el caso de que al entrar el flujo de la ejecución en el bucle, se quede permanentemente ejecutando su código por la inexistencia de una vía de salida. A una estructura de este tipo se le denomina bucle infinito, y puede originar un comportamiento irregular del programa y el bloqueo del sistema.

While...Wend

Esta estructura es similar a Do...Loop, ya que si el valor de la expresión es Verdadero, se ejecutarán las líneas contenidas en el bucle hasta que la expresión devuelva Falso.

Sin embargo, es menos flexible, ya que no permite situar la expresión a evaluar al final, ni tampoco tenemos medio de forzar la salida desde el interior, por lo que se recomienda emplear Do...Loop.

```
While <Expresión>
    ' código
    .....
    .....
    .....
Wend
```

For...Next

Esta estructura ejecuta el conjunto de líneas contenidas en su interior un número determinado de veces. El elemento Contador, controla el número de iteración realizada en cada momento,

inicializándose al comienzo del bucle con el valor de Principio y terminando al alcanzar el valor de Fin.

```
For Contador = Principio To Fin [Step Incremento]
    ' código
    .....
    .....
    [Exit For]
    ' código
    .....
    .....
Next [Contador]
```

Por defecto, las iteraciones se realizarán en incrementos de uno, pero si necesitamos que ese incremento sea diferente lo especificaremos en la cláusula Step Incremento, donde Incremento será el valor que utilizará el contador para actualizarse.

Existe también la posibilidad de realizar un incremento negativo, es decir, un decrecimiento. Para ello, el valor inicial del contador deberá ser superior a Fin, siendo necesario especificar el decrecimiento aunque sea en uno, ya que de lo contrario, el bucle no llegará a ejecutarse al detectar que el valor de comienzo es superior al de fin.

Si en algún momento al ejecutar esta estructura, necesitamos salir sin haber cumplido todas las iteraciones necesarias, emplearemos la instrucción Exit For, que desviará el control a la primera línea después de Next.

El procedimiento mostrado en el Código fuente 65, contiene varios ejemplos de uso de esta estructura.

```
Public Sub Main()
Dim lnContador As Integer ' variable que actúa como contador
Dim lnValor As Integer
' bucle normal
For lnContador = 1 To 20
    ' mostramos la iteración actual del bucle
    MsgBox "Iteración número " & lnContador, , ""
Next
' especificando un incremento
For lnContador = 1 To 20 Step 7
    ' con esta variable contamos
    ' las iteraciones realizadas
    lnValor = lnValor + 1

    ' mostramos el paso actual
    MsgBox "Se ha ejecutado el bucle " & lnValor & " veces", , ""
Next
' con decremento
For lnContador = 20 To 12 Step -1
    ' mostramos la iteración actual del bucle
    MsgBox "Iteración número " & lnContador, , ""
Next
' aquí no se especifica el valor
' del decremento, por lo que el
' bucle no se ejecutará el ser
' el valor de inicio del contador
' superior al valor final
For lnContador = 20 To 12
    ' mostramos la iteración actual del bucle
    MsgBox "Iteración número " & lnContador, , ""
Next
```

```

' en este caso salimos antes de cumplir
' todas las iteraciones
For lnContador = 1 To 15

    If lnContador = 8 Then
        Exit For
    End If

Next
End Sub

```

Código fuente 65

GoTo

La instrucción GoTo, sin ser una estructura de control, realiza al igual que estas una bifurcación en el flujo del programa. La diferencia con las estructuras, es que GoTo realiza la bifurcación de forma incondicional, y las estructuras proporcionan un medio para permitir realizar o no las oportunas bifurcaciones.

Goto NombreEtiqueta

La bifurcación se realiza hacia lo que se denomina una etiqueta de código, que consiste en un bloque de líneas encabezadas por un identificador y dos puntos.

```

Public Sub Main()
Dim lnUno As Integer
Dim lnDos As Integer
lnUno = 452
' se desvía la ejecución
GoTo Desvio
lnDos = lnUno * 2 ' esta línea no se ejecutará
' -----
' etiqueta de código
Desvio:
MsgBox "Estamos ejecutando la etiqueta de código", , "Aviso"
End Sub

```

Código fuente 66

El uso de GoTo y etiquetas de código en los procedimientos, salvo contadas excepciones como puede ser el tratamiento de errores, es muy desaconsejable debido a la poca estructuración del código y complejidad de mantenimiento que producen; por este motivo y sólo en casos muy determinados, desaconsejamos al lector el uso de este tipo de codificación, ya que por lo general, siempre existirá una forma más estructurada de realizar las tareas.

En el Código fuente 67 vamos a ver dos fuentes que realizan la misma tarea, en uno se utiliza GoTo junto a una etiqueta, en el otro se hace una llamada a un procedimiento para realizar el mismo trabajo.

```

' Versión con GoTo
Public Sub Main()
' declaramos variables y les
' asignamos valor

```

```

Dim lcNombre As String
Dim lnNum As Integer
lcNombre = "Aurora"
' enviamos el control del programa a
' la etiqueta VerMsg, las líneas de
' código entre Goto y la etiqueta
' no se ejecutarán
GoTo VerMsg
lnNum = 111
lcNombre = lcNombre & lnNum
VerMsg:
MsgBox "El valor de lcNombre es: " & lcNombre
End Sub
' -----
' Versión con llamada a procedimiento:
Public Sub Main()
' declaramos variables y les
' asignamos valor
Dim lcNombre As String
Dim lnNum As Integer
lcNombre = "Aurora"
' llamamos al procedimiento VerMsg
' y le pasamos como parámetro la variable
' lcNombre, al terminar de ejecutarse
' devolverá el control a esta línea
' con lo que el resto de líneas de
' este procedimiento si se ejecutarán
VerMsg lcNombre
lnNum = 111
lcNombre = lcNombre & lnNum
End Sub
' -----
Public Sub VerMsg(rcNombre As String)
MsgBox "El valor del Nombre es: " & rcNombre
End Sub

```

Código fuente 67

Anidación de estructuras

Esta técnica de escritura del código consiste en incluir una estructura de control dentro de otra estructura ya existente. La estructura anidada o interior puede ser del mismo o diferente tipo que la estructura contenedora o exterior, siendo posible además, anidar más de una estructura, y a varios niveles de profundidad. Veamos en el Código fuente 68, un procedimiento con varios ejemplos.

```

Public Sub Main()
' declaración de variables
Dim lnContador As Integer
Dim lnNumero As Integer
' estructura For exterior
For lnContador = 1 To 20
' estructura If interior
If lnContador > 10 Then
    MsgBox "Hemos pasado de 10", , "Aviso"
End If

Next
' -----
lnNumero = InputBox("Introducir un número", " ")
' estructura Select exterior
Select Case lnNumero

```

```

Case 20
    MsgBox "El número es 20", , ""

Case 30 To 40
    ' estructura For interior
    For lnContador = lnNúmero To lnNúmero + 5
        MsgBox "Contador del bucle " & lnContador

        ' estructura If interior
        ' con nuevo nivel de anidamiento
        If lnContador = 35 Then
            MsgBox "El contador es 35", , ""
        End If

    Next
Case Else
    MsgBox "No hacemos nada"
End Select
End Sub

```

Código fuente 68

Al escribir estructuras anidadas, hemos de poner cuidado para evitar que parte de la estructura de nivel interior, quede fuera de la estructura de nivel exterior, ya que esto provocaría un error de ejecución, como muestra la Figura 64.



Figura 64. Error en anidamiento de estructuras.

Arrays

Como se explicó en el apartado Datos, un array es un tipo de dato compuesto, consistente en un conjunto de elementos del mismo tipo de dato, al que se le asigna un nombre de variable para poder identificarlo en el código del programa. También se les denomina tabla, matriz o vector.

A cada elemento del array se accede mediante un índice, que indica el número de posición que ocupa dicho elemento dentro del array.

Si tuviéramos que representar un array gráficamente, podríamos hacerlo mediante un conjunto de casillas, en cada una de las cuales se encontraría un elemento del array. Cada casilla tendría un número (índice) que identificaría de forma única su posición.

Array Articulos	0 "mesa"	1 "silla"	2 "sofá"	3 "armario"
-----------------	-------------	--------------	-------------	----------------

Figura 65. Presentación gráfica de un array.

Declaración

Para declarar una variable que va a contener un array, procederemos de igual forma que para una variable normal, con la diferencia de que debemos indicar entre paréntesis, junto al nombre del array, el número de elementos que va a contener.

```
Dim lcArticulos(3) As String
```

Código fuente 69

El Código fuente 69 crearía un array de ámbito local, con cuatro elementos de tipo cadena de caracteres. Si el lector, ha leído bien, cuatro elementos; esto es debido a que la primera posición de un array, por defecto es cero, como se observa en la Figura 66.

Array lcArticulos	lcArticulos(0) ""	lcArticulos(1) ""	lcArticulos(2) ""	lcArticulos(3) ""
-------------------	----------------------	----------------------	----------------------	----------------------

Figura 66. Array de tipo cadena con los elementos sin valor (cadena vacía).

El modo que tenemos de crear un array en el que cada elemento pueda ser de un tipo distinto de dato, es declarando el array de tipo Variant.

Asignación

El modo de asignar o recuperar el valor de una posición de un array es también similar a una variable normal, sólo que tendremos que especificar el elemento a manipular en el índice del array. Veamos el Código fuente 70.

```
Public Sub Main()
' declaración de variables
Dim lcArticulos(3) As String
Dim lcValor As String
' asignar valores a los elementos del array
lcArticulos(0) = "mesa"
lcArticulos(1) = "silla"
lcArticulos(2) = "sofá"
lcArticulos(3) = "armario"
' pasar el valor de uno de los elementos
' del array a una variable
lcValor = lcArticulos(2)      ' sofá
End Sub
```

Código fuente 70

Establecer los límites en los índices

Por defecto, el primer índice de un array es cero, sin embargo, es posible cambiar esta configuración de dos maneras.

Option Base

Sintaxis

```
Option Base IndiceInferior
```

Utilizando esta instrucción en la zona de declaraciones del módulo, conseguiremos que el primer índice de un array sea 0 o 1, que son los dos valores que admite IndiceInferior. El ejemplo anterior por consiguiente, podría cambiar de la forma que se muestra Código fuente 71.

```
' zona de declaraciones
Option Base 1
' -----
Public Sub Main()
' declaración de variables
Dim lcArticulos(4) As String
' asignar valores a los elementos del array
lcArticulos(1) = "mesa"
lcArticulos(2) = "silla"
lcArticulos(3) = "sofá"
lcArticulos(4) = "armario"
End Sub
```

Código fuente 71

Esta instrucción afecta a todos los arrays que se declaren en el módulo en que es utilizada. Si declaramos un array con ámbito público, que tiene 1 como primer índice, seguirá manteniendo esta configuración de índices aunque se utilice en otro módulo que tenga Option Base 0.

To

Sintaxis

```
NombreArray(IndiceInferior To IndiceSuperior)
```

Empleando esta palabra clave en la declaración de un array, entre un número que represente el índice inferior y otro que sea el superior, crearemos un array en el que el rango de índices, será establecido por el programador, sin estar obligado a comenzar por 0 o 1. Vemos un ejemplo en el Código fuente 72.

```
Public Sub Main()
' declaración de variables
' establecer límites de índices
Dim lcArticulos(14 To 17) As String
' asignar valores a los elementos del array
```

```

lcArticulos(14) = "mesa"
lcArticulos(15) = "silla"
lcArticulos(16) = "sofá"
lcArticulos(17) = "armario"
End Sub

```

Código fuente 72

Esta cláusula es muy versátil, ya que podemos utilizar un número determinado de elementos para un array, asignándoles el intervalo de índice que mejor convenga al contexto del procedimiento o módulo, teniendo siempre en cuenta que si intentamos utilizar un número de índice fuera de ese intervalo, se producirá un error.

Recorrer los elementos de un array

Cuando nos encontramos en la situación de tener que manipular parte o todos los elementos de un array, del que no sepamos a priori cuantos elementos tiene, podemos utilizar las siguientes funciones de VB, que nos informan del número correspondiente al primer y último índice, del array pasado como parámetro.

- LBound(NombreArray, [Dimensión]). Devuelve el número del primer índice del array NombreArray. Si se trata de un array multidimensional, podemos indicar que dimensión queremos averiguar en el parámetro Dimensión.
- UBound(NombreArray, [Dimensión]). Devuelve el número del último índice del array NombreArray. Si se trata de un array multidimensional, podemos indicar que dimensión queremos averiguar en el parámetro Dimensión.

El valor devuelto por ambas funciones es de tipo Long.

La forma más cómoda de recorrer un array es utilizando un bucle For...Next, combinado con el uso de las funciones que se acaban de describir. En el Código fuente 73 tenemos un ejemplo.

```

Public Sub Main()
' declarar variables
Dim lcNombres(1 To 5) As String
Dim lnContador As Integer
' rellenar array
lcNombres(1) = "Carmen"
lcNombres(2) = "Rosa"
lcNombres(3) = "Luis"
lcNombres(4) = "Antonio"
lcNombres(5) = "Juan"
' recorrer el array
' mostrando su contenido
' y posición actual
For lnContador = LBound(lcNombres) To UBound(lcNombres)
    MsgBox "Contenido del elemento: " & lcNombres(lnContador), , _
        "Índice: " & lnContador
Next
End Sub

```

Código fuente 73

Existe sin embargo, una variante del bucle For...Next, que automatiza aun más este tipo de operaciones. Se trata de la estructura For Each...Next.

```
For Each Elemento In NombreArray
    ' código
    .....
    .....
    [Exit For]
    ' código
    .....
    .....
Next [Elemento]
```

En esta estructura, Elemento es una variable de tipo Variant, en la que se depositará el contenido del elemento del array en cada una de las iteraciones del bucle.

Veamos en el Código fuente 74 como quedaría el Código fuente 73.

```
Public Sub Main()
    ' declarar variables
    Dim lcNombres(1 To 5) As String
    Dim lvntElemento As Variant
    ' rellenar array
    lcNombres(1) = "Carmen"
    lcNombres(2) = "Rosa"
    lcNombres(3) = "Luis"
    lcNombres(4) = "Antonio"
    lcNombres(5) = "Juan"
    For Each lvntElemento In lcNombres
        MsgBox "Contenido del elemento: " & lvntElemento, , ""
    Next
End Sub
```

Código fuente 74

Arrays multidimensionales

Todos los arrays vistos hasta el momento han sido unidimensionales; se les denomina así porque tienen un único conjunto de datos o dimensión.

Los arrays multidimensionales por otro lado, son aquellos en los que cada elemento de primer nivel del array, es a su vez otro array, es decir, tienen más de un conjunto de datos. Veamos la Figura 67 para comprender mejor este concepto.

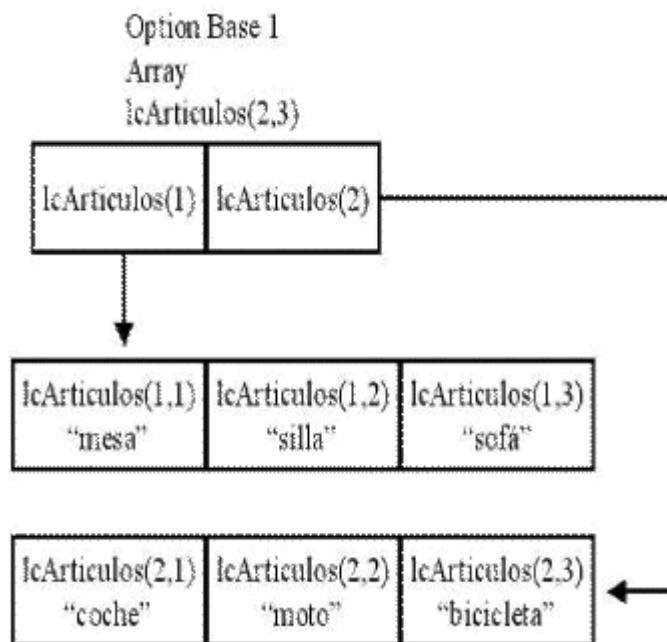


Figura 67. Array multidimensional.

El modo de declaración de un array de este tipo sería el que aparece en el Código fuente 75, suponiendo que tengamos Option Base a 1.

```
Dim lcArticulos(2,3) As String
```

Código fuente 75

De esta forma, declaramos un array con dos dimensiones. El primer número en la declaración indica la cantidad de dimensiones del array; el segundo número indica cuantos elementos va a tener cada dimensión. Así, para asignar en el código el valor sofá, lo haríamos como se indica en el Código fuente 76.

```
lcArticulos(1,3) = "sofá"
```

Código fuente 76

Para recorrer los elementos de un array multidimensional utilizaremos los bucles For...Next o For Each...Next, comentados anteriormente. La diferencia entre utilizar una u otra técnica para manipular el contenido del array, reside en que con For...Next tenemos un mayor control de los elementos a manipular, mientras que con For Each...Next, al implementar internamente el proceso, irá tomando las primeras posiciones de cada una de las dimensiones del array, después las segundas, y así sucesivamente. El lector puede comprobar este funcionamiento, ejecutando el procedimiento que aparece en el Código fuente 77.

```
Public Sub Main()
' declarar variables
```

```

Dim lcArticulos(2, 3) As String
Dim lnContador As Integer
Dim lnContador2 As Integer
Dim lvntElemento As Variant
' rellenar array
lcArticulos(1, 1) = "mesa"
lcArticulos(1, 2) = "silla"
lcArticulos(1, 3) = "sofá"
lcArticulos(2, 1) = "coche"
lcArticulos(2, 2) = "moto"
lcArticulos(2, 3) = "bicicleta"
' recorrer con bucle For...Next
For lnContador = LBound(lcArticulos, 1) To UBound(lcArticulos, 1)
    For lnContador2 = LBound(lcArticulos, 2) To UBound(lcArticulos, 2)
        MsgBox "Contenido del elemento: " & _
            lcArticulos(lnContador, lnContador2), , _
            "Posición: " & lnContador & "," & lnContador2
    Next
Next
' recorrer con For Each...Next
For Each lvntElemento In lcArticulos
    MsgBox "Contenido del elemento: " & lvntElemento, , ""
Next
End Sub

```

Código fuente 77

Arrays dinámicos

Un array estático es aquel que tiene un número fijo de elementos, establecido en la declaración, y no se pueden añadir o quitar. Es el tipo de array que hemos manejado hasta ahora.

Como contrapartida, tenemos los arrays dinámicos, que son aquellos que nos permiten agregar nuevos elementos durante la ejecución y eliminar elementos existentes.

Para crear un array dinámico, debemos declarar en primer lugar, un array sin especificar la dimensión o elementos.

```
Dim NombreArray() As Tipo
```

Cuando necesitemos añadir elementos, usaremos la instrucción ReDim, seguida del nombre del array, y la cantidad de elementos a crear.

```
ReDim NombreArray(NumElementos)
```

La instrucción ReDim elimina el contenido del array que se redimensiona. Si queremos conservar los elementos existentes, debemos utilizar la cláusula Preserve junto a ReDim de la siguiente forma.

```
ReDim Preserve NombreArray(NumElementos)
```

Lo que sí ocurrirá, y es un factor a tener muy en cuenta, es que si redimensionamos un array a un número de elementos más pequeño que el que tenía originalmente, los datos de las posiciones superiores se perderán.

En el Código fuente 78 se muestran dos procedimientos, en uno se manipula un array dinámico, y en el otro se visualiza.

```

Public Sub Main()
' declarar variables
Dim lcArticulos() As String
' redimensionar array
ReDim Preserve lcArticulos(2)
lcArticulos(1) = "mesa"
lcArticulos(2) = "silla"
VerArray lcArticulos ' mostrar array
' volver a redimensionar
ReDim Preserve lcArticulos(5)
lcArticulos(3) = "sofá"
lcArticulos(4) = "armario"
lcArticulos(5) = "cama"
VerArray lcArticulos ' volver a mostrar array
End Sub
' -----
Public Sub VerArray(ByRef vcDatos() As String)
Dim lnContador As Integer
' mostrar contenido del array pasado como parámetro
For lnContador = LBound(vcDatos) To UBound(vcDatos)
    MsgBox "Valor: " & vcDatos(lnContador), , ""
Next
End Sub

```

Código fuente 78

Array()

Esta función crea un array de tipo Variant utilizando la lista de valores que recibe como parámetro.

Sintaxis:

```
vntNombreArray = Array(ListaValores)
```

Veamos en el Código fuente 79, un sencillo ejemplo de esta función.

```

Public Sub Main()
Dim vntDatos As Variant
vntDatos = Array("limón", "naranja", "pera")
End Sub

```

Código fuente 79

Tipos definidos por el usuario

Un tipo definido por el usuario es un nuevo tipo de dato creado por el programador y formado por uno o más tipos de los existentes. Una vez creado, se pueden declarar variables de ese tipo de igual forma que hacemos con los tipos habituales.

Los tipos definidos por el usuario sólo se pueden crear a nivel de módulo (zona de declaraciones). En cuanto al ámbito de una variable con tipo de usuario, si dicha variable se va a utilizar en un módulo de código normal, se puede declarar pública; en el resto de módulos sólo podrá declararse privada o local.

Sintaxis:

```
Type NombreTipo
    NombreElemento1 As Tipo
    [NombreElementoN] As Tipo
    ....
    .....
End Type
```

Veamos en el Código fuente 80 un ejemplo de creación de tipo.

```
Option Explicit
Private Type Empleado
    Nombre As String
    Telefono As String
    FNacim As Date
End Type

' ----

Public Sub Main()
    ' declarar variable del nuevo tipo
    Dim lNuevoEmp As Empleado
    ' asignar valores a los elementos de
    ' la variable tipo
    lNuevoEmp.Nombre = "Juan García"
    lNuevoEmp.Telefono = "555.33.44"
    lNuevoEmp.FNacim = #9/20/69#
End Sub
```

Código fuente 80

Ahora creamos un nuevo tipo que sirva para almacenar direcciones y en el tipo Empleado definimos un nuevo elemento que sea de ese tipo. En el Código fuente 81 del procedimiento, la variable declarada como Empleado guardará ahora la dirección.

```
Option Explicit
Private Type Direccion
    Calle As String
    NumPortal As Integer
    Poblacion As String
End Type

' ----

Private Type Empleado
    Nombre As String
    ' el siguiente elemento del tipo
    ' es a su vez otro tipo de usuario
    Direcc As Direccion
    Telefono As String
    FNacim As Date
End Type

' ----

Public Sub Main()
    ' declarar variable del nuevo tipo
    Dim lNuevoEmp As Empleado
    ' asignar valores a los elementos de
    ' la variable tipo
    lNuevoEmp.Nombre = "Juan García"
    lNuevoEmp.Telefono = "555.33.44"
    lNuevoEmp.FNacim = #9/20/69#
    ' las siguientes líneas asignan valor
```

```
' a los elementos del tipo incluido
' a su vez dentro de otro elemento
lNuevoEmp.Direcc.Calle = "Olmos"
lNuevoEmp.Direcc.NumPortal = 21
lNuevoEmp.Direcc.Poblacion = "Madrid"
End Sub
```

Código fuente 81

El tipo Enumerado

Un tipo enumerado o una enumeración cómo también se le denomina, consiste en un identificador que contiene un conjunto de valores de tipo Long, a los que se accede mediante identificadores o constantes miembro, definidas para cada uno de los valores.

El modo de declaración de un tipo enumerado se realiza mediante la instrucción Enum, que tiene la siguiente sintaxis.

```
Enum Nombre
    Miembro [= Valor]
    Miembro [= Valor]
    .....
    .....
    .....
End Enum
```

Una declaración Enum consta de las siguientes partes:

- **Nombre.** El identificador que se asigna a la enumeración.
- **Miembro.** Identificador para cada elemento integrante de la enumeración.
- **Valor.** Opcional. Valor de tipo Long para un miembro de la enumeración. Si este valor no se proporciona, el primer miembro tendrá el valor cero, y los siguientes tendrán el valor del miembro anterior más uno.

Los tipos enumerados deben crearse en la zona de declaraciones del módulo. Son públicos por defecto, pudiendo acceder a ellos desde cualquier punto de la aplicación. Aunque un tipo enumerado toma valores por defecto, podemos asignar valores desde uno a todos los miembros del tipo, pudiendo comenzar las asignaciones por cualquiera de los miembros.

El Código fuente 82, declara un tipo enumerado con los valores por defecto.

```
Public Enum Impuestos
    Sociedades ' 0
    Patrimonio ' 1
    IRPF ' 2
    IVA ' 3
End Enum
```

Código fuente 82

En el Código fuente 83, asignamos valores comenzando por un miembro que no sea el primero.

```
Public Enum Impuestos
    Sociedades ' 0
    Patrimonio ' 1
    IRPF = 16 ' 16
    IVA ' 17
End Enum
```

Código fuente 83

En el Código fuente 84, realizamos dos asignaciones con valores negativos y positivos.

```
Public Enum Impuestos
    Sociedades = -6 ' -6
    Patrimonio ' -5
    IRPF = 208 ' 208
    IVA ' 209
End Enum
```

Código fuente 84

El modo de uso de una enumeración dentro de un procedimiento, pasa por declarar una variable del tipo correspondiente a la enumeración. En el momento de asignar un valor a dicha variable, aparecerá una lista con los valores disponibles, correspondientes a los miembros de la enumeración.



Figura 68. Asignación de valores desde una enumeración.

En este punto, el programador puede seleccionar un valor de la lista, o bien asignar otro diferente, siempre que sea de tipo Long.

El uso de enumeraciones presenta la ventaja de que podemos agrupar valores que tengan aspectos comunes, por ejemplo, los tipos de IVA.

```
Public Enum TiposIVA
    Reducido = 6
    Ordinario = 16
    Incrementado = 32
End Enum
```

Código fuente 85

También se puede conseguir este objetivo empleando constantes, pero no lograremos el nivel de agrupación de valores que proporciona una enumeración.

Una ventaja inmediata del uso de enumeraciones podemos verla al utilizar la función MsgBox(). En VB hay definida una enumeración denominada VbMsgBoxResult, que contiene los valores de botones que puede mostrar la llamada a esta función. De esta forma, en lugar de utilizar una variable numérica para almacenar el resultado de la llamada a MsgBox(), podemos utilizar una variable del tipo enumerado, tal y como se muestra en el Código fuente 86.

```
Private Sub Form_Load()
Dim l1Respuesta As VbMsgBoxResult
l1Respuesta = MsgBox("¿Comprobar valor?", vbYesNoCancel, "Aviso")
If l1Respuesta = vbCancel Then
    MsgBox "Operación cancelada"
End If
End Sub
```

Código fuente 86

Comprobación de tipos

Cuando en Visual Basic asignamos valores entre variables de distinto tipo, el lenguaje se encarga de realizar las comprobaciones oportunas de tipos de dato, convirtiendo el valor a asignar al tipo de la variable que lo recibe, como vemos en el Código fuente 87.

```
Public Sub Main()
Dim lnNumero As Integer
Dim lcNombre As String
lnNumero = 2000
' al asignar un número a una variable
' de cadena, se convierte el número a
' cadena
lcNombre = lnNumero ' "2000"
' -----
lcNombre = "434"
' al asignar una cadena a una variable numérica
' se convierte la cadena a número
lnNumero = lcNombre ' 434
End Sub
```

Código fuente 87

En este ejemplo no tenemos mayores problemas porque al asignar una cadena que contiene números a una variable de tipo numérico, se realiza una conversión automática de tipo. Sin embargo, si intentamos asignar una cadena que no puede ser convertida a número se producirá el denominado error de tipos. Veámoslo en el Código fuente 88 y en la Figura 69.

```
lcNombre = "mesa"
lnNumero = lcNombre ' error de tipos
```

Código fuente 88



Figura 69. Error de tipos.

Para prevenir este problema, el lenguaje nos provee de un conjunto de funciones que nos servirán para comprobar el tipo de dato de la expresión pasada como parámetro, ayudándonos a evitar el error de conversión.

- IsNumeric().

Sintaxis:

`IsNumeric(Expression)`

Devuelve un valor lógico que indica si la expresión pasada como parámetro se puede convertir o no a número.

- IsDate().

Sintaxis:

`IsDate(Expression)`

Devuelve un valor lógico que indica si la expresión pasada como parámetro se puede convertir o no a fecha.

- TypeName().

Sintaxis:

`TypeName(variable)`

Devuelve una cadena que informa del tipo de dato de la variable pasada como parámetro. Dicha cadena puede contener los siguientes valores: Byte, Integer, Long, Single, Double, Currency, Decimal, Date, String, Boolean, Error, Empty, Null, Object.

- VarType().

Sintaxis:

`VarType(variable)`

Devuelve un número indicando el tipo de dato contenido en el parámetro. Estos números están disponibles a través de un conjunto de constantes predefinidas para facilitar su identificación: vbInteger, vbString, vbLong, etc.

Mediante estas funciones, el anterior caso de error podemos evitarlo como se indica en el Código fuente 89.

```
Public Sub Main()
Dim lnNumero As Integer
Dim lcNombre As String
lcNombre = "mesa"
' comprobar primero el tipo
' de dato a asignar a la variable
If IsNumeric(lcNombre) Then
    lnNumero = lcNombre
Else
    MsgBox "No se pudo asignar la variable. No es un número", , ""
End If
End Sub
```

Código fuente 89

Funciones propias del lenguaje

Esta parte del lenguaje consiste en un conjunto más o menos extenso de rutinas de apoyo al programador, que le sirven de ayuda en el trabajo habitual de desarrollo. Visual Basic dispone de un amplio número de funciones de muy diferente tipo, algunas de las cuales se comentan a continuación. Recomendamos al lector, que dado el gran número de funciones, consulte el sistema de ayuda del lenguaje, ya que incluirlas todas aquí alargaría en exceso este apartado.

Numéricas

- **Abs(nNum).** Devuelve el valor absoluto del número pasado como parámetro.

```
lnNum = Abs(32.657) ^ 33
```

Código fuente 90

- **Int(nNum) – Fix(nNum).** Devuelven la parte entera del número pasado como parámetro. La diferencia entre ambas funciones es que con números negativos Int() devuelve el entero negativo menor o igual que nNum, y Fix() devuelve el entero negativo mayor o igual que nNum.

```
lnNum = Int(-32.657) ^ -33
lnNum = Fix(-32.657) ^ -32
```

Código fuente 91

- **Sgn(nNum).** Devuelve un número indicando el signo del parámetro numérico. Si nNum es positivo devuelve 1, si es negativo devuelve -1, y si es cero devuelve 0.

```
1nNum = Sgn(21) ' 1
1nNum = Sgn(-21) ' -1
1nNum = Sgn(0) ' 0
```

Código fuente 92

- **Round(xExpresion [, nDecimales]).** Redondea la expresión numérica xExpresion, pasada como parámetro, con la precisión indicada en nDecimales.

```
1dNumero = Round("15,80") ' 16
1dNumero = Round(32.2135, 3) ' 32,214
```

Código fuente 93

- **Randomize nNum.** Prepara el generador de número aleatorios basándose en el número pasado como parámetro, si no se pasa el parámetro, se usa el valor del reloj del sistema como número base. Una vez invocado Randomize, se utilizará la función Rnd() para generar números aleatorios.
- **Rnd(nNum).** Devuelve un número aleatorio de tipo Single, el parámetro nNum influirá en el modo de generar el número aleatorio basándose en una secuencia de números.

Si nNum es menor que cero, se devuelve el mismo número siempre. Si es mayor que cero o no se usa, se devuelve el siguiente número de la secuencia. Si es cero devuelve el último número generado.

- **Sqr(nNum).** Devuelve un número de tipo Double resultado de calcular la raíz cuadrada del número pasado como parámetro.

```
1dNum = Sqr(8) ' 2,82842712474619
```

Código fuente 94

- **Exp(nNum).** Devuelve un número de tipo Double resultado de elevar *e* a una potencia.

```
1dNum = Exp(12) ' 162754,791419004
```

Código fuente 95

- **Log(nNum).** Devuelve un número de tipo Double que especifica el logaritmo de un número.

```
1dNum = Log(16) ' 2,77258872223978
```

Código fuente 96

Cadenas de caracteres

- Len(cCadena). Devuelve un número Long con la longitud de caracteres de una cadena, incluidos los espacios en blanco.

También sirve para averiguar los bytes necesarios para almacenar una variable.

```
1lCad = Len(" hola ") ' 6
```

Código fuente 97

- Space(nEspacios). Devuelve una cadena de espacios en blanco según el número pasado como parámetro.

```
1cCadena = Space(5) & "hola" ' "hola"
```

Código fuente 98

- String(nNum, cCaracter). Devuelve una cadena de caracteres repetidos iguales a cCaracter, un número nNum de veces.

```
1cCadena = String(10, "Z") ' "ZZZZZZZZZZ"
```

Código fuente 99

- StrComp(cCadena1, cCadena2 [,nTipoCompara]). Devuelve un número Variant Integer con el resultado de la comparación entre dos cadenas de caracteres.
 - cCadena1, cCadena2. Cadenas a comparar.
 - nTipoCompara. Constante que indica el tipo de comparación a efectuar; la Tabla 21 muestra los valores disponibles.

Constante	Valor	Descripción
vbUseCompareOption	-1	Realiza la comparación según el valor de Option Compare
vbBinaryCompare	0	Realiza una comparación binaria

VbTextCompare	1	Realiza una comparación de texto
VbDatabaseCompare	2	Realiza la comparación según los valores de la base de datos. Sólo se puede usar con Access.

Tabla 21. Constantes de comparación

Si no se utiliza este parámetro, se tomará el valor establecido en Option Compare.

Los valores devueltos pueden ser:

- -1. cCadena1 es menor que cCadena2.
- 0. cCadena1 es igual que cCadena2.
- 1. cCadena1 es mayor que cCadena2.
- Null. Alguna de las dos cadenas es Null.

Veamos un ejemplo en el Código fuente 100.

```
Public Sub Main()
    Dim lcCorta As String
    Dim lcLarga As String
    Dim lnResulta As Integer

    lcCorta = "es la pequeña"
    lcLarga = "esta cadena es mayor"
    lnResulta = StrComp(lcCorta, lcLarga) ' -1

End Sub
```

Código fuente 100

- InStr([nInicio,] cCadenaBuscar, cCadenaBuscada[, nTipoCompara]). Devuelve un valor Variant Long que indica la primera ocurrencia de una cadena dentro de otra. Los parámetros de esta función son los siguientes:
 - nInicio. Indica el carácter de comienzo de la búsqueda, si no se especifica comienza por el primero.
 - cCadenaBuscar. Cadena en la que se realiza la búsqueda.
 - cCadenaBuscada. Cadena que se busca dentro de cCadenaBuscar.
 - nTipoCompara. Constante que indica el tipo de comparación a efectuar (ver valores en función StrComp()).

Los valores devueltos son los siguientes:

- Null. Si una o ambas cadenas son Null.

- 0. Si cCadenaBuscar es longitud cero, o no se encuentra cCadenaBuscada, o nInicio es mayor que la longitud de cCadenaBuscar.
- nInicio. Cuando el valor de cCadenaBuscada es una cadena de longitud cero.
- Posición encontrada. Cuando cCadenaBuscada se encuentre dentro de cCadenaBuscar.

Vemos un ejemplo en el Código fuente 101

```
lcCadenaBuscar = "El mueble está repleto"
l1Posicion = InStr(2, lcCadenaBuscar, "mueble") ' 4
```

Código fuente 101

- InStrRev(cCadenaBuscar, cCadenaBuscada[, nInicio[, nTipoCompara]]). Al igual que en InStr(), esta función busca una cadena dentro de otra, pero comenzando la búsqueda desde el final de cCadenaBuscar. Nótese que la sintaxis varía ligeramente con respecto a InStr(), aunque la descripción de los parámetros es la misma.
- Left(cCadena, nLongitud). Extrae comenzando por la parte izquierda de cCadena, una subcadena de nLongitud de caracteres.

```
lcCadena = Left("Efectivamente", 4) ' "Efec"
```

Código fuente 102

- Right(cCadena, nLongitud). Extrae comenzando por la parte derecha de cCadena, una subcadena de nLongitud de caracteres.

```
lcCadena = Right("Efectivamente", 4) ' "ente"
```

Código fuente 103

- Mid(cCadena, nInicio [, nCaractExtraer]). Extrae de cCadena, una subcadena de nCaractExtraer caracteres, comenzando en la posición nInicio. Si cCadena es Null, esta función devuelve Null. Si nInicio es mayor que el número de caracteres contenidos en cCadena, se devuelve una cadena de longitud cero. Si no se especifica nCaractExtraer, se devuelven todos los caracteres desde nInicio hasta el final de cCadena.

```
lcCadenaExtraer = "El bosque encantado"
lcCadena = Mid(lcCadenaExtraer, 11, 7) ' "encanta"
```

Código fuente 104

- Mid(varCadena, nInicio[, nCaractReemplazar]) = cCadenaSust / (Instrucción). Mid utilizado como instrucción, reemplaza una cantidad nCaractReemplazar de caracteres dentro de una variable varCadena, utilizando una cadena cCadenaSust y comenzando en la posición nInicio de varCadena.

```
lcCadenaExtraer = "El bosque encantado"
Mid(lcCadenaExtraer, 4, 5) = "teclado" ' "El teclae encantado"
```

Código fuente 105

- Replace(cCadena, cEncontrar, cReemplazar [, nInicio [, nNumCambios [, nTipoCompara]]]). Busca una subcadena dentro de otra cadena, efectuando una sustitución por otra subcadena, un número determinado de veces.
 - cCadena. Cadena en la que se va a realizar la sustitución.
 - cEncontrar. Cadena a sustituir.
 - cReemplazar. Cadena de reemplazo.
 - nInicio. Posición de cCadena desde la que se va a empezar la búsqueda, por defecto es 1.
 - nNumCambios. Número de sustituciones a realizar, el valor predeterminado es -1, que indica que se realizarán todos los reemplazos posibles.
 - nTipoCompara. Constante que indica el tipo de comparación a efectuar (ver valores en función StrComp()).

El resultado es una nueva cadena con las sustituciones realizadas. En el caso de no obtener el resultado esperado, los valores devueltos pueden ser los siguientes:

- Cadena vacía. Si cCadena está vacía o nInicio es mayor que su longitud.
- Un error. Si cCadena es Null.
- Copia de cCadena. Si cEncontrar está vacía.
- Copia de cCadena con las ocurrencias de cEncontrar eliminadas. Si cReemplazar está vacía.
- Copia de cCadena. Si nNumCambios es 0.

```
Public Sub Main()
  Dim lcCadena As String
  Dim lcResulta As String

  lcCadena = "En el bosque encantado habita un hado"
  lcResulta = Replace(lcCadena, "ado", "cambio")
  ' valor de lcResulta:
  ' "En el bosque encantcambio habita un hcambio"

End Sub
```

Código fuente 106

- StrReverse(cCadena). Invierte el orden de los caracteres de cCadena, devolviendo el resultado en otra cadena. Si cCadena es Null, se produce un error.

```
lcResulta = StrReverse("A ver quien entiende esto")
' Resultado de StrReverse():
' otse edneitne neiug rev A
```

Código fuente 107

- Trim(cCadena), LTrim(cCadena), RTrim(cCadena). Estas funciones eliminan los espacios en blanco en la parte izquierda de la cadena pasada como parámetro (LTrim), en la parte derecha (RTrim), o en ambos lados (Trim).

```
lcCadenaEspacios = " probando "
lcCadenaNoEsp = LTrim(lcCadenaEspacios) ' "probando "
lcCadenaNoEsp = RTrim(lcCadenaEspacios) ' " probando"
lcCadenaNoEsp = Trim(lcCadenaEspacios) ' "probando"
```

Código fuente 108

- LCase(cCadena), UCase(cCadena). Estas funciones convierten la cadena pasada como parámetro a minúsculas (LCase) o a mayúsculas (UCase).

```
lcCadena = "La Torre"
lcCadMayMin = LCase(lcCadena) ' "la torre"
lcCadMayMin = UCase(lcCadena) ' "LA TORRE"
```

Código fuente 109

- StrConv(cCadena, nConvert, LCID). Convierte la cadena cCadena en función del valor de nConvert, que puede ser un número o una constante de VB. Entre estas constantes, podemos destacar:

- vbUpperCase. Convierte la cadena a mayúsculas.
- vbLowerCase. Convierte la cadena a minúsculas.
- vbProperCase. Convierte la primera letra de cada palabra a mayúsculas.
- vbUnicode. Convierte la cadena a Unicode.
- vbFromUnicode. Convierte una cadena Unicode a la página de códigos del sistema.

Es posible especificar el identificador de la versión local del sistema, utilizando el parámetro LCID; si no se utiliza, se empleará el identificador por defecto.

```

lcCadena = "la torre de londres"
lcCadMayMin = StrConv(lcCadena, vbLowerCase) ' "la torre de londres"
lcCadMayMin = StrConv(lcCadena, vbUpperCase) ' "LA TORRE DE LONDRES"
lcCadMayMin = StrConv(lcCadena, vbProperCase) ' "La Torre De Londres"

```

Código fuente 110

- Format(xExpresión[, xFormato[, nPrimerDíaSemana[, nPrimerDíaAño]]]). Formatea la expresión xExpresión pasada como parámetro con un nombre o cadena de formato especificado en xFormato y la devuelve en una cadena.

El parámetro nPrimerDíaSemana es una constante de la Tabla 22 que indica el primer día de la semana.

Constante	Valor	Valor día semana
vbUseSystem	0	Utiliza el valor de API NLS.
VbSunday	1	Domingo (predeterminado)
VbMonday	2	Lunes
VbTuesday	3	Martes
vbWednesday	4	Miércoles
VbThursday	5	Jueves
VbFriday	6	Viernes
VbSaturday	7	Sábado

Tabla 22. Constantes para el parámetro nPrimerDíaSemana.

El parámetro nPrimerDíaAño es una constante de la Tabla 23 que indica el primer día del año:

Constante	Valor día año
vbUseSystem	Utiliza el valor de API NLS
vbFirstJan1	Valor por defecto. Empieza en la semana donde está el 1 de enero
vbFirstFourDays	Empieza en la primera semana del año que tenga al menos cuatro días
vbFirstFullWeek	Empieza en la primera semana completa del año

Tabla 23. Constantes para el parámetro nPrimerDíaAño.

- Nombres de formato para fecha:
 - General Date. Muestra una fecha en formato estándar dependiendo de la configuración del sistema, ejemplo: "10/05/98".
 - Long Date. Muestra la fecha en formato completo, ejemplo: "domingo 10 de mayo de 1998".
 - Medium Date. Muestra el formato de fecha mediana en función del idioma seleccionado en el sistema. Las pruebas efectuadas en este modo para la fecha 10/05/98 devolvieron la siguiente cadena: "10-may-aa", por lo que suponemos que debe existir algún bug en el tratamiento de fechas con este tipo de formato.
 - Short Date. Muestra la fecha en el formato corto del sistema, ejemplo: "10/05/98".
 - Long Time. Muestra la hora en el formato largo, ejemplo: "17:14:52".
 - Medium Time. Muestra la hora en el formato mediano del sistema, ejemplo: 17:15:40 pasa a "05:15".
 - Short Time. Muestra la hora en el formato corto del sistema, ejemplo: 17:16:39 pasa a "17:16".
- Caracteres utilizados en el formateo de fechas:
 - : . Separador de hora.
 - / . Separador de fecha.
 - c. Muestra la fecha en formato dd/mm/aa y la hora en formato hh:mm:ss.
 - d. Muestra el número de día sin cero a la izquierda, ejemplo: 7/5/98 devolvería "7".
 - dd. Muestra el número de día con cero a la izquierda, ejemplo: 7/5/98 devolvería "07".
 - ddd. Muestra el nombre abreviado del día, ejemplo: 7/5/98 devolvería "jue".
 - dddd. Muestra el nombre completo del día, ejemplo: 7/5/98 devolvería "jueves".
 - dddddd. Muestra la fecha en formato completo, ejemplo: 7/5/98 devolvería "jueves 7 de mayo de 1998".
 - w. Muestra un número correspondiente al día de la semana tomando el domingo como base (1), ejemplo: 7/5/98 devolvería "5".
 - ww. Muestra un número que corresponde a la semana del año de la fecha, ejemplo: 7/5/98 devolvería "19".
 - m. Muestra el número de mes sin cero a la izquierda, ejemplo: 7/5/98 devolvería "5".

- mm. Muestra el número de mes con cero a la izquierda, ejemplo: 7/5/98 devolvería "05".
- mmm. Muestra el nombre del mes en forma abreviada, ejemplo: 7/5/98 devolvería "may".
- mmmm. Muestra el nombre completo del mes, ejemplo: 7/5/98 devolvería "mayo".
- q. Muestra el número de trimestre del año a que corresponde la fecha, ejemplo: 7/5/98 devolvería "2".
- y. Muestra el número de día del año de la fecha, ejemplo: 7/5/98 devolvería "127".
- yy. Muestra un número de dos dígitos para el año, ejemplo: 7/5/98 devolvería "98".
- yyyy. Muestra un número de cuatro dígitos para el año, ejemplo: 7/5/98 devolvería "1998".
- h. Muestra un número correspondiente a la hora, sin cero a la izquierda, ejemplo: 08:05:37 devolvería "8".
- hh. Muestra un número correspondiente a la hora, con cero a la izquierda, ejemplo: 08:05:37 devolvería "08".
- n. Muestra un número correspondiente a los minutos, sin cero a la izquierda, ejemplo: 08:05:37 devolvería "5".
- nn. Muestra un número correspondiente a los minutos, con cero a la izquierda, ejemplo: 08:05:37 devolvería "05".
- s. Muestra un número correspondiente a los segundos, sin cero a la izquierda, ejemplo: 08:05:03 devolvería "3".
- ss. Muestra un número correspondiente a los segundos, con cero a la izquierda, ejemplo: 08:05:03 devolvería "03".
- tttt. Muestra la hora con el formato completo, ejemplo: "08:05:03".
- AM/PM am/pm A/P a/p. Muestran la franja horaria de la hora a formatear de la misma forma que el nombre del formato, ejemplos: 18:10:42 con AM/PM devolvería PM, con a/p devolvería p, y así con cada nombre de estos formatos.

Algunos ejemplos de formateo de fechas serían los que aparecen en el Código fuente 111.

```
lvtFecha = #5/7/98#
lcCadena = Format(lvtFecha, "Long Date") ' "jueves 7 de mayo de 1998"
lcCadena = Format(lvtFecha, "ddd dd/m/yyyy") ' "jue 07/5/1998"
```

Código fuente 111

- Nombres de formato para números:

- General Number. Muestra el número sin separadores de millar, para este y los demás ejemplos numéricos utilizaremos el número 125401.25 definido como Double, en este caso concreto devuelve "125401,25".
 - Currency. Muestra el número en formato de moneda, el ejemplo devuelve "125.401 Pts".
 - Fixed. Muestra como mínimo un dígito a la izquierda y dos a la derecha del separador decimal, el ejemplo devuelve "125401,25".
 - Standard. Muestra el separador de millar y la precisión decimal de Fixed, el ejemplo devuelve "125.401,25".
 - Percent. Muestra el número multiplicado por 100, con el signo de porcentaje y dos dígitos decimales, el ejemplo devuelve "12540125,00%".
 - Scientific. Muestra el número mediante notación científica, el ejemplo devuelve "1,25E+05".
 - Yes/No. Muestra "No" si el número es 0, en otro caso devuelve "Si" como el ejemplo.
 - True/False. Muestra "Falso" si el número es 0, en otro caso devuelve "Verdadero" como el ejemplo.
 - On/Off. Muestra "Desactivado" si el número es 0, en otro caso devuelve "Activado" como el ejemplo.
- Caracteres utilizados en el formateo de números:

- 0 . Marcador de dígito. Si la expresión a formatear tiene un dígito en la posición donde aparece el cero se muestra el dígito, en otro caso se muestra cero.

```
Format(" 123", "0###")  ' "0123"
```

Código fuente 112

- # . Marcador de dígito. Si la expresión a formatear tiene un dígito en la posición donde aparece #, se muestra el dígito, en otro caso no se muestra nada, ejemplo: "23" devolvería "23".

```
Format(" 23", "####")  ' "23"
```

Código fuente 113

- "." .Marcador de decimales; en función de la configuración del país se mostrará un punto o coma; si se especifica una precisión menor a mostrar, se efectuará un redondeo automático, ejemplo: "54.38" devolvería "54,4".

```
Format(54.38, "##.##") ' "54,4"
```

Código fuente 114

- % . Marcador de porcentaje; multiplica el número por 100 y le añade este carácter.

```
Format(28, "#####%") ' "2800%"
```

Código fuente 115

- "," . Marcador de millar; en función de la configuración del país se mostrará un punto o coma.

```
Format(2800, "###,###") ' "2.800"
```

Código fuente 116

- E- E+ e- e+ . Notación científica; si el número a formatear tiene a la derecha de un signo de precisión científica, un marcador 0 o # se mostrará en formato científico.

```
Format(5, "##E+##") ' "500E-2"
```

Código fuente 117

- - + \$ (). Literales, estos caracteres son mostrados en cualquier lugar de la cadena de formato donde sean situados.

```
Format(45000, "$#####(##)") ' "$450 (00)"
```

Código fuente 118

- \. Muestra el carácter situado a continuación de la barra inversa.

```
Format(45000, "#@\#\#\#\^\") ' "4@5000^"
```

Código fuente 119

- Caracteres utilizados en el formateo de cadenas:

- **@.** Marcador de carácter. Si la cadena de formato tiene un carácter en el lugar donde aparece este marcador, se muestra el carácter, en caso contrario, se muestra un espacio en blanco. Los caracteres de la cadena a formatear se rellenan por defecto de derecha a izquierda, excepto cuando se usa el carácter ! en la cadena de formato.

```
Format("hola", "@@@@@@@@") ' " hola"
```

Código fuente 120

- **&.** Marcador de carácter. Si la cadena de formato tiene un carácter en el lugar donde aparece este marcador, se muestra el carácter, en caso contrario, no se muestra nada.

```
Format("hola", "&&&&&&&&&") ' "hola"
```

Código fuente 121

- **<.** Convierte los caracteres de la cadena a minúscula.

```
Format("LETRA PEQUEÑA", "<") ' "letra pequeña"
```

Código fuente 122

- **>.** Convierte los caracteres de la cadena a mayúscula.

```
Format("letra grande", ">") ' "LETRA GRANDE"
```

Código fuente 123

- **!.** Rellena los caracteres de la cadena de izquierda a derecha.

```
Format("hola", "!!!!!!!!") ' "hola "
```

Código fuente 124

- **FormatNumber(xExpresión[, nDigitosDecimales [, nIndDigito [, nParentesisNeg [, nAgrupar]]]]).** Formatea como un número un valor.
 - **xExpresión.** Valor a formatear.
 - **nDigitosDecimales.** Número que indica la cantidad de decimales a mostrar en el número formateado. Si este valor es -1, se utilizará la configuración regional del sistema.

- **nIndDigito.** Constante de estado, que indica si en los valores fraccionales se mostrará un cero indicativo.
- **nParentesisNeg.** Constante de estado, que indica si se mostrará el número formateado entre paréntesis, cuando este sea negativo.
- **nAgrupar.** Constante de estado, que indica si el número se agrupará utilizando un delimitador de grupo. Los valores para las constantes de estado se muestran en la Tabla 24.

Constante	Valor	Descripción
VbTrue	-1	True
vbFalse	0	False
VbUseDefault	-2	Valor por defecto

Tabla 24. Constantes para valores de estado.

```
lcResulta = FormatNumber(-15.3255, 5, , vbTrue) ' "(15,32550)"
```

Código fuente 125

- **FormatPercent(xExpresión[, nDigitosDecimales [, nIndDigito [, nParentesisNeg [, nAgrupar]]]]).** Formatea una expresión como un porcentaje, incorporando el carácter "%". Los parámetros empleados, son los mismos que los utilizados en la función FormatNumber().

```
lcResulta = FormatPercent(128) ' "12.800,00%"
```

Código fuente 126

- **FormatCurrency(xExpresión[, nDigitosDecimales [, nIndDigito [, nParentesisNeg [, nAgrupar]]]]).** Formatea una expresión como valor de divisa, empleando el símbolo de divisa establecido en la configuración regional del panel de control del sistema. Los parámetros empleados, son los mismos que los utilizados en la función FormatNumber().

```
lcResulta = FormatCurrency(128) ' "128 Pts"
```

Código fuente 127

- **Asc(cCadena).** Devuelve un número Integer que corresponde al código del primer carácter de cCadena.

```
Asc ("Mesa") ' 77
```

Código fuente 128

- Chr(lCódigoCarácter). Devuelve un carácter correspondiente al código pasado como parámetro a esta función.

```
Chr (77) ' "M"
```

Código fuente 129

- Join(cCadenas() [, cDelimitador]). Devuelve una cadena formada por los elementos contenidos en un array.
 - cCadenas(). Array unidimensional que contiene las cadenas para formar el resultado de esta función.
 - cDelimitador. Cadena utilizada para separar las cadenas que componen el resultado. Por defecto se utiliza el espacio en blanco. Si se emplea una cadena de longitud cero, la cadena resultante contendrá todas las subcadenas unidas.

```
Public Sub Main()
Dim lcCadenas(1 To 5) As String
Dim lcResulta1 As String
Dim lcResulta2 As String
Dim lcResulta3 As String
' asignar valores al array de cadenas
lcCadenas(1) = "Aquí"
lcCadenas(2) = "usamos"
lcCadenas(3) = "una"
lcCadenas(4) = "nueva"
lcCadenas(5) = "función"
' utilizar Join() de diversas formas
lcResulta1 = Join(lcCadenas()) 'Aquí usamos una nueva función
lcResulta2 = Join(lcCadenas(), "==") 'Aquí==usamos==una==nueva==función
lcResulta3 = Join(lcCadenas(), "") 'Aquíusamosunanuevafunción
End Sub
```

Código fuente 130

- Split(cCadena [, cDelimitador [, nNumCadenas [, nTipoCompara]]]). Esta función es la inversa de Join(). Toma una cadena pasada como parámetro, y en función de los valores de configuración de esta función, devuelve un array cuyos elementos son subcadenas que forman la cadena original.
 - cCadena. Cadena que contiene las subcadenas que van a ser introducidas en el array. También se puede incluir el carácter o caracteres que actuarán como delimitador. Si la longitud de esta cadena es cero, el array devuelto no contendrá elementos.

- **cDelimitador.** Carácter empleado para identificar las cadenas a incluir en el array. Por defecto se utiliza el espacio en blanco. Si se emplea una cadena de longitud cero, el array resultante tendrá como único elemento, el valor de **cCadena**.
- **nNumCadenas.** Número de subcadenas a devolver. El valor -1 significa que se devolverán todas las subcadenas posibles.
- **nTipoCompara.** Constante que indica el tipo de comparación a efectuar (ver valores en función **StrComp()**).

```
Public Sub Main()
Dim lcOrigen As String
Dim lcValores() As String
Dim lnInd As Integer
lcOrigen = "Traspasar estos valores a un array"
lcValores = Split(lcOrigen)
For lnInd = 0 To UBound(lcValores)
    MsgBox lcValores(lnInd), , "Elemento " & lnInd & " del array"
Next
End Sub
```

Código fuente 131

Fecha y hora.

- **Date.** Devuelve la fecha actual del sistema.

```
ldtFecha = Date ' 20/03/98
```

Código fuente 132

- **Date = varFecha / Instrucción.** Establece la fecha actual del sistema mediante un valor de tipo fecha **varFecha**.

```
ldtFecha = #11/21/98# ' 21 de Noviembre de 1998
Date = ldtFecha
```

Código fuente 133

- **Time.** Devuelve la hora actual del sistema.

```
ldtHora = Time ' 11:49:55
```

Código fuente 134

- **Time = varHora / Instrucción.** Establece la hora actual del sistema mediante un valor asignado en **varHora**, este valor puede ser una cadena reconocible con formato de hora, por lo que **Time** hará la conversión automática a tipo fecha/hora, en caso contrario, se producirá un error.

```
ldtHora = #10:30:21#
Time = ldtHora
```

Código fuente 135

- Now. Devuelve la fecha y hora actual del sistema.

```
ldtFechaHora = Now ' 20/03/98 12:05:21
```

Código fuente 136

- FormatDateTime(xExpresion[, nFormato]). Formatea una expresión, retornando un valor de fecha/hora.

Vemos un ejemplo en el Código fuente 137

- xExpresion. Expresión a formatear.
- nFormato. Constante que especifica el formato que se aplicará a xExpresion, según la Tabla 25. Si no se utiliza este parámetro, se aplicará por defecto vbGeneralDate.

Constante	Valor	Descripción
vbGeneralDate	0	Muestra una fecha y/o hora en formato estándar, dependiendo de la configuración del sistema, ejemplo: "10/05/98".
vbLongDate	1	Muestra la fecha en formato completo, ejemplo: "domingo 10 de mayo de 1998".
vbShortDate	2	Muestra la fecha en el formato corto del sistema, ejemplo: "10/05/98".
vbLongTime	3	Muestra la hora en el formato largo, ejemplo: "17:14:52".
vbShortTime	4	Muestra la hora en el formato corto del sistema, ejemplo: 17:16:39 pasa a "17:16".

Tabla 25. Constantes de formato para fecha/hora.

```
' el resultado de este ejemplo en ambos casos es:
'"viernes 20 de noviembre de 1998"
cResulta = FormatDateTime("20-11-98", vbLongDate)
lcResulta = FormatDateTime(#11/20/1998#, vbLongDate)
```

Código fuente 137

- MonthName(nMes[, bAbreviado]). Devuelve el nombre del mes, en función del orden numérico que ocupa en el calendario.
 - nMes. Número correspondiente al mes: 1-Enero, 2-Febrero, etc.
 - bAbreviado. Valor lógico, que mostrará el nombre del mes abreviado en el caso de que sea True.

```
lcResulta = MonthName(4) ' abril
```

Código fuente 138

- WeekdayName(nDiaSemana[, bAbreviado [, nPrimerDiaSemana]]). Devuelve el nombre del día de la semana, en función del orden numérico que ocupa.
 - nDiaSemana. Número del día de la semana. El nombre devuelto, depende del valor establecido en el parámetro nPrimerDiaSemana.
 - bAbreviado. Valor lógico, que mostrará el nombre del día abreviado en el caso de que sea True.
 - nPrimerDiaSemana. Constante que indica el primer día de la semana (consultar la tabla para estas constantes en la función Format()).

```
lcDia = WeekdayName(6) ' sábado
```

Código fuente 139

DateAdd(cIntervalo, nNumIntervalo, dtFecha). Suma o resta un intervalo de tiempo nNumIntervalo representado en cIntervalo a la fecha dtFecha pasada como parámetro, devolviendo un valor de tipo fecha con el resultado. Si nNumIntervalo es positivo, se agrega tiempo a dtFecha, si es negativo se quita tiempo. El parámetro cIntervalo puede tener los valores de la Tabla 26, en función del espacio de tiempo a manejar. En el Código fuente 140 inicializamos una variable con una fecha y le aplicamos diferentes intervalos de tiempo.

Valor	Tipo de intervalo
yyyy	Año
Q	Trimestre
M	Mes
Y	Día del año
d	Día

w	Día de la semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

Tabla 26. Valores para el parámetro cIntervalo.

```

ldtFecha = #4/25/98#
' sumar a una fecha meses, trimestres y días
' del año. Restarle también meses
ldtNuevaFecha = DateAdd("m", 2, ldtFecha) ' 25/06/98
ldtNuevaFecha = DateAdd("m", -3, ldtFecha) ' 25/01/98
ldtNuevaFecha = DateAdd("q", 3, ldtFecha) ' 25/01/99
ldtNuevaFecha = DateAdd("y", 100, ldtFecha) ' 3/08/98

```

Código fuente 140

- DateDiff(cIntervalo, dtFecha1, dtFecha2 [, nPrimerDíaSemana [, nPrimerDíaAño]]). Calcula un intervalo de tiempo cIntervalo entre las fechas dtFecha1 y dtFecha2, devolviendo el resultado en un tipo Variant de valor Long.

Los valores disponibles para cIntervalo pueden consultarse en la función DateAdd(), y los valores para nPrimerDíaSemana y nPrimerDíaAño en la función Format().

En el Código fuente 141 se definen dos fechas y se realizan varias operaciones para calcular diferentes tipos de diferencias entre ellas.

```

' inicializar las variables con fechas
ldtFecha1 = #8/17/98#
ldtFecha2 = #4/21/99#
' calcular la diferencia de años
lvntDif = DateDiff("YYYY", ldtFecha1, ldtFecha2) ' 1
' calcular la diferencia trimestres
lvntDif = DateDiff("q", ldtFecha1, ldtFecha2) ' 3
' calcular la diferencia de semanas
lvntDif = DateDiff("ww", ldtFecha1, ldtFecha2) ' 35

```

Código fuente 141

- DatePart(cIntervalo, dtFecha [, nPrimerDíaSemana [, nPrimerDíaAño]]). Extrae el tipo de intervalo cIntervalo de una fecha dtFecha en forma de valor Variant Integer.

Los valores disponibles para nPrimerDíaSemana y nPrimerDíaAño pueden consultarse en la función Format().

En el Código fuente 142 vemos un fuente que extrae diferentes intervalos de una fecha.

```

' inicializar una variable de fecha
ldtFecha = #8/17/98#

' extraer el año
lvntParte = DatePart("yyyy", ldtFecha) ' 1998

' extraer el trimestre
lvntParte = DatePart("q", ldtFecha) ' 3

' extraer el día de la semana
lvntParte = DatePart("w", ldtFecha) ' 2

' extraer el día del año
lvntParte = DatePart("y", ldtFecha) ' 229

```

Código fuente 142

- Timer. Esta función devuelve un valor numérico Single con los segundos que han pasado desde la medianoche, se utiliza para medir el tiempo transcurrido desde el comienzo hasta el final de un proceso.

Vemos un ejemplo en el Código fuente 143

```

' tomar los segundos al iniciar
' el proceso
lfTiempoInicio = Timer
' comenzar el proceso
' .....
' tomar los segundos al finalizar
' el proceso
lfTiempoFin = Timer
' hallar la diferencia
lfTiempoTrans = lfTiempoFin - lfTiempoInicio

```

Código fuente 143

- DateSerial(nAño, nMes, nDía). Compone una fecha basándose en los números pasados como parámetro.

```

ldtFecha = DateSerial(1990, 5, 28) ' 28/05/90

```

Código fuente 144

- DateValue(cFecha). Compone una fecha basándose en la cadena pasada como parámetro, esta cadena ha de tener un formato que pueda ser convertido a fecha, en caso contrario, se producirá un error.

```

ldtFecha = DateValue("15 feb 1985") ' 15/02/85

```

Código fuente 145

- **TimeSerial(nHora, nMinutos, nSegundos).** Crea un valor de hora con los números pasados como parámetro.

```
ldtHora = TimeSerial(18, 20, 35) ' 18:20:35
```

Código fuente 146

- **TimeValue(cHora).** Devuelve un valor de hora creado con la cadena cHora pasada como parámetro; esta cadena ha de tener un contenido convertible a hora, en caso contrario, ocurrirá un error.

```
ldtHora = TimeValue("3:40:10 PM") ' 15:40:10
```

Código fuente 147

5

Técnicas

Recursividad

Denominamos *recursividad* a la capacidad que tiene un procedimiento en un determinado lenguaje para llamarse a sí mismo.

Uno de los ejemplos más habituales de técnica recursiva consiste en el cálculo del factorial de un número. Para resolver este problema, debemos efectuar el producto de dicho número por todos los valores comprendidos en el rango de 1 al número a calcular.

Debido a que este tipo de subrutinas devuelven un valor, se codificará en un Function, como se muestra en el Código fuente 148.

```
Public Function Factorial(ByVal vnNumero As Integer) As Double
If vnNumero = 0 Then
    Factorial = 1
Else
    Factorial = vnNumero * Factorial(vnNumero - 1)
End If
End Function
```

Código fuente 148

Comprobemos, en el Código fuente 149, el resultado obtenido al calcular el factorial del número 5.

```
Public Sub Main()
    Dim l1NumFact As Long
    l1NumFact = Factorial(5)      ' 120
End Sub
```

Código fuente 149

Búsqueda de datos

Cuando tenemos una cierta cantidad de datos, se hace preciso el uso de un algoritmo que nos ayude a localizar uno de esos datos dentro del conjunto disponible.

Las búsquedas toman sentido al utilizar en el programa arrays o ficheros, los cuales, al poder disponer de un elevado número de datos, nos permiten localizar rápidamente la información deseada.

Existen dos algoritmos básicos de búsqueda de datos, que vamos a aplicar en ejemplos con arrays.

Lineal

Este algoritmo de búsqueda, consiste en posicionarse en el primer elemento del array e ir buscando uno a uno hasta encontrar el dato que queremos, o llegar al final del array si la búsqueda no ha tenido éxito. En el Código fuente 150 se muestra este tipo de búsqueda.

```
Public Sub Main()
    Dim lcCiudades(1 To 8) As String
    Dim lcBuscar As String
    Dim lnContador As Integer
    lcCiudades(1) = "Londres"
    lcCiudades(2) = "París"
    lcCiudades(3) = "Roma"
    lcCiudades(4) = "Venecia"
    lcCiudades(5) = "Tokio"
    lcCiudades(6) = "Madrid"
    lcCiudades(7) = "Sevilla"
    lcCiudades(8) = "Barcelona"
    ' pedir al usuario que introduzca
    ' el valor a buscar
    lcBuscar = InputBox("¿Que ciudad buscamos?", " ")
    ' buscar desde principio a fin del array
    For lnContador = LBound(lcCiudades) To UBound(lcCiudades)
        If lcCiudades(lnContador) = lcBuscar Then
            MsgBox "Ciudad encontrada: " & lcCiudades(lnContador), , _
                    "Posición: " & lnContador
            Exit For
        End If
    Next
End Sub
```

Código fuente 150

Este algoritmo sólo es recomendable aplicarlo cuando trabajamos con pequeñas cantidades de datos, ya que de lo contrario, la búsqueda tomaría un excesivo tiempo para completarse.

Binaria

En el caso de que el array esté ordenado, la búsqueda binaria proporciona un algoritmo más veloz para la localización de datos, como vemos en el Código fuente 151, en el que se explican los pasos a realizar para localizar un valor en el array.

```

Public Sub Main()
    ' - realizar búsqueda binaria de una letra
    ' en un array
    ' - tener precaución de que en los elementos
    ' del array no falte ninguna letra, que
    ' vayan todas consecutivas o se puede
    ' entrar en un bucle infinito
    Dim lcLetras(9) As String
    Dim lcLetraBuscada As String
    Dim lnPosicCentral As Integer
    Dim lnPosicPrimera As Integer
    Dim lnPosicUltima As Integer
    lcLetras(1) = "a"
    lcLetras(2) = "b"
    lcLetras(3) = "c"
    lcLetras(4) = "d"
    lcLetras(5) = "e"
    lcLetras(6) = "f"
    lcLetras(7) = "g"
    lcLetras(8) = "h"
    lcLetras(9) = "i"
    lcLetraBuscada = InputBox("Introducir letra a buscar")
    lnPosicPrimera = LBound(lcLetras)
    lnPosicUltima = UBound(lcLetras)
    ' si la letra no está en el array, finalizar
    If lcLetraBuscada < lcLetras(lnPosicPrimera) Or _
        lcLetraBuscada > lcLetras(lnPosicUltima) Then
        MsgBox "Letra fuera de intervalo"
        Exit Sub
    End If
    ' si la letra está en el intervalo
    ' comenzar a buscar
    Do
        ' calcular la posición central del array
        lnPosicCentral = (lnPosicPrimera + lnPosicUltima) / 2
        ' si se encuentra la letra, finalizar
        If lcLetras(lnPosicCentral) = lcLetraBuscada Then
            MsgBox "Letra encontrada, posición: " & lnPosicCentral
            Exit Do
        End If
        ' si no se ha encontrado la letra:
        ' si la letra es menor que la que está
        ' en la posición central del array, asignar
        ' como última posición la que hasta ahora
        ' era la central
        If lcLetraBuscada < lcLetras(lnPosicCentral) Then
            lnPosicUltima = lnPosicCentral
        Else
            ' si la letra es mayor que la que está
            ' en la posición central del array, asignar
            ' como primera posición la que hasta ahora

```

```

    ' era la central
    lnPosicPrimera = lnPosicCentral
End If
Loop
End Sub

```

Código fuente 151

Ordenación de datos

La ordenación es la técnica que nos permite clasificar un conjunto de datos mediante un algoritmo basado en una condición o valor.

Existen diversos algoritmos de ordenación, dependiendo de la cantidad de datos a ordenar y de la rapidez en realizar la ordenación. Cuanto mayor número de datos debamos ordenar, mejor tendrá que ser el algoritmo para que emplee el menor tiempo posible.

En el ejemplo que sigue, se ha utilizado el algoritmo denominado de *burbuja*. Esta técnica consiste en comparar los elementos contiguos en el array y si no están en orden, se intercambian sus valores. Pasemos, en el Código fuente 152, a la rutina que realiza la ordenación y que contiene las explicaciones oportunas en los diferentes pasos.

```

Public Sub Main()
    ' ordenar por método burbuja
    Dim lnNumeros(5) As Integer
    Dim lnIndice As Integer
    Dim lnIndiceBis As Integer
    Dim lnValorTemp As Integer
    ' crear array
    lnNumeros(1) = 20
    lnNumeros(2) = 4
    lnNumeros(3) = 12
    lnNumeros(4) = 25
    lnNumeros(5) = 6
    For lnIndice = 1 To UBound(lnNumeros) - 1
        For lnIndiceBis = 1 To UBound(lnNumeros) - 1
            ' si valor del elemento inferior es mayor que
            ' el elemento superior, intercambiar utilizando
            ' una variable puente
            If lnNumeros(lnIndiceBis) > lnNumeros(lnIndiceBis + 1) Then
                lnValorTemp = lnNumeros(lnIndiceBis)
                lnNumeros(lnIndiceBis) = lnNumeros(lnIndiceBis + 1)
                lnNumeros(lnIndiceBis + 1) = lnValorTemp
            End If
        Next
    Next
End Sub

```

Código fuente 152

Manejo de errores

Desgraciadamente, por muy bien que planifiquemos nuestro código, siempre aparecerán errores de muy diverso origen.

Tenemos por un lado los errores producidos durante la escritura del código, aunque por suerte VB está preparado para avisarnos cuando escribimos alguna sentencia de forma incorrecta. Si intentamos introducir algo como lo que aparece en el Código fuente 153.

```
Dim lcCadena As String
lcCadena = "Astral"
If lcCadena = "hola"
```

Código fuente 153

La línea que contiene el If provocará inmediatamente un aviso de error de sintaxis por falta de Then, que nos evitará problemas en ejecución, ya que rápidamente rectificaremos la línea y asunto resuelto.

Por otra parte están los errores en tiempo de ejecución, que provocan la interrupción del programa una vez que se producen. En este tipo de errores, la sintaxis es correcta, pero no la forma de utilizarla, como ocurre en el Código fuente 154.

```
Public Sub Main()
Dim lcCadena As String
Dim lnNum As Integer
lcCadena = "A"
' al intentar ejecutar esta línea
' se produce un error ya que intentamos
' asignar una cadena a una variable
' de tipo Integer
lnNum = lcCadena
End Sub
```

Código fuente 154

Y también tenemos los errores lógicos que quizás son los que más quebraderos de cabeza puedan ocasionarnos. En este tipo de errores, el código está correctamente escrito, produciéndose el problema por un fallo de planteamiento en el código, motivo por el cual por ejemplo, el control del programa no entra en un bucle porque una variable no ha tomado determinado valor; sale antes de tiempo de un procedimiento al evaluar una expresión que esperábamos que tuviera un resultado diferente, etc.

Por estos motivos, y para evitar en lo posible los errores lógicos, conviene hacer un detallado análisis de lo que deseamos hacer en cada procedimiento, ya que este tipo de errores pueden ser los que más tiempo nos ocupen en descubrirlos. Como nadie está libre de errores, VB nos proporciona el objeto genérico *Err* para el tratamiento de errores en la aplicación, y la sentencia *On Error* para activar rutinas de tratamiento de errores.

Err

El objeto Err se instancia automáticamente al iniciarse el programa y proporciona al usuario información de los errores producidos en el transcurso de la aplicación; tiene un ámbito global, es decir, podemos usarlo directamente en cualquier punto del programa. Un error puede ser generado por Visual Basic, un objeto o el propio programador. Cuando se produce el error, las propiedades del objeto se rellenan con información sobre el error ocurrido y se reinician a cero al salir del procedimiento, con el método *Clear()*, o con la instrucción *Resume*.

- Propiedades.
 - Number. Número de error de los establecidos en VB o proporcionado por el programador.
 - Source. Nombre del proyecto en ejecución.
 - Description. Cadena informativa sobre la causa del error.
 - HelpFile. Ruta del fichero de ayuda para el error.
 - HelpContext. Contiene un identificador para la ayuda contextual sobre el error.
 - LastDllError. Código de error del sistema en la última llamada a una DLL.
- Métodos.
 - Clear. Inicializa a cero o cadena vacía las propiedades del objeto Err.
 - Raise (nNumber, cSource, cDescription, cHelpFile, nHelpContext). Provoca un error durante la ejecución del programa, los parámetros a pasar coinciden con las propiedades del objeto explicadas anteriormente.

On Error

Esta sentencia activa o desactiva una rutina de manejo de errores, que es una parte de código incluida en una etiqueta. On Error tiene los siguientes modos de empleo:

On Error GoTo Etiqueta

Veamos un ejemplo en el Código fuente 155.

```
Public Sub Main()
Dim lcCadena As String
Dim lnNum As Integer
' definimos un controlador de errores
On Error GoTo CompErr
lcCadena = "A"
' la siguiente linea provoca un error
' por lo que el control pasa a la
' etiqueta que tiene el controlador
lnNum = lcCadena
lcCadena = "otro valor"
lnNum = 232
Exit Sub
CompErr:
' tomamos información del error generado
' mediante el objeto Err
MsgBox "Error: " & Err.Number & vbCrLf &
      "Descripción: " & Err.Description
End Sub
```

Código fuente 155

Si en un procedimiento que incluye un controlador de errores, no ponemos la instrucción Exit Sub antes de la etiqueta que da comienzo al controlador, en el caso de que no se produzca un error, al llegar a la etiqueta, el código en ella incluido será ejecutado, y como es lógico nosotros sólo queremos que ese código se ejecute únicamente cuando haya un error. Por ese motivo las rutinas de control de error se sitúan en una etiqueta al final de un procedimiento y antes del comienzo de la etiqueta se pone una sentencia Exit Sub para que el control del programa salga del procedimiento en el caso de que no existan errores.

Si queremos que el control de la aplicación vuelva al lugar donde se originó el error, tendremos que utilizar la instrucción Resume en alguna de sus variantes:

- Resume. Terminado de controlar el error, el programa vuelve a la línea en donde se originó dicho error. Hemos de tener cuidado de solucionar el error en la etiqueta de control de errores, ya que de lo contrario, podríamos entrar en un bucle infinito.

De esta forma, el controlador de errores del ejemplo anterior se podría modificar como se indica en el Código fuente 156.

```
CompErr:
MsgBox "Error: " & Err.Number & vbCrLf &
      "Descripción: " & Err.Description
lcCadena = "22"

Resume
```

Código fuente 156

Si en el Código fuente 156 no hiciéramos una nueva asignación a la variable lcCadena con un contenido que pueda convertirse a número, al volver al código del procedimiento se detectaría de nuevo el error y nos volvería a mandar al controlador de errores, entrando en un bucle sin salida.

- Resume Next. Lo veremos más adelante.
- Resume Etiqueta. El control pasa al código indicado en una etiqueta de código una vez haya finalizado el controlador de errores.

On Error Resume Next

Cuando se produzca un error, el control del programa pasará a la siguiente línea a la que ha provocado el error. Con esta forma de controlar errores podemos hacer un seguimiento del error en los lugares del procedimiento más susceptibles de que se produzca; también nos proporciona más estructuración, ya que no desvía mediante saltos el flujo de la aplicación.

Vemos un ejemplo en el Código fuente 157.

```
Public Sub Main()
Dim lcCadena As String
Dim lnNum As Integer
' definimos un controlador de errores
On Error Resume Next
```

```

lcCadena = "A"
' la siguiente línea provoca un error
lnNum = lcCadena
' comprobamos el objeto Err
' si el número de error es diferente
' de cero, es que hay un error
If Err.Number <> 0 Then
    MsgBox "Error: " & Err.Number & vbCrLf & _
        "Descripción: " & Err.Description

    ' inicializar el objeto error
    ' y solucionar el error
    Err.Clear
    lcCadena = "22"
End If
lnNum = lcCadena
End Sub

```

Código fuente 157

On Error GoTo 0

Desactiva el manejador de error que haya actualmente en el procedimiento, con lo cual, si se produce un error, el flujo del programa no pasará a la etiqueta con el controlador de errores.

Como se ha comentado al ver el objeto Err, es posible para el programador generar un error para una determinada situación en la aplicación, en el Código fuente 158 vemos como provocarlo.

```

Public Sub Main()
    ' inicializar variables y dar valor
    Dim lcCadena As String
    lcCadena = "Alfredo"
    If lcCadena = "Alfredo" Then
        ' generar un error con el método Raise del objeto Err
        Err.Raise vbObjectError + 1200, , _
            "No se permite la asignación de esa cadena"
    End If
End Sub

```

Código fuente 158

6

Programación orientada a objeto (OOP)

¿Qué ventajas aporta la programación orientada a objetos?

La programación orientada a objeto aparece como una forma de acercar las técnicas de programación al modo en que la mente humana trabaja para resolver los problemas. Tradicionalmente, durante el proceso de creación de una aplicación, la atención del programador se centraba en la forma de resolver las tareas (procesos), y no los elementos (objetos) involucrados en esas tareas.

Pongamos por ejemplo un programa de facturación. Mediante programación procedural, podríamos emplear un pseudocódigo similar al Código fuente 159 para crear una factura.

```
LOCAL hFact AS Handle
LOCAL nNumFact AS Number
LOCAL cNombre AS Character
LOCAL nImporte AS Number
// abrir el fichero de facturas, la variable
// hFact hace las veces de manejador
// del fichero
hFact = AbrirFichero("Facturas")
// en las siguientes líneas,
// el usuario introduce valores
// en las variables que contendrán
// los datos de la factura
TomarValores(nNumFact, cNombre, nImporte)
// grabar los valores en el fichero de facturas
GrabarFactura(hFact, nNumFact, cNombre, nImporte)
```

```
// cerrar el fichero de facturas
CerrarFichero(hFact)
```

Código fuente 159

Como acabamos de ver, hacemos uso de un conjunto de rutinas o funciones aisladas e inconexas para resolver problemas puntuales, que si analizamos detenidamente, tienen un elemento común denominador, la factura.

Utilizando OOP (Oriented Object Programming), pensamos en primer lugar en la factura como un *objeto*, que tiene un número, fecha, importe, etc., (propiedades), que podemos visualizar, grabar, etc., (métodos). Estamos enfocando nuestra atención en los objetos de la aplicación y no en los procesos que manejan los objetos. Esto no significa que las rutinas que manejan los objetos sean menos importantes, sino que al abordar el diseño de esta manera, la factura pasa a convertirse en un componente, una entidad dentro de la aplicación y no un conjunto de funciones sin nexo común.

Con lo anteriormente expuesto, la grabación de una factura empleando técnicas OOP sería algo similar a lo que se puede ver en el Código fuente 160.

```
' declarar una variable que contendrá el objeto factura
Dim oFactura As Factura
' instanciar un objeto de la clase factura,
' y asignarlo a la variable de objeto
Set oFactura = New Factura
' ejecutar el método de captura de
' datos para la factura
oFactura.TomarValores
' grabar los datos de la factura a la tabla
oFactura.Grabar
' eliminar la referencia al objeto
Set oFactura = Nothing
```

Código fuente 160

El lector se estará preguntando dónde se encuentra el código que abre el fichero de datos en donde se graban las facturas, la inicialización de propiedades del objeto, etc. Bien, pues esta es una de las ventajas del uso de objetos, todo ese trabajo se encuentra encapsulado dentro del código del objeto, y si este se encuentra bien diseñado y construido, no necesitaremos saber nada en lo que respecta a esos puntos, ya que al crearse el objeto, en los diferentes métodos del mismo se realizará todo ese trabajo.

Objetos

Visto lo anterior, podemos decir que un objeto es un conjunto de código creado para acometer una tarea concreta y que puede ser manipulado como una unidad o entidad del programa. Dispone de información o propiedades para consultar y modificar su estado, y de rutinas o métodos que definen su comportamiento.

Clases

Una clase es el elemento que contiene la definición de las propiedades y métodos de un objeto, actuando como plantilla o plano para la creación de nuevos objetos de un mismo tipo.

Imaginemos una plancha utilizada en una imprenta para fabricar impresos, la plancha se puede identificar con la *Clase Impreso*, y cada impresos que se obtiene a partir de ella es un *Objeto Impreso*.

Una clase es una entidad abstracta, puesto que nosotros manejamos objetos, no clases, pero nos ayuda a mantener una organización. La clase Coche es algo intangible, mientras que un objeto Coche podemos tocarlo y usarlo. Es esta misma abstracción la que nos permite identificar a que clase pertenecen los objetos, ¿por qué motivo?, pues porque tomando el ejemplo de la clase coche, todos los objetos de esta clase no son exactamente iguales.

Un coche deportivo tiene una línea más aerodinámica, un motor más potente...; un coche familiar es más grande que un deportivo, su maletero es mayor, aunque el motor es menos potente...; una furgoneta sustituye el maletero por una zona de carga, puede soportar un mayor peso, etc. Pero todos los objetos de esta clase tienen una serie de características (propiedades) comunes: volante, ruedas, motor, etc., y un comportamiento (métodos) igual: arrancar, parar, girar, acelerar, etc., que nos permiten reconocerlos entre los objetos de otras clases. La capacidad de reconocer la clase a la que pertenecen los objetos es la abstracción.

Las diferencias entre objetos de una misma clase vienen dadas por las propiedades particulares de cada objeto. Continuando con el ejemplo de los coches, dos objetos de la clase coche familiar, pueden ser creados incluyendo en uno la propiedad asiento de cuero, y en el otro no. De igual manera, durante el periodo de duración del coche, su dueño puede cambiarle la propiedad radio dejándola vacía, o por otra más moderna. Los dos objetos seguirán perteneciendo a la clase coche familiar, pero existirán diferencias entre ambos impuestas por los distintos valores que contienen sus propiedades.

La organización de las clases se realiza mediante jerarquías, en las cuales, los objetos de clase inferior o clase hija heredan las características de la clase superior o clase padre.

La Figura 70 muestra como se organiza la jerarquía de medios de transporte. La clase superior define a los objetos más genéricos, y según vamos descendiendo de nivel encontramos objetos más específicos.

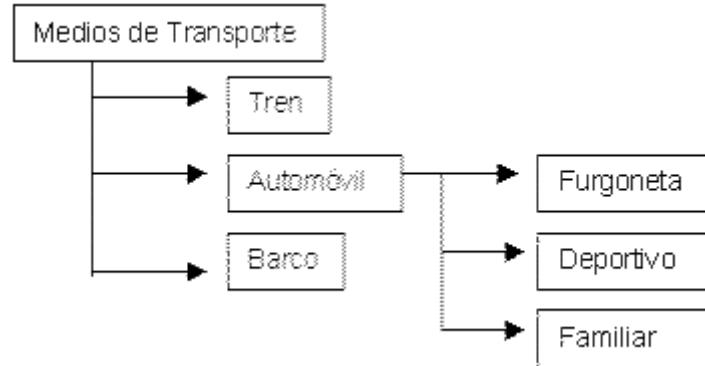


Figura 70. Jerarquía de la clase Medios de Transporte.

Clases abstractas

Una clase abstracta es aquella de la que no se pueden crear objetos, estando definida únicamente a efectos organizativos dentro de la jerarquía.

Tomemos de la jerarquía de la Figura 70, la rama Automóvil (Figura 71)

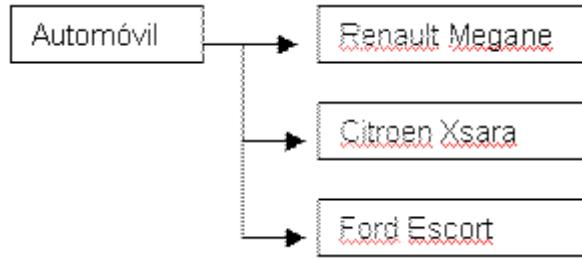


Figura 71. Rama Automóvil de la jerarquía de la clase Medios de Transporte.

Mientras que no podemos crear un objeto directamente de la clase Automóvil, si podemos hacerlo de cualquiera de las clases que dependen de ella. Esto es debido a que Automóvil como objeto no puede existir, es una entidad genérica, sienta las bases y especificaciones para que sus clases hijas puedan crear objetos automóviles *pero* de una determinada marca y modelo, que son los que sí podemos tocar y manejar.

Otro ejemplo de clase abstracta lo podría constituir la anterior clase Impreso, de la cual heredarían las clases Albarán, Factura y Carta. La clase Impreso no puede crear objetos, no existen objetos impreso, pero sí existen impresos de factura, carta, recibo, aviso, etc.

Relaciones entre objetos

Los objetos dentro de una aplicación se comunican entre sí mediante una serie de relaciones, a continuación hacemos un breve repaso de las mismas.

Herencia

En ocasiones, los objetos de una determinada clase, heredan a su vez de otra clase que ocupa un nivel superior en la jerarquía. Por ejemplo, un objeto de la clase Factura hereda de la clase Impreso, un objeto Albarán también hereda de la clase Impreso. A estas clases heredadas también se las conoce como subclases.

Cuando se crea una nueva clase heredada, la nueva clase dispone de todas las propiedades y métodos de la clase padre, mas el código implementado dentro de la propia clase.

Una forma de reconocer este tipo de relaciones, es realizar un análisis sintáctico de la misma usando las partículas "hereda de" o "es un", por ejemplo: "La clase Factura hereda de la clase Impreso" o "Un objeto Factura es un Impreso".

Pertenencia

Los objetos pueden contener o estar formados a su vez por otros objetos. Un objeto coche está formado por los objetos motor, volante, rueda, asiento, etc. Se dice en este caso que hay una relación de pertenencia, ya que existe un conjunto de objetos que pertenecen a otro objeto o se unen para formar otro objeto. A este tipo de relación se le denomina también Contenedora.

La forma de reconocer este tipo de relaciones, es realizar un análisis sintáctico de la misma usando la partícula "tiene un", por ejemplo: Un objeto de la clase Coche tiene un objeto de la clase Volante.

Utilización

Hay situaciones en que un objeto utiliza a otro para realizar una determinada tarea, sin que ello suponga la existencia de una relación de pertenencia entre dichos objetos.

Por ejemplo, un objeto Formulario puede utilizar un objeto Cliente para mostrar al usuario las propiedades del cliente, sin la necesidad de que el objeto cliente sea una propiedad del objeto formulario.

Nótese la importante diferencia entre esta relación y la anterior, ya que aquí, el objeto formulario a través de código, creará o le será pasado como parámetro un objeto cliente, para poder mostrarlo en el área de la ventana.

La forma de reconocer este tipo de relación, es realizar un análisis sintáctico de la misma, usando la partícula "usa un". Por ejemplo: Un objeto de la clase Form usa un objeto de la clase Cliente.

Elementos básicos de un sistema OOP

Encapsulación

Esta característica nos indica que las propiedades de un objeto sólo pueden ser manejadas por el propio objeto. De esta manera, la información del objeto permanece libre de modificaciones exteriores que pudieran hacer que el objeto se comportara de forma errónea.

Para recuperar/asignar el valor de una propiedad del objeto, se recomienda escribir métodos que comprueben el valor antes de asignarlo a la propiedad, y que realicen un formateo del valor a mostrar en el caso del acceso a la propiedad.

En VB disponemos de los métodos o procedimientos Property para lograr encapsulación, evitando el manejo directo de las propiedades del objeto.

Polimorfismo

Según este principio, dos métodos con el mismo nombre pueden realizar diferentes operaciones en función del objeto sobre el que se aplique. La ventaja de esta característica es que el programador no necesita preocuparse de cómo funciona el método en su interior, sólo ha de utilizarlo.

De esta forma, un método llamado Abrir, tendría un comportamiento diferente al ser usado con un objeto Ventana o con un objeto Fichero.

Herencia

Esta es la característica más importante de la OOP mediante la cual cuando definimos una clase hija que hereda de una superior, los objetos creados a partir de la clase hija, por el mero hecho de haber establecido la relación de herencia y sin tener que escribir ni una sola línea de código, contendrán todas las propiedades y métodos de la clase padre. Naturalmente, podremos añadir propiedades y métodos específicos para la clase hija que sirvan para completar su funcionalidad.

Tomemos como ejemplo una clase *Ventana*, que tiene como propiedades las coordenadas de la ventana, y un método *Mover()* para cambiarla de posición en pantalla. Ahora podemos crear una nueva clase llamada *VentNombre* que herede de *Ventana* y agregarle la propiedad *Nombre* para mostrar un título en la ventana, y el método *CambiarTam()* para redimensionarla, con lo que aprovecharemos todo el código de la clase padre y sólo tendremos que ocuparnos de crear la nueva funcionalidad para esta clase.

VB no permite en la actual versión una herencia *pura*, por decirlo de alguna manera. Podemos lograr una cierta comunicación entre las clases mediante el uso de interfaces, aunque no será el tipo de herencia al que estamos acostumbrados si hemos trabajado con otras herramientas que proporcionan herencia auténtica.

Reutilización

Este es uno de los objetivos perseguidos por la OOP. Un objeto bien diseñado, puede ser utilizado en otra aplicación, bien directamente o heredando una clase hija de él, que aproveche todo lo que tiene y a la que añadiríamos nuevas funcionalidades. Todo esto supone un considerable ahorro en el tiempo de desarrollo de la aplicación.

Creación de clases

Después de las anteriores nociones teóricas, veamos a continuación, los pasos necesarios para construir una clase en VB. El lector dispone en la aplicación [OOClien](#), de una clase básica como ejemplo, con sus propiedades y métodos.

Crear un módulo de clase

En primer lugar, debemos abrir desde el IDE de VB un nuevo proyecto, o uno existente. Seguidamente, mediante la opción de menú *Proyecto+Agregar módulo de clase*, insertaremos un módulo de clase en el proyecto. Este tipo de módulo contiene la definición y el código para una clase, por lo que deberemos añadir un módulo de este tipo por cada clase que incorporemos al proyecto.

Cada módulo de clase, dispone de una ventana de código en donde escribiremos las líneas de código correspondientes a sus propiedades y métodos.



Figura 72. Ventana de propiedades para asignar el nombre de una clase.

Los módulos de clase se guardan en ficheros con extensión .CLS; en nuestro ejemplo será Cliente.CLS.

Para abrir su ventana de código, procederemos igual que para un módulo de código estándar. Nos situaremos en la ventana de proyecto y haremos doble clic sobre el nombre del módulo de clase a codificar.

A continuación utilizaremos la ventana de propiedades para asignar en la propiedad Name el nombre de la clase (Figura 72), el resto de propiedades de la clase se utilizan para la creación de clases orientadas a datos.

Definir las propiedades de la clase

Las propiedades de una clase son variables declaradas a nivel del módulo de clase, que exponen su información bien directamente si se han declarado como públicas o a través de procedimientos Property, si se han declarado privadas.

Estas propiedades, como se ha comentado anteriormente, constituyen la información que contendrá cada objeto creado a partir de la clase.

¿Qué son los procedimientos Property?

Los procedimientos Property se utilizan para proporcionar acceso al valor de las propiedades privadas definidas dentro de un módulo de clase, y si es necesario, ejecutar código al asignar o recuperar el valor de la propiedad.

Para que el código externo a la clase pueda acceder a una propiedad privada, los procedimientos Property se deben declarar públicos.

Tipos de procedimientos Property

Disponemos de tres modos de comunicarnos con las propiedades de una clase.

- **Property Get.** Retorna el valor de una propiedad de la clase. Adicionalmente, este procedimiento puede contener el código para devolver el valor de la propiedad formateada, como pueda ser incluir el año completo para las fechas, puntos decimales en propiedades numéricas, etc. Este tipo de procedimiento es similar a Function, ya que devuelve un valor asignado al propio nombre del procedimiento.

En el Código fuente 161, podemos ver este procedimiento aplicado a la propiedad Nombre de la clase Cliente, en su modo más sencillo de uso, asignando el valor de la propiedad a la denominación del procedimiento.

```
Public Property Get Nombre() As String
Nombre = mcNombre
End Property
```

Código fuente 161

- **Property Let.** Asigna un valor a una propiedad de la clase. Gracias a este tipo de procedimientos, podemos comprobar que los valores para la propiedad son correctos antes de asignarlos definitivamente, así como realizar operaciones de formato con dichos valores.

En el Código fuente 162, vemos un procedimiento de este tipo para la propiedad Nombre de la clase Cliente. Como podrá comprobar el lector, la variable de propiedad mcNombre, es la que guarda el valor asignado al procedimiento Property.

```
Public Property Let Nombre(ByVal vcNombre As String)
mcNombre = vcNombre
End Property
```

Código fuente 162

En la propiedad Alta, queda más patente la forma de realizar comprobaciones sobre el valor que se asigna a la propiedad. Si el mes de la fecha que asignamos coincide con el de la fecha actual, se pasa el valor a la propiedad, en caso contrario, se avisa al usuario, sin modificar el valor que la propiedad tuviera.

```
Public Property Let Alta(ByVal vdtAlta As Date)
Dim lnMesActual As Integer
Dim lnMesAlta As Integer
lnMesAlta = Month(vdtAlta)
lnMesActual = Month(Date)
If lnMesAlta = lnMesActual Then
    mdtAlta = vdtAlta
Else
    MsgBox "El mes del alta ha de ser el actual", , "Error"
End If
End Property
```

Código fuente 163

El paso de parámetros a los procedimientos Property es igual que en el resto de procedimientos en VB, se puede hacer por valor o referencia. Lo más recomendable, si queremos mantener el mayor nivel de encapsulación, tal y como dictan las normas de programación orientada a objeto, es utilizar el paso por valor.

- **Property Set.** Este procedimiento es una variante de Let, ya que se utiliza para asignar un valor a una propiedad, con la particularidad de que el valor a asignar ha de ser un objeto.

Supongamos que en la aplicación de ejemplo, hemos añadido la propiedad mfrmCaja a la clase Cliente. La nueva propiedad contendrá un objeto de tipo formulario llamado frmDatos, como indica el Código fuente 164.

```
Private mfrmCaja As frmDatos
```

Código fuente 164

Como la propiedad es privada, proporcionamos su acceso creando el oportuno par de procedimientos Property, pero esta vez serán Get/Set en vez de Get/Let, debido a que la nueva propiedad contiene un objeto.

```
Public Property Set Caja(ByVal vfrmCaja As frmDatos)
Set mfrmCaja = vfrmCaja
End Property
' -----
Public Property Get Caja() As frmDatos
Set Caja = mfrmCaja
End Property
```

Código fuente 165

En otras herramientas de programación con capacidades de orientación a objeto, a los procedimientos Property se les conoce como *métodos de Acceso/Asignación*, lo cual es lógico, si pensamos que no dejan de ser métodos, aunque con una finalidad especial, que es la de tratar con las propiedades de la clase.

Tener que utilizar un método o procedimiento para acceder a una propiedad, puede resultar un poco chocante, ya que habituados a la forma de manejar las variables en el código, nuestra tendencia es utilizar las propiedades de la misma forma. Sin embargo, una de las normas de la OOP es no permitir *nunca* acceso directo a las propiedades de un objeto desde el exterior del mismo (la encapsulación no tendría sentido en ese caso), por este motivo se proporcionan los procedimientos Property en VB, y los métodos de Acceso/Asignación en otros lenguajes. A este respecto, un punto interesante se encuentra en el hecho de que las propiedades públicas de una clase, son convertidas internamente por VB en pares de procedimientos Get/Let, de forma que el lector puede hacerse una idea de la importancia que se le otorga a este aspecto en VB.

Insistiendo en la importancia que los procedimientos Property tienen en VB, también debemos comentar que para esta herramienta, las propiedades son identificadas sólo mediante este tipo de procedimiento. Como ejemplo, tenemos los asistentes que manipulan clases, al visualizar las propiedades de una clase, muestran los nombres de los procedimientos Property. Nosotros pensamos que una propiedad, es realmente el elemento de código que contiene el valor de esa propiedad. En el caso de VB, sería la variable de propiedad definida en el módulo de clase, mientras que los procedimientos Property cumplen la tarea de dar acceso al valor de la variable de propiedad desde el exterior de la clase, y formatear o validar el valor a establecer en la variable de propiedad.

Debido a esta estrecha relación entre las variables de propiedad y los procedimientos Property, emplearemos el término *propiedad*, para referirnos tanto a uno como otro elemento de una clase.

Ventajas de los procedimientos Property

Aunque el empleo de propiedades públicas en la clase, es la manera más fácil y rápida de definir propiedades, también es la menos recomendable. Siempre que sea posible es mejor utilizar la combinación, propiedades privadas – procedimientos Property. Algunas de las ventajas, se enumeran a continuación.

- **Encapsulación.** Supongamos que tenemos una cadena que debe ser convertida a mayúsculas antes de asignarla a la propiedad de un objeto. Empleando un método Property Let para esa propiedad, podemos hacer la conversión dentro del procedimiento en vez de realizar una conversión previa en cada punto del código antes de asignar la cadena a la propiedad. Con esta

forma de trabajo, por un lado nos despreocupamos de hacer la conversión en múltiples puntos a lo largo del código; y por otro al tener el código de conversión centralizado en un solo lugar, si necesitáramos hacerle un retoque en cualquier momento, sólo debemos modificar una vez. De la otra forma tendríamos que estar buscando en todo el proyecto los lugares en los que se realiza la conversión, y modificar las correspondientes líneas de código.

- **Hacer que la propiedad sea de Sólo lectura o Sólo asignación.** Puede darse el caso de que al crear un objeto, este tenga una propiedad que guarde la fecha y hora de creación para mantener cualquier tipo de estadística. Si necesitamos acceder a dicha propiedad, dispondremos del correspondiente Property Get para visualizar el contenido de la propiedad. Dado que no es conveniente modificar ese valor, para no influir en el resultado de la estadística, podemos hacer dos cosas:

- No escribir el procedimiento Property Let para esa propiedad. De esta forma, al ser la propiedad privada, no habrá medio de que desde el exterior del objeto pueda alterarse la propiedad que contiene la fecha.

El uso de procedimientos Property, no obliga a crear un conjunto Get/Let o Get/Set. Es perfectamente válido escribir sólo uno de ellos, y no crear el otro si no es necesario.

- Escribir el procedimiento Property Let declarándolo privado. Así, sólo el código de la propia clase podrá asignar valor a la propiedad, y como es natural, el programador ya se encargará de que el código de la clase no altere erróneamente los valores de la propiedad, y la estadística no resulte inexacta.

Piense el lector, que en este caso, si la propiedad se hubiese declarado pública, cualquier elemento de la aplicación podría alterar su valor accidentalmente, con los problemas que ello acarrearía.

- **Crear Propiedades Virtuales.** Mediante los procedimientos Property, es posible acceder a valores que no se corresponden directamente con una propiedad declarada en la clase, sino que son el resultado de las operaciones efectuadas entre varias propiedades.

Supongamos que en la clase Cliente necesitamos saber los días transcurridos entre la fecha de alta del cliente (propiedad Alta) y la fecha actual del sistema. La opción más lógica es declarar una nueva propiedad en la clase para que contenga este valor, siendo esto perfectamente válido. Pero existe otra vía para lograr este fin: crear un procedimiento Property Get, que realice el cálculo y devuelva el resultado.

```
Public Property Get DiasAlta() As Long
DiasAlta = DateDiff("d", Me.Alta, Date)
End Property
```

Código fuente 166

La forma de utilizarlo en el programa, sería igual que para el resto de los procedimientos Property que sí se corresponden con una propiedad de la clase.

- **Empleo de nombres de propiedad más naturales.** Ya hemos comentado los beneficios de utilizar unas normas de notación para las variables, aspecto al que las propiedades de una clase no pueden permanecer ajenas.

En la clase Cliente del ejemplo, hemos declarado una propiedad mnCodigo pública, mientras que el resto se han declarado privadas. Cuando en la aplicación utilicemos un objeto de esta clase para asignarle valores, lo haremos según el Código fuente 167.

```
' declarar una variable de la clase Cliente
' e instanciar un objeto de dicha clase
.....
.....
' asignar valores a algunas propiedades del objeto
loCliente.mnCodigo = 286
.....
loCliente.ciudad = "Venecia"
.....
```

Código fuente 167

Como la propiedad mnCodigo se ha declarado pública, al utilizarla en un objeto hemos de emplear, como es lógico, el mismo nombre para acceder a su contenido, lo que resulta un tanto incómodo al tener que recordar dentro del nombre, las partículas que indican el tipo de dato que contiene y el ámbito.

Para la propiedad mcCiudad, la situación es muy diferente. Al haberse declarado privada y disponiendo de procedimientos Property para acceder a su valor, emplearemos el nombre de tales procedimientos, en lugar usar directamente el nombre de la propiedad.

Ambos modos de trabajo son válidos, pero resulta evidente que el uso de los procedimientos Property para Ciudad, proporcionan un modo mucho más natural de interactuar con las propiedades de un objeto, constituyendo el mejor medio de comunicación entre el programador y la información del objeto. Este es otro de los argumentos que podemos apuntar a favor de los procedimientos Property, como ventaja frente a las propiedades públicas.

Una de las normas de la OOP es proporcionar modelos más naturales de trabajo. Cuando anotamos en un papel los datos de un cliente ponemos: "Codigo: 286, Ciudad: Venecia" y no "mnCodigo: 286, mcCiudad: Venecia". Del mismo modo, cuando accedamos a las propiedades de un objeto en VB, los nombres de esas propiedades, deberían estar lo más próximas al lenguaje habitual que sea posible.

Establecer una propiedad como predeterminada

Cuando usamos un control Label, es posible asignar un valor a su propiedad Caption en la forma mostrada en el Código fuente 168.

```
lblValor = "coche"
```

Código fuente 168

El hecho de no indicar la propiedad *Caption* en la anterior línea, se debe a que esta es la propiedad predeterminada del control/objeto, por lo que no será necesario ponerla para asignar/recuperar el valor que contiene.

De igual forma, si analizamos cual va a ser la propiedad o método que más se va a utilizar en una clase, podemos establecer esa propiedad o método como predeterminado, de manera que no sea necesario escribirlo en el código para usarlo. Sólo es posible definir una propiedad o método como predeterminado para cada clase. En la clase Cliente tomamos la propiedad *Nombre* como predeterminada.

Para ello, nos situaremos en el código del método o en el procedimiento *Property Get* de la propiedad a convertir en predeterminada y seleccionaremos la opción *Herramientas+Atributos del procedimiento* del menú de VB, que nos mostrará la ventana de la Figura 73.

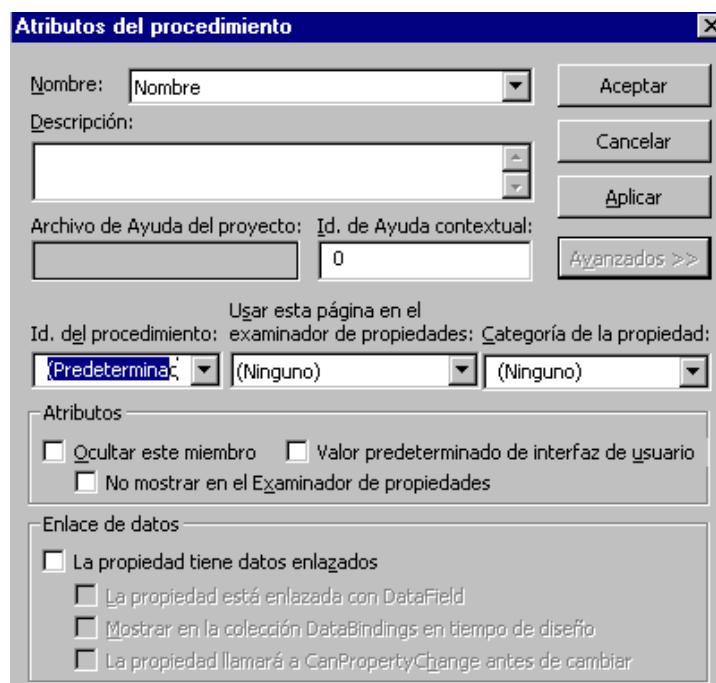


Figura 73. Establecer un elemento de la clase como predeterminado.

El siguiente paso consiste en seleccionar el valor *Predeterminado* en el *ListBox Id.del procedimiento*. Pulsaremos *Aceptar* para guardar los cambios realizados, con lo que a partir de ahora, para acceder a los valores de la propiedad *Nombre* de la clase Cliente podemos utilizar o no el nombre de la propiedad según se muestra en el Código fuente 169.

```
Dim loCliente As Cliente
Set loCliente = New Cliente
.....
.....
' accedemos a la propiedad Nombre
' de la siguiente forma
loCliente = "José Ramírez"
.....
.....
' o bien de esta otra
loCliente.Nombre = "José Ramirez"
```

Código fuente 169

Un inconveniente de las propiedades predeterminadas es que hemos de recordar en todo momento cuales, para poder asignarle el valor adecuado. En la clase Cliente por ejemplo, hemos de tener la precaución de no intentar asignar a la propiedad Nombre el valor del código.

Crear los métodos de la clase

Los métodos de una clase, definen la forma en que un objeto de esa clase se comportará y las acciones que llevará a cabo.

La forma de crear los métodos en VB es a través del uso de procedimientos Sub o Function, dependiendo de si el método a codificar deberá devolver o no un valor. La única diferencia existente es en el ámbito conceptual, a un procedimiento creado dentro de un módulo de clase se le denomina método, mientras que los creados en módulos estándar son simplemente procedimientos.

En lo que respecta al ámbito de los métodos, se aplicarán iguales normas que para las propiedades. Si queremos que un método sea accesible desde el exterior de la clase, se declarará público, si sólo es necesario a nivel interno, se declarará privado. Declarando un método privado, conseguimos que el código de ese método no sea ejecutado por procesos externos a la clase, que pudieran afectar negativamente al objeto al que pertenece.

La clase Cliente dispone del método NombreMay(), cuyo cometido es pasar a mayúsculas la cadena contenida en la propiedad Nombre del objeto.

```
Public Sub NombreMay()
Me.Nombre = UCase(Me.Nombre)
End Sub
```

Código fuente 170

¿Cuándo crear un procedimiento Property o un método?

Cuando queramos definir las características o atributos de un objeto, emplearemos un procedimiento Property. Por ejemplo, para manejar la *ciudad* de un objeto cliente crearemos el Property Ciudad.

Cuando se trate de definir el comportamiento o acciones de un objeto, utilizaremos métodos. Por ejemplo, la acción de *mostrar los datos* de un cliente puede perfectamente englobarse en un método.

Pese a lo anteriormente dicho, existen ocasiones en que es difícil determinar cuando emplear uno u otro sistema, por lo cual el empleo de un procedimiento Property o un método debe ser algo flexible y no basarse en unas normas rígidas. Para averiguar por ejemplo, los días que lleva dado de alta un cliente, podemos crear un procedimiento Property, ya que se puede considerar perfectamente como una propiedad del objeto. Sin embargo, también es factible definirlo como un método, ya que para obtener el número de días es necesario realizar una operación. Como podemos comprobar en este caso, el empleo de uno de los tipos de procedimiento queda en manos del criterio del programador.

Como ayuda ante estas situaciones, debemos analizar lo que necesitamos definir para el objeto, si tiene que ver con un nombre (Ciudad) crearemos procedimientos Property, si es con un verbo (MostrarDatos) crearemos un método.

Instanciar objetos de una clase

Una vez que hemos terminado de desarrollar una clase, para poder usarla hemos de crear *instancias* de la misma, ya que la clase no puede ser utilizada directamente.

Para comprender este concepto un poco mejor, volvamos al ejemplo de la clase Impreso, visto al comienzo del tema. La plancha de la imprenta, que representa a la clase Impreso, no puede ser utilizada directamente para llenar los datos de un albarán o factura. Debemos utilizar la plancha (clase) en la imprenta, para crear (instanciar) una factura (objeto) de papel, que será lo que llenaremos y enviaremos al cliente.

Por instanciar entendemos pues, la operación de crear un objeto a partir de la definición de una clase. Dichos objetos, serán los que utilizaremos en el código del programa.

Variables de objeto y tipos de enlace

Cuando declaramos una variable para asignarle un objeto, el modo de tipificación de dicha variable influirá posteriormente en el rendimiento de la aplicación, debido a que internamente se establece un enlace entre el código de la aplicación y el de la clase que vamos a utilizar. Existen dos tipos de enlace que mostramos a continuación.

- Enlace temprano (Early Binding). El enlace temprano consiste en declarar una variable con el tipo específico de objeto que va a utilizar. Las referencias en este caso, a métodos y propiedades para la variable de objeto se resuelven en tiempo de diseño, de manera que durante la ejecución no es necesario buscarlas a través del código, mejorando el rendimiento de la aplicación.

Una declaración early binding tendría la forma que aparece en el Código fuente 171.

```
Dim loCliente As Cliente
```

Código fuente 171

- Enlace tardío (Late Binding). Este tipo de enlace consiste en declarar la variable de objeto utilizando el tipo genérico Object.

Las situaciones en las que podemos utilizar este tipo de declaración, son cuando no conocemos la clase de objeto que vamos a utilizar, o bien cuando vamos a asignar a la misma variable diferentes tipos de objeto durante el transcurso del programa. En este caso, las referencias a métodos y propiedades para la variable de objeto se resuelven en tiempo de ejecución, afectando negativamente el rendimiento del programa.

Una declaración late binding tendría la forma que aparece en el Código fuente 172.

```
Dim loCliente As Object
```

Código fuente 172

La recomendación en este tipo de casos, es que al emplear variables de objeto, y siempre que conozcamos la clase de objeto que va a contener, utilicemos enlace temprano.

Existe una tercera forma de declaración, no indicando el tipo de dato para la variable, que es la que arrastra un mayor grado de penalización en el rendimiento, por lo que debe evitarse siempre que sea posible.

```
Dim loCliente
```

Código fuente 173

El Código fuente 173 no proporciona al compilador información alguna sobre el tipo de dato que debe contener la variable, creándose Variant por defecto, con lo cual, el compilador ha de encargarse de hacer las conversiones de tipos, y en el caso de objetos, el inconveniente añadido del enlace tardío.

Instanciar un objeto

Para crear una instancia de un objeto, una vez que se haya declarado la variable correspondiente, se utilizará la instrucción Set junto a la palabra clave New como se muestra el Código fuente 174. En este código se crea una nueva instancia de la clase Cliente y la asigna a la variable loCliente.

```
Set loCliente = New Cliente
```

Código fuente 174

El motivo de emplear Set para asignar un objeto a una variable, a diferencia de las variables comunes que no necesitan utilizar dicha instrucción, reside en el hecho de que una variable de objeto contiene una referencia (puntero) a un objeto y una variable común contiene un valor, de ahí el empleo de una sintaxis especial.

Una variante a este tipo de declaración, reside en usar la palabra clave New al declarar el objeto. Con este tipo de declaración, se crea la instancia la primera vez que se utilice la variable, con lo que ahorraremos el paso de instanciarlo.

```
' declarar el objeto con New
Dim loCliente As New Cliente
' en esta línea, al mismo tiempo que
' se crea una instancia del objeto
' de forma automática, se asigna un
' valor a una de sus propiedades
loCliente.mnCodigo = 286
```

Código fuente 175

Este tipo de declaración puede influir negativamente en el rendimiento del programa, debido a que cada vez que se utilice el objeto para llamar a un método o propiedad, VB ha de estar comprobando si ya se ha creado o no, una instancia de la clase para la variable.

Manipulación de propiedades y métodos

Una vez que el objeto se ha creado y asignado a una variable, esta se utilizará para manipular las propiedades del objeto y hacer llamadas a sus métodos, empleando el llamado en los lenguajes OOP *operador de envío* o *Send*, que en VB se representa con el signo de un punto ":".



Figura 74. Formato de uso para una variable de objeto.

```

' manipular una propiedad del objeto
loCliente.Ciudad = "Salamanca"
' llamar a un método del objeto
loCliente.MostrarDatos
  
```

Código fuente 176

Un consejo para aumentar el rendimiento de la aplicación, es emplear en lo posible sentencias cortas que hagan uso de un único objeto, en lugar de sentencias que impliquen múltiples referencias a varios objetos. Para explicar esta cuestión un poco mejor, supongamos que tenemos un objeto empresa, que contiene como propiedad otro objeto de tipo proveedor, y este último a su vez contiene una propiedad teléfono. Si queremos acceder al teléfono del objeto proveedor en una sola línea de código, podemos hacer lo que muestra el Código fuente 177.

```

Dim cTelf As String
Do While <expr>
    .....
    .....
    cTelf = oEmpresa.Proveedor.Telefono
    .....
    .....
Loop
  
```

Código fuente 177

Con este modo de trabajo, cada vez que recuperamos la propiedad Teléfono, se efectúan dos búsquedas en objetos, una para el objeto Empresa y otra para el objeto Proveedor. Si el bucle Do While debe realizar muchas iteraciones, esto supone una pérdida de rendimiento. Sin embargo, si empleamos la técnica mostrada en el Código fuente 178.

```

Dim cTelf As String
Dim oProv As Proveedor
  
```

```

oProv = oEmpresa.Proveedor
Do While <expr>
    .....
    cTelf = oProv.Telefono
    .....
Loop

```

Código fuente 178

Conseguiremos evitar una de las operaciones con los objetos, recuperando el valor únicamente de donde nos interesa.

Por una razón similar, además de un ahorro y claridad en el código, cuando debamos utilizar un mismo objeto en varias líneas de código seguidas, es conveniente el uso de la instrucción With...End With, que nos permitirá manipular un objeto sin usar continuamente su nombre.

```

With loCliente
    .Codigo = 286
    .Nombre = "Juan Palermo"
    .Direccion = "Escalinata 20"
    .Ciudad = "Salamanca"
    .MostrarDatos
End With

```

Código fuente 179

Eliminar el objeto

Un objeto utiliza los recursos del equipo, por lo que cuando no siga siendo necesario su uso, es necesario eliminar la referencia que mantiene la variable hacia el objeto, asignando a la variable la palabra clave *Nothing*.

```

' liberar los recursos utilizados por el objeto
Set loCliente = Nothing

```

Código fuente 180

Si hemos definido en un procedimiento, una variable local para un objeto, la referencia al objeto contenida en la variable será eliminada al finalizar el procedimiento.

La palabra clave Me

La palabra Me, se utiliza para establecer una referencia a un objeto, desde dentro del código del propio objeto. Este concepto, que puede parecer un rompecabezas, podemos entenderlo mejor con el siguiente ejemplo.

En el módulo CODIGO.BAS de la aplicación [OOMe](#), incluida con los ejemplos, hemos creado el procedimiento EliminarCiudad(), que recibe como parámetro, un objeto de la clase Cliente y asigna una cadena vacía a su propiedad Ciudad.

```
Public Sub EliminarCiudad(ByVal voCliente As Cliente)
voCliente.Ciudad = ""
End Sub
```

Código fuente 181

En el Código fuente 182, en el procedimiento Main() de la aplicación, creamos dos objetos de la clase Cliente, les asignamos valores a su propiedad Ciudad, y mediante el anterior procedimiento, borramos el contenido de la propiedad de uno de ellos.

```
Public Sub Main()
' declarar variables de objeto y
' asignarles valor
Dim loUnCliente As Cliente
Dim loOtroCliente As Cliente
Set loUnCliente = New Cliente
loUnCliente.Ciudad = "Roma"
Set loOtroCliente = New Cliente
loOtroCliente.Ciudad = "Estocolmo"
' eliminar valor de la propiedad Ciudad
' en uno de los objetos
EliminarCiudad loUnCliente
' mostrar contenido de la propiedad Ciudad de cada objeto
MsgBox "Ciudad del objeto loUnCliente: " & loUnCliente.Ciudad
MsgBox "Ciudad del objeto loOtroCliente: " & loOtroCliente.Ciudad
End Sub
```

Código fuente 182

Hasta ahora todo normal, pero llegados a este punto, vamos a crear un método en la clase Cliente, llamado QuitaCiudad(), que hará una llamada al procedimiento EliminarCiudad() para borrar el valor de la propiedad Ciudad del objeto. El procedimiento Main(), quedaría como el Código fuente 183.

```
Public Sub Main()
Dim loUnCliente As Cliente
Dim loOtroCliente As Cliente
Set loUnCliente = New Cliente
loUnCliente.Ciudad = "Roma"
Set loOtroCliente = New Cliente
loOtroCliente.Ciudad = "Estocolmo"
' primera forma de eliminar la propiedad
' deshabilitada para que no interfiera con
' la siguiente
'EliminarCiudad loUnCliente
' siguiente forma de eliminar la propiedad
loUnCliente.QuitaCiudad
MsgBox "Ciudad del objeto loUnCliente: " & loUnCliente.Ciudad
MsgBox "Ciudad del objeto loOtroCliente: " & loOtroCliente.Ciudad
End Sub
```

Código fuente 183

Cuando el objeto loUnCliente llama al método QuitaCiudad, se ejecuta el Código fuente 184.

```
Public Sub QuitaCiudad()
    EliminarCiudad Me
End Sub
```

Código fuente 184

En este método se llama al procedimiento EliminarCiudad, que recibe como ya vimos, un objeto de la clase Cliente, por lo tanto, ¿cómo podemos referirnos al objeto desde dentro de su propio código?. La respuesta es *Me*, que es pasado como parámetro a EliminarCiudad.

Usando *Me*, el objeto sabrá que se está haciendo uso de una de sus propiedades o métodos desde dentro de su propio código, o que se está pasando la referencia del objeto como parámetro a una rutina, este es el caso en el método *QuitaCiudad()*. Adicionalmente, el lector habrá comprobado que se están utilizando dos variables con objetos de la misma clase, pero sólo se borra la propiedad de uno de los objetos. Ya que los objetos de una misma clase comparten el código de sus métodos, ¿cómo sabe el código de la clase, el objeto sobre el que está actuando?

La respuesta a la anterior pregunta es la siguiente: cuando un objeto ejecuta uno de sus métodos, internamente se establece una referencia a las propiedades que contiene ese objeto, de forma que siempre se sabe sobre qué objeto actuar. Para el programador, dicha referencia está representada en *Me*.

Para terminar con las ventajas del uso de *Me*, diremos que proporciona una mayor legibilidad al código de una clase, ya que nos permite comprobar más fácilmente cuáles son las propiedades y métodos pertenecientes a la clase, como se muestra en el Código fuente 185.

```
Public Sub ComprobarValores()
    Dim Nombre As String
    ' podemos utilizar una variable con la misma
    ' denominación que una propiedad, ya que al usar Me
    ' se sabe cuando se quiere usar la variable
    ' y cuando la propiedad
    Nombre = "valor cualquiera"      ' aquí usamos la variable
    MsgBox "Contenido de la propiedad: " & Me.Nombre ' aquí usamos la propiedad
    ' ahora llamamos a un método de esta clase,
    ' lo identificamos rápidamente por el uso de Me
    Me.CalculaTotales
    ' a continuación llamamos a un procedimiento
    ' de la aplicación. Una de las formas de saberlo
    ' es teniendo la costumbre de utilizar siempre Me
    ' en las llamadas a los métodos de la propia clase,
    ' aunque hemos de tener en cuenta que si ponemos el
    ' nombre de un método sin Me, también será admitido
    VerFechaActual
End Sub
```

Código fuente 185

La palabra *Me* en VB equivale a *Self* o *This* en otros lenguajes orientados a objeto.

Emplear valores constantes en una clase

La definición de constantes en una clase, plantea el problema de que no pueden utilizarse desde el exterior de la misma. Si definimos dentro de un módulo de clase varias constantes.

```
Const Volum_Bajo = 1
Const Volum_Medio = 2
Const Volum_Alto = 3
```

Código fuente 186

Y tenemos un método o propiedad que recibe un valor relacionado con las constantes. Dentro del propio código de la clase podemos emplear las constantes.

```
Me.Volumen = Volum_Medio
```

Código fuente 187

Pero si intentamos utilizarlas desde un objeto fuera de la clase, se producirá un error.

```
loCliente.Volumen = Volum_Medio ' esta línea provocará un error
```

Código fuente 188

Para solventar este problema, VB proporciona los nuevos tipos enumerados, que definidos como públicos dentro de un módulo de clase, serán visibles desde el exterior de la misma. Consulte el lector el tema sobre elementos del lenguaje, apartado *El tipo Enumerado*, para una descripción detallada de los mismos. La aplicación [OOEnum](#) proporcionada como ejemplo, muestra la ya conocida clase Cliente, con las adaptaciones necesarias para emplear tipos enumerados. Dichos cambios se basan en la creación de una nueva propiedad que muestra el volumen de compras que realiza el cliente.

En primer lugar, declaramos en el módulo de la clase Cliente la nueva propiedad, como vemos en el Código fuente 189, que guardará el volumen de compras.

```
Private mvntVolumen As Variant
```

Código fuente 189

En el Código fuente 190, definimos en la zona de declaraciones de la clase, el tipo enumerado VolumCompras, que contendrá los valores permitidos para la propiedad mvntVolumen.

```
' crear valores constantes para la clase
' a través de un tipo enumerado
Public Enum VolumCompras
    Bajo = 1
    Medio = 2
    Alto = 3
End Enum
```

Código fuente 190

Creamos los procedimientos Property Volumen de acceso/asignación para la propiedad mvntVolumen (Código fuente 191).

```
Public Property Let Volumen(ByVal vvntVolumen As Variant)
mvntVolumen = vvntVolumen
End Property
' -----
Public Property Get Volumen() As Variant
Select Case mvntVolumen
Case Alto
    Volumen = "Alto"
Case Medio
    Volumen = "Medio"
Case Bajo
    Volumen = "Bajo"
End Select
End Property
```

Código fuente 191

Y por fin, para utilizar esta nueva cualidad en un objeto cliente, lo haremos según se muestra en el procedimiento Main() de esta aplicación (Código fuente 192).

```
Public Sub Main()
' declarar variable de objeto
Dim loCliente As Cliente
Set loCliente = New Cliente
.....
.....
loCliente.Volumen = Alto ' utilizamos el tipo enumerado
.....
.....
loCliente.MostrarDatos
End Sub
```

Código fuente 192

Como habrá observado el lector, la causa de declarar como Variant la propiedad mvntVolumen se debe a que al asignarle valores, utilizamos el tipo enumerado VolumCompras, pero al recuperar el valor de dicha propiedad con el método MostrarDatos(), necesitamos disponer de una cadena de caracteres que muestre al usuario la descripción del volumen de compras en lugar de un número que proporciona menos información.

También podíamos haber declarado mvntVolumen como Long para emplear el mismo valor que VolumCompras, realizando la conversión a cadena en el método MostrarDatos(). Todo depende de los requerimientos de la aplicación y el enfoque del programador.

Eventos predefinidos para una clase

Un evento es un mensaje que un objeto envía cuando se produce una situación determinada sobre dicho objeto, y que podrá ser respondido o no por el código de la aplicación. Una clase dispone de dos eventos predefinidos, que se producen al crear y destruir el objeto respectivamente.

El primero de estos eventos es *Initialize*, que se genera al instanciar un objeto de la clase. Si el lector conoce algún otro lenguaje OOP, este evento es similar al Constructor de la clase. La diferencia entre este y los constructores de otros lenguajes, es que *Initialize* no puede recibir parámetros.

El otro evento es *Terminate*, que se produce al eliminar la referencia que el objeto mantiene con la aplicación. Para eliminar dicha referencia podemos asignar *Nothing* al objeto, o se puede producir al finalizar el procedimiento en donde está contenido el objeto, si se ha declarado como local. Este evento es similar al Destructor de la clase en otros lenguajes OOP.

Para comprobar el trabajo realizado en estos eventos, disponemos de la aplicación de ejemplo [EventCls](#), que contiene la clase *Cliente*, utilizada en el apartado anterior, y a la que se le ha incorporado el Código fuente 193 en los métodos de evento predefinidos.

```
Private Sub Class_Initialize()
MsgBox "Se crea un objeto de la clase Cliente"
End Sub
' -----
Private Sub Class_Terminate()
MsgBox "Se elimina el objeto de la clase Cliente"
End Sub
' -----
Public Sub Main()
' declarar una variable para contener
' un objeto de la clase Cliente
Dim loCliente As Cliente
' instanciar un objeto de la clase Cliente
' aquí se produce el evento Initialize
Set loCliente = New Cliente
' .....
' eliminar el objeto
' aquí se produce el evento Terminate
Set loCliente = Nothing
```

Código fuente 193

Utilidades y herramientas para el manejo de clases

Generador de clases (Class Wizard)

Esta utilidad nos permite diseñar las clases o jerarquía de las mismas para una aplicación, proporcionándonos ayuda en la creación de las propiedades, métodos, eventos y colecciones de las clases, pero sólo en el ámbito de diseño, sin incorporar código.

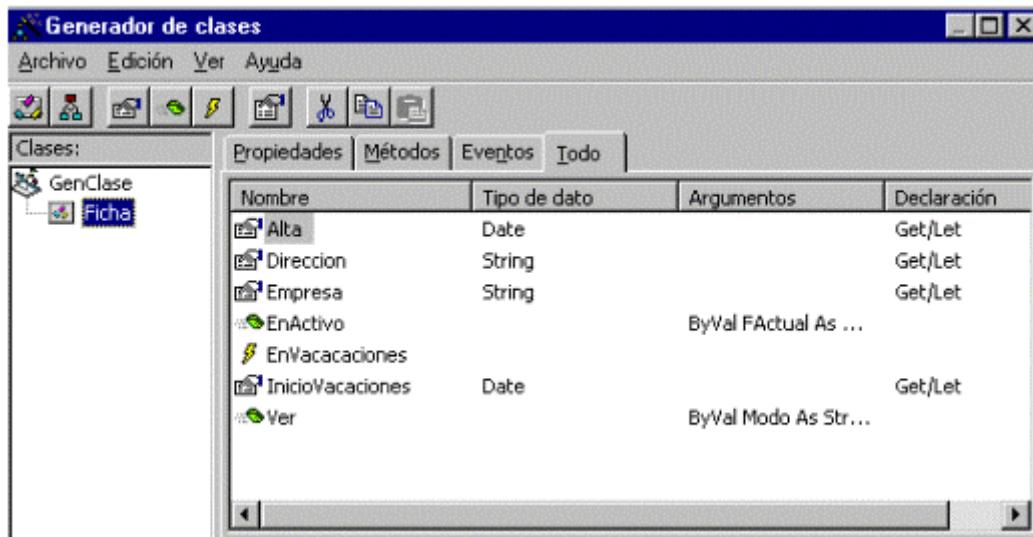


Figura 75. Generador de clases.

Una vez realizada la plantilla de la clase, el generador creará las definiciones para los diferentes elementos de la misma, debiendo encargarse el programador de escribir el código de los procedimientos.

En el panel izquierdo de esta ventana disponemos de las clases y colecciones incluidas en el proyecto actual. El panel derecho muestra los diferentes elementos de la clase o colección seleccionada. Podemos ver todos los elementos de la clase o según su categoría, en función de la ficha seleccionada en la ventana.

En la barra de herramientas disponemos de las opciones principales para trabajar con este asistente.

- **Agregar nueva clase.** Incorpora una nueva plantilla de clase al proyecto. Mediante esta ventana de diálogo damos nombre a la nueva clase. Es posible además documentar la clase incorporando información en la ficha Atributos de esta ventana. Esta ficha estará presente en la mayoría de las ventanas de creación de elementos de una clase, facilitándonos la tarea de proporcionar información sobre métodos, propiedades, etc.

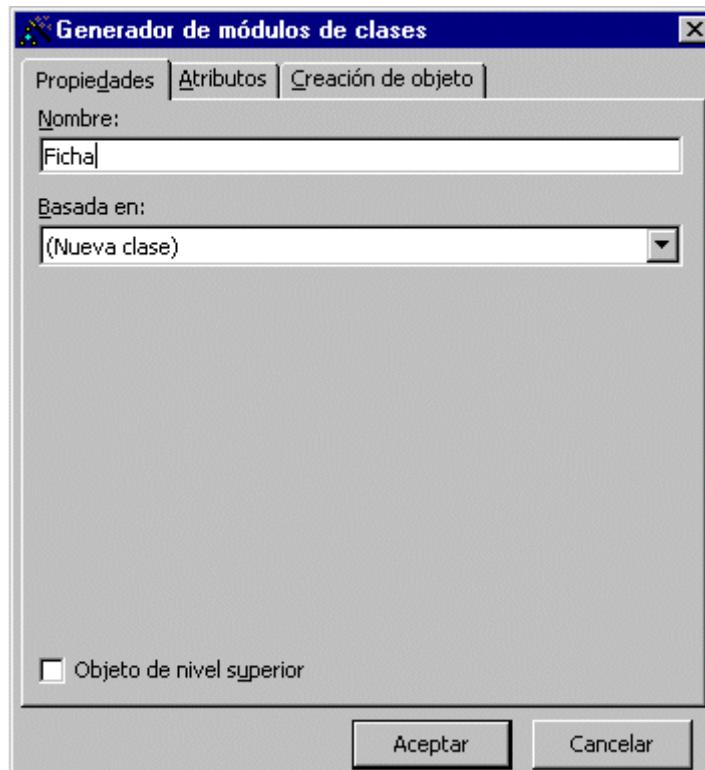


Figura 76. Ventana para agregar clase.

-  **Agregar nueva colección.** Crea una colección para guardar objetos de una clase existente en el proyecto o a partir de una nueva clase creada al mismo tiempo. La Figura 77 muestra la ventana para crear colecciones.

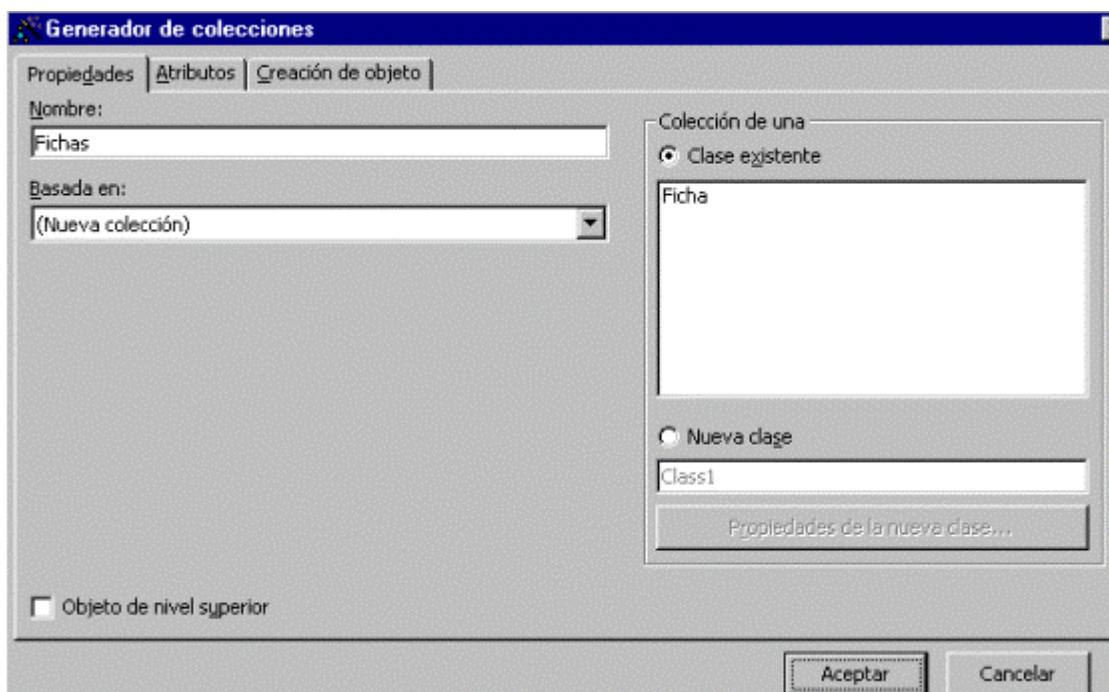


Figura 77. Ventana para agregar una colección.

-  **Agregar propiedad.** Esta opción permite incorporar una nueva propiedad a la clase, e indicar su tipo mediante la ventana de la Figura 78.

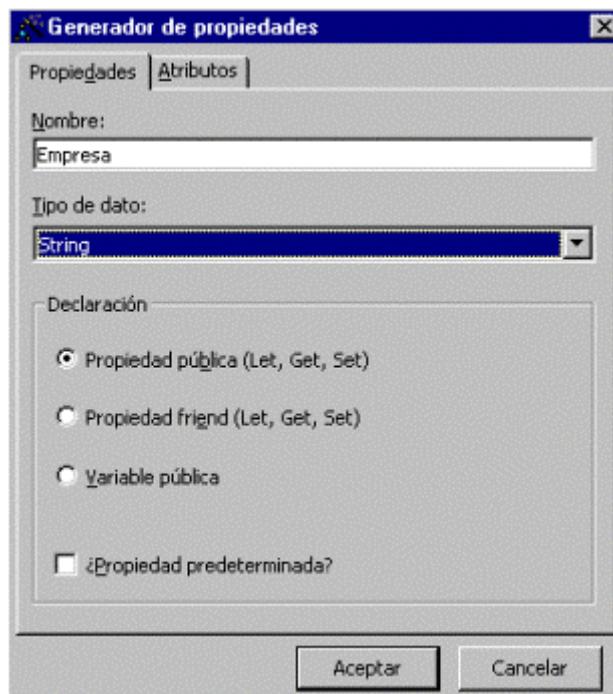


Figura 78. Ventana de creación de propiedades.

-  **Agregar método.** Mediante esta opción especificamos las características de un método para la clase: nombre, parámetros, tipo, etc.



Figura 79. Ventana de creación de métodos.

-  **Agregar evento.** Con esta opción creamos la declaración de un evento para la clase.



Figura 80. Ventana de definición de eventos.

Finalizada la creación de todos los componentes de la clase, el lector podrá comprobar que las definiciones resultantes de métodos, propiedades, etc., nos evitan una buena cantidad de trabajo a la hora de escribir código. El único inconveniente que podemos encontrar es que si estamos acostumbrados a una notación determinada en el código, es posible que la proporcionada por el generador no se adapte a la nuestra, lo que nos obligará a retocar parte del código generado.

Examinador de objetos (Object Browser)

Mediante esta estupenda herramienta, podemos visualizar las clases contenidas en las librerías estándar de objetos de Visual Basic, las que tengan establecida una referencia en el proyecto actualmente en desarrollo, y las clases propias de dicho proyecto. Adicionalmente es posible acceder desde el examinador al código de las clases propias del proyecto.

Supongamos que desde la aplicación de ejemplo OOCliente, iniciamos esta utilidad mediante la opción de menú de VB, Ver+Examinador de objetos o la tecla F2 y queremos que nos muestre todas las referencias a la palabra "ciudad" que existen en las clases de dicha aplicación. Para ello teclearemos dicha palabra en el ComboBox *Texto de búsqueda* y pulsaremos *Enter* o el botón de *Búsqueda*; el resultado nos mostrará cada una de las clases que contienen un miembro formado por esa palabra y los miembros correspondientes, teniendo en cuenta que el examinador de objetos también considera como clase a los módulos de código estándar, y como miembros a los procedimientos, métodos, propiedades, etc.

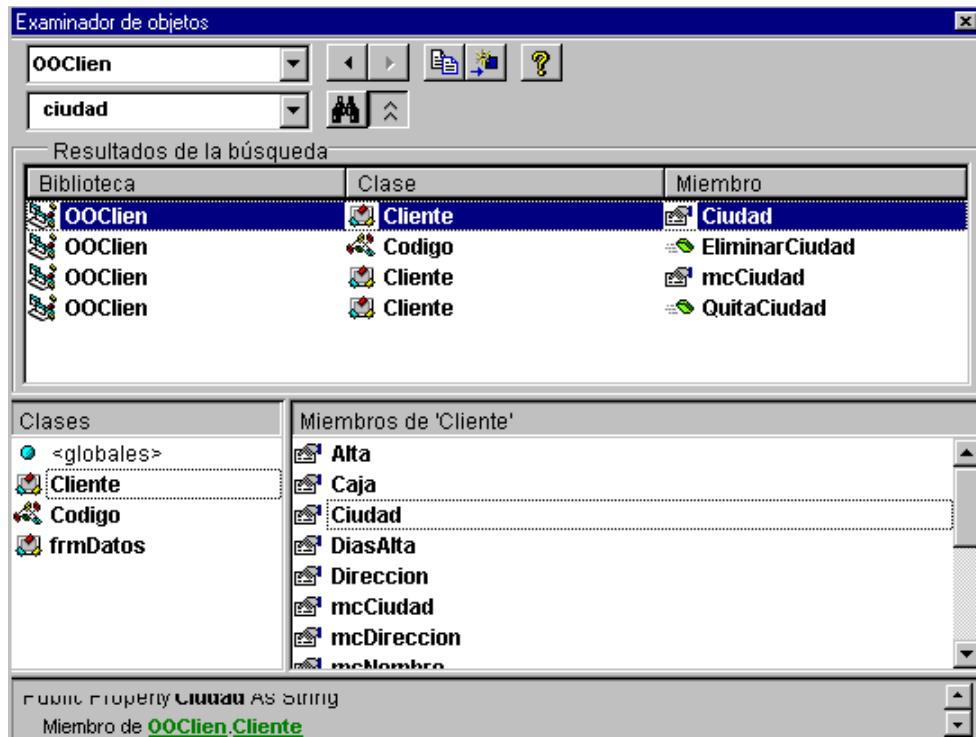


Figura 81. Examinador de objetos mostrando el resultado de una búsqueda.

Objetos del sistema

Visual Basic dispone de un conjunto de objetos, que son creados automáticamente cada vez que ejecutamos una aplicación, y destruidos al finalizar su ejecución. Estos objetos tienen un ámbito global, pueden usarse desde cualquier punto del programa, y su uso nos puede servir de apoyo en la programación de aspectos generales de nuestra aplicación, son los denominados *objetos del sistema*. A continuación, pasaremos a realizar una breve descripción de cada uno. Si el lector precisa de información más detallada, puede consultar la ayuda de VB, donde encontrará amplia información de cada uno de estos objetos.

- **App.** Proporciona información genérica sobre la aplicación que estamos ejecutando, tal como el nombre de su fichero ejecutable, título, versión, si es un ejecutable independiente o componente ActiveX, etc.
- **Clipboard.** Permite acceder y manipular el Portapapeles de Windows, realizando las acciones habituales del mismo: copiar, cortar y pegar.

Siempre que copiamos información al Portapapeles, hemos de borrar el contenido anterior utilizando el método *Clear()* de este objeto.

Es posible tener simultáneamente información de diferente tipo, como texto y gráficos, y recuperarla por separado. Pero debemos tener en cuenta, que cada vez que copiamos información en este objeto, desaparecerá la que hubiera anteriormente con el mismo tipo.

- **Debug.** Este objeto se emplea para enviar información a la ventana de Depuración de VB, estando disponible únicamente en tiempo de desarrollo de la aplicación.
- **Err.** Este objeto proporciona información sobre los errores producidos en tiempo de ejecución, de forma que puedan ser tratados por la aplicación. El programador también puede

utilizar este objeto para provocar errores, mediante su método `Raise()`, ante ciertas situaciones durante el transcurso del programa.

- **Printer.** Este objeto representa a una de las impresoras del sistema. Si disponemos de varias, hemos de emplear la colección `Printers`, para seleccionar una.

Un objeto `Printer` nos permite seleccionar el número de copias a imprimir de un documento, el tipo de fuente, tamaño del texto, etc.

- **Screen.** Emplearemos este objeto cuando necesitemos tener información sobre los formularios que se encuentran en pantalla, su tamaño, formulario y control activo, tipo de puntero del ratón, etc.

7

Desarrollo de un programa

Crear un nuevo proyecto

Como hemos visto en otros ejemplos, los elementos de un programa en VB se agrupan en el denominado proyecto: formularios, módulos de código, clases, etc.

En el proyecto de ejemplo [Primera](#), mostrado a continuación, se desarrolla un sencillo proyecto que incluye los mínimos elementos para dotar a un programa de una funcionalidad básica. Posteriormente se irán mostrando al lector nuevas características, que le permitirán mejorar sus aplicaciones.

Una vez iniciado el entorno de Visual Basic, comenzaremos seleccionando la opción de menú Archivo+Nuevo proyecto, eligiendo como tipo de proyecto el ya conocido EXE estándar, que nos mostrará en el entorno la ventana del diseñador visual del formulario.

En este momento, ya podríamos ejecutar el proyecto, que nos mostraría un formulario vacío. El lector puede pensar que lo ofrecido por defecto en el proyecto es poca cosa, sin embargo, comparado con otras herramientas de desarrollo para Windows, VB proporciona una gran cantidad de trabajo resuelto al programador, debido a que otros lenguajes requieren la creación del procedimiento de ventana, encargado de procesar los mensajes de Windows; ficheros de recursos; generación del ejecutable para realizar pruebas, etc.

Evidentemente, el simple formulario del que disponemos en este momento no es suficiente para el programa, pero nos ha evitado tener que escribir una considerable cantidad de código.

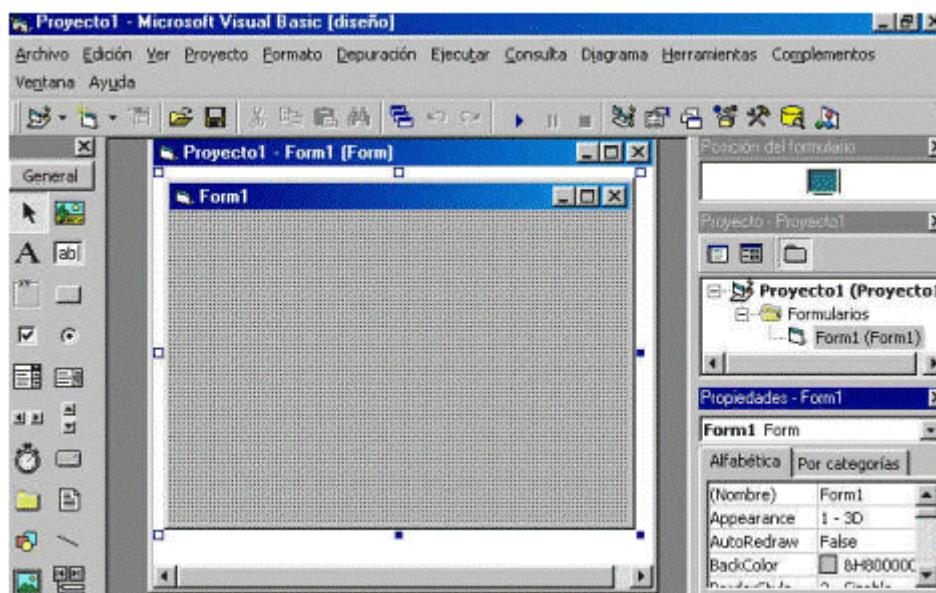


Figura 82. Entorno de VB mostrando un nuevo proyecto.

Configurar el entorno al inicio

Abriendo la ventana Opciones desde el menú de VB Herramientas+Opciones, la pestaña Entorno nos permite configurar detalles sobre el inicio de Visual Basic y el momento en que ponemos un programa en ejecución.

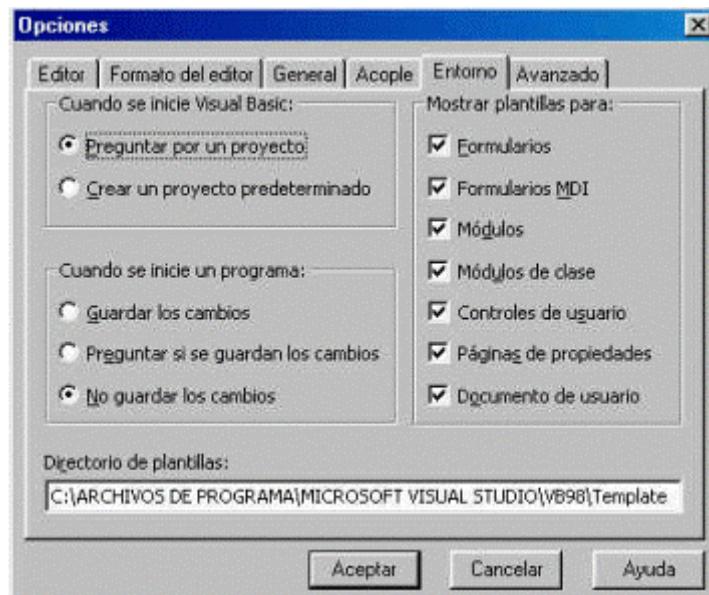


Figura 83. Opciones de configuración del entorno.

En el apartado Cuando se inicie Visual Basic, podemos establecer si se nos va a preguntar el tipo de proyecto a abrir, o se abrirá uno estándar.

En el apartado Cuando se inicie un programa, configuraremos el modo de grabación de los cambios efectuados hasta ese momento.

Acople de ventanas

Algunas de las ventanas que componen el entorno de desarrollo tienen la cualidad de poder acoplarse o pegarse a otras ventanas y a los laterales de la ventana principal de VB. Esta es una característica proporcionada al programador para que pueda organizarse de un modo más conveniente su espacio de trabajo.

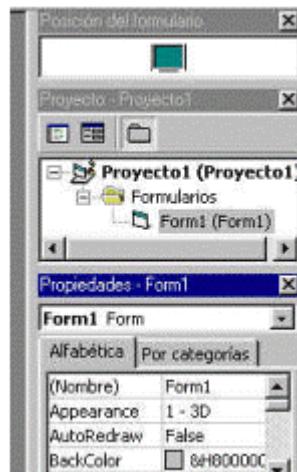


Figura 84. Ventana de proyecto acoplada a otras ventanas del IDE.

Una de las ventanas que por defecto se proporcionan acoplable es la correspondiente al explorador de proyecto, sin embargo, si no queremos disponer de esta característica, podemos deshabilitarla, pulsando con el botón derecho del ratón sobre la ventana acoplable y seleccionando de su menú contextual la opción **Acoplable** que se mostrará como una marca de verificación. Al desmarcar dicha opción, la ventana se comportará de manera estándar, como hija de una principal.

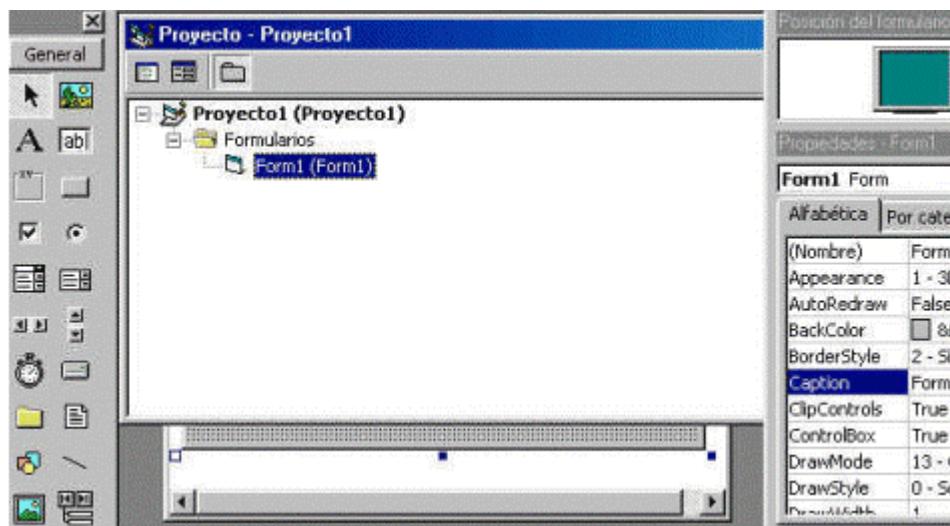


Figura 85. Ventana de proyecto sin característica de acople.

También es posible configurar el acople de ventana abriendo la ventana **Opciones** con la opción de menú de **VB Herramientas+Opciones**, y seleccionando la pestaña **Acople**, en donde podremos establecer que ventanas serán acopiables y cuales no.

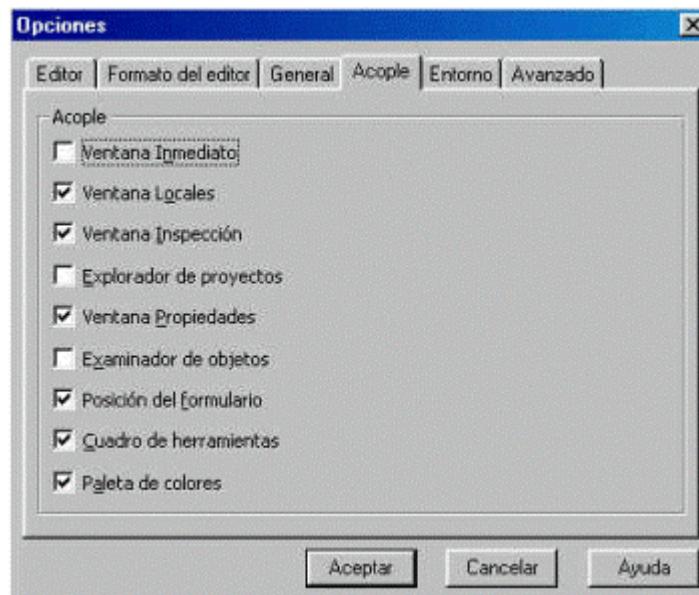


Figura 86. Opciones de acople de ventanas del entorno.

Las ventanas acopiables también disponen en su menú contextual de una opción que permite ocultarlas, de forma que ampliemos el espacio de trabajo para otras ventanas durante el tiempo que no sean necesarias.

Diseño del formulario

El primer paso que debemos dar es diseñar el interfaz de usuario del programa, es decir, el aspecto y elementos que tendrá el formulario para comunicarse con el usuario.

Esta labor la realizaremos mediante la ventana de diseño del formulario. Para acceder a esta ventana, nos situaremos en la ventana de proyecto y haremos doble clic sobre el formulario, o bien haremos un clic simple y seleccionaremos la opción de menú de VB Ver+Objeto.

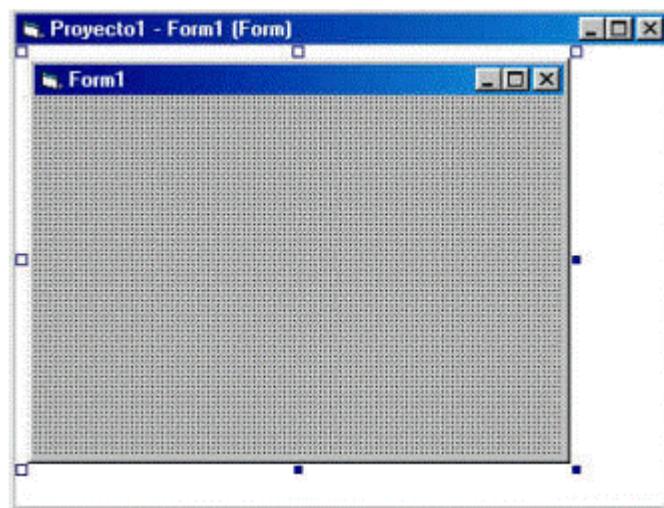


Figura 87. Ventana de diseño del formulario.

Esta ventana muestra el área del formulario sobre el que vamos a situar los controles que usaremos para comunicarnos con el usuario. Dispone de unas marcas de redimensión, que nos permiten variar visualmente el tamaño del formulario, pulsando y arrastrando sobre cualquiera de ellas.

La ventana de propiedades

Todos los elementos que vamos a manipular a partir de ahora, tanto formularios como controles, son objetos, que como ya sabemos, disponen de propiedades y métodos. Para acceder a las propiedades de un formulario durante la fase de diseño, usaremos la Ventana de Propiedades; que en el caso de estar oculta, podemos mostrar pulsando la tecla F4, mediante la opción de menú de VB Ver+Ventana Propiedades, o su botón de la barra de herramientas.



Figura 88. Botón para acceder a la ventana de propiedades.

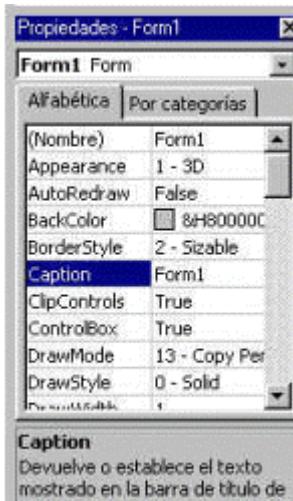


Figura 89. Ventana de propiedades.

En la parte superior de esta ventana, debajo del título, disponemos de una lista desplegable en la que podemos seleccionar el objeto al que vamos a manipular sus propiedades.

La columna izquierda, muestra los nombres de las propiedades del objeto seleccionado, mientras que la derecha corresponde a los valores de esas propiedades.

Podemos alterar el orden de presentación de las propiedades, pulsando sobre las pestañas situadas sobre las columnas de las propiedades. Si pulsamos en Alfabética, se mostrarán ordenadas por nombre, y si pulsamos Por categorías, se mostrarán agrupadas por grupos comunes de propiedades, además de poder contraer y expandir los grupos, pulsando sobre el símbolo existente junto a la categoría.

Para modificar una propiedad, debemos hacer clic sobre su valor y cambiarlo por uno nuevo.

Esta ventana dispone en la parte inferior de una breve descripción de la propiedad seleccionada. En el caso de que no necesitemos esta ayuda, podemos ocultarla haciendo clic con el botón derecho del ratón y desmarcando la opción Descripción, del menú contextual.

Propiedades del formulario

A continuación se muestran algunas de las propiedades más significativas de un objeto Form. Debido a que tanto formulario como controles disponen de un gran número de propiedades, las que no se indiquen aquí, pueden ser consultadas por el lector a través del sistema de ayuda de VB.

- Nombre. Esta es la propiedad más importante para cualquier objeto, ya que nos permite asignarle un identificador con el que podremos referirnos en el código a dicho objeto. Recomendamos al lector, consulte el apéndice dedicado a las convenciones de notación, en donde podrá encontrar ejemplos de como nombrar a los objetos de un proyecto (formularios, controles, etc.). En este caso, le hemos dado como nombre frmPrueba.
- BorderStyle. Permite establecer el tipo de borde para el formulario, que define el modo de comportamiento de la ventana: Sizable (redimensionable), Fixed Single (no redimensionable), etc. Este tipo de propiedades que contienen una lista de valores, muestran un botón en la propiedad, que permite abrir la lista y seleccionar el valor que necesitemos.
- Caption. Contiene una cadena con el título del formulario.
- BackColor. Establece el color de fondo del formulario. Esta propiedad muestra un botón al seleccionarse que abre un cuadro de diálogo, en el que al seleccionar la pestaña Paleta, nos permite elegir un nuevo color para el área del formulario.
- Icon. Esta propiedad muestra un botón en forma de puntos suspensivos, que abre un cuadro de diálogo al ser pulsado, para seleccionar un fichero de tipo ícono, que se asignará al formulario, mostrándolo en la parte superior izquierda, junto al título.
- MaxButton - MinButton. Estas propiedades permiten respectivamente mostrar en el formulario los botones de maximizar y minimizar que aparecen en la parte derecha del título. Al seleccionar una de ellas, permite abrir una lista con los valores a asignar.

En las propiedades con valores lógicos (True - False), es posible cambiar el valor existente haciendo doble clic sobre el nombre de la propiedad.

- ControlBox. Muestra u oculta el menú de control del formulario, dicho menú puede abrirse pulsando sobre el ícono en la parte superior izquierda del formulario.
- Picture. Con esta propiedad, podemos seleccionar un fichero que contenga una imagen para situarla como fondo del formulario. Para eliminar la imagen, haremos clic en el valor de esta propiedad y pulsaremos la tecla Suprimir.
- StartUpPosition. Permite seleccionar la posición de pantalla en la que se mostrará el formulario: asignada por Windows, centro de pantalla, etc.
- WindowState. Contiene un valor que indica el modo de visualización del formulario: normal, maximizado o minimizado.
- Top - Left. Contienen respectivamente las coordenadas superior e izquierda en donde será mostrado el formulario.
- Height - Width. Contienen respectivamente la altura y anchura del formulario.

Después de haber modificado alguna de las propiedades, el formulario podría presentar el aspecto que aparece en la Figura 90.

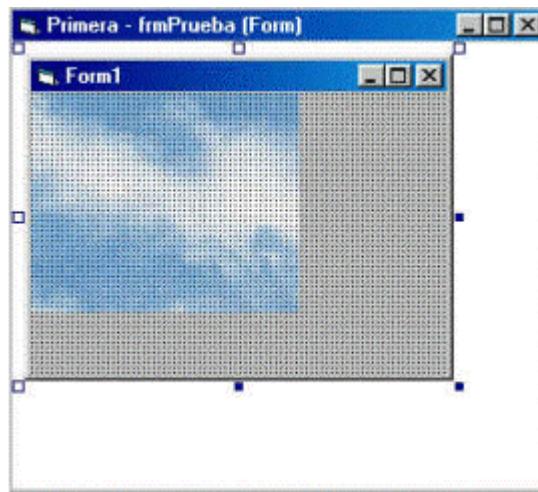


Figura 90. Diseñador de formulario después de modificar propiedades.

El utilizar un gráfico en la propiedad Picture, proporciona mayor vistosidad al formulario, sin embargo, encontramos el inconveniente de que el gráfico asignado a esta propiedad no se ajusta al tamaño del formulario, presenta el tamaño real del gráfico. Este problema será solventado empleando el control Image, que explicaremos más adelante. Por el momento, eliminaremos la imagen de la propiedad Picture, para poder trabajar con mayor claridad sobre la superficie del formulario.

Ventana de posición del formulario

Existe un medio más gráfico de establecer la posición de visualización del formulario, consistente en usar la Ventana de Posición del formulario. Dicha ventana puede abrirse, si no está ya disponible, mediante la opción de menú Ver+Ventana de Posición del formulario, o el botón de la barra de herramientas.



Figura 91. Botón para acceder a la ventana de posición del formulario.



Figura 92. Ventana de posición del formulario.

Esta ventana muestra la representación de un monitor, dentro del cual, aparecen los formularios existentes en el proyecto.

Haciendo clic, y arrastrando sobre la representación de alguno de los formularios mostrados, podemos situarlo en una nueva posición. Igualmente, es posible con el botón derecho del ratón, abrir un menú contextual con opciones para situar el formulario en posiciones predefinidas.

Esta ventana es particularmente útil cuando tenemos en el proyecto varios formularios y necesitamos disponer de una representación visual de sus posiciones antes de ejecutar.

Es posible que si esta ventana se encuentra oculta al abrir un proyecto, no se muestren los formularios. Este comportamiento erróneo puede ser debido a algún bug, por lo que para poder trabajar correctamente debemos tener esta ventana visible antes de abrir un proyecto.

Si al lector se le ha dado este caso, sólo debe reabrir el mismo proyecto sobre el que estaba trabajando (guardando los cambios previamente) con esta ventana abierta.

Asignar un nombre al proyecto

Cuando creamos un nuevo proyecto, VB le asigna por defecto el nombre Proyecto1, pero si necesitamos que tenga un nombre más descriptivo, podemos cambiarlo mediante la opción de menú de VB Proyecto+Propiedades de <NombreProyecto>..., que nos mostrará la ventana de propiedades del proyecto. En esta ventana podemos asignar un nuevo nombre al proyecto en el campo Nombre de proyecto.

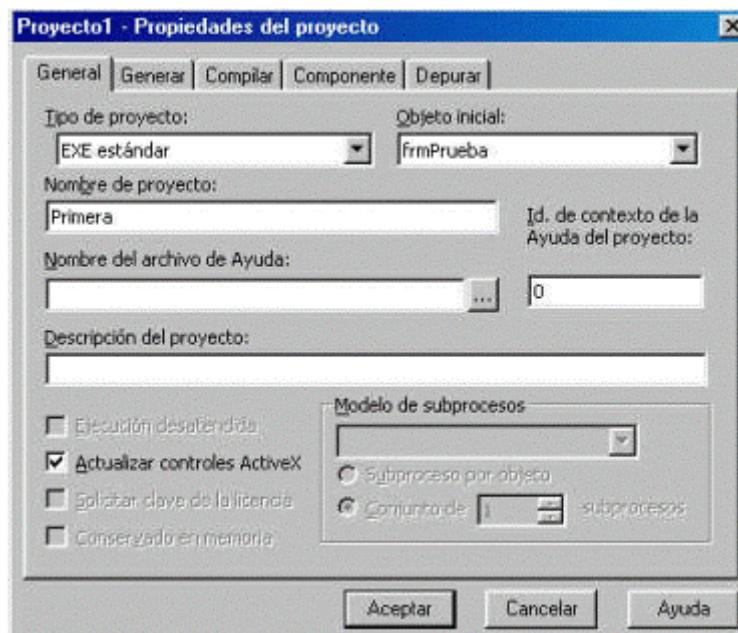


Figura 93. Ventana de propiedades del proyecto.

Controles

Los controles son aquellos elementos que podemos situar dentro del formulario y permiten la comunicación con el usuario, mostrando información y esperando respuesta por su parte.

En primer lugar hemos de abrir el Cuadro de herramientas, una ventana que, a modo de paleta, contiene los controles disponibles para ser insertados en el formulario. Podemos acceder a ella con la opción de menú Ver+Cuadro de herramientas o su botón en la barra de herramientas de VB.



Figura 94. Botón para acceder al cuadro de herramientas.



Figura 95. Cuadro de herramientas de VB.

Podemos insertar un control del Cuadro de herramientas en el formulario de dos formas:

- Hacer doble clic en el control, con lo cual se insertará directamente en el centro del formulario con un tamaño predeterminado.
- Hacer clic en el control, el puntero del ratón cambiará de forma. Pasar al formulario, y en el lugar donde queramos situar el control, pulsar y arrastrar el ratón, dibujando el control. Al soltar, aparecerá el control en el formulario.

Al situar el cursor del ratón sobre un elemento de esta ventana, se mostrará una cadena flotante con el nombre del control sobre el que estamos situados.

Cada vez que insertemos un control, tomará una serie de propiedades por defecto, que podremos cambiar para adaptar a los requerimientos de la aplicación.

Label

Visualiza un texto informativo para el usuario. Se trata de un control estático, ya que el usuario no puede interaccionar con él, sin embargo, mediante código es posible cambiar el texto mostrado. Uno de los usos más corrientes para los controles Label, es indicar al usuario el tipo de valor que debe contener otro control situado junto a él.



Icono del control Label en el Cuadro de herramientas.

Propiedades

- Caption. Muestra la cadena que se visualizará al usuario.

La Figura 96 muestra el aspecto del formulario después de incluir dos controles Label.

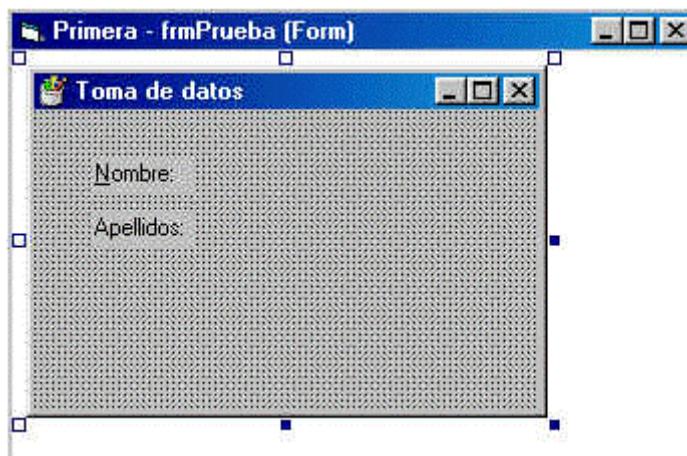


Figura 96. Formulario mostrando controles Label.

TextBox

Consiste en un cuadro en el que el usuario puede escribir información.



Figura 97. Icono del control TextBox en el Cuadro de herramientas.

Propiedades

- Text. Cadena de caracteres que se visualiza en el control.
- Enabled. Indica si el control está habilitado (puede escribirse en él) o no.
- MaxLength. Número máximo de caracteres que pueden introducirse en el control. Si se asigna 0, no hay límite de caracteres.
- BorderStyle. Indica el tipo de borde para el control. Los valores permitidos son:
 - 0. None. El control se muestra sin borde.
 - 1. Fixed Single. El control aparece con borde.

- PasswordChar. Asignando un carácter en esta propiedad, dicho carácter será visualizado en el TextBox en lugar del carácter real que teclee el usuario. Esta propiedad está especialmente indicada para tomar valores de tipo contraseña de usuario.
- ToolTipText. Introduciendo una cadena en esta propiedad, dicha cadena será visualizada en una viñeta flotante cuando el usuario sitúe el ratón sobre el control. De esta manera, podemos incluir una breve descripción sobre el cometido del control.
- MultiLine. Permite que el control disponga de una o varias líneas para editar el texto. Los valores admitidos son:
 - True. Se puede editar más de una línea de texto.
 - False. Sólo es posible escribir una línea en el control.
- ScrollBars. Valor que indica si un TextBox tiene barras de desplazamiento y de qué tipo. Obviamente, esta propiedad tiene sentido utilizarla cuando configuramos el TextBox para editar varias líneas. La Tabla 27 muestra las constantes disponibles para esta propiedad.

Constante	Valor	Descripción
vbSBNone	0	(Predeterminado) Ninguna
vbHorizontal	1	Horizontal
vbVertical	2	Vertical
vbBoth	3	Ambas

Tabla 27. Valores para la propiedad ScrollBars.

A continuación, pasaremos al formulario y dibujaremos dos TextBox, uno para introducir un nombre, y en el otro los apellidos, a los que se ha asignado en la propiedad nombre: txtNombre y txtApellidos respectivamente.

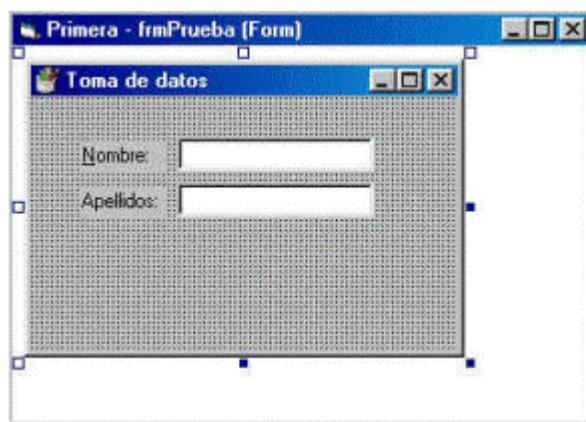


Figura 98. Formulario mostrando los TextBox creados.

En estos TextBox se ha eliminado el valor que por defecto tiene su propiedad Text, para que inicialmente no presenten ningún valor en el formulario. También se ha introducido una breve descripción de la tarea del control en la propiedad ToolTipText, de manera que el usuario sepa qué valor introducir en cada uno.

Ubicación precisa de controles

La ventana de diseño del formulario tiene una serie de puntos o cuadrícula, que nos permiten situar y dimensionar con mayor precisión un control en una posición determinada. Por defecto, el espacio de separación entre puntos es grande, lo que causa que el ajuste preciso de un control sea difícil.

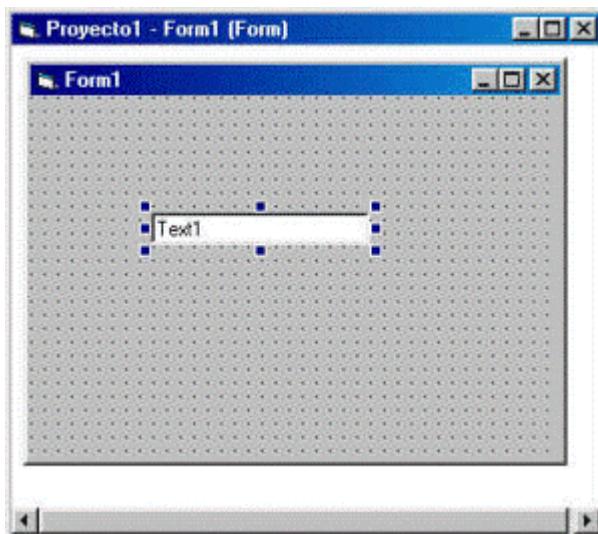


Figura 99. Diseñador de formulario mostrando cuadrícula para ajuste de controles.

Este inconveniente puede ser solventado en la ventana Opciones (menú de VB Herramientas + Opciones), pestaña General, apartado Opciones de la cuadrícula.

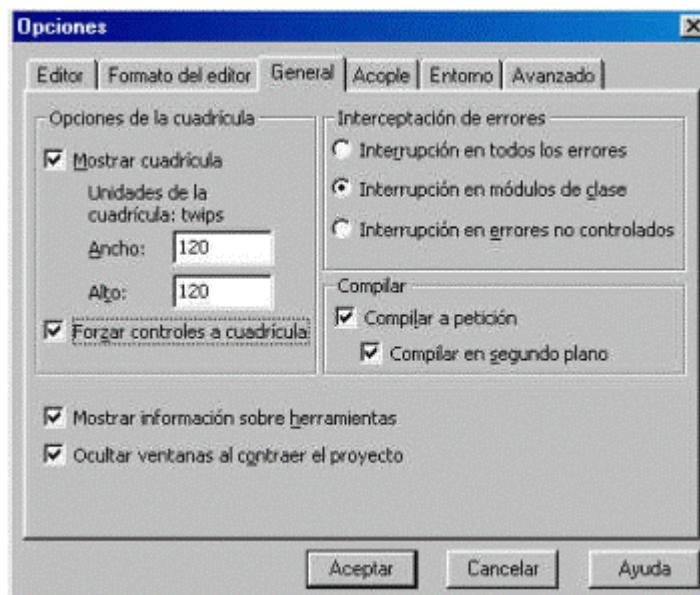


Figura 100. Opciones del entorno, pestaña General.

Si desmarcamos la opción Forzar controles a la cuadrícula, VB no obligará a que el control se pegue a los puntos de la cuadrícula del formulario, pudiendo situarlo entre dos filas de puntos. Si además, no queremos utilizar la cuadrícula como guía, desmarcaremos también la opción Mostrar cuadrícula.

Otra solución reside en reducir el espacio de separación entre puntos de la cuadrícula, que indican los campos Ancho y Alto; situando un valor por ejemplo de 50.

Operaciones con grupos de controles

Para realizar una acción sobre un conjunto de controles de un formulario, en primer lugar debemos seleccionar los controles necesarios. Para ello, situaremos el ratón sobre un punto y haremos clic, arrastrando y abarcando los controles a seleccionar; finalmente soltaremos y reconoceremos los controles seleccionados porque muestran sus marcas de dimensión.

También es posible seleccionar varios controles, haciendo clic sobre cada control, manteniendo pulsada la tecla Control o Shift (Mayúscula).

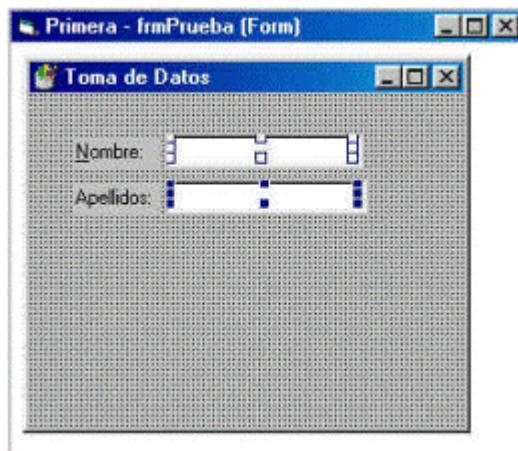


Figura 101. Formulario con grupo de controles seleccionados.

Una vez seleccionados, podemos mover los controles haciendo clic sobre cualquiera de los seleccionados y arrastrando a una nueva posición. Si queremos moverlos con el teclado tendremos que utilizar las teclas de desplazamiento, mientras mantenemos pulsada la tecla Control.

También es posible redimensionar los controles, utilizando las teclas de desplazamiento y manteniendo pulsada la tecla Mayúsculas; esto es en el caso de controles seleccionados, ya que si necesitamos redimensionar un único control, podemos hacerlo también con el ratón.

Si necesitamos realizar alguna operación de alineación de un grupo de controles, situación en el formulario, modificación de espacio entre controles, etc.; recurriremos a las opciones del menú Formato de VB, que el lector puede aplicar sobre una selección de controles para ver el resultado.

Una vez que establezcamos la ubicación definitiva de los controles, sobre todo en formularios con un elevado número, es conveniente seleccionar la opción Bloquear controles del menú Formato, que impedirá la modificación de posición y tamaño accidentalmente de cualquiera de ellos. Seleccionando de nuevo esta opción, desbloquearemos los controles, permitiendo nuevamente su modificación.

Al seleccionar un grupo de controles de diferente tipo, la ventana de propiedades muestra las propiedades comunes a todos ellos. La Figura 102 muestra esta ventana, después de haber seleccionado un control Label y un TextBox.

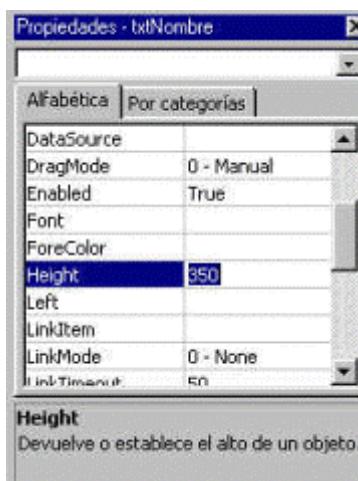


Figura 102. Propiedades comunes en un grupo de controles seleccionados.

Si modificamos alguna de estas propiedades, los cambios afectarán a todo el grupo de controles seleccionado.

Copiar y pegar controles

Supongamos que debemos diseñar un formulario que incluye varios TextBox con las mismas características (a excepción del nombre). En este caso, después de crear y configurar el primero de estos controles, lo seleccionaremos y copiaremos al Portapapeles mediante alguna de las acciones estándar de Windows: teclas Control+C, opción Copiar del menú Edición, etc.

A continuación lo pegaremos empleando también el modo habitual de Windows: teclas Control+V, opción Pegar del menú Edición, etc. Antes de ser pegado, VB nos mostrará el aviso que nos muestra la Figura 103.

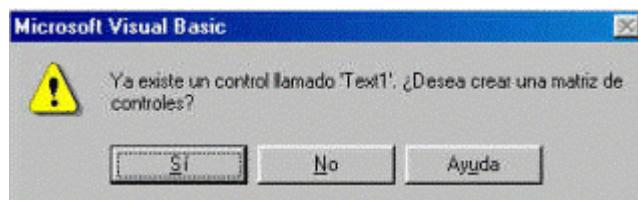


Figura 103. Aviso para crear un array de controles al pegar un control en el formulario.

Al detectar que se está pegando un control idéntico a uno existente, es posible crear un array o matriz de controles con las mismas propiedades. Contestaremos de momento No a este aviso, ya que los arrays de controles son tratados en un apartado posterior.

El nuevo control pegado, aparecerá en la parte superior izquierda del formulario. Desde aquí, lo situaremos en una nueva posición, y repetiremos esta acción tantas veces como controles necesitemos pegar.

Conceptos de foco de la aplicación y cambio de foco

Un control que tiene el foco de la aplicación, es aquel que recibe todos los eventos y acciones que toma el usuario, hasta que este decide cambiar el foco a otro control del mismo u otro formulario.

Las formas de cambiar el foco entre controles de un mismo formulario son: pulsando el ratón sobre el nuevo control o con la tecla Tabulador.

Mientras que el uso del ratón nos permite cambiar el foco entre controles sin seguir un orden preestablecido, empleando el teclado, el orden del cambio de foco viene determinado por la propiedad TabIndex. Esta propiedad, disponible en todos los controles, contiene el número de orden en el que cambiará el foco entre los controles al pulsar Tabulador, de manera que durante el diseño del formulario, podemos alterar dicho orden, adaptándolo a nuestros requerimientos.

Cada vez que agregamos un nuevo control, la propiedad TabIndex toma un valor asignado automáticamente por Visual Basic. Dicho valor podemos cambiarlo si queremos establecer un orden distinto para el paso de foco entre controles, teniendo en cuenta que no puede haber dos controles con el mismo valor en esta propiedad.

CommandButton

Representa un botón que al ser pulsado, desencadena una determinada acción. Es posible definir una tecla de atajo al igual que en los menús, en la propiedad Caption, mediante el uso del signo "&" junto a una letra, con lo cual quedará subrayada dicha letra.



Figura 104. Icono del control CommandButton en el Cuadro de herramientas.

Propiedades

- **Caption.** Cadena que se utiliza como título del botón. Si anteponemos el signo '&' a una de las letras, dicha letra aparecerá subrayada, indicando un atajo de teclado, de forma que podemos accionar el botón pulsando la combinación de teclas Alt+LetraResaltada.
- **Style.** Permite establecer el modo en que puede ser visualizado el botón. Con la apariencia estándar de Windows o incluyendo una imagen.
 - **Standard.** Apariencia estándar de Windows.
 - **Graphical.** Mostrando una imagen.
- **Picture.** Si la propiedad Style del botón está en modo gráfico, podemos asignar a esta propiedad un fichero de imagen para que sea mostrado por este control.
- **MousePointer.** Contiene el tipo de cursor que será mostrado al pasar el ratón sobre este control.
- **MouseIcon.** Si a la propiedad MousePointer se le asigna el valor Custom, podemos establecer en esta propiedad un ícono que hará las veces de cursor sobre el botón.

- Default. Contiene un valor lógico. Cuando sea True, al pulsar el usuario la tecla Enter, se producirá el mismo efecto que si se hubiera hecho clic con el ratón en el botón: se desencadenará el evento Click() de este control. Consulte el lector el apartado Codificación de eventos en este mismo tema, sobre los eventos para controles y su forma de codificación.
- Cancel. Contiene un valor lógico. Cuando sea True, al pulsar el usuario la tecla Escape, se producirá el mismo efecto que si se hubiera hecho clic con el ratón en el botón: se desencadenará el evento Click() de este control.

En el formulario se han añadido dos CommandButton; cmdCancelar, que se mostrará al estilo normal, y cmdAceptar, que contendrá una imagen y además un cursor personalizado.

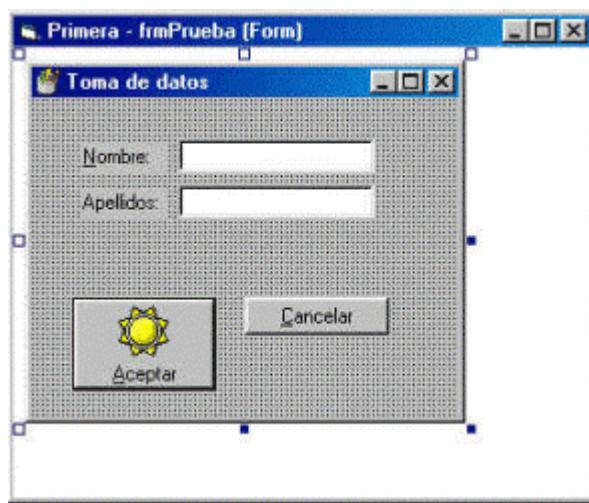


Figura 105. Formulario mostrando los controles CommandButton.

CheckBox

La finalidad de este control es proporcionar al usuario una elección del estilo Verdadero/Falso. Para ello, dispone de un pequeño cuadro que podemos marcar (signo de verificación) para aceptar la opción que propone el control, o desmarcar para no aceptarla.



Figura 106. Icono del control CheckBox en el Cuadro de herramientas.

Propiedades

- Value. Indica el estado del CheckBox, pudiendo utilizarse alguna de las siguientes constantes:
 - Unchecked. No marcado, valor por defecto.
 - Checked. Marcado.
 - Grayed. Atenuado. En este estado el control se muestra marcado y bajo un tono gris, esto no significa sin embargo, que el control se encuentre deshabilitado, sigue estando operativo.

- Style. Define el aspecto y comportamiento del control. Las constantes empleadas con este control son:
 - vbButtonStandard. El control se muestra en su forma habitual.
 - vbButtonGraphical. El control se muestra con el aspecto de un CommandButton. Al marcarlo, quedará hundido hasta que se vuelva a desmarcar.

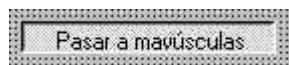


Figura 107. CheckBox gráfico marcado.

- Alignment. Esta propiedad permite establecer la posición del texto y del control. Los valores a emplear aparecen en la Tabla 28.

Constante	Valor	Descripción
vbLeftJustify	0	Valor por defecto. El control se encuentra en la parte izquierda y el texto asociado en la derecha.
vbRightJustify	1	El control se encuentra en la parte derecha y el texto asociado en la izquierda.

Tabla 28. Constantes para la propiedad Alignment del control CheckBox.

En el formulario del ejemplo situaremos un CheckBox, con el nombre chkMayusc, que nos servirá para convertir a mayúsculas el contenido de uno de los TextBox cuando se marque este control, y pasarlo a minúsculas cuando se desmarque. Lo veremos al tratar la codificación de eventos para el formulario.

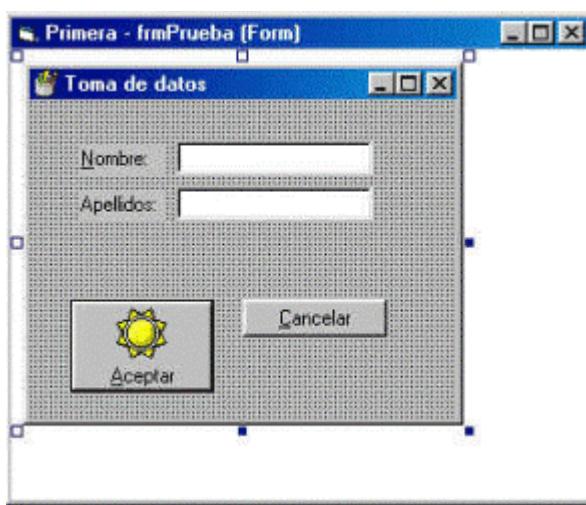


Figura 108. CheckBox incluido en el formulario de pruebas.

Con este control ya hemos completado la fase de diseño del formulario. Solamente quedaría agregar un control Label adicional, al que daremos el nombre lblCuentaLetras, junto al TextBox txtNombre,

que nos servirá para visualizar el número de caracteres introducidos en txtNombre. Este y el resto de aspectos sobre la manipulación de código, serán tratados en el apartado dedicado a la codificación de eventos.

La ventana de código

Un formulario, al igual que sucede con los módulos de código estándar y de clase, dispone de su propia ventana de código, en donde se podrán escribir tanto procedimientos normales como el código para los eventos de los objetos que contenga.

Podemos acceder a esta ventana de las siguientes formas:

- Desde la ventana de diseño del formulario, mediante la opción de menú de VB Ver+Código, o el menú contextual de esta ventana de diseño, opción Ver código.
- Desde el explorador de proyectos, seleccionando el formulario y abriendo el menú contextual, opción Ver código.

```

Private Sub Form_QueryUnload(Cancel As Integer, Unload As Boolean)
  Dim InRespuesta As Integer
  InRespuesta = MsgBox("¿Finalizar la ejecución del programa?", vbQuestion + vbOKCancel, "")
  If InRespuesta = vbCancel Then
  Cancel = 1
  End If
  End Sub
  
```

Figura 109. Ventana de código del formulario.

En este apartado vamos a realizar un detallado análisis de esta ventana, de forma que podamos sacar el máximo provecho de todas sus funcionalidades.

Configuración del editor de código

En la ventana Opciones de VB (opción de menú Herramientas+Opciones), disponemos de la pestaña Editor, en la que podemos configurar varias características.

- Comprobación automática de sintaxis. Comprueba cada línea de código al terminar de escribirla.
- Requerir declaración de variables. Obliga a declarar las variables en el código, situando una declaración Option Explicit en cada nuevo módulo agregado al proyecto.
- Lista de miembros automática. Muestra una lista de valores disponibles para ciertos elementos del lenguaje que se vayan a escribir como tipos de datos, propiedades, métodos, etc.

- Información rápida automática. Muestra una breve descripción sobre funciones y parámetros necesarios.
- Sugerencias de datos automáticas. En modo de depuración, muestra el valor de variables y expresiones del código que esté en ejecución.
- Sangría automática. Al introducir una tabulación en una línea de código, las siguientes líneas comenzarán en la misma tabulación.

Pulsaremos la tecla Tabulador o Mayúsculas+Tabulador para indentar o quitar la indentación respectivamente a una línea o conjunto de líneas seleccionado. También podemos realizar esta misma operación mediante dos opciones del menú Edición: Aplicar sangría y Anular sangría.

- Ancho de tabulación. Permite establecer la cantidad de espacios entre cada tabulación del código.
- Modificar texto con arrastrar y colocar. Permite mover porciones de código entre las ventanas de código, y hacia las ventanas del depurador.
- Ver módulo completo de forma predeterminada. Muestra todo el código del módulo o por procedimientos.
- Separador de procedimientos. Visualiza una línea que indica el final de un procedimiento y comienzo del siguiente.

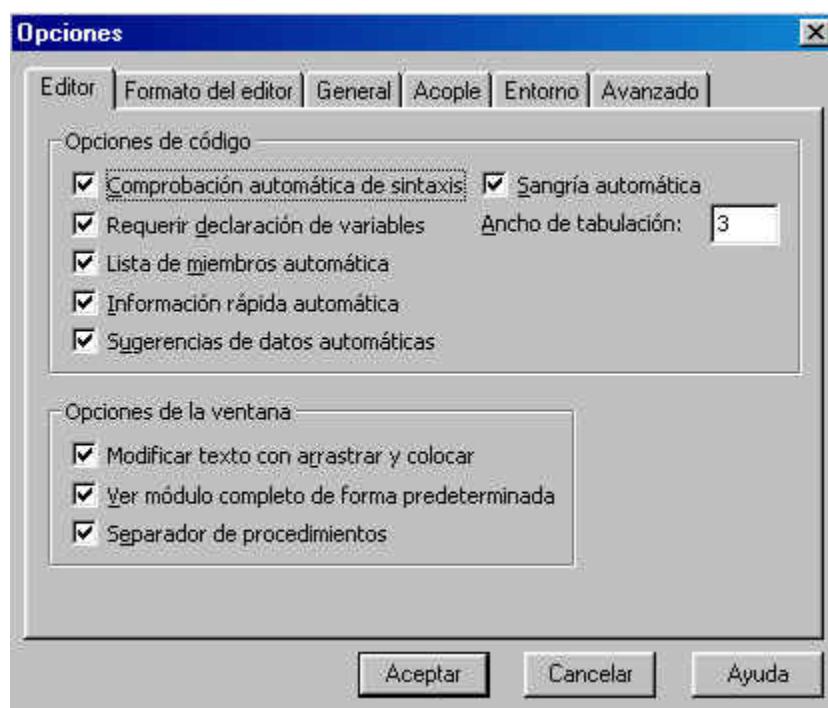


Figura 110. Opciones del editor de código.

El editor de código trabaja con un tipo de letra por defecto y una configuración de colores para los diferentes elementos del lenguaje. La pestaña Formato del editor en esta misma ventana de opciones nos permite también modificar estas características para adaptarlas a nuestras preferencias.

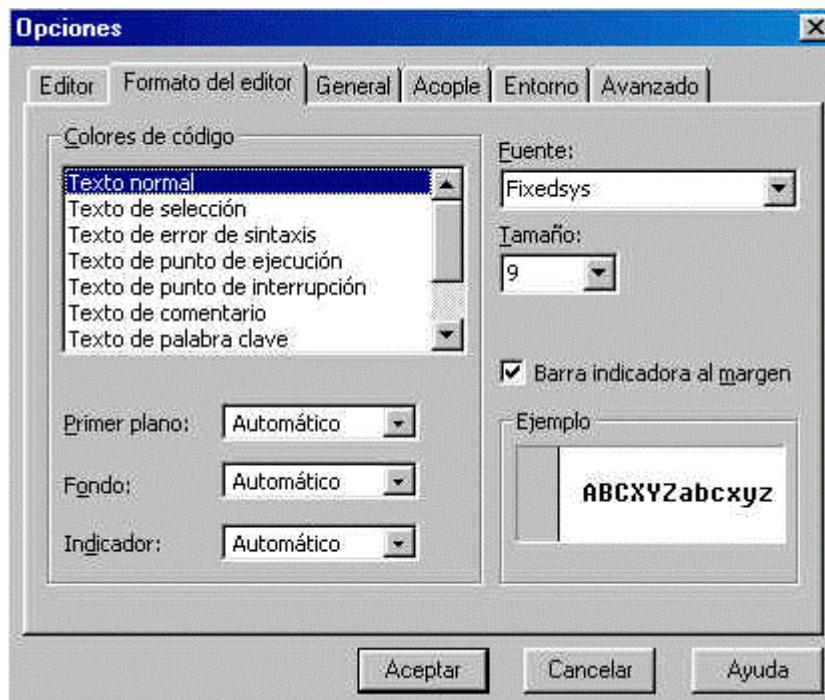


Figura 111. Opciones para el formato del editor de código.

- Colores de código. Lista de elementos del código sobre los que se puede modificar la configuración de colores para identificarlos más rápidamente.
- Fuente. Tipo de letra mostrado en el editor.
- Tamaño. Tamaño del tipo de letra.
- Barra indicadora al margen. Permite mostrar una columna en el margen izquierdo del editor en la que se podrán visualizar los puntos de ruptura del depurador, marcadores, etc.

Modos de visualización del código

Dentro de esta ventana podemos mostrar todo el código contenido o por procedimientos independientes pulsando sobre los botones situados en la parte inferior izquierda.



Figura 112. Botones de la ventana de código para mostrar el código completo o por procedimientos.

División del área de edición.

Para evitar tener que movernos constantemente entre dos zonas de un procedimiento o del módulo, podemos dividir la ventana en dos partes independientes.

```

Private Sub cmdCancelar_Click()
    'Unload FrmPrueba ' -Me- es más recomendable
    Me.cmdAceptar.Caption = "VALE"
    Me.txtApellidos.MaxLength = 5
    Me.txtNombre.PasswordChar = "="
End Sub

Private Sub cmdAceptar_Click()
    MsgBox "Nombre: " & Me.txtNombre.Text & vbCrLf & _
        "Apellidos: " & Me.txtApellidos.Text, vbInformation
End Sub

```

Figura 113. Ventana de código dividida en zonas independientes.

El modo de conseguir esta característica es mediante el menú de VB Ventana+Dividir, o haciendo clic y arrastrando en el indicador de división de la ventana, situado en la parte derecha, encima de la barra de desplazamiento vertical.

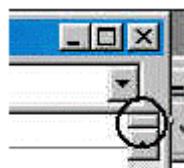


Figura 114. Indicador de división de la ventana de código.

Para devolver la ventana a su estado inicial, volveremos a desplazar el indicador de división a su lugar original o seleccionaremos de nuevo la opción de menú.

Búsquedas de código

La opción Edición+Buscar, del menú de VB, nos muestra un cuadro de diálogo en el que podemos realizar búsquedas y reemplazamiento de código, indicando el área de búsqueda (procedimiento, módulo, etc.) y otra serie de parámetros que nos permitirán ajustar dicha búsqueda.

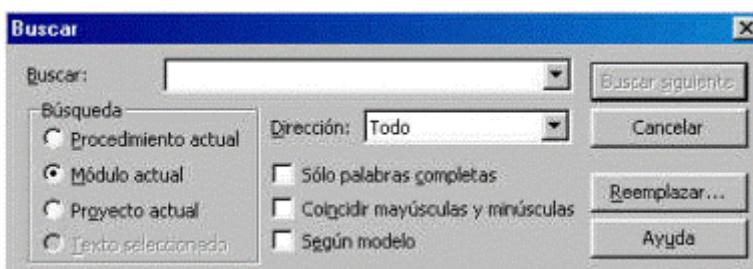


Figura 115. Cuadro para realizar búsquedas en la ventana de código.

Asistentes durante la escritura

El menú Edición dispone de una serie de opciones que facilitan el proceso de escritura del código, mostrando diferentes listas de valores correspondientes al elemento a codificar o ayuda en línea.

- Mostrar propiedades y métodos. Visualiza una lista con las propiedades y métodos disponibles para un objeto, al poner el punto a continuación del objeto.

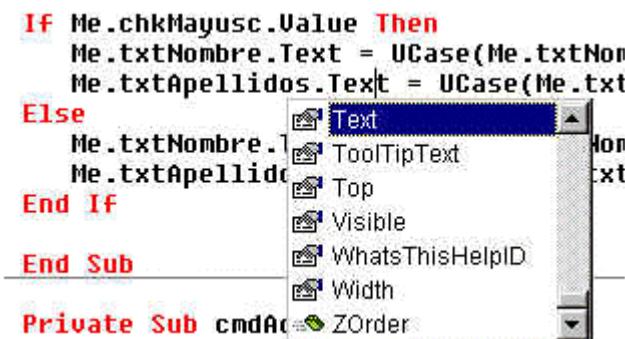


Figura 116. Lista de propiedades y métodos para un control TextBox.

- Mostrar constantes. Abre una lista con los posibles valores a asignar a una variable de enumeración, o a un parámetro que admite un conjunto de constantes.

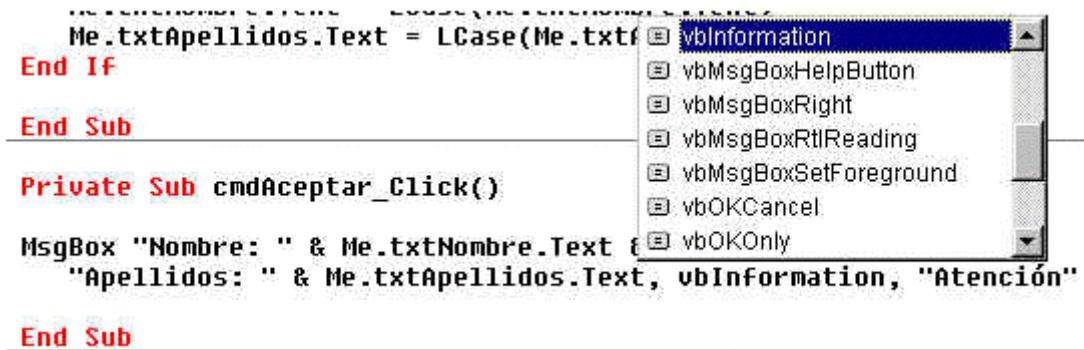


Figura 117. Lista de constantes disponibles para un parámetro de MsgBox().

- Información rápida. Muestra información sobre la sintaxis del elemento seleccionado en la ventana de código.
- Información de parámetros. Muestra el formato de llamada a un procedimiento o función en una viñeta flotante.

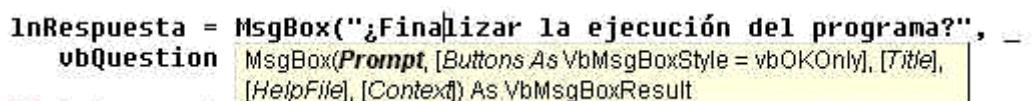


Figura 118. Información de parámetros para MsgBox().

- Palabra completa. Finaliza la escritura de una palabra si a VB le es posible reconocerla.

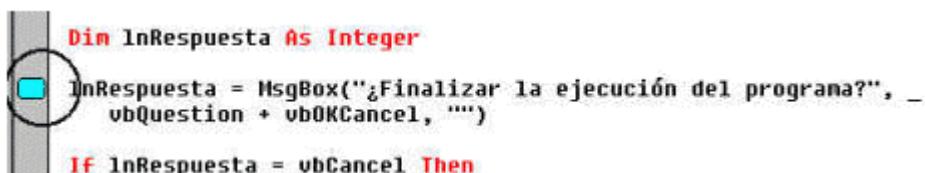
Marcadores

Un marcador es una señal que se establece en el margen de la ventana de edición y nos permitirá desplazarnos rápidamente entre todos los establecidos en el código del proyecto.

Los marcadores no se conservan entre sesiones de trabajo con el proyecto, por lo que al abrir nuevamente un proyecto, deberemos establecer otra vez los marcadores que necesitemos.

El menú de VB Edición+Marcadores, contiene las siguientes opciones para el manejo de marcadores en el código.

- Alternar marcador. Añade un nuevo marcador o elimina uno existente. El nuevo marcador se reconoce por una señal junto a la línea de código.



```

Dim InRespuesta As Integer
InRespuesta = MsgBox("¿Finalizar la ejecución del programa?", _
vbQuestion + vbOKCancel, "")
If InRespuesta = vbCancel Then

```

Figura 119. Marcador establecido para una línea de código.

- Marcador siguiente. Nos sitúa en el siguiente marcador del código.
- Marcador anterior. Nos sitúa en el marcador anterior del código.
- Borrar todos los marcadores. Elimina todos los marcadores establecidos.

Barras de herramientas

Visual Basic dispone de una serie de barras de herramientas, que incluyen las opciones más usuales organizadas por categorías, evitando la navegación entre los menús para las tareas más repetitivas.

La barra más conocida es la Estándar, que aparece por defecto debajo de los menús de VB. Todas las barras pueden acoplarse a los laterales de la ventana principal del IDE o situarse en modo flotante. Para visualizar las barras, debemos seleccionar la opción de menú Ver+Barras de herramientas, que mostrará un submenú con las barras existentes.

Otro medio más fácil de mostrar una barra, consiste en hacer clic con el botón derecho del ratón sobre una barra visible, lo que abrirá un menú contextual con las barras disponibles. La Figura 120 muestra la barra Edición.

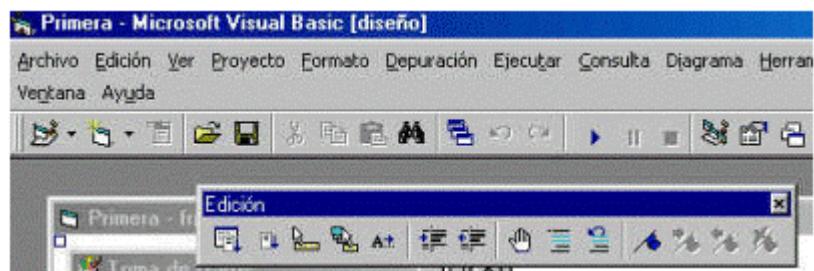


Figura 120. Entorno de VB con la barra de herramientas Edición en modo flotante.

Otra opción muy interesante, consiste en modificar las barras existentes, añadiendo o quitando elementos, y la posibilidad de crear nuevas barras personalizadas.

Para crear una nueva barra, seleccionaremos la opción Personalizar en el submenú que contiene las barras de herramientas, apareciendo la Figura 121.

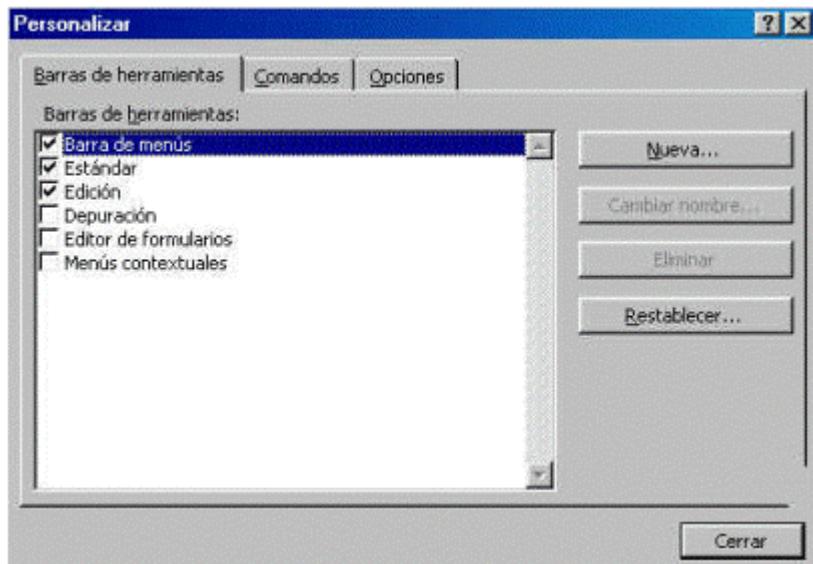


Figura 121. Personalizar barras de herramientas en VB.

A continuación, pulsaremos el botón Nueva, que nos pedirá el nombre que le vamos a dar a la barra, como podemos ver en la Figura 122.

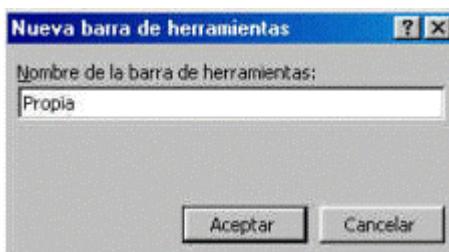


Figura 122. Asignar nombre a la barra de herramientas.

Al aceptar este cuadro, se creará la nueva barra de herramientas vacía. Es posible que quede debajo de la ventana Personalizar, por lo que deberemos mover esta última hasta localizar la barra, de manera que tengamos ambas accesibles.

Pulsaremos la pestaña Comandos en la ventana Personalizar, que nos mostrará todos los comandos disponibles en los menús de VB. Haciendo clic y arrastrando hasta la barra de herramientas, iremos confeccionando una barra de manera personalizada, con las opciones que estimemos más necesarias.

Los anteriores pasos son también válidos para las barras propias de VB, ya que todas son modificables.

Mientras estemos en modo de personalización de las barras, si hacemos clic con el botón derecho sobre alguno de sus botones, o pulsamos el botón Modificar selección de la ventana Personalizar, se abrirá un menú contextual que nos permitirá cambiar el ícono que muestran, eliminar el botón de la barra y otras opciones varias.

Terminada la creación de nuestra barra personalizada, pulsaremos el botón Cerrar en la ventana Personalizar, con lo que quedará lista para usar.



Figura 123. Nueva barra de herramientas personalizada.

La ventana Personalizar, dispone además de la pestaña Opciones, en la que podemos hacer que los menús de VB se muestren mediante un efecto animado, que los iconos de las barras de herramientas aumenten de tamaño y que se visualicen los atajos de teclado en las viñetas flotantes de los botones.

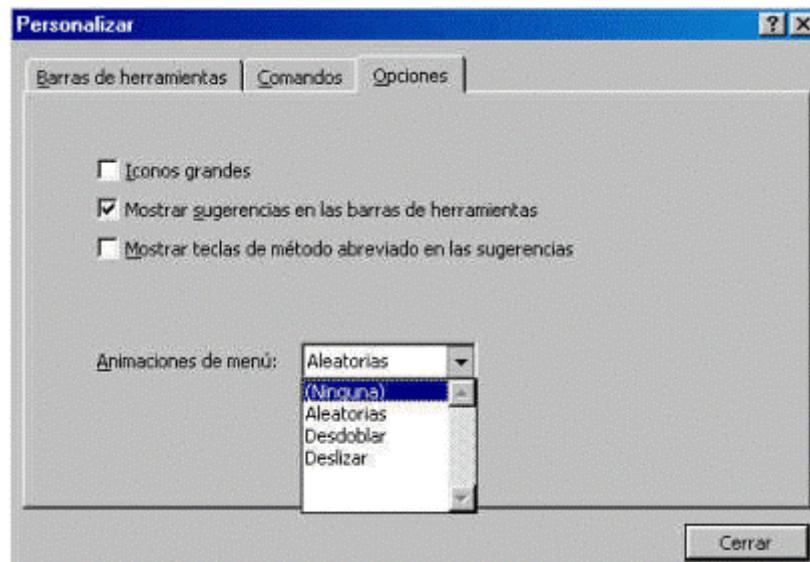


Figura 124. Opciones de configuración para barras de herramientas.

Existen dos opciones para la edición de código que se encuentran en la barra de herramientas Edición, pero que sin embargo no disponemos en el menú del mismo nombre, se trata de Bloque con comentarios y Bloque sin comentarios, que permiten comentar y descomentar respectivamente un conjunto de líneas de código seleccionadas. Dada su enorme utilidad a la hora de escribir el código, hemos considerado necesario comentar este detalle para que sea tenido en cuenta por el lector.

Los botones de la barra de herramientas correspondientes son los resaltados en la Figura 125.



Figura 125. Barras de herramientas Edición.

Codificación de eventos

Un evento es un suceso que se desencadena dentro de un formulario o control, debido a una acción del usuario, un mensaje enviado por otra aplicación o por el sistema operativo.

Los eventos se codifican empleando una declaración de tipo Sub, por lo que formalmente, no hay diferencias entre un evento y un procedimiento normal. Sin embargo, en el ámbito conceptual existe una importante diferencia entre un evento y un procedimiento.

La ejecución de un procedimiento es labor del programador, que es quien debe poner expresamente en el código, la llamada a dicho procedimiento para que sea ejecutado.

Un evento es algo que no sabemos cuando va a suceder, o tan siquiera si va a suceder. Un formulario, por ejemplo, puede tener un botón que no sea pulsado por el usuario durante el transcurso del programa; por lo que el código asociado a esa pulsación no se ejecutará.

Por este motivo, el programador no realiza la llamada al evento, ya que desconoce cuando este se va a producir. Sólo debe escribir el código que desea que se ejecute cuando se desencadene el evento.

Cada formulario dispone de su propia ventana de código, en donde podremos escribir el código para los eventos del formulario, controles y procedimientos normales.

La disposición del código en esta ventana es igual que la comentada en el tema Elementos del Lenguaje, apartado El módulo de código, con una pequeña variación respectiva al código del propio formulario y controles.

Una vez abierta esta ventana, la lista desplegable Objeto, nos muestra todos los objetos (formulario y controles) para los que podemos escribir código en sus eventos.

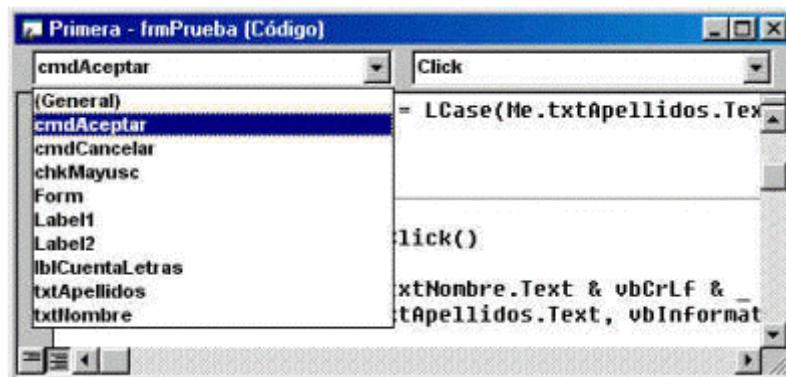


Figura 126. Ventana de código de formulario con lista de objetos disponibles.

Después de seleccionar un objeto, pasaremos a la lista desplegable Procedimiento, en donde encuentran todos los eventos disponibles para ese objeto. Despues de seleccionar el que necesitemos, se mostrará su declaración en la que podremos introducir las líneas de código necesarias.

Para saber cual es el evento correcto que debemos codificar, analizaremos que acción (sobre un objeto) tomará el usuario, a la que tengamos que responder. Identificada dicha acción, buscaremos el objeto en la lista de la ventana de código y seguidamente el evento en la otra lista de esta misma ventana, que tenga un nombre identificativo del problema a resolver.

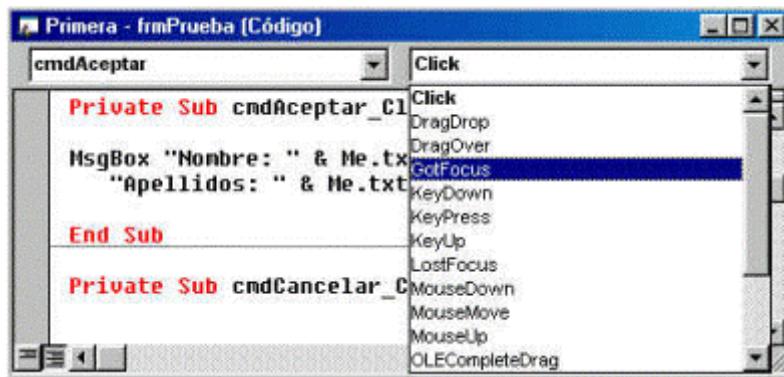


Figura 127. Ventana de código de formulario con lista de procedimientos disponibles para el objeto seleccionado.

Por ejemplo, supongamos que necesitamos enviar un mensaje al usuario cada vez que haga doble clic sobre el formulario. Para ello, seleccionaremos de la lista Objeto el elemento Form, y en la lista Procedimiento el evento DblClick. El código a escribir en este procedimiento de evento sería el mostrado en la Figura 128.

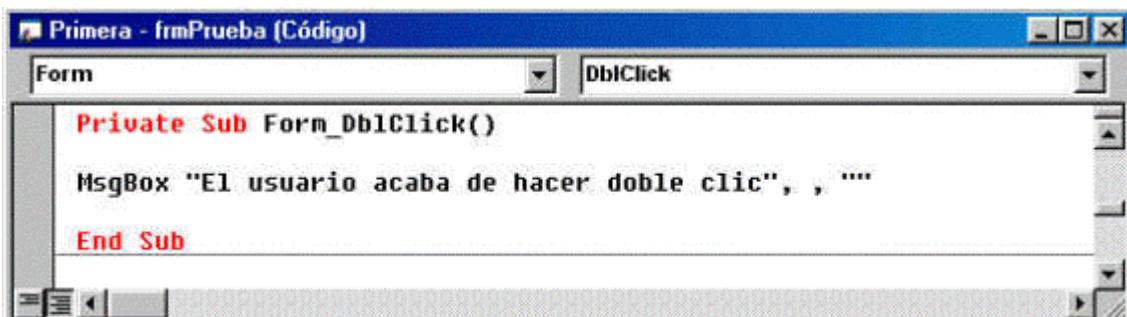


Figura 128. Código para el evento DblClick() del formulario.

Cada objeto dispone de un evento por defecto, que suele corresponder al evento que se dispara en más ocasiones para ese objeto. La forma de averiguar cual es consiste en hacer doble clic sobre el objeto en la ventana de diseño del formulario, lo que nos situará en la ventana de código y en el evento por defecto del objeto. Para un TextBox, este evento es Change(); para un CommandButton es Click(), etc.

En la ventana de código de este formulario, el lector puede ver el código fuente de los eventos codificados para los diferentes controles y el propio formulario, con los comentarios correspondientes, por lo que evitaremos repetirlos aquí.

En esta ventana, también es posible escribir procedimientos normales que no correspondan a eventos del formulario o controles, para ello, el programador debe escribir el procedimiento igual que si lo hiciera en una ventana de código estándar. Una vez escritos, estos procedimientos podrán localizarse, abriendo la lista Objeto, de la ventana, y situándose en la zona General. A continuación, abriremos la lista Procedimiento, que mostrará todas las rutinas de este tipo escritas en el código del formulario.

Generar el ejecutable del proyecto

Después de escribir el código para los objetos del formulario, si damos por concluido el desarrollo del proyecto, debemos crear el fichero ejecutable (EXE), ya que hasta el momento, todas las ejecuciones se hacían en modo de prueba desde el entorno de desarrollo.

El fichero EXE, será el que permita al usuario, ejecutar nuestro programa sin necesidad de tener instalado VB. Para ello, debemos seleccionar la opción Archivo+Generar <NombreProyecto.exe>..., del menú de VB, que nos mostrará un cuadro de diálogo para indicar la ruta en donde se generará el ejecutable.

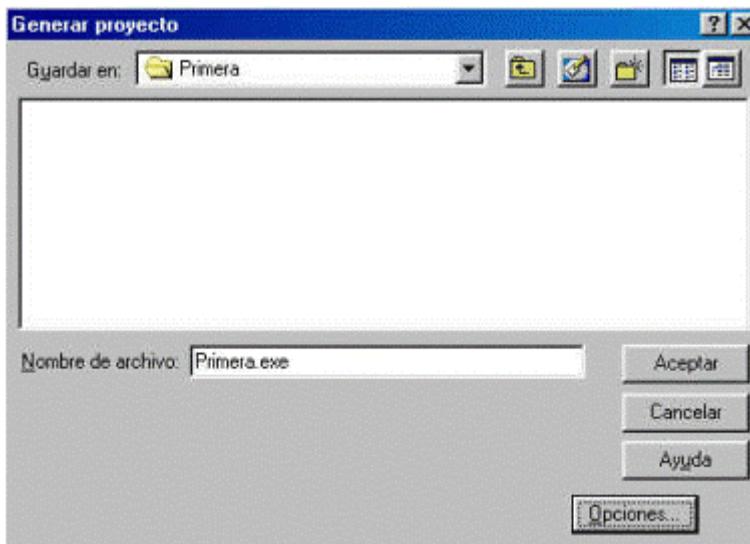


Figura 129. Cuadro de diálogo para generar el ejecutable.

Pulsando el botón Opciones de este cuadro, podemos establecer diversos parámetros que modificarán el modo de generación estándar del ejecutable, como el tipo de compilación, versión del programa, ícono, etc.

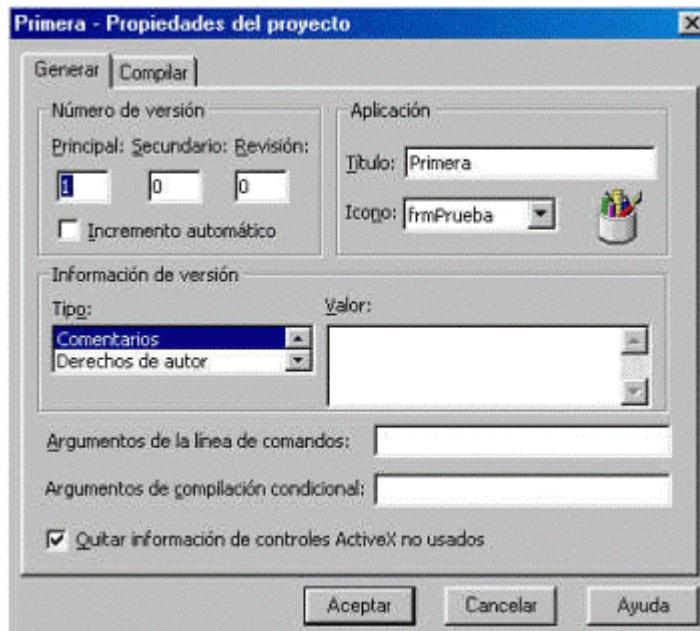


Figura 130. Opciones de generación del ejecutable.

Después de modificar los aspectos que sean necesarios, pulsaremos Aceptar en el cuadro de diálogo principal, lo que creará el fichero EXE que será el que debamos utilizar para ejecutar la aplicación.

Iniciar el formulario desde código

Cada vez que ejecutamos un proyecto típico de VB, con un formulario, en el que no hemos modificado ninguna propiedad del proyecto, dicho formulario es cargado y mostrado automáticamente al usuario, lo cual es una ventaja y una comodidad en situaciones que no requieran el uso de más de un formulario en el proyecto.

Esta situación cambia cuando por ejemplo, necesitamos modificar una propiedad del formulario antes de visualizarlo.

Se adjunta a este apartado, el proyecto [IniciarCodigo](#), que contiene las pruebas aquí mostradas.

En este caso, en lugar de usar el modo de carga automática empleado hasta ahora, debemos incluir un módulo de código al proyecto y escribir un procedimiento Main(). En este procedimiento escribiremos el nombre que hayamos dado al formulario en diseño, modificando las propiedades que sean necesarias y mostrándolo finalmente, como vemos en el Código fuente 194.

```
Public Sub Main()
    Form1.Caption = "pruebas con código"
    Form1.Show
End Sub
```

Código fuente 194

Una de las ventajas de esta técnica es que podemos arrancar el programa por distintos formularios. Supongamos por ejemplo, un proyecto que contiene dos formularios, llamados frmUno y frmDos. Si iniciamos desde un procedimiento Main(), podemos escribir el Código fuente 195 para que el usuario seleccione el formulario que quiere mostrar.

```
Public Sub Main()
    Dim lcIniciar As String

    lcIniciar = InputBox("Formulario a iniciar", "Atención")

    Select Case lcIniciar
        Case "uno"
            frmUno.Show

        Case "dos"
            frmDos.Show
    End Select
End Sub
```

Código fuente 195

Variables globales de formulario

El anterior ejemplo, sin embargo, no es el más óptimo para manipular formularios empleando código, ya que utiliza la técnica conocida como variables globales ocultas de formulario.

Cuando trabajamos con objetos, y un formulario lo es, el modo correcto de usarlos, es declarar una variable con el tipo de la clase correspondiente e instanciar un objeto de dicha clase, asignándolo a la variable.

Sin embargo, para facilitar las cosas, VB nos permite, sólo en el caso de formularios, utilizar directamente el nombre del formulario como si fuera una variable.

Mediante esta técnica, la primera vez que se haga referencia al formulario mediante su nombre en el código, VB crea internamente una variable global y le asigna una instancia del formulario. Toda esta operación es totalmente transparente de cara al programador, dando la impresión de estar trabajando directamente con ese formulario sin necesidad de emplear variables.

Pero ¿cómo deberíamos actuar si necesitáramos crear dos copias del mismo formulario y mostrar ambas al usuario?. El sistema empleado hasta ahora no tendría efecto. El Código fuente 196 no darían el resultado esperado, ya que estaríamos haciendo referencia a la misma instancia del objeto formulario.

```
Public Sub Main()  
  
Load Form1  
Form1.Show  
  
Load Form1  
Form1.Show  
End Sub
```

Código fuente 196

Variables declaradas de formulario

El problema que acabamos de plantear se soluciona con un poco de código adicional. Declaramos dos variables con el tipo perteneciente a la misma clase de formulario, las instanciamos, y trabajamos con ellas como lo que son: dos objetos distintos aunque pertenecientes a la misma clase.

```
Public Sub Main()  
  
Dim lfrmUno As Form1  
Dim lfrmDos As Form1  
  
Set lfrmUno = New Form1  
Load lfrmUno  
lfrmUno.Caption = "ventana uno"  
lfrmUno.Show  
  
Set lfrmDos = New Form1  
Load lfrmDos  
lfrmDos.Caption = "ventana DOS"  
lfrmDos.Show  
  
End Sub
```

Código fuente 197

Como conclusión a este apartado, podemos afirmar que el uso de variables para formulario declaradas explícitamente, y el procedimiento Main(), nos proporcionan un mayor control sobre la aplicación.

Asistente para crear procedimientos

La opción de menú Herramientas+Agregar procedimiento, nos muestra el cuadro de diálogo del mismo nombre, mediante el que podemos escribir el nombre para un nuevo procedimiento, seleccionando qué tipo y ámbito de procedimiento será creado.

Una vez aceptado este cuadro, se creará la declaración vacía para dicho procedimiento, que deberemos completar con el código correspondiente, parámetros, etc.

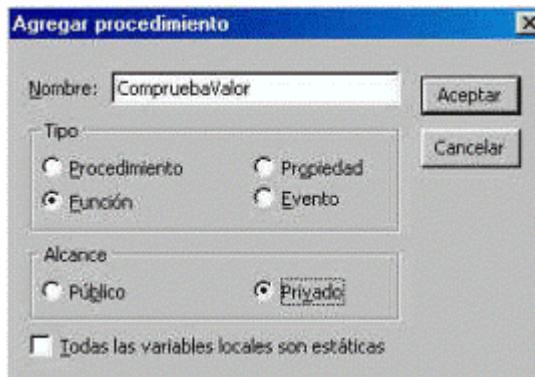


Figura 131. Cuadro de diálogo para la creación de procedimientos.

Image

El control Image muestra el contenido de un fichero gráfico. Como se comentó anteriormente, es más adecuado para visualizar un fondo en un formulario que la propiedad Picture del mismo.



Figura 132. Icono del control Image en el Cuadro de herramientas.

El proyecto [Imagenes](#) nos muestra como podemos utilizar este control en combinación con un formulario.

Después de crear un nuevo proyecto, dibujamos sobre el formulario un control Image y le asignamos un fichero gráfico, como muestra la Figura 133.

Seguidamente, situamos y redimensionamos el control Image de forma que ocupe todo el área de trabajo del formulario, aunque si la imagen tiene un tamaño más pequeño que el tamaño asignado al control, se seguirá mostrando el tamaño original de la imagen.

Para conseguir que el fichero gráfico asociado a un control Image ocupe todo el tamaño del control, nos situaremos en la ventana de propiedades de este control y asignaremos a la propiedad Stretch el valor True. Esta acción hará que la imagen se adapte al tamaño del control (Figura 134).

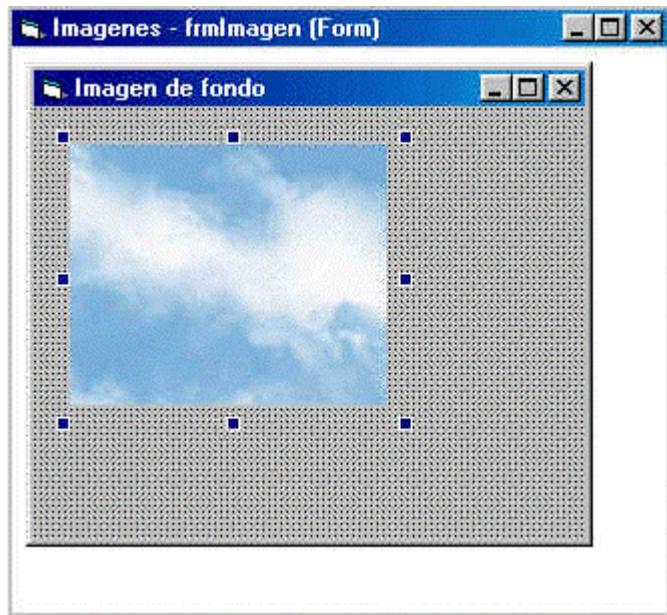


Figura 133. Formulario con control Image en su interior.

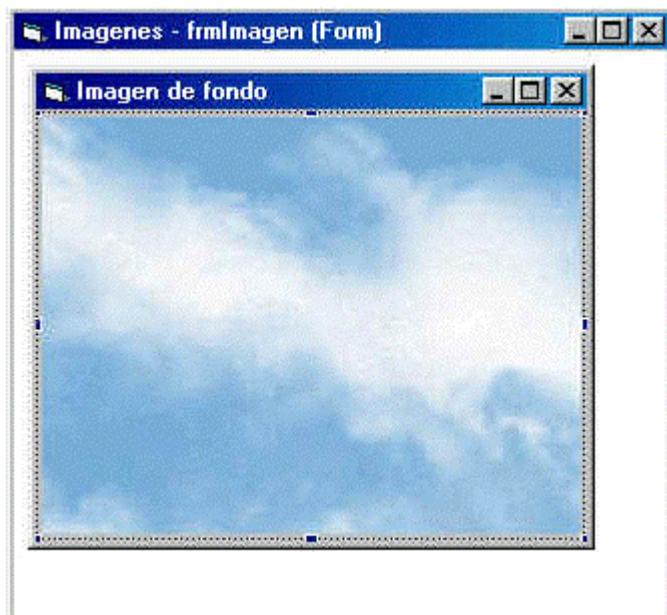


Figura 134. Control Image con su gráfico adaptado al área del formulario.

Ahora podemos ejecutar el programa mostrando el formulario con el tamaño correcto de la imagen de fondo. Pero nos encontramos con un nuevo inconveniente, como el formulario es redimensionable, cuando aumentamos su tamaño, el nuevo área del formulario no muestra la imagen de fondo. Esto es debido a que el control Image no se adapta automáticamente al tamaño del formulario.

La solución más rápida a este problema, pero menos profesional, consiste en hacer que el formulario tenga un tamaño fijo, pero ¿qué sucede cuando el formulario deba ser redimensionable?

La respuesta está en detectar cuando el formulario cambia de tamaño y cambiar al mismo tiempo el tamaño del control Image para adaptarlo al formulario. El evento del formulario que se produce

cuando este cambia de tamaño es `Resize()`, y es en el que debemos escribir el código correspondiente a esta acción, como se muestra en el Código fuente 198.

```
Private Sub Form_Resize()
    Me.imgNubes.Height = Me.Height
    Me.imgNubes.Width = Me.Width
End Sub
```

Código fuente 198

Las siguientes líneas del evento adaptan el tamaño del control al que acaba de tomar el formulario, por lo que ambos estarán en todo momento sincronizados.

Arrays de controles

Un array de controles contiene un grupo de controles del mismo tipo, que comparten un código común. En el ejemplo [ArrayControles](#), el lector dispone del proyecto utilizado para mostrar esta técnica.

Crearemos un nuevo proyecto y añadiremos al formulario un control `TextBox`. Una vez modificadas las propiedades necesarias, haremos clic sobre el control y lo copiaremos al Portapapeles mediante alguna de las formas habituales en Windows: `Control+C`, menú Edición+Copiar, etc.

A continuación pegaremos el control que acabamos de copiar, mostrándonos VB el aviso que muestra la Figura 135.

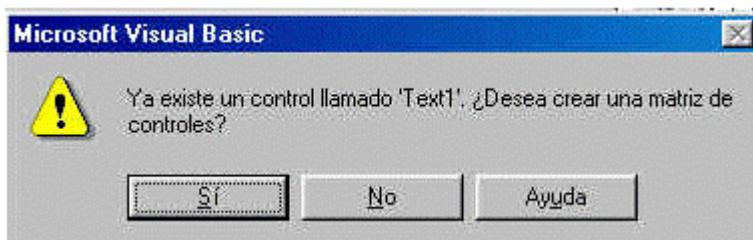


Figura 135. Aviso de VB antes de pegar un control en el formulario.

Responderemos **Si**, creándose en ese momento un array de controles, que contendrá el primer control y el que acabamos de pegar. Este último, se situará en la parte superior izquierda del formulario. Desde esta posición, lo moveremos hasta un lugar más adecuado del formulario.

Esta operación la repetiremos tantas veces como controles necesitemos agregar al formulario, teniendo en cuenta que el aviso para crear un array de controles no se volverá a producir, puesto que el array ya está creado.

Abriendo la lista de elementos en la ventana de propiedades, podemos identificar los controles del array por el nombre y la posición que ocupan, indicada entre paréntesis.

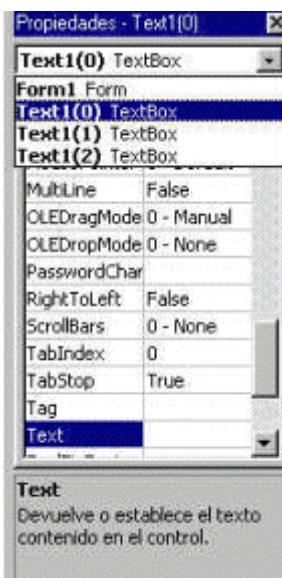


Figura 136. Ventana de propiedades mostrando la lista de elementos.

Para modificar una propiedad en particular de alguno de los controles del array, seleccionaremos dicho control en el formulario, y realizaremos la modificación en la ventana de propiedades. Como acabamos de hacer para el control que ocupa la posición 1 en el array, cambiando su propiedad Width a un valor menor que el resto.

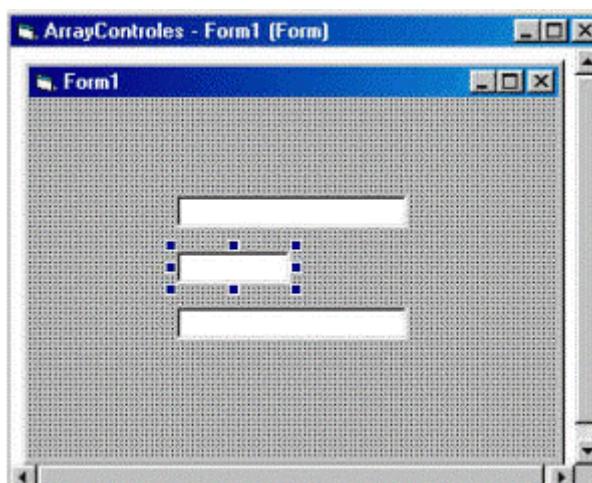


Figura 137. Control del array modificado.

En cuanto al código correspondiente a los eventos de un array de controles, se utiliza un mismo procedimiento de evento para todos los controles. Veamos, en el Código fuente 199 el evento correspondiente al doble clic, en el que se han escrito una serie de acciones para cuando se produzca.

```
Private Sub Text1_DblClick(Index As Integer)
' según sea el control del array
' se realizarán diferentes acciones
Select Case Index
Case 0
    ' pasar a mayúsculas el contenido del control

```

```

Me.Text1(0).Text = UCASE(Me.Text1(0).Text)
Case 1
    ' pasar a minúsculas el contenido del control
    Me.Text1(1).Text = LCASE(Me.Text1(1).Text)
Case 2
    ' borrar el contenido del control
    Me.Text1(2).Text = ""
End Select

```

Código fuente 199

Observe el lector, que cuando trabajamos con arrays de controles, los eventos a escribir, reciben el parámetro Index, que corresponde al número de control, dentro del array, que ha provocado el evento.

Creación de controles en tiempo de ejecución

Cuando necesitemos agregar a un formulario, varios controles del mismo tipo sin conocer el número de antemano, podemos crearlos dinámicamente durante la ejecución, como muestra el proyecto [ControlesEjec](#), que se acompaña como ejemplo.

La técnica empleada para ello, consiste en añadir al formulario un control, que nos servirá como base para crear copias a partir del mismo, durante la ejecución.

Dicho control, se configurará como un array de controles, y se ocultará, ya que será el que sirva como muestra para el resto, y debe permanecer inaccesible.

Pasemos al ejemplo. Como es habitual, crearemos en primer lugar, un nuevo proyecto Exe estándar. En el formulario de este proyecto vamos a crear copias de un control CheckBox en tiempo de ejecución, un número indeterminado de veces. Para ello, dibujamos un control de este tipo en el formulario, situándolo en la parte superior derecha y modificando algunas propiedades: a Visible le asignamos False, a Index el valor cero, Caption la dejamos vacía y en Nombre le asignamos chkMarca.. A continuación agregamos dos CommandButton, que se encargarán de crear y eliminar los controles en ejecución.

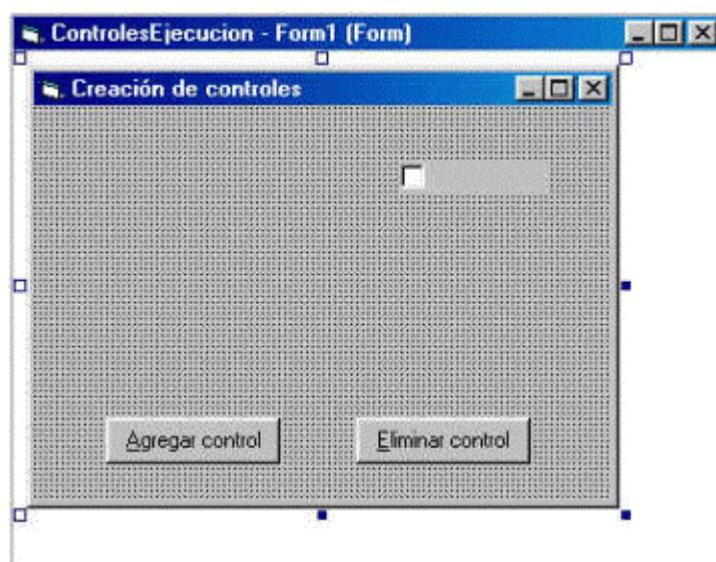


Figura 138. Formulario presentando el CheckBox base y el CommandButton para agregar nuevos controles.

Seguidamente pasaremos al módulo de código del formulario. En primer lugar declaramos dos variables a nivel de módulo, para controlar la coordenada superior en la que se visualizará cada nuevo control y el número de controles creados.

```
Option Explicit

Private mnNumControles As Integer
Private mnTop As Integer
```

Código fuente 200

La variable mnTop debe ser inicializada con un valor que asignamos en el evento Load() del formulario.

```
Private Sub Form_Load()
' asignar un valor de inicio
' para la coordenada superior
' del primer control a crear
mnTop = 405
End Sub
```

Código fuente 201

Seguidamente pasamos al código de los CommandButton que añaden y eliminan los controles CheckBox en tiempo de ejecución. Como puede comprobar el lector, la instrucción utilizada para agregar un nuevo control es Load, seguida del control a crear, en este caso, un nuevo elemento del array de controles.

Para eliminar uno de los controles creados, utilizaremos la instrucción Unload.

```
Private Sub cmdAgregar_Click()
' esta variable controla el número
' de controles creados dinámicamente
mnNumControles = mnNumControles + 1
' agregar nuevo control
Load chkMarca(mnNumControles)
' asignarle propiedades
chkMarca(mnNumControles).Top = mnTop
chkMarca(mnNumControles).Left = 360
chkMarca(mnNumControles).Caption = "Marca " & mnNumControles
' visualizarlo
chkMarca(mnNumControles).Visible = True
' actualizar la variable que
' controla la posición superior
' del control
mnTop = mnTop + 300
End Sub
*****
Private Sub cmdEliminar_Click()
' eliminar último control creado
Unload chkMarca(mnNumControles)
' actualizar la variable que contiene
' el número de controles
```

```

mnNumControles = mnNumControles - 1
' actualizar la variable que
' controla la posición superior
' del control
mnTop = chkMarca(mnNumControles).Top
End Sub

```

Código fuente 202

Propiedades accesibles en tiempo de diseño y en tiempo de ejecución

La mayoría de las propiedades para formularios y controles pueden modificarse tanto en la fase de diseño como en ejecución.

Existen, no obstante, propiedades que por sus especiales características, sólo pueden ser modificadas en diseño o en ejecución.

Se acompaña como ejemplo, el proyecto [DisenoEjec](#), en el que se realizan algunas pruebas de cambio de propiedades en diversas situaciones, que explicamos a continuación.

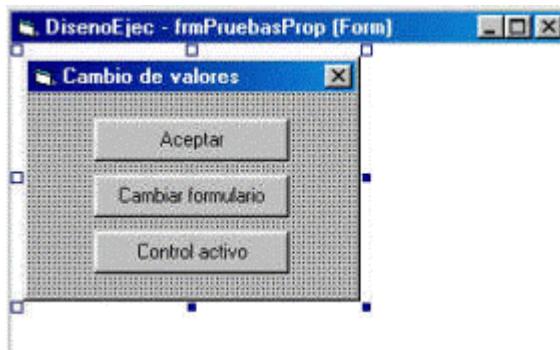


Figura 139. Formulario para las pruebas de modificación de propiedades.

- Propiedad modificable en diseño y en ejecución. La propiedad Caption representa a este tipo de propiedades que pueden ser modificadas en todo momento. En este caso, al cargar el formulario, se cambia el valor de esta propiedad en el botón cmdAceptar.

```

Private Sub Form_Load()
    Me.cmdAceptar.Caption = "cambiado"
End Sub

```

Código fuente 203

- Propiedad modificable sólo en tiempo de diseño. Un representante de este tipo de propiedades, es ControlBox, que está disponible en los objetos Form. Nos permite establecer o eliminar el menú de control de un formulario, pero sólo en diseño. Si intentamos ejecutar el Código fuente 204, asociado al botón cmdCambiarForm, del formulario de ejemplo, se producirá un error.

```
Private Sub cmdCambiarForm_Click()
    Me.ControlBox = False
End Sub
```

Código fuente 204

- Propiedad modificable sólo en tiempo de ejecución. La propiedad ActiveControl, de los objetos Form, sólo puede ser manipulada en código. El lector puede comprobarlo observando la ventana de propiedades, ya que allí no aparece.

Esta propiedad devuelve el control del formulario que tiene el foco en ese momento, como muestra el Código fuente 205 del botón cmdControlAct.

```
Private Sub cmdControlAct_Click()
    Dim lctlControl As Control
    Set lctlControl = Me.ActiveControl
    MsgBox "El control activo es " & lctlControl.Name, , ""
End Sub
```

Código fuente 205

8

El depurador

¿Qué es un depurador?

Un depurador, es una utilidad o conjunto de utilidades que se suministran junto a una herramienta de programación, para permitir al programador, realizar un seguimiento exhaustivo, de todos los aspectos del código existentes en su programa, de forma que pueda detectar y corregir los posibles errores producidos durante la fase de desarrollo de la aplicación.

El depurador es un elemento tan importante como pueda serlo el compilador, y de su versatilidad a la hora de tratar el código, mediante puntos de interrupción, evaluación de expresiones y otros aspectos propios de este tipo de componente, depende que podamos desarrollar más eficiente y rápidamente nuestro trabajo.

Visual Basic incorpora un excelente conjunto de herramientas de depuración, que nos permitirán abordar la ejecución del código durante la fase de desarrollo de muy distintas formas, ayudándonos en la eliminación de todos los errores de programación existentes.

Para un mayor detalle sobre el lugar en el que se encuentran las diferentes órdenes y opciones del depurador, recomendamos al lector consultar el tema El Entorno de Desarrollo (IDE). No obstante, se indicarán el botón de la barra de herramientas y combinación de teclado para aquellos elementos accesibles mediante el menú de VB.

La aplicación [Depurar](#), se incluye como ejemplo para que el lector pueda realizar las pruebas necesarias con este elemento de VB.

Acciones básicas

Estas acciones proporcionan el funcionamiento mínimo para el depurador. Veamos una breve descripción de las mismas, y a continuación su forma de uso.

- **Iniciar / Continuar.** Con esta acción, comenzaremos la ejecución del programa desde el entorno de desarrollo de VB, o reanudaremos la ejecución, si se encontraba interrumpida.



- **Interrumpir.** Para detener momentáneamente la ejecución de la aplicación y entrar en el modo de interrupción, utilizaremos este comando.



- **Terminar.** Esta acción, finaliza la ejecución del programa cuando se realiza desde el entorno de VB.



- **Reiniciar.** Si la ejecución del programa está interrumpida, el uso de este comando hace que comience de nuevo la ejecución, de igual forma que si hubiéramos pulsado el botón Iniciar. (Mayús+F5).

Para probar estas órdenes, insertemos un nuevo formulario en un proyecto, asignándole un tamaño que nos permita acceder a las opciones del entorno de VB e incluyendo un control CheckBox y un OptionButton. Pulsemos acto seguido, el botón Iniciar, de forma que el formulario se ponga en ejecución. Si a continuación pulsamos Interrumpir, entraremos en el modo de interrupción, pasando el foco a la ventana Inmediato del entorno de VB, que veremos posteriormente.

Pulsando nuevamente Iniciar, se reanudará la ejecución del formulario. Ahora haremos clic en los dos controles del formulario y seleccionaremos la orden Reiniciar del depurador. Lo que pasará a continuación es que, al ejecutarse el formulario de nuevo desde el comienzo los controles aparecerán vacíos, en su estado inicial.

Modo de interrupción

Se puede definir como una parada momentánea cuando se ejecuta un programa desde el entorno de VB. Las formas de invocar este modo son las siguientes:

- Al llegar la ejecución del programa a un punto de interrupción.
- Introduciendo en el código una instrucción Stop.
- Al pulsar la combinación de teclado Ctrl+Enter.
- Al producirse un error de ejecución.
- Cuando una inspección se define para que entre en este modo, al devolver un valor verdadero cuando es evaluada.

Puntos de interrupción

Un punto de interrupción, es una línea de código con una marca especial, que produce la detención del programa, al llegar la ejecución a ella. Es posible establecer puntos de interrupción en cualquier línea ejecutable de la aplicación. No están permitidos en líneas de declaración o comentarios. El comando Alternar puntos de interrupción de VB, nos permite establecer/quitar un punto de interrupción en el código de la aplicación.



También es posible insertar un punto de interrupción, haciendo clic en el margen, junto a la línea que deberá provocar la interrupción. La Figura 140 muestra la ventana de código con un punto de interrupción establecido.

```

Public Sub Mensaje()
    Dim lcTexto As String
    lcTexto = "Variable con el texto del mensaje"
    MsgBox lcTexto, , "Depurando"
    Asigna
End Sub

```

Figura 140. Ventana de código con punto de interrupción.

Una vez que hayamos establecido uno o más puntos de interrupción en el código del programa, lo iniciaremos. Si durante la ejecución, el flujo de la aplicación pasa por alguna de las líneas de código marcadas como de interrupción, el programa se detendrá provisionalmente, pasando el foco a la ventana de código de VB que contiene la línea definida como punto de interrupción, este tipo de ejecución se conoce como modo de interrupción.

```

Option Explicit

Private Sub cmdAceptar_Click()
    Dim lcCadena As String
    lcCadena = "nombre"
    MsgBox "Se ha pulsado el botón Aceptar"
End Sub

```

Figura 141. Ejecución detenida en un punto de interrupción.

Aquí podemos comprobar el estado de las expresiones, variables, ejecutar paso a paso, etc., o volver a la ejecución normal pulsando Continuar.

Para eliminar un punto de interrupción utilizaremos el mismo botón de la barra de herramientas o combinación de teclado usada para establecerlo. Si queremos eliminar todos los puntos de interrupción del programa, usaremos el comando Borrar todos los puntos de interrupción, de VB.



(Ctrl+Mayús+F9)

Ejecución paso a paso

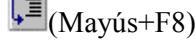
Una vez detenido el programa en un punto de interrupción, la ejecución paso a paso nos permite controlar la ejecución del código situado en el depurador. Las diferentes formas de ejecución paso a paso son las siguientes:

- **Paso a paso por instrucciones.** Ejecuta el código del programa línea a línea, si llega a la llamada a un procedimiento, el depurador se introduce en el código de ese procedimiento, procediendo a ejecutarlo también línea a línea.



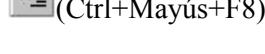
(F8)

- **Paso a paso por procedimientos.** Es similar a paso por instrucciones, que ejecuta cada instrucción del programa. Pero en este caso, si llega a la llamada a un procedimiento, el depurador ejecuta el procedimiento por completo sin mostrarlo al programador, y se sitúa en la siguiente línea después de la llamada al procedimiento.



(Mayús+F8)

- **Paso a paso para salir.** Ejecuta las líneas del procedimiento actual hasta situarse en la primera línea posterior a la llamada del procedimiento en donde nos encontrábamos.



(Ctrl+Mayús+F8)

- **Ejecutar hasta el cursor.** Ejecuta las líneas del programa hasta el punto donde está situado el cursor. Si hubiera un punto de ruptura antes de llegar, la ejecución se pararía en ese punto. (Ctrl+F8).

Adicionalmente, en este modo de ejecución por líneas, disponemos de las siguientes órdenes:

- **Establecer instrucción siguiente.** Esta opción hace que la ejecución se desplace al punto seleccionado por el usuario, sin ejecutar las líneas intermedias que pudiera haber entre la línea activa en ese momento y la seleccionada como instrucción siguiente. Es posible establecer, una línea anterior o posterior a la que se ha ejecutado. Esta característica no está disponible entre procedimientos diferentes.



(Ctrl+F9)

- **Mostrar instrucción siguiente.** Sitúa el cursor en la siguiente línea a ejecutar. Esta instrucción es muy útil, si estamos revisando el código y no recordamos en qué lugar se encuentra la siguiente instrucción del programa a ejecutar.



Inspecciones

Una inspección, es una expresión definida por el programador, que será evaluada dentro de un contexto, ámbito o zona de alcance durante la ejecución del programa. En función del resultado de dicha evaluación, se realizará una acción determinada por el programador.

Supongamos que disponemos de un formulario con un CommandButton cmdAceptar. Al pulsar dicho botón, se ejecutará el Código fuente 206.

```
Private Sub cmdAceptar_Click()

Dim lcCadena As String
Dim lnNumero As Integer

lcCadena = "agregar cadena"

lnNumero = 2350
lnNumero = lnNumero * 2

lcCadena = lcCadena & " al nombre"

End Sub
```

Código fuente 206

Lo que necesitamos en este código, es controlar cuando el valor de la variable lnNumero es superior a 3000. Para ello, vamos a emplear una expresión de inspección.

Emplearemos la opción Depuración+Agregar inspección, del menú de VB para añadir nuevas inspecciones, lo que mostrará la ventana Agregar inspección, Figura 142, en donde introduciremos la expresión que queremos controlar.



Figura 142. Ventana de edición de inspecciones.

Los elementos de esta ventana son los siguientes:

- Expresión. Permite introducir la variable o expresión que será evaluada.
- Procedimiento. En este control podemos seleccionar el procedimiento en el que se realizará la inspección, o todos los procedimientos del módulo.
- Módulo. Aquí podemos indicar un módulo en el que se realizará la inspección, o todos los módulos del proyecto.

Aunque es posible indicar que la inspección se realice en todos los procedimientos y todos los módulos del proyecto, lo más recomendable es acotar dicha inspección a un procedimiento y módulo si es posible. La razón de esto, reside en que cuando la aplicación entra en un procedimiento en el que hay definida una inspección, dicha inspección es evaluada tras la ejecución de cada línea de código, lo cual nos permite deducir que cuantos más procedimientos deba comprobar una inspección, más lento se ejecutará el programa.

- Expresión de inspección. Si estamos situados en un procedimiento con una inspección definida, se muestra su valor en la ventana Inspección.
- Modo de interrupción cuando el valor sea Verdadero. Cuando durante la ejecución de la aplicación, la evaluación de la expresión a inspeccionar de un resultado verdadero, se interrumpirá la ejecución de igual forma que si se hubiera definido un punto de interrupción en el código.
- Modo de interrupción cuando el valor cambie. Es igual que el anterior, excepto en que entramos en el modo de interrupción cuando cambia el valor de la variable a inspeccionar.

Cuando sea necesario modificar alguno de los valores de una inspección ya establecida, emplearemos la opción Editar inspección (Ctrl+W), del mismo menú Depuración. Esta orden mostrará, la misma ventana usada para agregar inspecciones, pero con la inspección seleccionada, lista para editar.

La ventana de Inspección

Esta ventana es la empleada cuando entramos en el modo de depuración, para comprobar las diferentes inspecciones que hemos definido en la aplicación.



Expresión	Valor	Tipo	Contexto
Len(lcCadena) > 20	Falso	Boolean	frmBasico.cmdAceptar_Click
InNúmero	2350	Integer	frmBasico.cmdAceptar_Click

Figura 143. Ventana Inspección visualizando expresiones durante la depuración de un programa.

Las inspecciones son organizadas en forma de filas y columnas. Cada fila o expresión, se compone de la propia expresión a evaluar, el valor actual, resultado de haberla evaluado, el tipo de valor resultante de la expresión y el contexto o lugar del código en que está definida.

Según vayamos ejecutando la aplicación paso a paso, esta ventana reflejará los cambios que se produzcan en las inspecciones, de forma que podamos localizar comportamientos extraños en el código.

Inspección rápida

Esta opción, nos permite ver en modo de interrupción, el resultado de una expresión para la cual no se ha definido una inspección. Para ello, sólo debemos marcar la expresión a evaluar y seleccionar este comando en el menú de VB.

 (Mayús+F9)

La Figura 144, nos muestra la ventana de código en modo interrupción, con una expresión seleccionada, sobre la que se efectúa una inspección rápida.

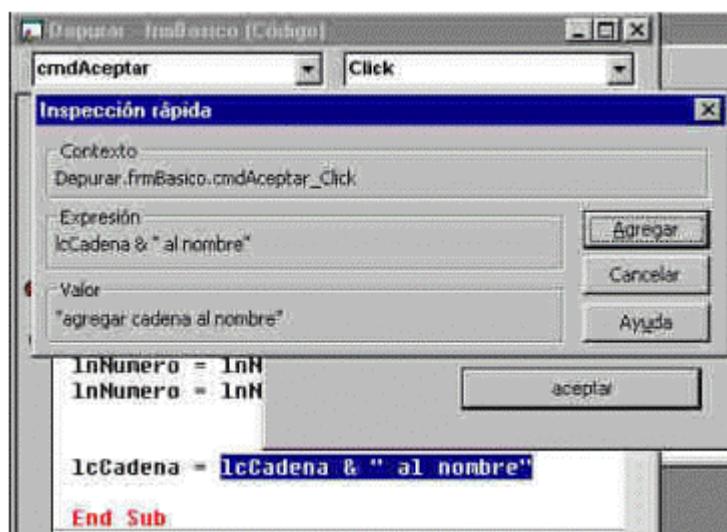


Figura 144. Inspección rápida sobre una expresión.

Como podrá apreciar el lector, la información aquí es muy similar a la ofrecida por la ventana Inspección. Disponemos del contexto en el que se ejecuta la expresión, la expresión seleccionada y el resultado de la misma, existiendo la posibilidad de agregar la expresión a la ventana Inspección.

Si intentamos ver los resultados de las inspecciones cuando no se está ejecutando la aplicación, tanto la ventana de Inspección como la de Inspección rápida no proporcionarán valores, indicándonos que la inspección está fuera de contexto.

La ventana Inmediato

Esta ventana se abre automáticamente, desde el momento en que ejecutamos una aplicación en el entorno de desarrollo de VB, aunque si la aplicación ocupa toda la pantalla, no la veremos a no ser que entremos en el modo de interrupción o cambiemos la tarea activa al entorno de VB.

Si una vez en VB, esta ventana no está disponible, podemos abrirla mediante la combinación de teclado Ctrl+G, o su botón de la barra de herramientas: 

Como su propio nombre indica, podemos ver de forma inmediata, el resultado de expresiones del código que está en ese momento en ejecución, y hacer asignaciones de valores a variables y objetos, siempre que el ámbito de los mismos esté accesible.

La Figura 145, muestra esta ventana durante la ejecución de un programa, en concreto el procedimiento de pulsación de un CommandButton. Si queremos ver el valor de una variable o una expresión, debemos escribir la instrucción Print, o el símbolo ?, y a continuación la variable o expresión a comprobar. Después pulsaremos Enter, obteniendo el resultado en la línea siguiente de la ventana. Para ejecutar una sentencia desde esta ventana, la escribiremos directamente y pulsaremos Enter. El código a evaluar en esta ventana, también podemos copiarlo y pegarlo desde la ventana de código en ejecución.

El formulario frmBasico, al que pertenece el código que estamos depurando, y todos sus controles también pueden ser analizados en esta ventana. Como puede comprobar el lector en la figura, estamos visualizando el Caption del formulario, para después cambiar dicha propiedad y el ancho.

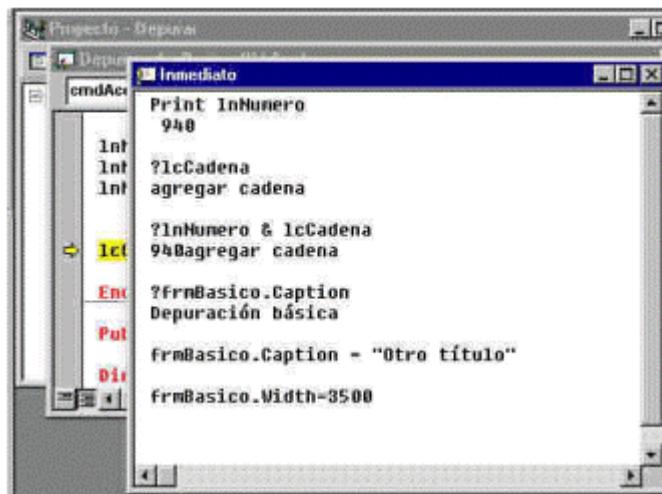


Figura 145. Ventana Inmediato durante la ejecución del programa.

La ventana Locales

Esta ventana visualiza las variables locales declaradas en el procedimiento actual.

La Figura 146, muestra esta ventana, cuyos elementos describimos a continuación:

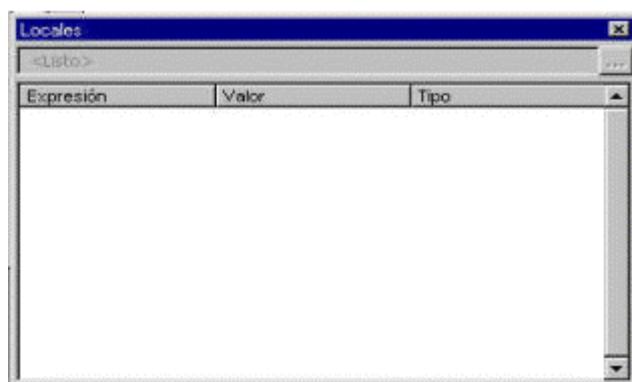


Figura 146. Ventana de información de variables locales.

- Contexto. Situado debajo del título de esta ventana, este elemento nos informa del punto del código que se está ejecutando.
- Pila de llamadas. Muestra la ventana de pila de llamadas, que veremos después de este apartado.
- Expresión. Esta columna muestra la lista de variables definidas en el procedimiento. Si estamos situados en un módulo de clase, la palabra Me hará referencia al objeto en ejecución de dicha clase. Consulte el lector, el apartado La palabra clave Me, incluido en el tema Programación orientada a objeto. En el caso de estar en un módulo de código estándar, la primera variable será el nombre del módulo.

Tanto la palabra Me, como el nombre de un módulo estándar, y las variables que contengan objetos, irán acompañadas de los iconos y , que al ser pulsados, desplegarán y contraerán respectivamente, la lista de propiedades y variables a nivel de módulo, que contiene dicho elemento.

Esta ventana no permite el acceso a variables globales

- Valor. Muestra el valor de la variable. Es posible modificar el valor visualizado, haciendo clic sobre él, y tecleando el nuevo valor. Una vez modificado, pulsaremos Enter, Flecha Arriba o Flecha Abajo, para aceptar el nuevo valor. Si dicho valor no fuera correcto, el valor seguiría en modo de edición. Para cancelar la edición pulsaremos Esc.
- Tipo. Muestra el tipo de la variable.

La Figura 147, muestra esta ventana durante la ejecución de un programa, una vez que se ha entrado en el modo de interrupción, para depurar el código. Los valores reflejados se irán actualizando según vayamos ejecutando el código línea a línea.

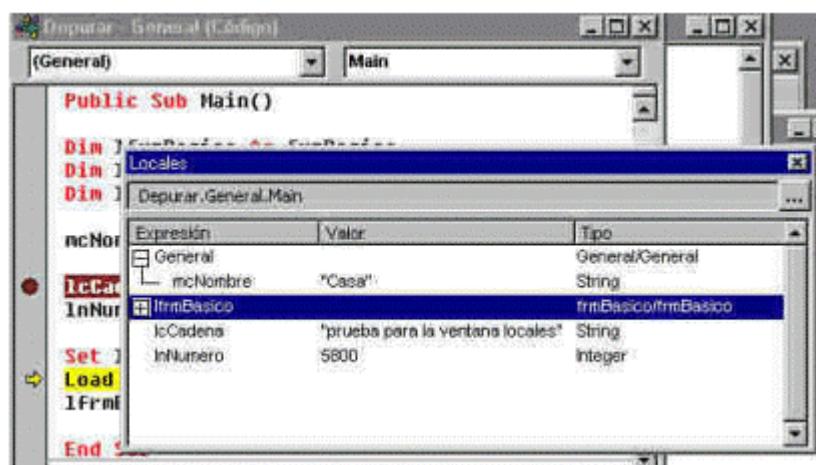


Figura 147. Ventana Locales, mostrando el valor de las variables del procedimiento Main().

Pila de llamadas

Durante la ejecución de un programa, el procedimiento activo es situado en una lista que controla las llamadas entre procedimientos. Si dicho procedimiento, llama a su vez a otro, el nuevo es agregado a la lista pasando a ocupar el primer lugar y así sucesivamente. La ventana Pila de llamadas, muestra

esta lista de procedimientos apilados, de forma que el programador pueda comprobar la relación de llamadas efectuadas entre sí por los procedimientos en ejecución.

El Código fuente 207, muestra la entrada a una aplicación mediante un procedimiento Main(), este hace una llamada a otro procedimiento Segundo(), y este a su vez, a otro llamado Tercero().

```
Public Sub Main()
    MsgBox "Este es el primer procedimiento de la pila"
    Segundo
End Sub
' -----
Public Sub Segundo()
    MsgBox "Este es el segundo procedimiento en la pila"
    Tercero
End Sub
' -----
Public Sub Tercero()
    MsgBox "Este es el tercer procedimiento en la pila"
End Sub
```

Código fuente 207

Si realizamos una ejecución paso a paso desde el modo de interrupción, y al llegar al procedimiento Tercero(), abrimos la ventana de la pila de llamadas (Ctrl+L), se mostrará igual que la Figura 148.

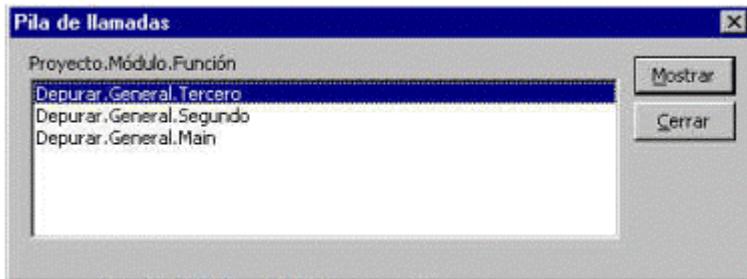


Figura 148. Ventana Pila de llamadas durante la ejecución de un programa.

Como podemos observar, cada llamada de la ventana, muestra en este orden: el proyecto en ejecución, el módulo y la función, procedimiento, método, etc., que se está ejecutando. La primera línea indica el procedimiento que actualmente se está ejecutando. Si seleccionamos cualquier otra línea, y pulsamos el botón Mostrar, se mostrará la ventana de código de ese procedimiento, en el punto de llamada a uno de los procedimientos de la pila, indicado por la flecha verde.

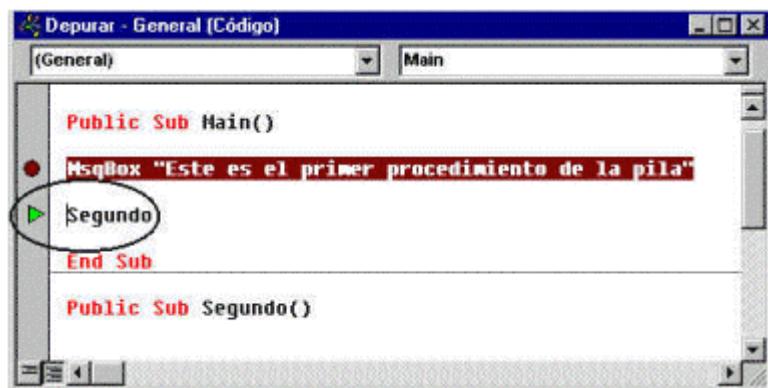


Figura 149. Resultado de la pulsación del botón Mostrar en la ventana Pila de llamadas.

Estilos de formularios

Aplicaciones SDI y MDI

Una aplicación con interfaz SDI (Interfaz de documento sencillo), dispone de un sólo formulario, que se ocupa de todas las labores de interacción con el usuario. Un ejemplo de este tipo de programas es el Bloc de notas de Windows, en el que sólo se puede abrir un fichero para ser editado.

Una aplicación de interfaz MDI (Interfaz de documento múltiple), dispone de un formulario principal o MDI, que actuará como contenedor del resto de formularios (documentos) abiertos en el programa; a estos formularios se les denomina formularios secundarios MDI o hijos MDI. Como ejemplo de este tipo de aplicación tenemos Word.

Un formulario MDI, tiene un comportamiento ligeramente distinto de un formulario normal, ya que sólo admite cierto tipo de controles. No se pueden utilizar controles como CommandButton o TextBox, ya que en este tipo de aplicaciones, se presupone que existirá al menos un formulario hijo MDI, que será el que pueda tener este tipo de controles.

Tampoco es posible eliminar su menú de control, no existiendo propiedad ControlBox para ello.

El medio más habitual para que un formulario MDI cargue formularios secundarios será a través de un menú. Los menús serán tratados en un tema posterior.

Por último, una aplicación de documento múltiple sólo puede incorporar un formulario de tipo MDI y tantos formularios MDI secundarios como sea preciso.

Por su parte, los formularios MDI secundarios tienen también ciertas peculiaridades, se visualizan dentro del área de trabajo del formulario MDI principal, y no pueden ser desplazados fuera de dicho formulario, permaneciendo en todo momento en su interior.

Al maximizar un formulario secundario MDI, su barra de título desaparecerá, y su valor se mostrará junto al título del formulario MDI.

Un formulario secundario MDI no puede visualizarse como modal. El tipo de ejecución modal y no modal, puede consultarla el lector más adelante en este mismo tema.

Si al comenzar a ejecutar un proyecto de tipo MDI, se hace referencia en primer lugar a un formulario secundario MDI, sin cargar en primer lugar el formulario MDI principal, automáticamente se cargará el formulario MDI principal y a continuación el secundario.

En el ejemplo [PruebaMDI](#), el lector dispone de un proyecto con interfaz MDI, cuyos pasos de creación detallaremos seguidamente.

Como es habitual, debemos crear un nuevo proyecto EXE estándar y modificar las propiedades que consideremos oportuno. En este caso le daremos al proyecto el nombre PruebaMDI e indicaremos que el objeto inicial será un Sub Main, mientras que al formulario que se incluye por defecto le daremos el nombre frmHijo.

El siguiente paso, consistirá en seleccionar la opción de menú de VB Proyecto+Agregar formulario MDI, que añadirá un formulario de este tipo al programa, que llamaremos mdiPrincipal. Este tipo de formulario lo podemos reconocer porque tiene un tono más oscuro en el color de fondo que un formulario normal, como vemos en la Figura 150.

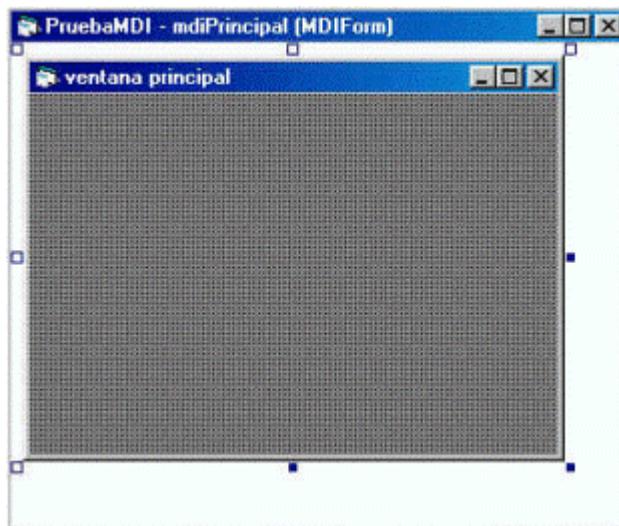


Figura 150. Diseñador de formulario MDI.

Continuaremos agregando un módulo de código con la opción Proyecto+Agregar módulo, en el que escribiremos el procedimiento Main() que se ocupará de cargar y mostrar los formularios de la aplicación.

```
Public Sub Main()
Dim lmdiPrincipal As mdiPrincipal
Dim lfrmHijo As frmHijo
```

```
Set lmdiPrincipal = New mdiPrincipal
Load lmdiPrincipal
lmdiPrincipal.Show

Set lfrmHijo = New frmHijo
Load lfrmHijo
lfrmHijo.Show

End Sub
```

Código fuente 208

Si ejecutamos en este momento el programa, se cargarán y mostrarán los formularios, pero el formulario frmHijo no se comportará como secundario MDI, ya que se podrá desplazar a cualquier parte la pantalla, sin estar contenido dentro del formulario mdiPrincipal.

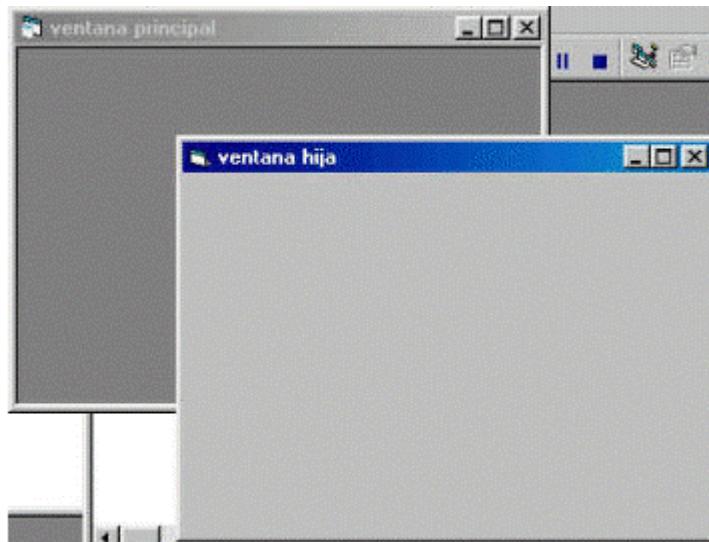


Figura 151. El formulario frmHijo no está contenido dentro de mdiPrincipal.

Para conseguir que frmHijo se comporte como secundario MDI, debemos asignar a su propiedad MDIChild el valor True. A partir de ahora, cuando ejecutemos el proyecto, el comportamiento de frmHijo será el esperado.

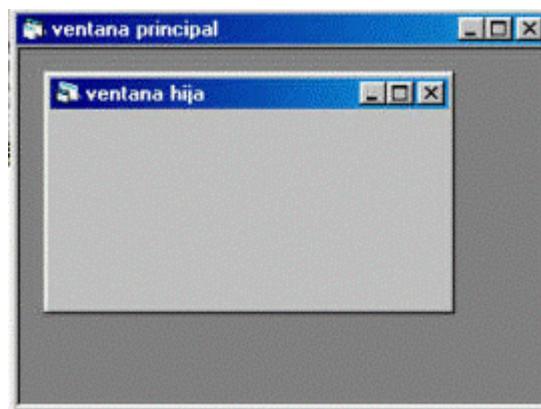


Figura 152. El formulario frmHijo sí se comporta ahora como secundario MDI.

Formularios modales y no modales

La forma en que dentro de una sesión de trabajo con Windows podamos cambiar el foco entre las diferentes ventanas existentes, viene determinada por el estilo de dichas ventanas: modal o no modal.

Modal

Un formulario modal precisa que el usuario realice alguna acción sobre dicho formulario, antes de poder cambiar el foco a otro formulario de la misma o de otra aplicación.

Existen los siguientes tipos de formulario modal:

Modal a la aplicación

Al ser mostrado un formulario de este tipo, no se permite cambiar el foco a otro formulario dentro de la misma aplicación, hasta que el usuario no haya completado la información solicitada o respondido a la pregunta planteada.

Como ejemplo de este tipo de formularios, se acompaña el proyecto [ModalApp](#). Este proyecto parte del ejemplo mostrado en el apartado anterior, conteniendo un formulario MDI principal y uno secundario MDI, al que hemos añadido un nuevo formulario, asignándole el nombre frmDatos, proporcionándole los siguientes controles, para que el usuario introduzca el nombre y apellidos, como podemos ver en la Figura 153.

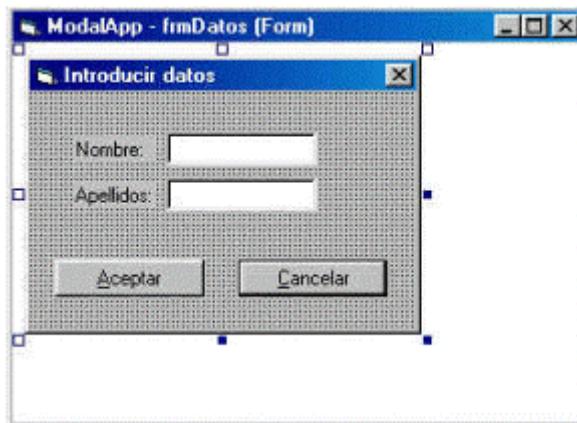


Figura 153. Formulario frmDatos que funcionará como formulario modal a la aplicación.

Aunque no es necesario, en la propiedad BorderStyle del formulario hemos establecido el valor Fixed Dialog, de forma que su apariencia sea igual a los cuadros de diálogo estándar de Windows.

Para mostrar este formulario en el proyecto, agregamos un botón en el formulario frmHijo, con el nombre cmdTomarDatos, y escribimos el Código fuente 209, que carga y muestra el formulario modal.

```
Private Sub cmdTomarDatos_Click()
    Dim lfrmDatos As frmDatos
```

```
Set lfrmDatos = New frmDatos
Load lfrmDatos
lfrmDatos.Show vbModal

End Sub
```

Código fuente 209

Como habrá observado el lector, al visualizar este formulario le pasamos como parámetro la constante `vbModal`, que como su nombre indica, lo visualizará con estilo modal a la aplicación, es decir, hasta que no cerremos el formulario, no podremos pasar el foco a otro de la misma aplicación.

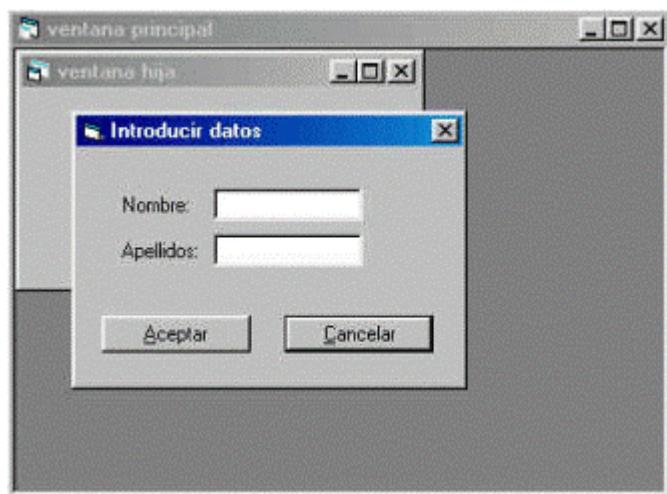


Figura 154. Aplicación mostrando el formulario modal.

Visual Basic nos proporciona, al seleccionar el menú Proyecto+Agregar formulario, varios tipos de formularios modales preconfigurados (Figura 155), para ahorrar tiempo de desarrollo al el programador.

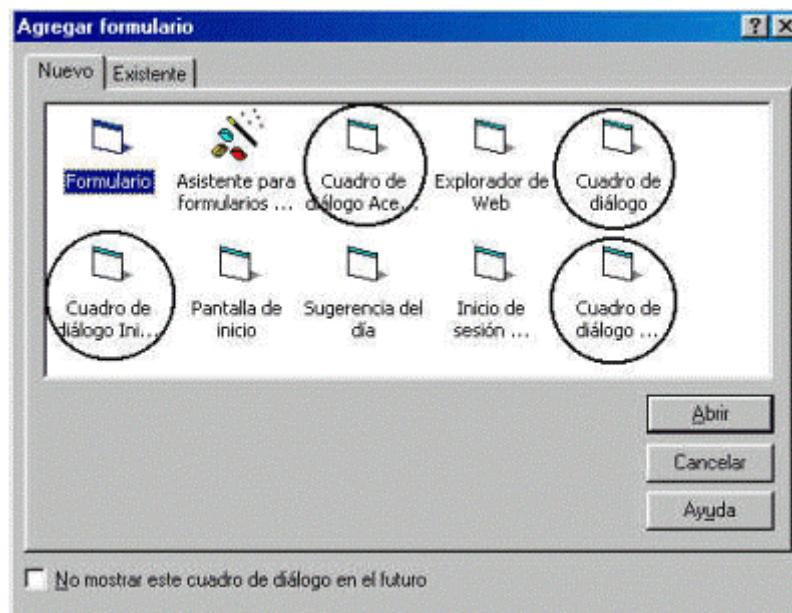


Figura 155. Formularios modales preconfigurados por VB.

En el caso de que sólo sea necesario mostrar al usuario un mensaje, disponemos de la función MsgBox(), vista en el tema Elementos del lenguaje, que también proporciona un formulario modal, sin necesidad de diseñarlo manualmente.

Modal al sistema

Al ser mostrado un formulario de este tipo, el sistema queda detenido hasta que el usuario no responda a dicho formulario. No siendo posible cambiar el foco a ningún formulario, ni dentro de la aplicación, ni de cualquier otra aplicación que esté en ejecución.

Los formularios creados manualmente por el programador en VB, no pueden ser modales al sistema, sólo es posible mostrarlos modales a la aplicación, como acabamos de ver en el punto anterior.

Sin embargo, podemos recurrir a la función MsgBox(), pasando en el segundo parámetro (el correspondiente a los botones) la constante vbSystemModal. De esta forma la ventana del mensaje tendrá un comportamiento modal al sistema.

```
MsgBox("Dato no válido", vbSystemModal + vbCritical, "Error")
```

Código fuente 210

Es posible también, que al utilizar MsgBox() y la mencionada constante, el comportamiento de la ventana de mensaje no sea del todo adecuado, permaneciendo dicha ventana, visible en todo momento, pero permitiendo al usuario cambiar y operar con las ventanas del resto de aplicaciones en funcionamiento, posiblemente consecuencia de un error interno de esta función o de Visual Basic.

No Modal

Un formulario no modal, permite cambiar el foco libremente a cualquier otro formulario que haya en ese momento abierto en el sistema, ya sea de nuestra o de otra aplicación.

El ejemplo más claro lo tenemos en Word. Cada documento abierto estará contenido en una ventana distinta, entre las que podremos cambiar el foco, o bien pasar a otra aplicación que haya en ese momento en ejecución.

Otros controles básicos

OptionButton

Muestra una opción que se puede activar o desactivar. Utilizando varios controles de este tipo en un formulario, proporcionan un conjunto de opciones auto excluyentes, es decir, sólo puede estar una de ellas seleccionada.



Figura 156. Icono del control OptionButton en el Cuadro de herramientas.

Un control OptionButton es muy similar a un CheckBox, en cuanto a que ambos proporcionan un estado de seleccionado/no seleccionado. La diferencia entre ellos radica en que utilizados en grupo, dentro del conjunto de OptionButton, sólo podemos tener uno seleccionado, mientras que en el conjunto de CheckBox, es posible seleccionar más de uno al mismo tiempo.

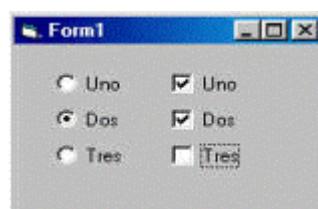


Figura 157. Sólo es posible seleccionar un OptionButton al mismo tiempo, pero podemos seleccionar varios CheckBox.

Propiedades

- **Value.** Contiene un valor lógico, True o False, que indica si el control está o no pulsado. Estableciendo esta propiedad a True en tiempo de diseño, podemos hacer que uno de los controles aparezca seleccionado al comenzar a ejecutar la aplicación.

El resto de propiedades, en su mayoría, realizan las mismas funciones que para un CheckBox, por lo que evitaremos repetirlas de nuevo.

En el siguiente ejemplo, [Colores](#), hemos dibujado tres OptionButton para proporcionar al usuario la elección de un color. Cada vez que haga clic en uno, se desactivará el que hubiera pulsado hasta ese momento.



Figura 158. Formulario mostrando un conjunto de opciones mediante controles OptionButton.

Esta misma situación no podría ser resuelta mediante CheckBox, ya que si utilizamos un único CheckBox, no podemos representar varios valores a seleccionar, y tampoco serviría el utilizar tres CheckBox, ya que podríamos seleccionar más de uno al mismo tiempo, con lo que no sería posible establecer opciones auto excluyentes.

Desde el código de la aplicación, también podemos manipular las selecciones de un conjunto de OptionButton. Siguiendo con el ejemplo anterior, si agregamos un CommandButton y le añadimos el Código fuente 211, podemos cambiar el OptionButton seleccionado a uno diferente, el resultado lo vemos en la Figura 159.

```
Private Sub cmdCambiar_Click()
Me.optRojo.Value = True
End Sub
```

Código fuente 211



Figura 159. Al pulsar el CommandButton, cambiamos el OptionButton seleccionado desde código.

Frame

El uso de OptionButton nos puede plantear un problema: ¿Qué podemos hacer cuando en un mismo formulario necesitamos incluir dos grupos de selecciones diferentes?

Planteemos la siguiente situación: en un formulario tenemos que incluir controles OptionButton para que el usuario seleccione un medio de transporte: Automóvil, Tren, Avión, y una dirección: Norte, Sur, Este, Oeste. Este ejemplo está disponible en el proyecto [OpcFrame](#).

Si insertamos todos estos controles juntos, el formulario no sabrá diferenciar cuál pertenece a un grupo y cuál a otro, se tomarán todos como un único grupo de opciones.

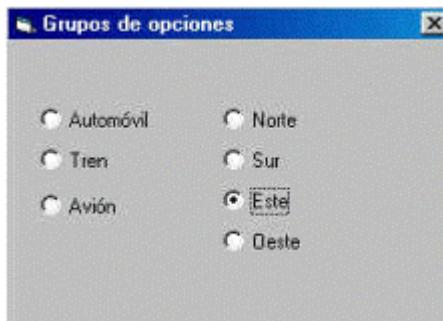


Figura 160. El formulario no detecta dos grupos de OptionButton. Sólo es posible seleccionar un control.

Para solucionar este problema, disponemos del control Frame, que nos permite establecer grupos de controles.



Figura 161. Icono del control Frame en el Cuadro de herramientas.

Este control actúa como contenedor de controles, podemos incluir en su interior varios controles del mismo o diferente tipo, con la ventaja de que cuando sea necesario mover el Frame a un lugar diferente del formulario, los controles contenidos también serán desplazados, ya que son detectados por este control.

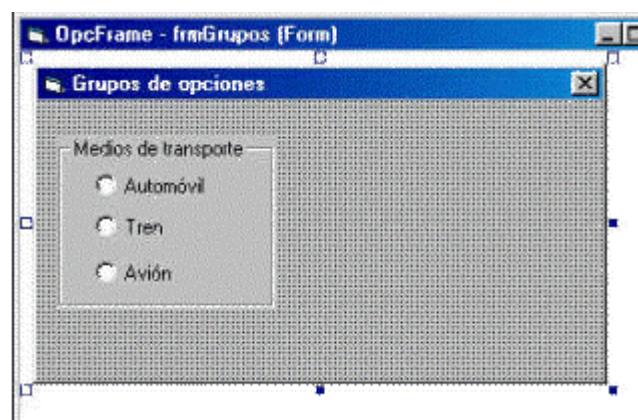


Figura 162. Control Frame y grupo de OptionButton.

Gracias a esta capacidad de detección de controles, podemos situar dos Frame en un mismo formulario y varios OptionButton en cada uno de ellos, de manera que podemos seleccionar dos controles de opción, uno de cada Frame.

Para conseguir el objetivo antes comentado, dibujaremos un Frame en el formulario, y a continuación en su interior, los OptionButton para los medios de transporte, como se muestra en la Figura 162. La propiedad Caption del Frame, nos servirá para mostrar una cadena con el título del control.

Repetiremos la operación para crear el grupo de OptionButton referentes a la dirección, de forma que al ejecutar, podamos establecer una selección en cada grupo de opciones, como se muestra en la Figura 163.



Figura 163. Formulario mostrando grupos de opciones en controles Frame.

En cuanto al código que podemos escribir para un OptionButton, su evento principal es Click(), que se produce cuando el usuario hace clic sobre este control.

Para mostrar algunos ejemplos de codificación de estos controles, disponemos del proyecto [OpcionBut](#), cuyo formulario se muestra en la Figura 164.



Figura 164. Formulario del proyecto OpcionBut.

Al hacer clic sobre los OptionButton del Frame Dirección, se marcará, en función de la opción pulsada, el OptionButton Automóvil o Motocicleta. El Código fuente 212 muestra el evento Click de uno de los OptionButton.

```
Private Sub optOeste_Click()
Me.optMotocicleta.Value = True
End Sub
```

Código fuente 212

Otra de las operaciones de este formulario, consiste en visualizar u ocultar el CommandButton cmdPrueba, según se haga clic en el OptionButton optMostrar u optOcultar, cuyos eventos vemos en el Código fuente 213.

```
Private Sub optMostrar_Click()
Me.cmdPrueba.Visible = True
End Sub

' -----
Private Sub optOcultar_Click()
Me.cmdPrueba.Visible = False
End Sub
```

Código fuente 213

ListBox

Visualiza una lista de elementos de tipo carácter, de los que podemos seleccionar uno o varios. Si el número de elementos es superior a los que puede mostrar el control, se añadirá automáticamente una barra de desplazamiento.



Figura 165. Icono del control ListBox en el Cuadro de herramientas.

Propiedades

- List. Array con el conjunto de valores que son mostrados por el control. Al seleccionar esta propiedad en la ventana de propiedades, debemos pulsar el botón que contiene para abrir la lista de valores. Cada vez que escribamos un nuevo valor, pulsaremos la combinación de teclas Control+Enter para pasar a una nueva línea vacía.
- ListIndex. Devuelve o asigna un número que corresponde al elemento actualmente seleccionado en la lista.
- IntegralHeight. Esta propiedad permite obligar a que se muestren los elementos por completo en la lista, los valores a usar son los siguientes.

- True. Ajusta el tamaño del control para que los elementos de la lista se muestren por completo.
- False. No ajusta el tamaño del control, por lo que los elementos en la parte superior e inferior de la lista se verán de forma parcial.
- Sorted. Permite ordenar alfabéticamente los elementos de la lista si el valor de esta propiedad es True.
- Style. Establece el modo en que se visualizarán y seleccionarán los elementos de la lista.
 - Standard. Modo habitual, al seleccionar un elemento, se queda todo el nombre con el color de selección.
 - CheckBox. Junto a cada elemento de la lista se muestra una casilla de tipo CheckBox. Al seleccionar uno, se marca/desmarca la casilla.
- Selected. Array de valores lógicos, con el mismo número de elementos que la propiedad List. Cada elemento indica si está o no seleccionado el valor de la lista. Estos valores se pueden modificar desde el código.
- MultiSelect. Permite realizar selecciones de varios elementos de la lista simultáneamente. Los valores y constantes para esta propiedad son:
 - vbMultiSelectNone - 0. No se pueden efectuar selecciones múltiples. Valor por defecto.
 - vbMultiSelectSimple - 1. Selección múltiple simple. Al hacer clic sobre un elemento o pulsar la tecla Espacio, se selecciona dicho elemento.
 - vbMultiSelectExtended - 2. Selección múltiple extendida. Al pulsar la tecla Mayúsculas y hacer clic, o pulsar Mayúsculas y una de las teclas de dirección (Flecha Arriba, Abajo, Izquierda, Derecha), se seleccionan todos los elementos existentes entre la última selección y la actual.

Para agregar o eliminar elementos de la lista en ejecución, disponemos de los métodos AddItem() y RemoveItem().

El ejemplo [Lista](#), permite realizar varias operaciones sobre un control ListBox, como añadir/eliminar elementos, mostrar los valores ordenados, recorrer la lista mostrando sólo los elementos seleccionados, etc.

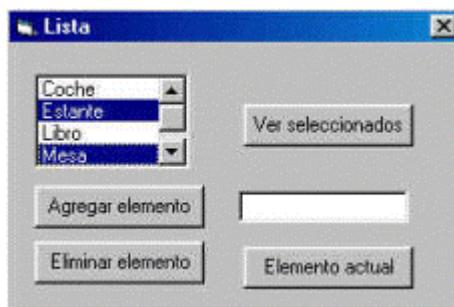


Figura 166. Formulario frmLista con las opciones disponibles.

En el Código fuente 214 se muestra el código fuente de los eventos para los controles del formulario.

```

Private Sub cmdAgregar_Click()
' añadir un nuevo elemento a la lista
Me.lstValores.AddItem Me.txtAgregar.Text
Me.txtAgregar.Text = ""
End Sub
' -----
Private Sub cmdElemento_Click()
' visualizar el elemento actualmente
' seleccionado
MsgBox "Elemento seleccionado: " & _
    Me.lstValores.List(Me.lstValores.ListIndex)
End Sub
' -----
Private Sub cmdEliminar_Click()
' eliminar de la lista el elemento actual
Me.lstValores.RemoveItem Me.lstValores.ListIndex
End Sub
' -----
Private Sub cmdVerSelec_Click()
' mostrar mediante un mensaje,
' sólo los elementos seleccionados
Dim lnContador As Integer
For lnContador = 0 To Me.lstValores.ListCount - 1
    If Me.lstValores.Selected(lnContador) Then
        MsgBox Me.lstValores.List(lnContador), , _
            "Elemento seleccionado"
    End If
Next
End Sub

```

Código fuente 214

ComboBox

Este control es una combinación de TextBox y ListBox, ya que contiene una zona en la que el usuario puede editar texto y otra con una lista de valores.



Figura 167. Icono del control ComboBox en el Cuadro de herramientas.

Propiedades

- Style. Determina el comportamiento de la lista y el modo de edición del cuadro de texto en este control. Las constantes disponibles se muestran en la Tabla 29.

Constante	Valor	Descripción
VbComboDropDown	0	(Predeterminado) Cuadro combinado desplegable. Incluye una lista desplegable y un cuadro de texto. El usuario puede seleccionar datos en la lista o

		escribir en el cuadro de texto
VbComboSimple	1	Cuadro combinado simple. Incluye un cuadro de texto y una lista, que no se despliega. El usuario puede seleccionar datos en la lista o escribir en el cuadro de texto. El tamaño de un cuadro combinado simple incluye las partes de edición y de lista. De forma predeterminada, el tamaño de un cuadro combinado simple no muestra ningún elemento de la lista. Incremente la propiedad Height para mostrar más elementos de la lista.
VbComboDrop-DownList	2	Lista desplegable. Este estilo sólo permite la selección desde la lista desplegable

Tabla 29. Constantes disponibles para la propiedad Style de ComboBox.

Debido a las propiedades comunes con los controles antes mencionados, esta propiedad es la única destacable en el control ComboBox.

El proyecto [ListaComb](#), contiene un sencillo ejemplo de uso de este control. El ComboBox cboValores, inicialmente no contiene valores. Deberá ser el usuario el que los introduzca en la zona de edición del control; al pulsar Enter, la cadena tecleada se añadirá a la lista. Para conseguirlo, debemos codificar el evento KeyPress() del ComboBox.

```
Private Sub cboValores_KeyPress(KeyAscii As Integer)
' al pulsar Enter, el contenido del cuadro
' de texto se añade a la lista
If KeyAscii = vbKeyReturn Then
    Me.cboValores.AddItem Me.cboValores.Text
    Me.cboValores.Text = ""
End If
End Sub
```

Código fuente 215



Figura 168. ComboBox cboValores mostrando la lista de valores.

Timer

Un control Timer actúa como un temporizador que permite ejecutar código a intervalos determinados de tiempo.

Carece de interfaz de usuario, es decir, en tiempo de ejecución permanecerá oculto, por lo que al dibujarlo en el formulario, toma un tamaño fijo que no puede ser alterado.

En el evento Timer() de este control, se debe escribir el código a ejecutar cada vez que cumplan los intervalos de tiempo especificados.



Figura 169. Icono del control Timer en el Cuadro de herramientas.

Propiedades

- Interval. Indica en milisegundos, el tiempo que transcurrirá entre cada ejecución del evento Timer().
- Enabled. Al asignar True a esta propiedad, el temporizador se pondrá en funcionamiento, mientras que al asignarle False, se detendrá.

El proyecto [Temporizador](#), muestra uno de los modos de uso de este control. Se trata de un formulario que contiene el control Timer tmrTemporizador, y al pulsar el botón cmdIniciar, se pondrá en marcha el Timer, que se ocupará en cada intervalo de tiempo ejecutado, de traspasar una letra desde el control txtOrigen a txtDestino. Finalizado el traspaso entre los dos TextBox, se detendrá el Timer.

Veamos el Código fuente 216 y la Figura 170 de este proyecto.

```
' variable para saber que letra
' tomar en cada momento del TextBox
' origen
Private mnLetraActual As Integer

' -----
Private Sub cmdIniciar_Click()
' poner en marcha el temporizador
Me.tmrTemporizador.Enabled = True
End Sub

' -----
Private Sub tmrTemporizador_Timer()
' calcular la letra a tomar
mnLetraActual = mnLetraActual + 1
' traspasar la posición de la letra desde
' el TextBox origen al destino
Me.txtDestino.Text = Me.txtDestino.Text &
    Mid(Me.txtOrigen.Text, mnLetraActual, 1)
' al completar el traspaso de texto,
' detener el temporizador
If Me.txtOrigen.Text = Me.txtDestino.Text Then
    Me.tmrTemporizador.Enabled = False
End If
End Sub
```

Código fuente 216

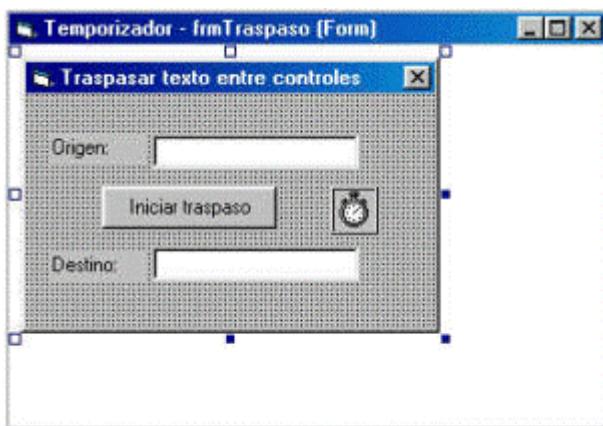


Figura 170. Formulario del ejemplo para el control Timer.

DriveListBox, DirListBox y FileListBox

Estos controles constituyen una variante del control ListBox estándar, que permiten, en el caso de que los sincronicemos mediante código, la navegación entre las unidades, directorios y ficheros del sistema.

DriveListBox

Este control muestra unidades de disco y permite cambiar a otras unidades accesibles en el sistema.



Figura 171. Icono del control DriveListBox en el Cuadro de herramientas.

Propiedades

- Drive. Devuelve o establece una cadena con una unidad válida, ya sea disco flexible, fijo o unidad de red.

DirListBox

Visualiza directorios y subdirectorios del sistema, permitiendo la navegación a través de los mismos.



Figura 172. Icono del control DirListBox en el Cuadro de herramientas.

Propiedades

- Path. Devuelve o establece una cadena con la ruta de acceso.

FileListBox

Muestra una lista de ficheros perteneciente al directorio indicado por su propiedad Path.



Figura 173. Icono del control FileListBox en el Cuadro de herramientas.

Propiedades

- Path. Cadena con la ruta de la cual se van a mostrar ficheros.
- Pattern. Cadena con el patrón del tipo de ficheros que mostrará el control. Por defecto se utilizan todos los tipos de fichero "*.*". Es posible utilizar caracteres comodín, como "* .exe", e indicar varios tipos de archivos, utilizando el punto y coma (;) como separador "* .bmp; *.gif".
- ReadOnly. Valor lógico, que permite mostrar en la lista ficheros de sólo lectura, cuando esta propiedad sea True.
- System - Archive - Hidden. Estas propiedades contienen un valor lógico, que permiten mostrar en la lista, ficheros del sistema, con el atributo modificado y ocultos, cuando la correspondiente propiedad sea True.

Para mostrar el funcionamiento conjunto de estos controles, el lector dispone del proyecto [Visualizador](#), que consiste en un visualizador de ficheros gráficos. El formulario del programa dispone del ComboBox cboTipoFi, en el que podremos seleccionar los tipos de ficheros a visualizar, por defecto serán .JPG.

Por otra parte se incluye un control Image para visualizar el fichero seleccionado y los tres controles de lista comentados en este apartado.



Figura 174. Formulario con los controles Drive-Dir-ListBox para visualizar ficheros gráficos.

En el Código fuente 217 se incluye el código del proyecto para las diferentes acciones que tome el usuario sobre los controles.

```
Private Sub cboTipoFi_Click()
' establecer el tipo de fichero
' al seleccionarlo de la lista
Me.filFicheros.Pattern = "*." & Me.cboTipoFi.Text
End Sub

' -----
Private Sub cboTipoFi_KeyPress(KeyAscii As Integer)
' establecer el tipo de fichero
' manualmente y pulsando Enter
If KeyAscii = vbKeyReturn Then
    Me.filFicheros.Pattern = "*." & Me.cboTipoFi.Text
End If
End Sub

' -----
Private Sub dirCarpeta_Change()
' al cambiar la ruta del directorio
' cambiar la ruta de la lista de
' ficheros
Me.filFicheros.Path = Me.dirCarpeta.Path
End Sub

' -----
Private Sub drvUnidad_Change()
' al cambiar la unidad actual
' cambiar la ruta de directorios
Me.dirCarpeta.Path = Left(Me.drvUnidad.Drive, 2) & "\"
End Sub

' -----
Private Sub filFicheros_Click()
' al seleccionar un fichero de la lista
' mostrarlo en el control Image
Me.imgImagen.Picture = LoadPicture(Me.dirCarpeta.Path & "\" & _
    Me.filFicheros.FileName)
End Sub
```

Código fuente 217

Validación de controles

Hasta la llegada de Visual Basic 6, cuando en un formulario de entrada de datos era necesario comprobar que los valores introducidos en los respectivos controles de dicho formulario fueran correctos, el programador debía recurrir a efectuar las validaciones oportunas en el evento `LostFocus()` del control que lo necesitara, o bien en el `CommandButton` u otro control encargado de aceptar el conjunto de todos los controles del formulario, o recurrir a otra técnica desarrollada por el programador para este cometido.

Sin embargo, el uso de `LostFocus()` entraña el problema de que es un evento producido cuando el control a validar ya ha perdido el foco, como indica su propio nombre, por lo que si su información no es adecuada, debemos volver a asignar el foco al control que lo tenía. Esto conlleva el problema

añadido de que en ciertos contextos de validación, el flujo de la aplicación puede caer en un bucle infinito que bloquee el sistema.

Afortunadamente, a partir de VB 6 los controles incorporan la propiedad CausesValidation y el evento Validate(), que empleados conjuntamente, proporcionan al programador un medio más eficaz de realizar la validación del contenido de los controles de un formulario.

CausesValidation contiene un valor lógico que si es establecido a True en un primer control, hará que al intentar pasar el foco a un segundo control, y siempre que este segundo control también tenga True en esta propiedad, sea llamado antes de cambiar el foco el evento Validate() del primer control. Es en este evento donde el programador deberá escribir el código de validación para el control, y en el caso de que no sea correcto, establecer el parámetro Cancel de este evento a True, para forzar a mantener el foco en el primer control al no haber cumplido la regla de validación. De esta manera disponemos de un medio más flexible y sencillo de manejar las reglas de validación establecidas para los controles.

El esquema de funcionamiento de esta técnica de validaciones se muestra en la Figura 175

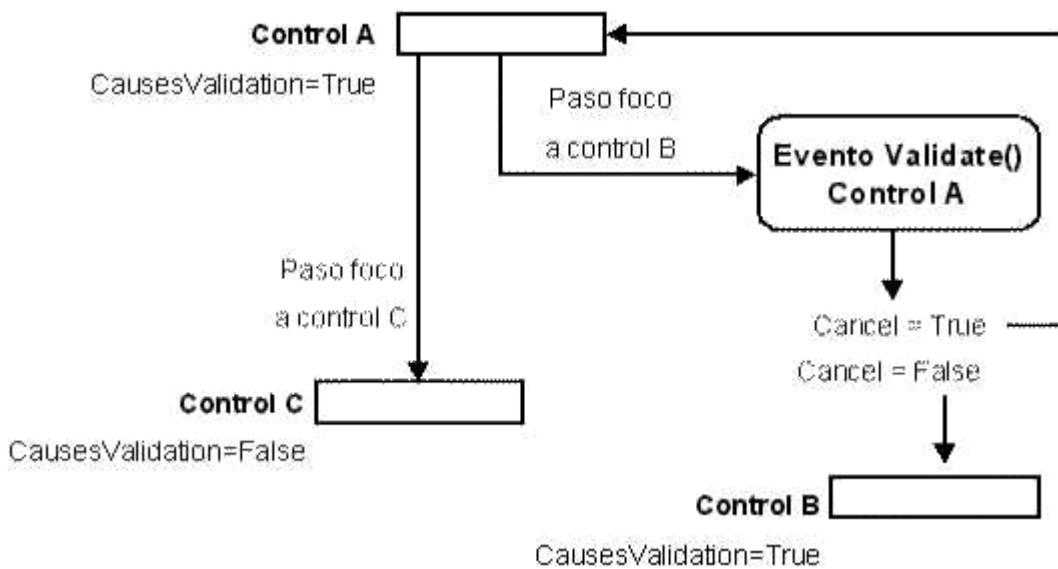


Figura 175. Esquema de funcionamiento de validaciones.

Según el esquema de funcionamiento mostrado en la anterior figura, al pasar del control A al B, se ejecuta el código de Validate(), si dentro de este evento no se cumplen las reglas de validación y se establece Cancel a True, el foco se devuelve al control A. En el caso de que todo sea correcto y Cancel valga False, el foco pasará al control B.

Para la situación de paso de foco desde el control A al C, no existirán impedimentos, ya que al tener el control C su propiedad CausesValidation establecida a False, no se ejecutará Validate() y siempre llegarán al control.

Sin embargo, lo que puede no llegar a ser tan intuitivo en este último caso es que una vez situados en el control C, si intentamos pasar el foco al control B no será posible ya que este si tiene el Valor True en CausesValidation, ejecutándose el Validate() del control A.

El ejemplo que el lector puede encontrar en [ValidarCtls](#), consiste en un formulario de entrada de datos, en el que todos los controles tienen el valor True en CausesValidation excepto el botón cmdCancelar. Adicionalmente, en el TextBox txtCodigo se han incluido las líneas del Código fuente 218 en su método Validate(), de modo que el usuario sólo pueda introducir números.

```
Private Sub txtCodigo_Validate(Cancel As Boolean)
If Not IsNumeric(Me.txtCodigo.Text) Then
    MsgBox "Sólo se permiten valores numéricos en este campo"
    Cancel = True
End If
End Sub
```

Código fuente 218

Una vez en ejecución, si intentamos introducir un valor incorrecto en el control txtCodigo y cambiar a un control que no sea cmdCancelar, se mostrará el aviso que nos muestra la Figura 176.

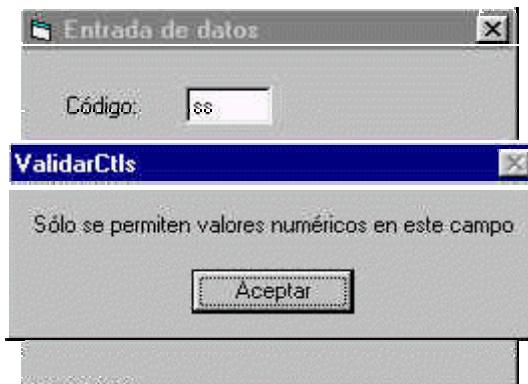


Figura 176. Aviso de error en la validación de los datos.

Diseño de menús

Descripción de un menú

Un menú es uno de los componentes más habituales en los programas Windows. Se basa en un conjunto de opciones desplegables a partir de una serie de opciones principales, que permiten organizar el acceso a las diferentes partes de la aplicación. A efectos de programación, cada opción de menú es tratada y se maneja de la misma forma que cualquier control dentro del formulario.

Sin ser imprescindible su uso, si es útil en el ámbito organizativo, ya que evita la sobrecarga de controles en el formulario. Hemos de tener en cuenta que si no incorporamos un menú, tendremos que utilizar otro control, un botón de comando por ejemplo, por cada opción que queramos ejecutar dentro del formulario, lo que ocupa más espacio y resulta menos estético, debido a la sobrecarga de controles que se produciría dentro del formulario.

Si la aplicación a desarrollar, se basa en una interfaz MDI, el uso de menú resulta casi obligatorio, ya que al estar un formulario MDI restringido al uso de un buen número de controles, el menú es el mejor medio de acceder a las opciones y formularios secundarios del programa.

Características de un menú

Un menú está compuesto por un conjunto de opciones principales o nivel superior, que se disponen en la parte más próxima al título del formulario. De cada opción del nivel principal, se despliega un conjunto de opciones o menú de nivel inferior dependientes del principal. A su vez, desde las opciones de este segundo nivel se pueden seguir abriendo sucesivamente niveles más inferiores.

Aunque la posibilidad de desplegar varios niveles de menús es muy interesante, de forma que podamos alcanzar un alto nivel de organización, no conviene, sin embargo, hacer un uso abusivo de esta cualidad, ya que un usuario puede tomar un mal concepto de una aplicación que le obliga a descender muchos niveles de menús para seleccionar una opción que es muy utilizada a lo largo de la aplicación.

Propiedades de un control Menú

Puesto que una opción de menú es un control, al igual que el resto de controles tiene una serie de propiedades que vemos a continuación.

- **Name.** Nombre del control.
- **Caption.** Texto que muestra la opción.
- **Enabled.** Valor lógico que habilita o deshabilita la opción. Cuando está deshabilitada, aparece en gris y el usuario no puede seleccionarla.
- **Checked.** Dato lógico que cuando es verdadero, muestra una marca en la opción de menú, para informar al usuario de que esa opción está activada.
- **Visible.** Si su valor es False, oculta la opción. Por defecto es True.
- **Index.** Permite definir una opción como parte de un array de controles u opciones de menú. Esta propiedad es útil para crear nuevas opciones de menú en tiempo de ejecución.
- **Shortcut.** Contiene el valor de una tecla de método abreviado o acelerador de teclado, como F5 o Ctrl+G. Cuando se define un acelerador, la combinación de teclas aparece junto a la descripción del menú. La ventaja de un acelerador es que no necesitamos navegar a través de las opciones del menú para llegar a una determinada. Si la que queremos tiene definido un acelerador, tecleándolo lanzaremos directamente la opción de ese menú.
- **WindowList.** En un formulario MDI, si damos el valor True a esta propiedad en una opción de nivel superior, el programa creará de forma transparente al usuario un submenú con las ventanas que tiene abiertas la aplicación, marcando la ventana activa.

En cuanto a eventos, el único disponible para el programador es Click(), que se dispara al seleccionar el usuario una opción del menú. Aquí deberemos incluir el código que se ejecutará para la opción de menú.

El editor de Menús

Para crear un menú en el formulario hemos de hacerlo usando la ventana de edición de menús, la cual podemos abrir de varias maneras:

- Hacer clic con el botón derecho del ratón dentro del formulario, para visualizar el menú contextual del mismo. Dentro de este menú tenemos una opción para abrir el editor de menús.
- Pulsar el botón *Editor de menús* en la barra de herramientas de VB.
- Seleccionar el menú *Herramientas + Editor de menús* de VB.

- Usar la combinación de teclado *Ctrl+E*.

Al abrir el editor de menús por primera vez, obtenemos una ventana similar a la Figura 177.

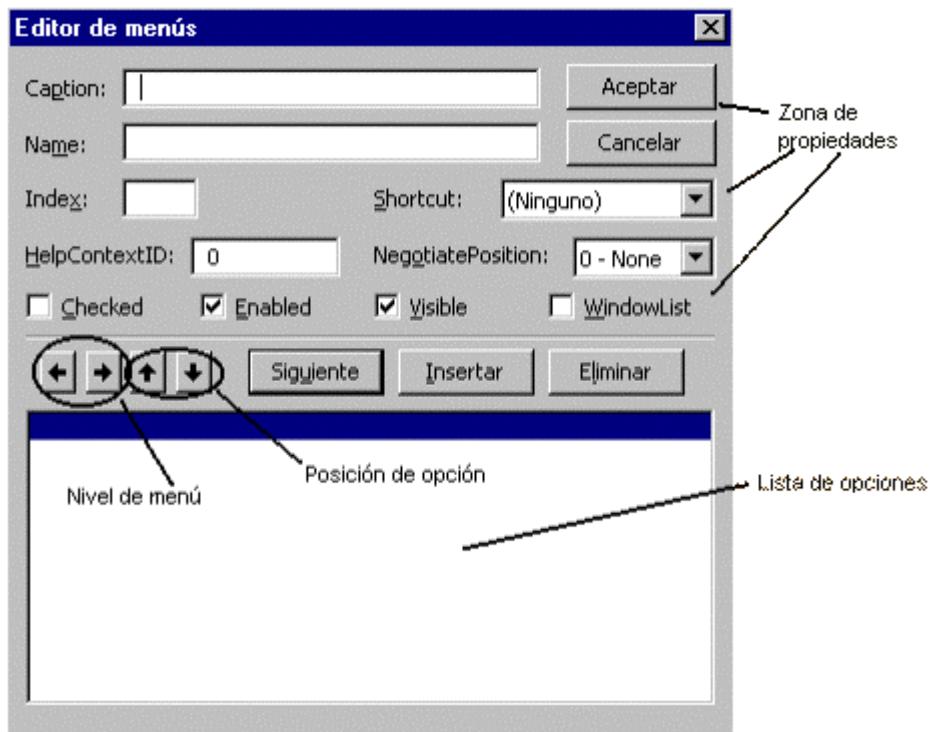


Figura 177. Editor de menús.

Los pasos para crear las opciones del menú son los siguientes:

- Introducir el Caption o título de la opción, que es la cadena de caracteres que aparecerá en el menú. Si deseamos que esta opción tenga una tecla de acceso rápido o hotkey, hemos de situar el carácter & precediendo a la letra que queramos actúe como hotkey. De esta forma, la letra aparecerá subrayada en el menú y podremos seleccionar la opción con la combinación de teclado *Alt+hotkey* si pertenece al nivel principal. Si es una opción incluida en un nivel inferior, debemos abrir primero el nivel y después pulsar la tecla de acceso rápido.
- Asignar un valor a Name, que es el nombre del control, y nos servirá para hacer referencia a él dentro del código.
- En este punto podemos asignar a la opción algún tipo de variante: habilitarla, marcarla, ocultarla, asignarle un acelerador de teclado o Shortcut, etc.
- Completadas todas las propiedades de la opción, pulsamos Siguiente, con lo que pasará a la parte inferior de la ventana, en la zona de opciones ya creadas
- Un aspecto muy importante a tener en cuenta es el nivel en el que vamos a situar la opción. Si deseamos que esté situada en un nivel inferior, hemos de hacer clic en el botón con la flecha a la derecha y un grupo de puntos aparecerá en la zona de opciones indicando que esa opción es de nivel inferior. Cuantos más grupos de puntos haya, mayor profundidad tendrá la opción. De forma inversa, si queremos que una opción suba de nivel, hemos de hacer clic en el botón con flecha a la izquierda. Si la opción no tiene puntos, es que pertenece al nivel principal.

- Es posible cambiar la posición de una opción, haciendo clic en los botones con las flechas arriba o abajo la desplazaremos en una u otra dirección.
- Pulsando el botón *Insertar* crearemos una nueva opción vacía entre las opciones ya creadas.
- Pulsando el botón *Eliminar* borraremos la opción actualmente resaltada.
- Podemos agrupar diferentes opciones dentro de un mismo menú usando separadores o líneas divisorias. Simplemente hemos de poner un guión "-" en el Caption de la opción de menú.

Añadir código a una opción de menú

Una vez terminado el diseño del menú, tenemos dos formas de incluir el código que debe ejecutar:

- Desde la ventana de diseño del formulario, seleccionar la opción de menú, lo que abrirá la ventana de código del formulario, situándonos en el método Click(), que como hemos visto anteriormente es el único disponible para este control.
- Abrir la ventana de código del formulario, y buscar nosotros directamente en la lista de objetos el control a codificar.

Creación de un menú

Vamos a ver una aplicación que contenga un menú en el que sus opciones muestren las propiedades descritas anteriormente.

Tomamos del grupo de ejemplos la aplicación llamada [Menu](#). Una vez cargada en VB, abrimos su único formulario *frmMenu* que dispone de la siguiente estructura de menú:

```
Archivo
--- Usuarios (Enabled: False)
--- Entradas (Checked: True)
--- Salir
Contabilidad
--- Apuntes
--- Consultas
---- Cuenta
---- Fecha
Clientes
--- Altas
--- Bajas
--- (Separador)
--- Enviando correo
Proveedores
--- Altas (Shortcut: Ctrl+F)
--- Bajas
--- Consultas (Shortcut: Ctrl+Y)
Fuente (Visible: False / este será un menú emergente)
--- Arial
--- Courier
--- Roman
```

Dinámico

---Dinámico1 (Index: 1)

Algunas de las propiedades ya están asignadas en tiempo de diseño, otras las modificaremos en ejecución. Así pues, ejecutamos la aplicación y en el primer menú podemos ver la opción *Usuarios* deshabilitada, aparece en gris, y *Enviando correo* que está marcada.

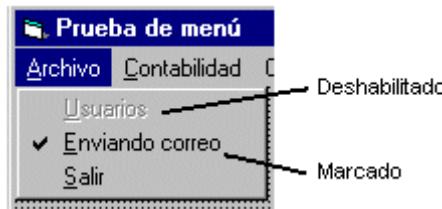


Figura 178. Aplicación Menu. Menú Archivo.

Desde aquí podemos probar el uso de un hotkey. Si pulsamos por ejemplo la tecla "S", se ejecutará el código asociado a la opción *Salir*, que finalizará el programa descargando el formulario. El fuente se muestra en el Código fuente 219.

```
Private Sub mnuArSalir_Click()
Unload Me
End Sub
```

Código fuente 219

Podemos cambiar el estado de la opción *Usuarios* pulsando el botón *Habilitar/Deshabilitar* del formulario, que tiene el código que aparece en el Código fuente 220.

```
Private Sub cmdHabilDeshab_Click()
If Me.mnuArUsuarios.Enabled Then
    Me.mnuArUsuarios.Enabled = False
Else
    Me.mnuArUsuarios.Enabled = True
End If
End Sub
```

Código fuente 220

De la misma forma, se puede cambiar la marca de la opción *Enviando correo*, pulsando el botón *Enviar correo*. Veámoslo en el Código fuente 221.

```
Private Sub cmdMarcar_Click()
If Me.mnuArEntradas.Checked Then
    Me.mnuArEntradas.Checked = False
Else
    Me.mnuArEntradas.Checked = True
End If
End Sub
```

Código fuente 221

Como muestra de menú con varios niveles de opciones, tenemos el menú *Contabilidad*, en el que la opción *Consultas* despliega a su vez otro submenú.



Figura 179. Aplicación Menu. Menú Contabilidad con todos los submenús desplegados.

En el menú Proveedores disponemos de dos opciones con acelerador de teclado

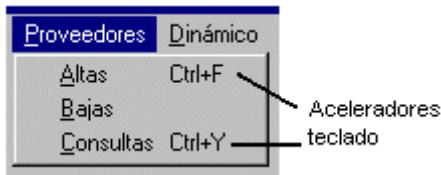


Figura 180. Aplicación Menu. Menú Proveedores.

Al pulsar alguna de estas combinaciones de teclado aparecerá un mensaje informándonos de que hemos seleccionado esa opción.

```
Private Sub mnuPrAltas_Click()
MsgBox "Menú Proveedores - Opción Altas"
End Sub
'-----
Private Sub mnuPrConsultas_Click()
MsgBox "Menú Proveedores - Opción Consultas"
End Sub
```

Código fuente 222

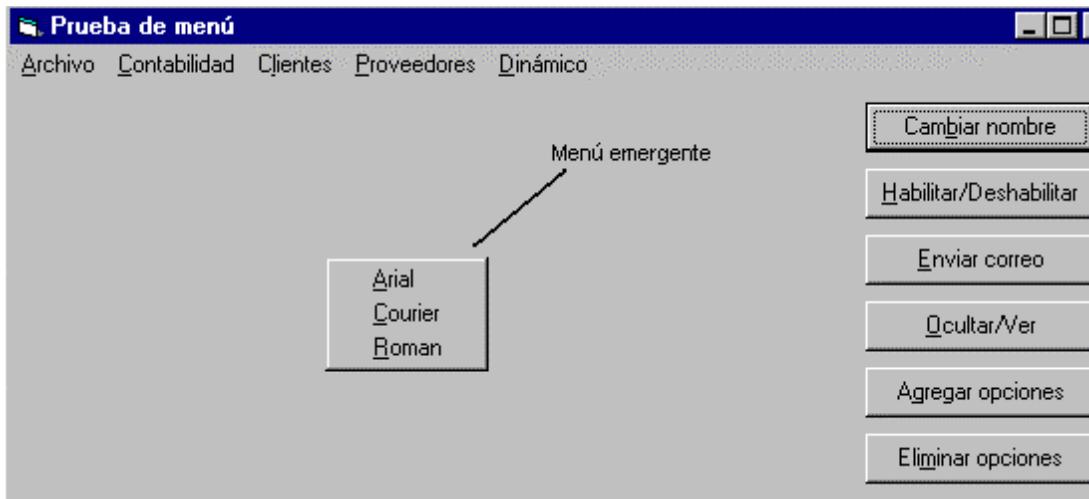


Figura 181. Aplicación Menu mostrando el menú emergente.

Durante el diseño del menú, a la opción de nivel superior *Fuente*, se le ha quitado la marca en la propiedad *Visible*. Esto provoca que al ejecutar el programa no aparezca en la barra de menús ya que *Visible* es *False*. La causa de hacer esto es conseguir un menú emergente de los que aparecen al pulsar el botón derecho del ratón, como muestra la Figura 181. La forma de comprobar la pulsación del ratón en el formulario es mediante el método *MouseDown()*. El parámetro *Button* nos informa del botón pulsado, si ha sido el derecho, visualizamos el menú *Fuente* con el método *PopupMenu* del formulario.

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = vbRightButton Then
    PopupMenu Me.mnuFuente, vbPopupMenuRightButton, , , Me.mnuFuRoman
End If
End Sub
```

Código fuente 223

La sintaxis de *PopupMenu* es la siguiente:

```
PopupMenu cNombreMenu, xConstantes, x, y, mnuNegrita
```

- *cNombreMenu*. Nombre del menú que se va a presentar como emergente.
- *xConstantes*. Valores que especifican la ubicación y el comportamiento del menú. Los valores para la ubicación son los siguientes:
 - *vbPopupMenuLeftAlign*. El lado izquierdo del menú se sitúa en *x*.
 - *vbPopupMenuCenterAlign*. El menú se sitúa centrado.
 - *vbPopupMenuRightAlign*. El lado derecho del menú se sitúa en *x*.

Los valores para el comportamiento son los siguientes:

- *vbPopupMenuLeftButton*. Sólo se pueden seleccionar las opciones del menú con el botón izquierdo del ratón; esta es la opción por defecto.
- *vbPopupMenuRightButton*. Se pueden seleccionar las opciones del menú también con el botón derecho del ratón.
- *x, y*. Coordenadas para situar el menú en el formulario.
- *mnuNegrita*. Nombre de una de las opciones del menú que deseamos que aparezca resaltada en negrita.

Es posible cambiar el título de una opción en tiempo de ejecución simplemente asignando una nueva cadena a su propiedad *Caption*. El botón *Cambiar nombre* cambia el título de la opción *Apuntes* del menú *Contabilidad* como indica el Código fuente 224.

```
Private Sub cmdCambiaCaption_Click()
Me.mnuCoApuntes.Caption = "Nuevo nombre"
End Sub
```

Código fuente 224

Si queremos que una opción no sea visible para el usuario, ponemos a *False* su propiedad *Visible*, como hacemos en el botón *Ocultar/Ver*.

```
Private Sub cmdOcultarVer_Click()
If Me.mnuPrBajas.Visible Then
    Me.mnuPrBajas.Visible = False
Else
    Me.mnuPrBajas.Visible = True
End If
End Sub
```

Código fuente 225

Con los botones *Agregar opciones* y *Eliminar opciones* añadimos o quitamos opciones al menú *Dinámico* en tiempo de ejecución. Para conseguir esta funcionalidad utilizamos un array de controles *Menu*, de forma que a la propiedad *Index* de la opción *Dinámico1* le damos el valor 1 (podríamos haberle puesto 0, pero por cuestiones de controlar el número de opciones creadas es más sencillo usar 1). También hemos definido la variable *mnOpcionesDinam* a nivel de formulario para que actúe como contador de las opciones que llevamos creadas.

Una vez hecho esto, cada vez que pulsamos el botón para agregar, utilizaremos la instrucción *Load* y el array de opciones con el nuevo número de opción a crear. Cuando pulsamos el botón para eliminar, usaremos la instrucción *Unload* y el array de controles con el número de opción a suprimir. El Código fuente 226 contiene los métodos involucrados en estas operaciones.

```
' zona de declaraciones
Option Explicit
Private mnOpcionesDinam As Integer
'-----
Private Sub Form_Load()
' inicializamos el contador
mnOpcionesDinam = 1
End Sub
'-----
Private Sub cmdAgregar_Click()
' sumamos al contador
mnOpcionesDinam = mnOpcionesDinam + 1
' creamos una nueva opción de menú con la instrucción Load
Load Me.mnuDiDinam(mnOpcionesDinam)
Me.mnuDiDinam(mnOpcionesDinam).Caption = "Dinámico" & CStr(mnOpcionesDinam)
End Sub
'-----
Private Sub cmdEliminar_Click()
' si el contador es mayor de 1 permitimos borrar opciones,
' en caso contrario no borramos ya que se produciría un error
If mnOpcionesDinam > 1 Then
    ' eliminamos una opción del menú descargándola
    Unload Me.mnuDiDinam(mnOpcionesDinam)
    ' ajustamos el contador
    mnOpcionesDinam = mnOpcionesDinam - 1
End If
```

End Sub

Código fuente 226

Menús en formularios MDI

Un menú en un formulario MDI funciona de la misma manera que en una aplicación SDI, la única salvedad reside en que al abrir una ventana secundaria MDI, si esta tiene su propio menú, dicho menú sustituirá al del formulario MDI mientras la ventana secundaria se encuentre en funcionamiento. Al descargarse volverá a mostrarse el menú del formulario MDI.

La propiedad *WindowList* al ser usada en el menú de un formulario MDI nos permite definir un menú en el que sus opciones mostrarán los nombres de las ventanas secundarias abiertas.

En la aplicación de ejemplo *MenuMDI* se ha incorporado esta característica. Disponemos de dos formularios: *frmPrincipal* que es el formulario MDI y *frmDocumento* que es el formulario que va a actuar como documento dentro del MDI.

En el editor de menús de *frmPrincipal*, la opción *mnuVentana* tiene marcada la propiedad *WindowList*, con lo que estará activada la capacidad de mostrar en este menú los nombres de las ventanas secundarias abiertas dentro de la MDI.

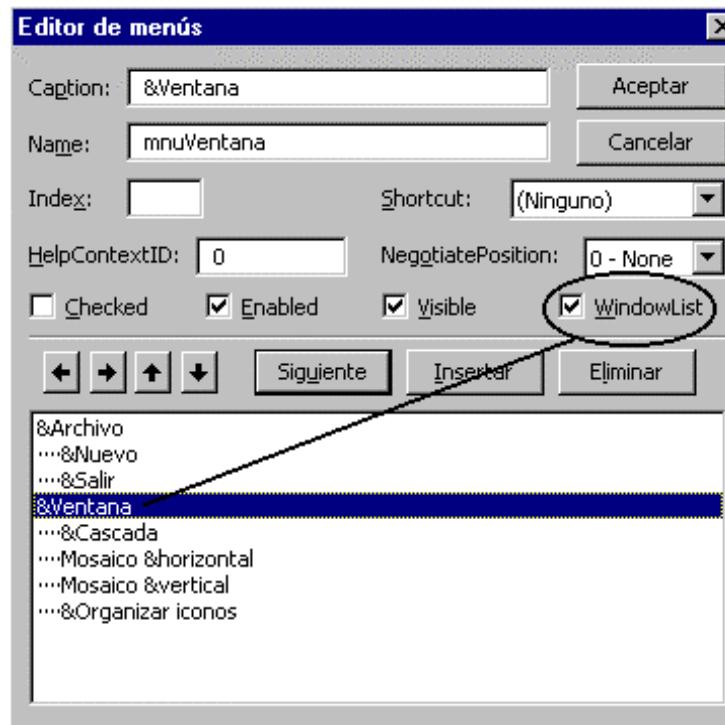


Figura 182. Editor de menú con propiedad *WindowList* activada.

Ejecutamos ahora la aplicación. Para trabajar con un nuevo documento seleccionamos la opción *Nuevo* del menú *Archivo* que crea una nueva ventana secundaria y controla su número gracias a un contador definido a nivel de módulo.

Private Sub mnuArNuevo_Click()

```
Dim frmDocum As frmDocumento
' instancia un nuevo objeto de la clase frmDocumento
Set frmDocum = New frmDocumento
' suma el contador
mnNumDocumento = mnNumDocumento + 1
frmDocum.Caption = "Documento" & mnNumDocumento
frmDocum.Show
End Sub
```

Código fuente 227

Abrimos varios documentos más con esta opción y pasamos al menú *Ventana*. Este menú es el típico de muchas aplicaciones MDI Windows, en el que las primeras opciones son para organizar las diferentes ventanas secundarias de la aplicación, y las restantes sirven para cambiar el foco entre esas ventanas secundarias. Para organizar las ventanas disponemos del método *Arrange* del formulario MDI. El Código fuente 228 muestra las rutinas encargadas de la organización de ventanas.

```
Private Sub mnuVeCascada_Click()
' organiza las ventanas en cascada
Me.Arrange vbCascade
End Sub
'-----
Private Sub mnuVeIconos_Click()
' organiza las ventanas iconizadas
Me.Arrange vbArrangeIcons
End Sub
'-----
Private Sub mnuVeMosHoriz_Click()
' organiza las ventanas en mosaico horizontal
Me.Arrange vbTileHorizontal
End Sub
'-----
Private Sub mnuVeMosVert_Click()
' organiza las ventanas en mosaico vertical
Me.Arrange vbTileVertical
End Sub
```

Código fuente 228

Las siguientes opciones en este menú corresponden cada una a una de las ventanas abiertas dentro del formulario MDI. Estas opciones son creadas, mantenidas y eliminadas por la aplicación sin que tengamos que escribir ni una sola línea de código para ello. Lo que supone una gran ventaja y ahorro de tiempo al programador.



Figura 183. Menú de aplicación MDI con opciones de ventanas abiertas.

Manipulación de ficheros

¿Por qué usar ficheros?

Durante el desarrollo de una aplicación, puede darse el caso de que nos encontremos en la necesidad de guardar una determinada cantidad de información para ser usada en una ejecución posterior de esa misma aplicación o de otra que necesite consultar dichos datos.

Debido al diseño del programa, puede que no sea conveniente o que no estemos interesados en emplear una base de datos para guardar la información. Alguno de los motivos de esta decisión pueden ser: el que la cantidad de datos a manipular sea pequeña y no merezca la pena el uso de un gestor de datos, o bien, que vayamos a realizar un seguimiento de la ejecución del programa, grabando ciertas acciones del usuario en un fichero de log, por lo cual será más conveniente el uso de un fichero de texto para efectuar dicho seguimiento.

Ante las anteriores situaciones o cualquier otra que se pudiera plantear, Visual Basic nos proporciona un amplio conjunto de instrucciones y una jerarquía de objetos para la manipulación de ficheros; con los que podemos efectuar operaciones de lectura/escritura tanto de texto como de información binaria sobre ficheros.

En los siguientes apartados veremos las instrucciones básicas para abrir y cerrar ficheros. Posteriormente y en función del tipo de acceso, se mostrará el modo de leer y escribir información en tales ficheros.

Apertura de un fichero

- Open. Mediante esta instrucción, podemos abrir un fichero y dejarlo preparado para efectuar sobre él operaciones de lectura/escritura.

Sintaxis.

```
Open cNombreFichero [For ModoApertura] [Access TipoAcceso]
[TipoBloqueo] As [#]nNúmeroFichero [Len = nLongitudRegistro]
```

- cNombreFichero. Cadena con el nombre del fichero a abrir. Puede incluir la ruta donde se encuentra el fichero. En el caso de que no se especifique la ruta, se abrirá el fichero en el directorio actual de la aplicación.
- ModoApertura. Una de las siguientes palabras clave, que indican como se va a abrir el fichero en función de las operaciones que vayamos a efectuar sobre él: Append, Binary, Input, Output, Random.
- TipoAcceso. Palabra clave que establece las operaciones a realizar en el fichero: Read, Write o Read Write.
- TipoBloqueo. Palabra clave que especifica el tipo de bloqueo a realizar sobre el fichero: Shared, Lock Read, Lock Write, Lock Read Write.
- nNúmeroFichero. Valor numérico comprendido en el rango de 1 a 511, utilizado por Open para identificar cada uno de los ficheros abiertos por la aplicación.
- nLongitudRegistro. Número igual o inferior a 32767. Si el fichero se abre para acceso aleatorio, este número indica la longitud del registro. Si el fichero se abre para acceso secuencial, indica el número de caracteres situados en la zona de memoria al realizar las operaciones de lectura y escritura. La cláusula Len se ignora cuando el modo de apertura es Binary.

Si el archivo indicado en cNombreFichero no existe al intentar abrirlo, se creará un nuevo archivo vacío con ese mismo nombre cuando el modo de apertura sea uno de los siguientes: Append, Binary, Output, Random.

Si al intentar abrir un fichero, ya estuviera abierto y bloqueado para no permitir otros accesos, o el modo de apertura fuera Input, se produciría un error.

Un fichero abierto se puede abrir de nuevo con un número distinto, siempre y cuando el modo de apertura sea Binary, Input y Random. Sin embargo, para abrir un fichero en los modos Append y Output, dicho fichero debe estar cerrado y no ser utilizado por ningún otro proceso.

En lo que respecta al número identificativo del fichero abierto, para asegurarnos que no intentamos utilizar un número ya en uso por otro proceso, podemos emplear la función FreeFile(), que devuelve un número de fichero libre para poder ser utilizado con Open.

Sintaxis.

```
FreeFile [ (nNúmeroIntervalo) ]
```

- nNúmeroIntervalo. Especifica el rango del que se va a tomar el número de fichero:

- Valor por defecto, devuelve un número de 1 a 255.
- Devuelve un número de 256 a 511.

Cierre de un fichero

Para cerrar uno o más ficheros abiertos con la instrucción Open, se utilizan las siguientes instrucciones:

- Close. Cierra uno o varios ficheros abiertos, pasando como parámetro los números de fichero.

Sintaxis.

`Close [ListaNumFichero]`

- `ListaNumFichero`. Indica el número o números de fichero que se van a cerrar de la siguiente forma:

`Close #NumFicheroA, #NumFicheroB, ...`

Si no se indica el número de fichero, se cierran todos los ficheros abiertos.

- Reset. Cierra todos los ficheros abiertos, volcando previamente a disco, el contenido de todas las zonas de memoria reservadas a los ficheros.

Modos de acceso a ficheros

En el apartado dedicado a la instrucción Open, hemos visto que hay varias formas de abrir un fichero, en función de como necesitemos escribir o acceder a la información que contiene.

En los siguientes apartados, veremos el resto de instrucciones y funciones relacionadas con la manipulación de ficheros, dependiendo del modo de acceso a efectuar sobre los mismos. Utilizaremos como ejemplo la aplicación [ManipFic](#), y a través de las opciones de menú de su ventana principal mdiInicio, veremos las diferentes operaciones a efectuar según el modo de acceso.

Esta aplicación comienza mediante un procedimiento Main(), en el cual usamos el objeto del sistema App, para establecer como directorio de trabajo el mismo directorio en el que está situada la aplicación, según muestra el Código fuente 229.

```
Public Sub Main()
Dim lmdiInicio As mdiInicio
' establecer como directorio actual
' el directorio donde está situada la
' aplicación
ChDir App.Path
' cargar la ventana principal
Set lmdiInicio = New mdiInicio
Load lmdiInicio
lmdiInicio.Show
End Sub
```

Código fuente 229

Acceso Secuencial

Este es el modo de acceso más sencillo a un fichero. Se utiliza habitualmente para leer o escribir texto en el fichero.

Apertura

Cuando abramos un fichero para este tipo de acceso, utilizaremos alguna de las siguientes palabras clave en la cláusula For de la instrucción Open:

- Input. Abre el fichero para leer la información que contiene desde el principio del mismo.
- Output. Abre el fichero para escribir información. Si dicho fichero ya existe, su contenido se elimina, en el caso de que no exista, se crea un nuevo fichero.
- Append. Abre el fichero para escribir. La nueva información se añade a partir de los últimos datos existentes.

Lectura

Para leer información de un fichero abierto en modo de acceso secuencial, emplearemos las siguientes instrucciones:

- Input #. Lee información de un fichero secuencial asignándola a variables.

Sintaxis.

Input #nNumFichero, [ListaVariables]

- nNumFichero. Número de fichero del que se va a leer.
- ListaVariables. Lista de variables separadas por comas o punto y coma, en las que se irá introduciendo la información leída.

Para comprobar como funciona esta instrucción, seleccionaremos la opción Secuencial + lectura Input del menú de mdiInicio, que ejecutará el Código fuente 230.

```
Private Sub mnuSeLecInput_Click()
Dim lcFichero As String
Dim lcNombreFich As String
Dim lnManip As Integer
Dim lcUna As String
Dim lcDos As String
Dim lcTres As String
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso secuencial y lectura
Open "SecuenLee.txt" For Input As lnManip
' tomar varias líneas del fichero y traspasarlas
' a variables
Input #lnManip, lcUna, lcDos, lcTres
' visualizar las variables
MsgBox "lcUna: " & lcUna, , "Contenido de la variable"
```

```

MsgBox "lcDos: " & lcDos, , "Contenido de la variable"
MsgBox "lcTres: " & lcTres, , "Contenido de la variable"
' cerrar el fichero
Close #lnManip
End Sub

```

Código fuente 230

- Line Input #. Lee una línea de un fichero secuencial y la asigna a una variable. Cualquier carácter usado como delimitador, es incluido en la variable.

Sintaxis.

Line Input #nNumFichero, cVariable

- nNumFichero. Número de fichero del que se va a leer.
- cVariable. Variable de tipo cadena, en la que se va a introducir el valor de la línea del fichero.

Ejecutando la opción del formulario principal Secuencial+Lectura Line Input, leeremos el mismo fichero que en el ejemplo anterior, pero tomando la información por líneas del fichero en vez de por los delimitadores.

El Código fuente 231 muestra como se realiza esta tarea.

```

Private Sub mnuSeLecLineInput_Click()
Dim lcFichero As String
Dim lcNombreFich As String
Dim lnManip As Integer
Dim lcUna As String
Dim lcDos As String
Dim lcTres As String
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso secuencial y lectura
Open "SecuenLee.txt" For Input As lnManip
' tomar varias líneas del fichero y traspasarlas
' a variables
Line Input #lnManip, lcUna
Line Input #lnManip, lcDos
Line Input #lnManip, lcTres
' visualizar las variables
MsgBox "lcUna: " & lcUna, , "Contenido de la variable"
MsgBox "lcDos: " & lcDos, , "Contenido de la variable"
MsgBox "lcTres: " & lcTres, , "Contenido de la variable"
' cerrar el fichero
Close #lnManip
End Sub

```

Código fuente 231

- Input(). Función que toma un número determinado de caracteres de un fichero secuencial y los asigna a una variable. Entre los caracteres leídos se pueden incluir delimitadores, finales de línea y retorno de carro, etc.

Sintaxis.

Input (cNumCaracteres, #nNumFichero)

- nNumCaracteres. Número de caracteres a leer del fichero.
- nNumFichero. Número de fichero del que se va a leer.

La opción Secuencial+Lectura función Input de la aplicación de ejemplo, toma la información del fichero de texto empleado anteriormente. Como podrá observar el lector, el texto es leído según el número de caracteres que se necesiten extraer, sin tener en cuenta delimitadores, finales de línea o cualquier otro elemento, como vemos en el Código fuente 232.

```
Private Sub mnuSeLecFuncInput_Click()
Dim lcFichero As String
Dim lcNombreFich As String
Dim lnManip As Integer
Dim lcUna As String
Dim lcDos As String
Dim lcTres As String
Dim lcCuatro As String
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso secuencial y lectura
Open "SecuenLee.txt" For Input As lnManip
' tomar varias cadenas del fichero y traspasarlas
' a variables
lcUna = Input(14, #lnManip)
lcDos = Input(9, #lnManip)
lcTres = Input(17, #lnManip)
lcCuatro = Input(12, #lnManip)
' visualizar las variables
MsgBox "lcUna: " & lcUna, , "Contenido de la variable"
MsgBox "lcDos: " & lcDos, , "Contenido de la variable"
MsgBox "lcTres: " & lcTres, , "Contenido de la variable"
MsgBox "lcCuatro: " & lcCuatro, , "Contenido de la variable"
' cerrar el fichero
Close #lnManip
End Sub
```

Código fuente 232

Escritura

Para escribir los datos en un fichero secuencial, emplearemos las siguientes instrucciones:

- Print #. Escribe información en un fichero secuencial, la salida de los datos a escribir se realiza de la misma forma que si empleáramos la instrucción Print para mostrar datos en pantalla.

Sintaxis.

Print #nNumFichero, [ListaExpresiones]

- nNumFichero. Número de fichero en el que se va a escribir.

- ListaExpresiones. Expresiones a escribir en el fichero. Cada una de las expresiones puede tener el siguiente formato:

- [Spc(nCaracteres) | Tab [(nColumna)]] [Expresión]
 [cDelimitExpr]
- Spc(nCaracteres). Inserta nCaracteres en blanco al escribir la expresión en el fichero.
 - Tab(nColumna). Sitúa el punto de escritura de la expresión en la columna indicada por nColumna.
 - Expresión. Valor a escribir en el fichero.
 - cDelimitExpr. Carácter, por ejemplo punto y coma ";", para indicar que finaliza la escritura de una expresión y comienza la siguiente.

Veamos un ejemplo de este tipo de escritura en ficheros, seleccionando la opción Secuencial+Escritura Print de la aplicación de ejemplo, que contiene el Código fuente 233.

```
Private Sub mnuSeEscPrint_Click()
Dim lnManip As Integer
Dim lbValor As Boolean
Dim ldtFecha As Date
Dim lxSinValor
Dim lxError
' tomar un manipulador de fichero libre
lnManip = FreeFile
' asignar valores a variables cuyo contenido
' se va a escribir en el fichero
lbValor = True
ldtFecha = CDate("05/09/1998")
lxSinValor = Null
lxError = CVErr(250)
' abrir el fichero para acceso secuencial y escritura
Open "SecuenEsc1.txt" For Output As lnManip
' escribir datos en el fichero
Print #lnManip, "prueba"; "de escritura"; "con PRINT"
Print #lnManip, "prueba "; "de escritura "; "con PRINT"
Print #lnManip,
Print #lnManip, "prueba"; Spc(2); "de escritura"; Spc(4); "con PRINT"
Print #lnManip, Tab(2); "prueba "; Tab(16); "de escritura"; Tab(30); " con
PRINT"
Print #lnManip,
Print #lnManip, lbValor
Print #lnManip, ldtFecha
Print #lnManip, lxSinValor
Print #lnManip, lxError
' cerrar el fichero
Close #lnManip
MsgBox "Finalizada la escritura con Print", , "Acceso Secuencial"
End Sub
```

Código fuente 233

Si abrimos el fichero resultante para visualizar su contenido mediante el Bloc de notas de Windows, podremos observar que la escritura de la información especial tal como valores lógicos, fecha, error, etc., es escrita adaptándose a la configuración local del sistema operativo. En lo que respecta al texto normal, ha de ser el programador el encargado de incluir los espacios o tabuladores necesarios para que el texto a escribir sea legible.

- Write #. Escribe una lista de expresiones a un fichero secuencial.

Sintaxis.

```
Write #nNumFichero, [ListaExpresiones]
```

La lista de expresiones a escribir, irá separada por espacios, coma o punto y coma. La ventaja de esta instrucción es que incluye automáticamente los delimitadores en el fichero e inserta un final de línea al escribir la última expresión de la lista, lo que facilita la posterior lectura del fichero.

La opción Secuencial+Escritura Write, escribe una serie de cadenas de texto en un fichero. Si abrimos el fichero después de esta operación, comprobaremos que en dichas cadenas se ha incluido automáticamente un delimitador, para facilitar su posterior lectura. En el Código fuente 234 podemos ver el código de esta opción.

```
Private Sub mnuSeEscWrite_Click()
Dim lnManip As Integer
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso secuencial y escritura
Open "SecuenEsc2.txt" For Output As lnManip
' escribir datos en el fichero
Write #lnManip, "prueba"; "de escritura"; "con WRITE"
Write #lnManip, "prueba "; "de escritura "; "con WRITE"
Write #lnManip, "escritura del número:"; 512
' cerrar el fichero
Close #lnManip
MsgBox "Finalizada la escritura con Write", , "Acceso Secuencial"
End Sub
```

Código fuente 234

En cuanto a la información especial: fechas, valores lógicos, errores, etc. El tratamiento que Write hace de los mismos, difiere del realizado por Print. Write escribe este tipo de datos según unas normas de formato estándar que no tienen en cuenta la configuración local del sistema.

La opción Secuencial+Escritura Añadir con Write, agrega al fichero utilizado en el anterior ejemplo con Write una serie de líneas con este tipo de información, de manera que podamos ver la diferencia entre la escritura con Print y Write, además de comprobar como se agregan datos a un fichero ya existente, manteniendo la información que hubiera en el mismo.

El fuente que aparece en el Código fuente 235 pertenece a esta opción.

```
Private Sub mnuSeEscAdd_Click()
Dim lnManip As Integer
Dim lbValor As Boolean
Dim ldtFecha As Date
Dim lxSinValor
Dim lxError
' tomar un manipulador de fichero libre
lnManip = FreeFile
' asignar valores a variables cuyo contenido
' se va a escribir en el fichero
```

```

lbValor = True
ldtFecha = CDate("05/09/1998")
lxSinValor = Null
lxError = CVErr(250)
' abrir el fichero para acceso secuencial y escritura,
' la información a escribir se añadirá a la ya existente
Open "SecuenEsc2.txt" For Append As lnManip
' escribir datos en el fichero
Write #lnManip,
Write #lnManip, lbValor
Write #lnManip, ldtFecha
Write #lnManip,
Write #lnManip, lxSinValor
Write #lnManip, lxError
' cerrar el fichero
Close #lnManip
MsgBox "Finalizada la escritura con Write (añadir)", , "Acceso Secuencial"
End Sub

```

Código fuente 235

Acceso Aleatorio

En un fichero diseñado para acceso aleatorio, la información está organizada en registros y estos a su vez en campos.

Apertura

Para abrir un fichero con acceso aleatorio, emplearemos la palabra clave Random en la cláusula For de la instrucción Open, y especificaremos la longitud del registro mediante la cláusula Len de la misma instrucción.

Escritura

- Put. Esta instrucción escribe un registro en un fichero abierto para acceso aleatorio. Si se intentan escribir más datos para un registro, que los especificados en la cláusula Len de la instrucción Open, se producirá un error.

Sintaxis.

Put #nNumFichero, [nNumRegistro], cVariable

- nNumFichero. Número de fichero en el que se va a escribir.
- nNumRegistro. Número de registro a grabar. Si se omite, se escribe el siguiente registro que corresponda después de haber ejecutado la última instrucción Get o Put.
- cVariable. Nombre de la variable que contiene los datos a escribir en el fichero.

Para manejar registros, la mejor manera es emplear tipos definidos por el usuario. En los ejemplos de acceso aleatorio, se ha definido a nivel de módulo el tipo Ficha, cuyo fuente vemos en el Código fuente 236.

```

Private Type Ficha
    Codigo As String * 2
    Nombre As String * 10
    Ciudad As String * 15
End Type

```

Código fuente 236

La opción de menú Aleatorio+Escritura de los ejemplos, nos muestra como escribir registros en un fichero utilizando esta instrucción y una variable del tipo Ficha.

```

Private Sub mnuAlEscritura_Click()
Dim lnManip As Integer
Dim lFicha As Ficha
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso aleatorio
Open "Aleatorio.txt" For Random As lnManip Len = Len(lFicha)
' escribir registros en el fichero
lFicha.Codigo = "10"
lFicha.Nombre = "Pedro"
lFicha.Ciudad = "Burgos"
Put #lnManip, 1, lFicha
lFicha.Codigo = "20"
lFicha.Nombre = "Antonio"
lFicha.Ciudad = "Salamanca"
Put #lnManip, 2, lFicha
lFicha.Codigo = "40"
lFicha.Nombre = "Raquel"
lFicha.Ciudad = "Ávila"
Put #lnManip, 3, lFicha
lFicha.Codigo = "45"
lFicha.Nombre = "Ana"
lFicha.Ciudad = "Barcelona"
Put #lnManip, 4, lFicha
lFicha.Codigo = "80"
lFicha.Nombre = "Verónica"
lFicha.Ciudad = "Madrid"
Put #lnManip, 5, lFicha
lFicha.Codigo = "82"
lFicha.Nombre = "Luis"
lFicha.Ciudad = "Valencia"
Put #lnManip, 6, lFicha
' cerrar el fichero
Close #lnManip
MsgBox "Finalizada la escritura con Put", , "Acceso Aleatorio"
End Sub

```

Código fuente 237

Lectura

- Get. Lee un registro de un archivo abierto en modo aleatorio, almacenando el registro en una variable.

Sintaxis.

```
Get #nNumFichero, nNumRegistro,cVariable
```

- nNumFichero. Número de fichero del que se va a leer.
- nNumRegistro. Número de registro a leer. Si se omite, se lee el siguiente registro que corresponda después de haber ejecutado la última instrucción Get o Put.
- cVariable. Nombre de la variable en la que se va a situar el registro leído.

La opción Aleatorio+Lectura de la aplicación de ejemplo, ilustra la forma de recorrer los registros del fichero grabado en el ejemplo anterior y mostrarlos al usuario.

```
Private Sub mnuAlLectura_Click()
Dim lcFichero As String
Dim lcNombreFich As String
Dim lnManip As Integer
Dim lFicha As Ficha
Dim lnRegistro As Integer
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso aleatorio
Open "Aleatorio.txt" For Random As lnManip Len = Len(lFicha)
' inicializar contador de registros
lnRegistro = 1
' recorrer el fichero hasta el final
Do While Not EOF(lnManip)
    ' tomar un registro del fichero
    Get #lnManip, lnRegistro, lFicha
    ' visualizar el registro
    MsgBox "Código: " & lFicha.Codigo & vbCrLf & _
        "Nombre: " & lFicha.Nombre & vbCrLf & _
        "Ciudad: " & lFicha.Ciudad, , "Contenido del registro"
    lnRegistro = lnRegistro + 1
Loop
' cerrar el fichero
Close #lnManip
End Sub
```

Código fuente 238

La función EOF(nNumFichero) utilizada aquí, sirve para saber cuando hemos llegado al final del fichero, cuyo número es pasado como parámetro, evitando que se produzca un error al intentar leer datos pasada la última posición del fichero. Esta función, devolverá False mientras se pueda leer información del fichero, y True una vez que se sobrepase la última posición del mismo.

Acceso Binario

El acceso binario a un fichero, nos permite manipular la información del mismo byte a byte, o dicho de otro modo carácter a carácter, sin tener en cuenta si la información del fichero está grabada bajo una determinada estructura de registros o no.

Apertura

Para abrir un fichero que disponga de acceso binario, emplearemos la palabra clave Binary en la cláusula For de la instrucción Open.

Escritura

Escribiremos información en un fichero abierto en modo binario mediante la instrucción Put, vista anteriormente al tratar el acceso aleatorio. En el caso de acceso binario, el parámetro nNumRegistro hará referencia a la posición o byte del fichero en la que se comienza a escribir.

La opción Binario+Escritura de la aplicación de ejemplo, nos muestra como escribir en un fichero, usando este tipo de acceso. En el Código fuente 239 vemos el fuente de este punto.

```
Private Sub mnuBiEscritura_Click()
Dim lnManip As Integer
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso binario
Open "Binario.txt" For Binary As lnManip
' escribir información en el fichero
Put #lnManip, , "En ese parque hay diez árboles"
' cerrar el fichero
Close #lnManip
MsgBox "Finalizada la escritura con Put", , "Acceso Binario"
End Sub
```

Código fuente 239

Lectura

Leeremos los datos de un fichero abierto en modo binario mediante la instrucción Get, vista en el modo de acceso aleatorio. En acceso binario, el parámetro nNumRegistro hará referencia al número de bytes a leer del fichero.

En el Código fuente 240 podemos ver el código perteneciente a la opción Binario + lectura de la aplicación de ejemplo. En primer lugar, leemos una parte de la cadena escrita en el anterior ejemplo, para mostrarla al usuario y posteriormente sobreescrivimos en el fichero dicha cadena por otra distinta, volviendo a mostrar el resultado.

```
Private Sub mnuBiLectura_Click()
Dim lnManip As Integer
Dim lcCadena As String * 4
' tomar un manipulador de fichero libre
lnManip = FreeFile
' abrir el fichero para acceso binario
Open "Binario.txt" For Binary As lnManip
' leer varios caracteres y pasarlos
' a una variable
Get #lnManip, 19, lcCadena
' mostrar el resultado
MsgBox "Valor leído del fichero: " & lcCadena, , "Acceso Binario"
' grabar una cadena distinta
```

```

' en la misma posición de la que se ha leído
Put #lnManip, 19, " 10 "
' volver a efectuar la operación de lectura
Get #lnManip, 19, lcCadena
MsgBox "Nuevo valor leído de la misma posición fichero: " _
    & lcCadena, , "Acceso Binario"
' cerrar el fichero
Close #lnManip
End Sub

```

Código fuente 240

Otras instrucciones para manejo de ficheros

A continuación se relaciona un grupo adicional de instrucciones y funciones que también se emplean en las operaciones con fichero.

- **Seek().** Función que devuelve la posición actual del fichero utilizado como parámetro. Si es un fichero abierto para acceso aleatorio, esta función devuelve el número de registro en el que está posicionado el fichero, para otros modos de acceso, devuelve el byte actual.

Sintaxis.

Seek (nNumFichero)

- **nNumFichero.** Número de fichero del que se averigua la posición.

- **Seek.** Esta instrucción establece la posición del fichero sobre la que se va a realizar la siguiente operación de lectura o escritura.

Sintaxis.

Seek #nNumFichero, nPosicion

- **nNumFichero.** Número de fichero en el que se establece la posición.
- **nPosicion.** Posición del fichero a establecer.

Si después de usar esta instrucción, utilizamos Get o Put especificando una posición en el fichero, esta última prevalecerá sobre la indicada en Seek.

- **Loc().** Función que informa sobre la posición actual del fichero

Sintaxis.

Loc (nNumFichero)

- **nNumFichero.** Número de fichero del que se obtiene la posición.

Si el fichero está en modo binario, se obtiene la posición del último byte utilizado; si está en modo aleatorio, el último registro; finalmente si el acceso es secuencial, devuelve la posición del último byte dividida por 128.

- **LOF().** Devuelve el tamaño de un fichero abierto.

Sintaxis.

`LOF (nNumFichero)`

- `nNumFichero`. Número de fichero del que obtendremos su tamaño.
- `FileLen()`. Función que devuelve el tamaño de un fichero.

Sintaxis.

`FileLen (cNombreFich)`

- `cNombreFich`. Nombre del fichero y ruta opcional, del que vamos a obtener su tamaño.

La diferencia entre esta función y `LOF()` reside en que utilizando `FileLen()` no es necesario que el fichero esté abierto para conocer su tamaño.

- `Kill`. Borra ficheros del disco.

Sintaxis.

`Kill cNombreFich`

- `cNombreFich`. Nombre del fichero/s y ruta opcional, que vamos a eliminar, es posible emplear los caracteres comodín "*" y "?" para borrar grupos de ficheros.

Hemos de asegurarnos de que el fichero/s a eliminar están cerrados o de lo contrario ocurrirá un error.

- `Name`. Esta instrucción permite cambiar de nombre un fichero o directorio.

Sintaxis.

`Name cNombreFichActual As cNombreFichNuevo`

- `cNombreFichActual`. Nombre del fichero existente al que vamos a cambiar de nombre, opcionalmente puede incluir la ruta de acceso.
- `cNombreFichNuevo`. Nombre del nuevo fichero y ruta opcional. No se puede utilizar el nombre de un fichero ya existente.

Si el nombre de fichero empleado tanto en `cNombreFichActual` como en `cNombreFichNuevo` es igual, y las rutas de acceso son distintas, el fichero se cambiará de posición.

Las siguientes instrucciones, si bien no operan directamente con ficheros, son necesarias en cuanto a la organización de los mismos.

- `FileCopy`. Copia un fichero

Sintaxis.

`FileCopy cNombreFichOrigen, cNombreFichDestino`

- `cNombreFichOrigen`. Cadena con el nombre y ruta opcional del fichero que se va a copiar.

- `cNombreFichDestino`. Cadena con el nombre y ruta opcional del fichero en el que se va a efectuar la copia.

Si se intenta utilizar esta instrucción sobre un fichero abierto, se provocará un error.

- `FileDateTime`. Esta función retorna un valor de tipo fecha, que informa de la fecha y hora de creación de un fichero.

Sintaxis.

`FileDateTime (cNombreFich)`

- `cNombreFich`. Cadena con el nombre y ruta opcional del fichero.

- `FileAttr`. Esta función devuelve información sobre un fichero abierto con `Open`.

Sintaxis.

`FileAttr (nNumFichero, nTipoInfo)`

- `nNumFichero`. Número de fichero abierto.
- `nTipoInfo`. Número que especifica el tipo de información, si es 1, se devolverá el modo del fichero, si es 2, se devolverá información sobre el selector de archivos del sistema operativo. Cuando se utiliza 1 en este parámetro, los valores que podemos obtener son los siguientes:

- 1. Entrada.
- 2. Salida.
- 4. Aleatorio.
- 8. Añadir.
- 32. Binario.

- `GetAttr`. Devuelve un valor que identifica los atributos de un fichero, directorio o volumen.

Sintaxis.

`GetAttr (cNombreFich)`

- `cNombreFich`. Cadena con el nombre y ruta opcional del fichero.

El valor devuelto por esta función puede ser uno de los mostrados en la Tabla 30.

Valor	Constante	Descripción
0	<code>vbNormal</code>	Normal.
1	<code>vbReadOnly</code>	Sólo lectura.
2	<code>vbHidden</code>	Oculto.

4	vbSystem	Archivo de sistema.
16	vbDirectory	Directorio o carpeta.
32	VbArchive	El archivo ha sido modificado después de efectuar la última copia de seguridad.

Tabla 30. Valores devueltos por la función GetAttr().

- SetAttr. Establece los atributos de un fichero.

Sintaxis.

`SetAttr(cNombreFich, nAtributos)`

- `cNombreFich`. Cadena con el nombre y ruta opcional del fichero.
- `nAtributos`. Uno o más valores numéricos o constantes que indica los atributos a establecer. Dichos valores son los mismos que devuelve la función GetAttr() excepto `vbDirectory`.

El atributo se debe establecer con el fichero cerrado, en caso contrario, se producirá un error.

- MkDir. Crea un nuevo directorio.

Sintaxis.

`MkDir cNombreDir`

- `cNombreDir`. Cadena con el nombre del directorio a crear. Si no se indica la unidad de disco, el directorio se crea en la unidad actual.

- RmDir. Borra un directorio existente.

Sintaxis.

`RmDir cNombreDir`

- `cNombreDir`. Cadena con el nombre del directorio a borrar. Si no se indica la unidad de disco, se borra el directorio de la unidad actual.

Si el directorio que se desea borrar contiene ficheros, se producirá un error. Se deben eliminar previamente los ficheros del directorio antes de proceder a borrarlo.

- CurDir. Devuelve una cadena con la ruta de acceso actual.

Sintaxis.

`CurDir [(cUnidad)]`

- cUnidad. Cadena con la unidad de disco de la que se desea saber la ruta de acceso, si no se indica, se devuelve la ruta de la unidad actual.
- ChDir. Cambia el directorio actual.

Sintaxis.

ChDir cNombreDir

- cNombreDir. Cadena con el nuevo directorio, puede incluir también la unidad de disco, sin embargo, la unidad predeterminada no cambia aunque se cambie el directorio en una unidad diferente.

- ChDrive. Cambia la unidad de disco actual.

Sintaxis.

ChDrive cUnidad

- cUnidad. Cadena con la nueva unidad.

Objetos para el manejo de ficheros

Visual Basic 6 incorpora una interesante y seguramente esperada novedad para la manipulación de ficheros, consistente en un modelo de objetos denominado File System Objects (Objetos del sistema de ficheros), FSO a partir de ahora. Decimos esperada novedad, puesto que ya que esta es una herramienta que progresivamente ha incorporado la tecnología de objetos para todas las tareas a realizar, en este aspecto del manejo de archivos sólo disponíamos del tradicional conjunto de instrucciones para la apertura, lectura y escritura de información en ficheros, echándose en falta cada vez más una jerarquía de clases para realizar este trabajo.

Gracias a estos objetos, podremos realizar las operaciones habituales de lectura y escritura en modo secuencial sobre un fichero (para acceso aleatorio y binario debemos seguir empleando la instrucción Open), pero no sólo disponemos de objetos para la manipulación de ficheros, esta jerarquía nos permite acceder también a las carpetas y unidades instaladas en nuestro equipo.

La aplicación de ejemplo para este apartado, [ManipFSO](#), muestra al igual que el anterior ejemplo, a través del formulario principal mdiInicio, diversas operaciones efectuadas con los objetos de este modelo, de los que seguidamente veremos las propiedades y métodos principales.

Drive

Facilita información sobre una unidad instalada en el equipo: disco duro, CD ROM, disco RAM, etc.

Propiedades

- AvailableSpace. Informa sobre la cantidad de espacio disponible en la unidad.
- DriveLetter. Devuelve la letra identificativa de la unidad.
- DriveType. Constante que indica el tipo de unidad instalada según la Tabla 31.

Constante	Descripción
CDRom	CD-ROM
Fixed	Disco Duro
RamDisk	Disco RAM o Disco en memoria
Remote	Disco de Red
Removable	Removable
Unknown	Desconocido

Tabla 31. Constantes para la propiedad DriveType.

- FileSystem. Sistema de archivos empleado por la unidad: FAT, NTFS, etc.
- SerialNumber. Número de serie de la unidad.
- TotalSize. Tamaño total que tiene la unidad.
- VolumeName. Establece o devuelve el nombre de volumen para la unidad.

La opción de menú Drive+Mostrar información, en el programa, ejecuta las líneas del Código fuente 241, que toman la unidad C: instalada en el equipo del usuario y visualizan algunas de sus propiedades.

```

Private Sub mnuDrMostrar_Click()
Dim lfsOfichero As FileSystemObject
Dim ldrvUnidad As Drive
Dim lcInfo As String
Dim lcTipoUnidad As String
' crear objeto FSO
Set lfsOfichero = New FileSystemObject
' obtener objeto Drive
Set ldrvUnidad = lfsOfichero.GetDrive("c:")
' mostrar propiedades del objeto
' comprobar el tipo de unidad
Select Case ldrvUnidad.DriveType
Case CDRom
    lcTipoUnidad = "CDRom"

Case Fixed
    lcTipoUnidad = "Disco Duro"
Case RamDisk
    lcTipoUnidad = "Disco en memoria"
Case Removable
    lcTipoUnidad = "Removable"
Case Remote
    lcTipoUnidad = "Red"
Case Unknown
    lcTipoUnidad = "Desconocida"
End Select
lcInfo = "Unidad: " & ldrvUnidad.DriveLetter & vbCrLf
lcInfo = lcInfo & "Tipo: " & lcTipoUnidad & vbCrLf

```

```

lcInfo = lcInfo & "Volumen: " & ldrvUnidad.VolumeName & vbCrLf
lcInfo = lcInfo & "Tamaño: " &
    FormatNumber(ldrvUnidad.TotalSize, 0) & vbCrLf
lcInfo = lcInfo & "Espacio disponible: " &
    FormatNumber(ldrvUnidad.AvailableSpace, 0) & vbCrLf
lcInfo = lcInfo & "Sistema de ficheros: " & ldrvUnidad.FileSystem & vbCrLf
MsgBox lcInfo, , "Información del objeto Drive"
End Sub

```

Código fuente 241

Folder

Proporciona información sobre una carpeta que reside en una unidad.

Propiedades

- Attributes. Devuelve los atributos establecidos en una carpeta o fichero. La Tabla 32 muestra las constantes disponibles para los atributos.

Constante	Valor	Descripción
Normal	0	Archivo normal. No se establecen atributos.
ReadOnly	1	Archivo de sólo lectura. El atributo es de lectura o escritura.
Hidden	2	Archivo oculto. El atributo es de lectura o escritura.
System	4	Archivo del sistema. El atributo es de lectura o escritura.
Volume	8	Etiqueta del volumen de la unidad de disco. El atributo es de sólo lectura.
Directory	16	Carpeta o directorio. El atributo es de sólo lectura.
Archive	32	El archivo cambió desde la última copia de seguridad. El atributo es de lectura o escritura.
Alias	64	Vínculo o método abreviado. El atributo es de sólo lectura.
Compressed	128	Archivo comprimido. El atributo es de sólo lectura.

Tabla 32. Constantes para la propiedad Attributes en carpetas y ficheros.

- DateCreated. Devuelve la fecha y hora en que se creó la carpeta o fichero.
- DateLastAccessed. Devuelve la fecha y hora en que se efectuó el último acceso a la carpeta o fichero.
- Drive. Devuelve la letra de la unidad a la que pertenece la carpeta o fichero.
- Files. Colección de objetos File que pertenecen a la carpeta actual.
- Name. Devuelve o establece el nombre de la carpeta o fichero.
- Size. Devuelve el tamaño de la carpeta (suma de los ficheros y subcarpetas contenidos en la carpeta utilizada) o fichero.
- SubFolders. Colección de objetos Folder contenidos en la carpeta actual.
- ShortName. Nombre de la carpeta o fichero en el formato 8+3 caracteres.
- Type. Cadena con una descripción de la carpeta o fichero.

Métodos

- Copy(). Copia una carpeta o fichero a una ubicación diferente dentro de la misma o de otra unidad.

Sintaxis.

```
Copy(cDestino [, bSobrescribir])
```

- cDestino. Cadena con la nueva ubicación.
- bSobrescribir. Opcional. Valor lógico que será True (predeterminado), para sobrescribir la carpeta en el caso de que exista en cDestino, y False cuando no se sobrescriba.

- Delete(). Borra una carpeta o fichero.

Sintaxis.

```
Delete([bForzar])
```

- bForzar. Opcional. Valor lógico que al ser True borrará las carpetas o ficheros con el atributo de sólo lectura. Si es False (predeterminado) no se borrará este tipo de carpetas o ficheros.

- Move(). Cambia la ubicación de una carpeta o fichero.

Sintaxis.

```
Move(cDestino)
```

- cDestino. Cadena con la nueva ubicación para la carpeta o fichero.

Debemos tener la precaución de poner el carácter indicador de carpeta en el destino para evitar errores, tal como muestra la siguiente línea, en las que un objeto Folder se mueve de carpeta.

```
oFolder.Move "c:\correo\"
```

Las diferentes opciones del menú Folder, en la aplicación de ejemplo, muestran información sobre el nombre de una carpeta introducida por el usuario, efectuando también diversas acciones. El código de tales opciones se muestra en el Código fuente 242.

```
Private Sub mnuFoMostrar_Click()
Dim lcNombreCarpeta As String
Dim lcInfo As String
Dim lfs0Fichero As FileSystemObject
Dim lfolCarpeta As Folder
Dim lfolSubCarpeta As Folder
Dim lfilFichero As File
' solicitar datos al usuario
lcNombreCarpeta = InputBox("Nombre de la carpeta", _
    "Mostrar propiedades de una carpeta")
' crear objeto FSO
Set lfs0Fichero = New FileSystemObject
' obtener objeto carpeta
Set lfolCarpeta = lfs0Fichero.GetFolder(lcNombreCarpeta)
' mostrar propiedades de la carpeta
lcInfo = "Nombre de carpeta: " & lfolCarpeta.Name & vbCrLf
lcInfo = lcInfo & "Nombre en formato 8+3: " & lfolCarpeta.ShortName & vbCrLf
lcInfo = lcInfo & "Tipo: " & lfolCarpeta.Type & vbCrLf
lcInfo = lcInfo & "Tamaño: " & FormatNumber(lfolCarpeta.Size, 0) & vbCrLf
lcInfo = lcInfo & "Creada el " & _
    FormatDateTime(lfolCarpeta.DateCreated, vbShortDate) & vbCrLf
lcInfo = lcInfo & "Usada por última vez el " & _
    FormatDateTime(lfolCarpeta.DateLastAccessed, vbShortDate) & vbCrLf
MsgBox lcInfo, , "Propiedades de la carpeta"
' recorrer la colección Files de la
' carpeta mostrando información sobre
' cada objeto fichero
If lfolCarpeta.Files.Count > 0 Then
    For Each lfilFichero In lfolCarpeta.Files
        lcInfo = "Nombre: " & lfilFichero.Name & vbCrLf
        lcInfo = lcInfo & "Tamaño: " & FormatNumber(lfilFichero.Size, 0) & vbCrLf
        lcInfo = lcInfo & "Ruta: " & lfilFichero.Path & vbCrLf
        lcInfo = lcInfo & "Nombre corto: " & lfilFichero.ShortName & vbCrLf

        MsgBox lcInfo, , "Fichero de la carpeta " & lfolCarpeta.Name
    Next
End If
' recorrer la colección SubFolders de la carpeta
' mostrando el nombre de cada SubCarpeta
If lfolCarpeta.SubFolders.Count > 0 Then
    For Each lfolSubCarpeta In lfolCarpeta.SubFolders
        MsgBox "SubCarpeta: " & lfolSubCarpeta.Name
    Next
End If
End Sub
*****
```

```
Private Sub mnuFoCopiar_Click()
Dim lcNombreCarpeta As String
Dim lcCopiarEnCarpeta As String
Dim lfs0Fichero As FileSystemObject
Dim lfolCarpeta As Folder
' obtener datos del usuario
lcNombreCarpeta = InputBox("Nombre de la carpeta origen", _
    "Copiar carpeta")
lcCopiarEnCarpeta = InputBox("Nombre de la carpeta destino", _
    "Copiar carpeta")
```

```

' crear objeto FSO
Set lfsоФichero = New FileSystemObject
' obtener objeto carpeta
Set lfolCarpeta = lfsоФichero.GetFolder(lcNombreCarpeta)

' copiar el contenido de la carpeta origen
' en la carpeta destino
lfolCarpeta.Copy lcCopiarEnCarpeta
End Sub
*****
Private Sub mnuFoBorrar_Click()
Dim lcNombreCarpeta As String
Dim lfsоФichero As FileSystemObject
Dim lfolCarpeta As Folder
' obtener datos del usuario
lcNombreCarpeta = InputBox("Nombre de la carpeta a borrar", _
    "Borrar carpeta")
' crear objeto FSO
Set lfsоФichero = New FileSystemObject
' obtener objeto carpeta
Set lfolCarpeta = lfsоФichero.GetFolder(lcNombreCarpeta)

' borrar la carpeta
lfolCarpeta.Delete True
End Sub
*****
Private Sub mnuFoMover_Click()
Dim lcNombreCarpeta As String
Dim lcMoverA As String
Dim lfsоФichero As FileSystemObject
Dim lfolCarpeta As Folder
' obtener datos del usuario
lcNombreCarpeta = InputBox("Nombre de la carpeta", _
    "Mover carpeta")
lcMoverA = InputBox("Nueva ubicación para la carpeta", "Mover carpeta")

' crear objeto FSO
Set lfsоФichero = New FileSystemObject
' obtener objeto carpeta
Set lfolCarpeta = lfsоФichero.GetFolder(lcNombreCarpeta)

' mover contenido de carpeta
lfolCarpeta.Move lcMoverA
End Sub

```

Código fuente 242

File

Realiza operaciones con archivos relacionadas con la administración de los mismos, proporcionando también información acerca de sus características. No crea ni modifica el contenido de ficheros, ya que para eso disponemos del objeto TextStream que veremos más adelante. Consulte el lector las propiedades y métodos del anterior objeto Folder, ya que son también aplicables a los objetos File.

En el Código fuente 243 se muestran las líneas de código de las opciones contenidas en el menú File del ejemplo, que realizan diferentes operaciones con este tipo de objetos.

```

Private Sub mnuFiMostrar_Click()
Dim lfsоФichero As FileSystemObject
Dim lfilFichero As File

```

```

Dim lcNombreFichero As String
Dim lcInfo As String
' obtener datos del usuario
lcNombreFichero = InputBox("Nombre del fichero", _
    "Introducir el nombre del fichero")
' crear objeto FSO
Set lfsoFichero = New FileSystemObject
' obtener objeto fichero
Set lfilFichero = lfsoFichero.GetFile(lcNombreFichero)
' mostrar propiedades del objeto fichero
lcInfo = "Nombre: " & lfilFichero.Name & vbCrLf
lcInfo = lcInfo & "Tipo: " & lfilFichero.Type & vbCrLf
lcInfo = lcInfo & "Creado: " & _
    FormatDateTime(lfilFichero.DateCreated, vbShortDate) & vbCrLf
lcInfo = lcInfo & "Tamaño: " & FormatNumber(lfilFichero.Size, 0) & vbCrLf
lcInfo = lcInfo & "Ubicación: " & lfilFichero.ParentFolder & vbCrLf
lcInfo = lcInfo & "Ruta: " & lfilFichero.Path
MsgBox lcInfo, , "Información del objeto File"
End Sub
*****
Private Sub mnuFiCopiar_Click()
Dim lcFicheroOrigen As String
Dim lcFicheroDestino As String
Dim lfsoFichero As FileSystemObject
Dim lfilFichero As File
' obtener datos del usuario
lcFicheroOrigen = InputBox("Fichero origen", _
    "Copiar fichero")
lcFicheroDestino = InputBox("Fichero destino", _
    "Copiar fichero")

' crear objeto FSO
Set lfsoFichero = New FileSystemObject
' obtener objeto File
Set lfilFichero = lfsoFichero.GetFile(lcFicheroOrigen)

' copiar fichero en carpeta destino
lfilFichero.Copy lcFicheroDestino
End Sub
*****
Private Sub mnuFiBorrar_Click()
Dim lcNombreFichero As String
Dim lfsoFichero As FileSystemObject
Dim lfilFichero As File
' obtener datos del usuario
lcNombreFichero = InputBox("Nombre del fichero", "Borrar fichero")
' crear objeto FSO
Set lfsoFichero = New FileSystemObject
' obtener objeto File
Set lfilFichero = lfsoFichero.GetFile(lcNombreFichero)
' borrarlo
lfilFichero.Delete
End Sub
*****
Private Sub mnuFiMover_Click()
Dim lcFicheroOrigen As String
Dim lcFicheroDestino As String
Dim lfsoFichero As FileSystemObject
Dim lfilFichero As File
' obtener datos del usuario
lcFicheroOrigen = InputBox("Nombre del fichero a mover", _
    "Mover fichero")

lcFicheroDestino = InputBox("Destino del fichero a mover", _
    "Mover fichero")

' crear objeto FSO

```

```

Set lfsoFichero = New FileSystemObject
' obtener objeto fichero
Set lfilFichero = lfsoFichero.GetFile(lcFicheroOrigen)

' mover el fichero especificado
lfilFichero.Move lcFicheroDestino
End Sub

```

Código fuente 243

FileSystemObject

Objeto principal del modelo. Facilita la creación y manipulación de ficheros y carpetas; igualmente proporciona objetos File y Folder para obtener información de sus propiedades.

Propiedades

- Drives. Colección de objetos Drive conectados al equipo.

Métodos

- BuildPath(). Crea una ruta para archivos.

Sintaxis.

```
BuildPath(cRuta, cNombre)
```

- cRuta. Ruta de archivos existente.
- cNombre. Nombre que combinado con cRuta, creará la nueva ruta.

- GetTempName(). Devuelve una cadena con un nombre generado aleatoriamente, que puede utilizarse en conjunción con el método CreateTextFile() para realizar operaciones con ficheros temporales.

```
lcNombre = loFSO.GetTempName      ' radC8BOF.tmp
```

- GetAbsolutePathName(). Devuelve una cadena con una ruta absoluta, creada en base a una ruta pasada como parámetro

Sintaxis.

```
GetAbsolutePathName(cRuta)
```

- cRuta. Cadena con una ruta de ficheros.

- DriveExists(). Devuelve un valor lógico que indica si la unidad pasada como parámetro existe (True) o no (False).

Sintaxis.

```
DriveExists(cUnidad)
```

- cUnidad. Cadena de caracteres con la especificación de la unidad.
- GetDrive(). Retorna un objeto Drive si coincide con la especificación de unidad pasada como parámetro.

Sintaxis.

`GetDrive(cUnidad)`

- cUnidad. Cadena de caracteres con la especificación de la unidad.
- GetDriveName(). Devuelve una cadena con el nombre de la unidad en función de la ruta pasada como parámetro.

Sintaxis.

`GetDriveName(cRuta)`

- cRuta. Cadena con la ruta de la que se va a devolver el nombre de la unidad.
- CopyFolder(). Copia una carpeta en una nueva ruta.

Sintaxis.

`CopyFolder(cOrigen, cDestino [, bSobrescribir])`

- cOrigen. Cadena con la carpeta que se va a copiar, puede incluir caracteres comodín para copiar varias carpetas que estén contenidas en el origen.
- cDestino. Cadena con la carpeta en la que se va a realizar la copia.
- bSobrescribir. Opcional. Valor lógico que indica si las carpetas que ya existan en el destino se van a sobrescribir (True) o no (False).
- CreateFolder(). Crea una nueva carpeta

Sintaxis.

`CreateFolder(cCarpeta)`

- cCarpeta. Cadena con el nombre de la carpeta a crear.
- DeleteFolder(). Borra una carpeta.

Sintaxis.

`DeleteFolder(cCarpeta [, bForzar])`

- cCarpeta. Cadena con el nombre de la carpeta a borrar.
- bForzar. Opcional. Valor lógico para indicar que se borrarán las carpetas que tengan el atributo de lectura (True) o no (False).
- FolderExists(). Devuelve un valor lógico True en el caso de que exista la carpeta pasada como parámetro, y False si no existe.

Sintaxis.

`FolderExists (cCarpeta)`

- `cCarpeta`. Cadena con el nombre de la carpeta que vamos a comprobar.
- `GetFolder()`. Devuelve un objeto `Folder` correspondiente al nombre de la carpeta pasada como parámetro. Para mayor información sobre objetos `Folder`, consulte el lector el punto dedicado a ellos en este apartado.

Sintaxis.

`GetFolder (cCarpeta)`

- `cCarpeta`. Cadena con el nombre de la carpeta a recuperar.
- `GetParentFolderName()`. Devuelve el nombre de la carpeta de nivel superior de la ruta pasada como parámetro

Sintaxis.

`GetParentFolderName (cRuta)`

- `cRuta`. Cadena con la ruta de la que vamos a averiguar la carpeta de nivel superior.
- `GetSpecialFolder()`. Devuelve una carpeta especial del sistema, según una de las constantes pasada como parámetro, que podemos ver en la Tabla 33.

Constante	Valor	Descripción
<code>WindowsFolder</code>	0	La carpeta Windows contiene archivos instalados por el sistema operativo Windows.
<code>SystemFolder</code>	1	La carpeta Sistema contiene bibliotecas, fuentes y unidades de dispositivo.
<code>TemporaryFolder</code>	2	La carpeta Temp se utiliza para almacenar archivos temporales. Su ruta se encuentra en la variable del entorno TMP.

Tabla 33. Constantes para obtener una carpeta especial.

Sintaxis.

`GetSpecialFolder (kCarpeta)`

- `kCarpeta`. Constante con el tipo de carpeta a recuperar.
- `MoveFolder()`. Mueve una o varias carpetas a una nueva ubicación.

Sintaxis.

`MoveFolder(cOrigen, cDestino)`

- `cOrigen`. Cadena con la carpeta que se va a mover, puede incluir caracteres comodín para mover varias carpetas que estén contenidas en el origen.
- `cDestino`. Cadena con la carpeta en la que se va a depositar la carpeta origen.
- `CopyFile()`. Copia uno o más ficheros en una nueva ubicación.

Sintaxis.

`CopyFile(cOrigen, cDestino [, bSobrescribir])`

- `cOrigen`. Cadena con el fichero que se va a copiar, es posible emplear caracteres comodín.
- `cDestino`. Cadena con la ruta en la que se va a realizar la copia.
- `bSobrescribir`. Opcional. Valor lógico que indica si los ficheros que ya existan en el destino se van a sobrescribir (True) o no (False).
- `CreateTextFile()`. Crea un nuevo fichero.

Sintaxis.

`CreateTextFile(cFichero [, bSobrescribir [, bUnicode]])`

- `cFichero`. Cadena con el nombre del fichero a crear.
- `bSobrescribir`. Opcional. Valor lógico que indica si los ficheros existentes se pueden sobrescribir (True) o no (False).
- `bUnicode`. Opcional. Valor lógico que indica si el fichero se creará de tipo Unicode (True) o ASCII (False).
- `DeleteFile()`. Borra un fichero.

Sintaxis.

`DeleteFile(cFichero [, bForzar])`

- `cFichero`. Nombre del fichero a borrar, pudiendo emplear caracteres comodín.
- `bForzar`. Opcional. Valor lógico para indicar que se borrarán los ficheros que tengan el atributo de lectura (True) o no (False).
- `FileExists()`. Devuelve un valor lógico True en el caso de que exista el archivo pasado como parámetro, y False si no existe.

Sintaxis.

`File(cArchivo)`

- `cArchivo`. Cadena con el nombre del archivo a comprobar.

- **GetFile()**. Devuelve un objeto File correspondiente a la ruta pasada como parámetro. Para mayor información sobre objetos File, consulte el lector el punto dedicado a ellos en este apartado.

Sintaxis.

`GetFile(cRuta)`

- **cRuta**. Cadena con la ruta y nombre del fichero a recuperar.

- **MoveFile()**. Mueve uno o varios archivos a una nueva ubicación.

Sintaxis.

`MoveFile(cOrigen, cDestino)`

- **cOrigen**. Cadena con la ruta de los archivos a mover, puede incluir caracteres comodín para mover varios archivos.
- **cDestino**. Cadena con la ruta en la que se van a depositar los archivos.

- **OpenTextFile()**. Abre el fichero pasado como parámetro, devolviendo un objeto TextStream para realizar operaciones de lectura y escritura en dicho fichero.

Sintaxis.

`OpenTextFile(cFichero [, kModoAcceso [, bCrear [, kFormato]]])`

- **cFichero**. Cadena con el nombre del fichero a abrir.
- **kModoAcceso**. Opcional. Una de las constantes que aparecen en la Tabla 34 para el modo de acceso al fichero.

Constante	Valor	Descripción
ForReading	1	Abrir un archivo sólo para lectura. No puede escribir en este archivo.
ForAppending	8	Abrir un archivo y escribir al final del archivo.
ForWriting	2	Abrir un archivo para escritura.

Tabla 34. Constantes para el modo de acceso al fichero.

- **bCrear**. Opcional. Valor lógico que indica si es posible crear el fichero en el caso de que no exista (True) o no es posible crear el fichero (False).
- **kFormato**. Opcional. Constante que indica el formato para el fichero.

Constante	Valor	Descripción
TristateUseDefault	-2	Abrir el archivo utilizando el valor predeterminado del sistema.
TristateTrue	-1	Abrir el archivo como Unicode.
TristateFalse	0	Abrir el archivo como ASCII.

Tabla 35. Constantes para el formato del fichero.

Las opciones del menú FSO en el formulario mdiInicio, muestran algunos ejemplos del uso de este tipo de objeto en la manipulación de carpetas y archivos, tal y como podemos ver en el Código fuente 244, que contiene el código de dichas opciones.

```

Private Sub mnuFSCarpetas_Click()
Dim loFSO As FileSystemObject
Dim lfolEspecial As Folder
Dim lfilFichero As File
Set loFSO = New FileSystemObject
' crear nuevas carpetas
If Not loFSO.FolderExists("c:\pruebas") Then
    loFSO.CreateFolder "c:\pruebas"
    loFSO.CreateFolder "c:\pruebas\datos"
End If
' copiar el contenido de una carpeta en otra
loFSO.CopyFolder "c:\mis documentos", "c:\pruebas\" 
' nombre de la carpeta de nivel superior
MsgBox "Carpeta superior de la ruta 'c:\pruebas\datos' --> " & _
    loFSO.GetParentFolderName("c:\pruebas\datos")
' eliminar carpeta
loFSO.DeleteFolder "c:\pruebas\datos"
' cambiar de posición una carpeta
loFSO.MoveFolder "c:\pruebas", "c:\mis documentos\" 
' recuperar una carpeta del sistema y mostrar
' el nombre de los ficheros que contiene
Set lfolEspecial = loFSO.GetSpecialFolder(WindowsFolder)
For Each lfilFichero In lfolEspecial.Files
    MsgBox lfilFichero.Name, , "Ficheros de la carpeta Windows"
Next
End Sub
*****
Private Sub mnuFSFicheros_Click()
Dim loFSO As FileSystemObject
Dim lfolEspecial As Folder
Dim lfilFichero As File
Dim lcNombreFichero As String
Dim lcInfo As String
Set loFSO = New FileSystemObject
' obtener datos del usuario
lcNombreFichero = InputBox("Introducir el nombre de un fichero", _
    "Pruebas con ficheros")
' comprobar si existe el fichero
If Not loFSO.FileExists(lcNombreFichero) Then
    MsgBox "El fichero " & lcNombreFichero & " no existe"
    Exit Sub
End If
' tomar un objeto fichero del nombre
' introducido por el usuario y mostrar
' algunas propiedades

```

```

Set lfilFichero = loFSO.GetFile(lcNombreFichero)
lcInfo = "Nombre: " & lfilFichero.Name & vbCrLf
lcInfo = lcInfo & "Tamaño: " & FormatNumber(lfilFichero.Size, 0) & vbCrLf
lcInfo = lcInfo & "Tipo: " & lfilFichero.Type & vbCrLf
MsgBox lcInfo, , "Información del fichero " & lcNombreFichero
' copiar el fichero en una carpeta
loFSO.CopyFile lcNombreFichero, "c:\mis documentos\", True
' mover el fichero a una carpeta
loFSO.MoveFile lcNombreFichero, "c:\windows\temp\"
' borrar el fichero de su ubicación original
loFSO.DeleteFile lcNombreFichero
' crear fichero vacío
loFSO.CreateTextFile "registra.txt"
End Sub

```

Código fuente 244

TextStream

Este objeto proporciona un conjunto de propiedades y métodos para realizar operaciones de lectura y escritura sobre un archivo secuencial.

Propiedades

- **AtEndOfLine**. Valor lógico que al ser verdadero indicará que el puntero del archivo está situado al final de una línea.
- **AtEndOfStream**. Valor lógico que al ser verdadero indicará que el puntero del archivo está situado al final del archivo.
- **Column**. Devuelve el número de columna en la que está situado el puntero del archivo.
- **Line**. Devuelve el número de línea en la que está situado el puntero del archivo.

Métodos

- **Close()**. Cierra el fichero.
- **Read()**. Lee un número de caracteres del fichero.

Sintaxis

Read (nCaracteres)

- **nCaracteres**. Número de caracteres a leer.
- **ReadAll()**. Lee el contenido total del fichero y lo devuelve en una cadena. Este método no es muy recomendable ya que emplea muchos recursos, siendo más conveniente la lectura por líneas o por grupos de caracteres.
- **ReadLine()**. Lee una línea del fichero y la devuelve en una cadena.
- **Skip()**. Avanza un número de caracteres en la lectura de un fichero, sin devolverlos.

Sintaxis

`Skip (nCaracteres)`

- `nCaracteres`. Número de caracteres a avanzar.
- `SkipLine()`. Avanza una línea en la lectura de un fichero, sin devolver su contenido.
- `Write()`. Escribe una cadena en un fichero.

Sintaxis.

`Write (cCadena)`

- `cCadena`. Cadena de caracteres a escribir.
- `WriteBlankLines()`. Escribe una o más líneas en blanco en un fichero.

Sintaxis.

`WriteBlankLines (nLineas)`

- `nLineas`. Número de líneas en blanco a insertar.
- `WriteLine()`. Escribe una línea en un fichero que puede incluir una cadena.

Sintaxis.

`WriteLine ([cCadena])`

- `cCadena`. Opcional. Cadena a escribir en la línea.

En el Código fuente 245 se muestra el fuente que corresponde a las opciones del menú TextStream, encargadas de efectuar diversas tareas con ficheros mediante objetos de esta clase.

```
Private Sub mnuTeEscribir_Click()
Dim loFSO As FileSystemObject
Dim ltxsTexto As TextStream
Dim lcNombreFichero As String
Set loFSO = New FileSystemObject
' obtener datos del usuario
lcNombreFichero = InputBox("Introducir el nombre de un fichero", _
    "Crear y escribir en un fichero")
' comprobar si existe el fichero y
' crearlo si no existe
If Not loFSO.FileExists(lcNombreFichero) Then
    loFSO.CreateTextFile lcNombreFichero, True
End If
' abrir el fichero para escritura,
' añadiendo al final
Set ltxsTexto = loFSO.OpenTextFile(lcNombreFichero, ForAppending)
' escribir nuevas cadenas en el fichero
ltxsTexto.Write "cadena de caracteres para el fichero"
ltxsTexto.WriteBlankLines 2
ltxsTexto.WriteLine "nuevos datos"
ltxsTexto.WriteLine "terminar aquí"
' cerrar el fichero
ltxsTexto.Close
```

```

End Sub
*****
Private Sub mnuTeLeer_Click()
Dim loFSO As FileSystemObject
Dim ltxsTexto As TextStream
Dim lcNombreFichero As String
Dim lcCadena As String
Set loFSO = New FileSystemObject
' obtener datos del usuario
lcNombreFichero = InputBox("Introducir el nombre de un fichero", _
    "Abrir y leer de un fichero")
' comprobar si existe el fichero
If Not loFSO.FileExists(lcNombreFichero) Then
    MsgBox "El fichero " & lcNombreFichero & " no existe"
    Exit Sub
End If
' abrir el fichero para lectura
Set ltxsTexto = loFSO.OpenTextFile(lcNombreFichero, ForReading)
' leer varias cadenas y líneas
' agregando a una variable e
' incluyendo saltos en la lectura
lcCadena = ltxsTexto.Read(5)
ltxsTexto.Skip 30
lcCadena = lcCadena & ltxsTexto.Read(20)
ltxsTexto.Skip 10
lcCadena = lcCadena & ltxsTexto.ReadLine
ltxsTexto.SkipLine
lcCadena = lcCadena & ltxsTexto.ReadLine
MsgBox lcCadena, , "Resultado de la lectura de cadenas del fichero"
' cerrar el fichero
ltxsTexto.Close
End Sub
*****
Private Sub mnuTeCompleto_Click()
Dim loFSO As FileSystemObject
Dim ltxsTexto As TextStream
Dim lcNombreFichero As String
Dim lcCadena As String
Set loFSO = New FileSystemObject
' obtener datos del usuario
lcNombreFichero = InputBox("Introducir el nombre de un fichero", _
    "Abrir y leer un fichero por completo")
' comprobar que existe el fichero
If Not loFSO.FileExists(lcNombreFichero) Then
    MsgBox "El fichero " & lcNombreFichero & " no existe"
    Exit Sub
End If
' abrir el fichero para lectura
Set ltxsTexto = loFSO.OpenTextFile(lcNombreFichero, ForReading)
' leer el fichero por líneas y mostrarlas
' al usuario
Do While Not ltxsTexto.AtEndOfStream
    MsgBox "Línea del fichero: " & ltxsTexto.Line & vbCrLf & _
        "Contenido: " & ltxsTexto.ReadLine
Loop
' cerrar el fichero
ltxsTexto.Close
End Sub

```

Código fuente 245

Mejorar la funcionalidad del interfaz visual del programa

La importancia de diseñar un interfaz atractivo

Llegados a este punto, el lector puede desarrollar aplicaciones con plena funcionalidad, que incorporen los elementos principales en el entorno Windows.

Sin embargo, entre dos programas que resuelvan el mismo tipo de problema, existen ciertos detalles que pueden hacer prevalecer uno de ellos respecto a otro.

Tales detalles tienen que ver con el interfaz de usuario desarrollado, puesto que un usuario, ante dos aplicaciones distintas, enfocadas a resolver el mismo problema, siempre se decantará por la que ofrezcan un aspecto más atractivo, utilidades e información complementaria.

Es por este motivo, que en el presente tema, nos dedicaremos a tratar algunos aspectos del desarrollo: técnicas, controles, etc., que permitan dotar a nuestro programa de un interfaz más atractivo, favoreciendo su elección por parte del usuario.

CommonDialog

El control CommonDialog proporciona al programador, un medio de comunicación estándar con el usuario, para las operaciones más habituales de configuración del color, tipo de letra, impresora, apertura/grabación de ficheros y acceso a la ayuda.

El uso de este control proporciona ventajas evidentes. Por una parte, el programador no necesita emplear tiempo en desarrollar, por ejemplo, un formulario y su código correspondiente, para realizar la apertura o grabación de ficheros, el control ya incorpora todo lo necesario, sólo es necesario asignar valores a sus propiedades y mostrarlo. Por otro lado, el usuario de la aplicación sólo necesitará aprender una vez a utilizarlo, ya que al ser un elemento común a todas las aplicaciones Windows, en nuestras siguientes aplicaciones o las de otro programador, el modo de manejo será idéntico.

Si el Cuadro de herramientas de VB no presenta este control, podemos incluirlo abriendo la ventana Componentes, a la cual accederemos mediante el menú de VB, opción Proyecto+Componentes, o la combinación de teclado Ctrl+T. Una vez en esta ventana, seleccionaremos el componente Microsoft Common Dialog Control 6.0, tal como se muestra en la Figura 184.

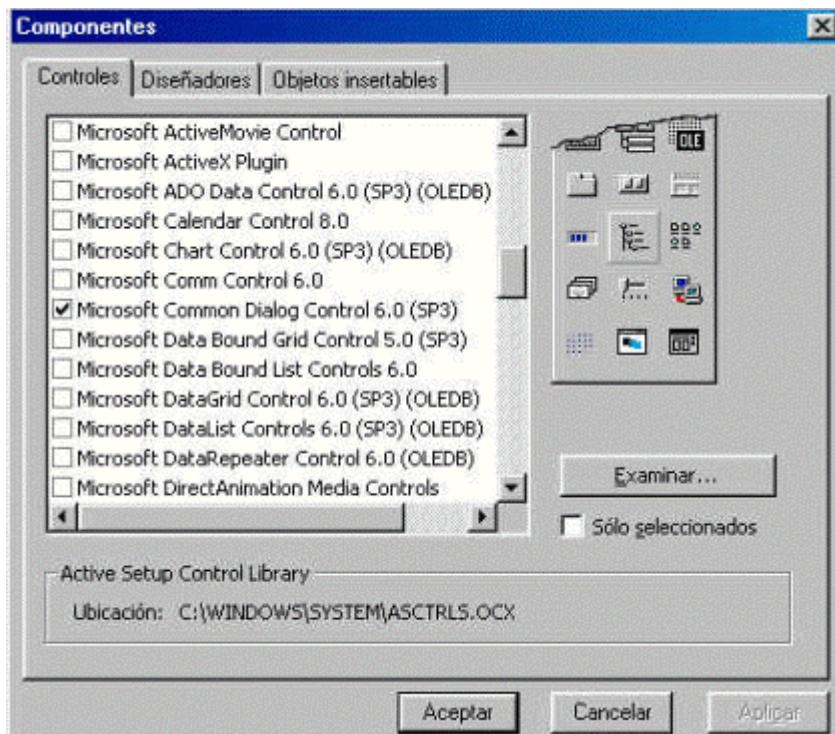


Figura 184. Incorporación del control CommonDialog al Cuadro de herramientas.



Figura 185. Icono del control CommonDialog en el Cuadro de herramientas.

Un control CommonDialog puede ser usado tanto en un form MDI como en uno normal. La forma de insertarlo en el formulario, es la misma que para cualquiera de los controles vistos anteriormente, con la particularidad de que una vez en el formulario, este control se presenta en forma de ícono, sin admitir ningún tipo de edición visual, por lo que todo el trabajo a realizar con él, debe hacerse desde la ventana de propiedades y el código del programa.

Para mostrar las posibilidades de este control, se acompaña el proyecto [CommonDlg](#), con un formulario que incluye un control de este tipo, un menú y un TextBox, como vemos en la Figura 186.

Una de las propiedades más importantes en un CommonDialog es Flags, a la que se asignará, en función del tipo de diálogo a mostrar una o más constantes para configurar dicho diálogo.



Figura 186. Formulario del ejemplo para el control CommonDialog.

Color

El cuadro de diálogo de color, permite seleccionar entre un conjunto de colores disponibles o configurar un color personalizado por el usuario. Al aceptar el diálogo, la propiedad Color del control, contendrá el valor del color seleccionado por el usuario.

Las constantes para la propiedad Flags, referentes al color se muestran en la Tabla 36.

Constante	Valor	Descripción
cdlCCIFullOpen	&H2	Se presenta todo el cuadro de diálogo, incluyendo la sección Definir colores personalizados.
cdlCCHelpButton	&H8	Hace que el cuadro de diálogo presente un botón Ayuda.
cdlCCPreventFullOpen	&H4	Desactiva el botón de comando Definir colores personalizados y evita que el usuario defina colores personalizados.
cdlCCRGRBInit	&H1	Establece el valor de color inicial del cuadro de diálogo.

Tabla 36. Constantes para la propiedad Flags usados en el CommonDialog de selección de color.

La opción de menú Configurar+Color, nos permite cambiar el color de fondo del formulario, mostrando el control dlgOperaciones con el diálogo de selección de color. El Código fuente 246 muestra el código para esta opción. Observe el lector cómo es posible incluir más de una constante en la propiedad Flags empleando el operador Or.

```
Private Sub mnuCoColor_Click()
' configurar diálogo,
' se puede asignar más de una constante usando
```

```

' el operador Or
' cdlCCPreventFullOpen: deshabilita botón "definir colores personalizados"
' cdlCCHelpButton: muestra botón ayuda
Me.dlgOperaciones.Flags = cdlCCPreventFullOpen Or cdlCCHelpButton

' mostrar diálogo para selección de color
Me.dlgOperaciones.ShowColor

' aplicar color seleccionado al color de
' fondo del formulario
Me.BackColor = Me.dlgOperaciones.Color

End Sub

```

Código fuente 246

Tipo de letra

Este cuadro nos permite seleccionar un tipo de fuente o letra para poder asignarlo a cualquier objeto de la aplicación que admita las propiedades de fuente. La Tabla 37 muestra los valores para la propiedad Flags que pueden usarse para este diálogo.

Constante	Valor	Descripción
cdlCFANSIOnly	&H400	Especifica que el cuadro de diálogo sólo permite la selección de una de las fuentes que utilicen el juego de caracteres de Windows. Si este indicador está activado, el usuario no podrá seleccionar una fuente que sólo contenga símbolos.
cdlCFApply	&H200	Activa el botón Aplicar en el cuadro de diálogo.
cdlCFBoth	&H3	Hace que el cuadro de diálogo enumere las fuentes de impresora y de pantalla disponibles. La propiedad hDC identifica el contexto de dispositivo asociado con la impresora.
cdlCFEffects	&H100	Especifica que el cuadro de diálogo permite los efectos de tachado, subrayado y color.
cdlCFFixedPitchOnly	&H4000	Especifica que el cuadro de diálogo selecciona sólo fuentes de tamaño de punto fijo.
cdlCFForceFontExist	&H10000	Especifica que se presentará un mensaje de error si el usuario intenta seleccionar una fuente o un estilo que no exista.
cdlCFHelpButton	&H4	Hace que el cuadro de diálogo presente un botón Ayuda.
cdlCFLimitSize	&H2000	Especifica que el cuadro de diálogo selecciona sólo tamaños de fuente dentro del intervalo especificado por las propiedades Min y Máx.
cdlCFNoFaceSel	&H80000	No hay ningún nombre de fuente seleccionado.

cdlCFNoSimulations	&H1000	Especifica que el cuadro de diálogo no permite simulaciones de fuente con Interfaz de dispositivo gráfico (GDI).
cdlCFNoSizeSel	&H200000	No hay ningún tamaño de fuente seleccionado.
cdlCFNoStyleSel	&H100000	No se seleccionó ningún estilo.
cdlCFNoVectorFonts	&H800	Especifica que el cuadro de diálogo no permite selecciones de fuentes vectoriales.
cdlCFPrinterFonts	&H2	Hace que el cuadro de diálogo enumere únicamente las fuentes aceptadas por la impresora, según lo especificado por la propiedad hDC.
cdlCFScalableOnly	&H20000	Especifica que el cuadro de diálogo sólo permite la selección de fuentes escalables.
cdlCFScreenFonts	&H1	Hace que el cuadro de diálogo enumere únicamente las fuentes de pantalla compatibles con el sistema.
CdlCFTTOnly	&H40000	Especifica que el cuadro de diálogo sólo permite la selección de fuentes TrueType.
CdlCFWYSIWYG	&H8000	Especifica que el cuadro de diálogo sólo permite la selección de fuentes que estén disponibles tanto en la impresora como en la pantalla. Si este indicador está activado, los indicadores cdlCFBoth y cdlCFScalableOnly también deben definirse.

Tabla 37. Constantes de la propiedad Flags usados en el CommonDialog Fuente.

Algunas propiedades específicas de CommonDialog para el tipo de letra son las siguientes.

- FontName. Cadena de caracteres con el nombre del tipo de letra a utilizar.
- FontSize. Tamaño del tipo de letra.
- FontBold - FontItalic - FontUnderline - FontStrikethru. Efectos a aplicar al tipo de letra: negrita, cursiva, subrayada, tachada. Estas propiedades contienen un valor lógico, que al ser True, indicarán que el efecto está activado, y al ser False, que está desactivado.

Un detalle de suma importancia, es que antes de visualizar el diálogo, debemos asignar a Flags alguna de estas constantes: cdlCFScreenFonts, cdlCFPrinterFonts o cdlCFBoth. De no hacerlo así, se producirá un error.

Para ilustrar la selección de un tipo de letra, la operación que vamos a efectuar, es cambiar el tipo de letra en el control Label del formulario, situado sobre el TextBox. La opción de menú que ejecuta esta operación es Configurar+Tipo de letra, y el código que contiene es el Código fuente 247.

```
Private Sub mnuCoLetra_Click()
' se debe utilizar la constante cdlCFScreenFonts
' o dará error al abrir el cuadro de diálogo
```

```

Me.dlgOperaciones.Flags = cdICFScreenFonts 'Or cdICFEffets

' mostrar cuadro diálogo selección de tipo de letra
Me.dlgOperaciones.ShowFont

' si no se selecciona una fuente, terminar el procedimiento
If Len(Trim(Me.dlgOperaciones.FontName)) = 0 Then
    Exit Sub
End If

Me.lblTitulo.FontName = Me.dlgOperaciones.FontName

' comprobar si un determinado flag está activado
' mediante la propiedad flags, el operador And
' y la constante a comprobar
If Me.dlgOperaciones.Flags And cdICFEffets Then
    MsgBox "efectos de letra está activado"
Else
    MsgBox "efectos de letra NO está activado"
End If

End Sub

```

Código fuente 247

Impresión

Para las labores de impresión existen dos cuadros de diálogo. Por defecto se muestra el diálogo Imprimir, pero existe la posibilidad de mostrar el cuadro Configurar impresión, para seleccionar y configurar una de las impresoras conectadas al sistema.

Las constantes para la propiedad Flags, referentes a la impresión se muestran en la Tabla 38.

Algunas propiedades específicas de CommonDialog para imprimir son las siguientes.

- Copies. Número de copias a imprimir.
- FromPage. Número de página inicial por donde se comenzará a imprimir.
- ToPage. Número de página en la que finalizará la impresión.

Constante	Valor	Descripción
cdlPDAllPages	&H0	Devuelve o establece el estado del botón de opción Todo.
cdlPDCollate	&H10	Devuelve o establece el estado de la casilla de verificación Intercalar.
cdlPDDisablePrintToFile	&H80000	Desactiva la casilla de verificación Imprimir en un archivo.
cdlPDHelpButton	&H800	Hace que el cuadro de diálogo presente el botón Ayuda.

cdlPDHidePrintToFile	&H100000	Oculta la casilla de verificación Imprimir en un archivo.
cdlPDNoPageNums	&H8	Desactiva el botón de opción Páginas y el control de edición asociado.
cdlPDNoSelection	&H4	Desactiva el botón de opción Selección.
cdlPDNoWarning	&H80	Evita la presentación de un mensaje de advertencia cuando no hay ninguna impresora predeterminada.
cdlPDPageNums	&H2	Devuelve o establece el estado del botón de opción Páginas.
cdlPDPrintSetup	&H40	Hace que el sistema presente el cuadro de diálogo Configurar impresora en vez del cuadro de diálogo Imprimir.
cdlPDPrintToFile	&H20	Devuelve o establece el estado de la casilla de verificación Imprimir en un archivo.
cdlPDRetnDC	&H100	Devuelve un contexto de dispositivo para la impresora seleccionada en el cuadro de diálogo. El contexto de dispositivo se devuelve en la propiedad hDC del cuadro de diálogo.
cdlPDRetnDefault	&H400	Devuelve el nombre de la impresora predeterminada.
cdlPDRetnIC	&H200	Devuelve un contexto de información para la impresora seleccionada en el cuadro de diálogo. El contexto de información proporciona una manera rápida de obtener información acerca del dispositivo sin crear un contexto de dispositivo. El contexto de información se devuelve en la propiedad hDC del cuadro de diálogo.
cdlPDSelection	&H1	Devuelve o establece el estado del botón de opción Selección. Si no se especifican cdlPDPageNums ni cdlPDSelection, el botón de opción Todo está activado.
cdlPDUseDevModeCopies	&H40000	Si un controlador de impresora no acepta varias copias, el establecimiento de este indicador desactiva el control Número de copias del cuadro de diálogo Imprimir. Si un controlador no acepta varias copias, el establecimiento de este indicador señala que el cuadro de diálogo almacena el número de copias solicitado en la propiedad Copies.

Tabla 38. Constantes para la propiedad Flags usados en el CommonDialog para los cuadros de diálogo Imprimir / Configuración de impresión.

Mediante la opción de menú Configurar+Imprimir, mostraremos el cuadro de diálogo del mismo nombre, utilizado por las aplicaciones para configurar la impresión de documentos. Lo vemos en el Código fuente 248.

```
Private Sub mnuCoImprimir_Click()
' deshabilitar el CheckBox de impresión
' a un fichero
Me.dlgOperaciones.Flags = cdlPDDisablePrintToFile
' mostrar diálogo para imprimir documentos
Me.dlgOperaciones.ShowPrinter
End Sub
```

Código fuente 248

En lo que respecta al diálogo de configuración de impresoras, la opción de menú del formulario mdiInicio Configurar+Impresora, visualiza dicho cuadro, como vemos en el Código fuente 249.

```
Private Sub mnuCoImpresora_Click()
' el cuadro de diálogo será el de selección
' y configuración de impresora
Me.dlgOperaciones.Flags = cdlPDPrintSetup
' mostrar diálogo para configuración de impresora
Me.dlgOperaciones.ShowPrinter
End Sub
```

Código fuente 249

Ayuda

En lo que respecta a la ayuda, el control CommonDialog no muestra ningún cuadro de diálogo específico, sino que visualiza directamente un fichero de ayuda conforme a los valores asignados a las propiedades que sobre la ayuda dispone este control, y que son las siguientes:

- HelpFile. Sirve para asignar una cadena con la ruta y nombre del fichero de ayuda a mostrar.
- HelpCommand. Constante que indica el tipo de ayuda a visualizar. La Tabla 39, contiene los valores disponibles para esta propiedad.

Constante	Valor	Descripción
cdlHelpCommandHelp	&H102&	Ejecuta una macro de Ayuda.
cdlHelpContents	&H3&	Presenta el tema de contenido de Ayuda que se haya definido en la opción Contents de la sección [OPTION] del archivo .HPJ.
cdlHelpContext	&H1&	Presenta Ayuda acerca de un contexto concreto. Cuando utilice este valor, también tiene que especificar un contexto mediante la propiedad HelpContext.

cdlHelpContextPopup	&H8&	Presenta, en una ventana emergente, un determinado tema de Ayuda identificado por un número de contexto definido en la sección [MAP] del archivo .HPJ.
cdlHelpForceFile	&H9&	Asegura que WinHelp presentará el archivo de Ayuda correcto. Si actualmente se está presentando el archivo de Ayuda correcto, no se realizará ninguna acción. Si se está presentando un archivo de Ayuda incorrecto, WinHelp abrirá el archivo correcto.
cdlHelpHelpOnHelp	&H4&	Presenta Ayuda acerca del uso de la aplicación de Ayuda propiamente dicha.
cdlHelpIndex	&H3&	Presenta el índice del archivo de Ayuda especificado. La aplicación sólo debe utilizar este valor con archivos de Ayuda de índice único.
cdlHelpKey	&H101&	Presenta Ayuda acerca de una palabra clave determinada. Cuando utilice este valor, también tiene que especificar una palabra clave mediante la propiedad HelpKey.
cdlHelpPartialKey	&H105&	Presenta el tema encontrado en la lista de palabras clave que coincide con la palabra clave pasada en el parámetro dwData si hay coincidencia exacta. Si hay varias coincidencias, se presenta el cuadro de diálogo Buscar, con los temas encontrados en el cuadro de lista Ir a. Si no hay coincidencia, se presenta el cuadro de diálogo Buscar. Para presentar el cuadro de diálogo Buscar sin pasar una palabra clave, utilice un puntero de tipo Long a una cadena vacía.
cdlHelpQuit	&H2&	Notifica a la aplicación de Ayuda que el archivo de Ayuda especificado ya no está en uso.
cdlHelpSetContents	&H5&	Determina el tema de contenido que se presenta cuando un usuario presiona la tecla F1.
cdlHelpSetIndex	&H5&	Establece el contexto especificado por la propiedad HelpContext como índice actual del archivo de Ayuda especificado por la propiedad HelpFile. Dicho índice permanece como actual hasta que el usuario consulta otro archivo de Ayuda diferente. Utilice este valor sólo para los archivos de Ayuda que tengan varios índices.

Tabla 39. Constantes disponibles para la propiedad HelpCommand.

- HelpKey. Cadena con el tema a mostrar del fichero de ayuda. Para poder utilizar esta propiedad, se debe establecer en la propiedad HelpCommand la constante cdlHelpKey.

- HelpContext. Contiene el identificador de contexto del tema de ayuda. Para poder utilizar esta propiedad, se debe establecer en la propiedad HelpCommand la constante cdlHelpContext.

Para ilustrar el uso de este control con ficheros de ayuda, disponemos de la opción Configurar+Ayuda en el formulario del ejemplo, que contiene el Código fuente 250.

```
Private Sub mnuCoAyuda_Click()
' asignar el fichero de ayuda a utilizar
Me.dlgOperaciones.HelpFile = "c:\windows\help\iexplore.hlp"
' indicar que vamos a buscar dentro del fichero
' de ayuda un tema concreto
Me.dlgOperaciones.HelpCommand = cdlHelpKey
' establecer el tema a buscar
' probar a pasar aquí una cadena vacía
Me.dlgOperaciones.HelpKey = "canales"
' mostrar la ayuda
Me.dlgOperaciones.ShowHelp
End Sub
```

Código fuente 250

Apertura de ficheros

Este cuadro nos permite seleccionar uno o varios ficheros para abrir y manipular su contenido. El control no es el encargado de la apertura del fichero seleccionado, su labor es devolvernos el nombre y ruta de dicho fichero/s, siendo la aplicación la encargada de la apertura y manipulación.

Debido a que tanto el diálogo de apertura como el de grabación son muy similares, las constantes empleadas en la propiedad Flags son las mismas para ambos tipos de cuadro.

Constante	Valor	Descripción
cdlOFNAllowMultiselect	&H200	Especifica que el cuadro de lista Nombre de archivo permita varias selecciones. El usuario puede seleccionar varios archivos en tiempo de ejecución presionando la tecla MAYÚS y utilizando las teclas FLECHA ARRIBA y FLECHA ABAJO para seleccionar los archivos deseados. Al terminar, la propiedad FileName devuelve una cadena que contiene los nombres de todos los archivos seleccionados. Los nombres de la cadena están delimitados por espacios en blanco.
cdlOFNCreatePrompt	&H2000	Especifica que el cuadro de diálogo pida al usuario la creación de un archivo que no existe actualmente. Este indicador establece automáticamente los indicadores cdlOFNPathMustExist y cdlOFNFileMustExist.
cdlOFNExplorer	&H80000	Usa la plantilla del cuadro de diálogo Abrir archivo de tipo Explorador. Funciona en Windows 95 y en Windows NT 4.0.

cdlOFNExtensionDifferent	&H400	Indica que la extensión del nombre de archivo devuelto es distinta de la extensión especificada por la propiedad DefaultExt. Este indicador no está definido si la propiedad DefaultExt es Null, si las extensiones coinciden o si el archivo no tiene extensión. El valor de este Indicador se puede comprobar después de cerrar el cuadro de diálogo.
cdlOFNFileMustExist	&H1000	Especifica que el usuario sólo puede introducir nombres de archivos existentes en el cuadro de texto Nombre de archivo. Si este indicador está activado y el usuario escribe un nombre de archivo no válido, se mostrará una advertencia. Este indicador establece automáticamente el indicador cdlOFNPathMustExist.
cdlOFNHelpButton	&H10	Hace que el cuadro de diálogo presente el botón Ayuda.
cdlOFNHideReadOnly	&H4	Oculta la casilla de verificación Sólo lectura.
cdlOFNLongNames	&H200000	Usa nombres de archivo largos.
cdlOFNNoChangeDir	&H8	Hace que el cuadro de diálogo restablezca como directorio actual el que lo era en el momento de Abrirse el cuadro de diálogo.
CdlOFNNoDereferenceLinks	&H100000	No resuelve la referencia en vínculos del sistema (también conocidos como accesos directos). De forma predeterminada, la elección de un vínculo hace que el sistema resuelva la referencia que contiene.
cdlOFNNoLongNames	&H40000	Nombres de archivos cortos.
cdlOFNNoReadOnlyReturn	&H8000	Especifica que el archivo devuelto no tendrá establecido el atributo de Sólo lectura y no estará en un directorio protegido contra escritura.
cdlOFNNoValidate	&H100	Especifica que el cuadro de diálogo común permite caracteres no válidos en el nombre de archivo devuelto.
cdlOFNOverwritePrompt	&H2	Hace que el cuadro de diálogo Guardar como genere un cuadro de mensajes si el archivo seleccionado ya existe. El usuario tiene que confirmar si desea sobrescribir el archivo.
cdlOFNPathMustExist	&H800	Especifica que el usuario sólo puede escribir rutas de acceso válidas. Si este indicador está activado y el usuario escribe una ruta no válida, se mostrará un mensaje de advertencia.

cdlOFNReadOnly	&H1	Hace que la casilla de verificación Sólo lectura esté activada inicialmente cuando se crea el cuadro de diálogo. Este indicador también señala el estado de la casilla de verificación Sólo lectura cuando se cierra el cuadro de diálogo.
cdlOFNShareAware	&H4000	Especifica que se pasarán por alto los errores por infracción al compartir.

Tabla 40. Constantes para la propiedad Flags usados en el CommonDialog para los diálogos Abrir / Guardar como.

En lo que se refiere a las propiedades de este control para el manejo de ficheros, podemos destacar las siguientes:

- DefaultExt. Contiene una cadena con la extensión predeterminada del fichero a seleccionar.
- DialogTitle. Contiene una cadena con el título para el cuadro de diálogo. Esta propiedad sólo funciona con los cuadros de diálogo de manejo de ficheros.
- FileName. Cadena con el nombre y la ruta del fichero a manipular. Esta propiedad puede establecerse antes de mostrar el cuadro, para disponer de un fichero inicial a seleccionar, o emplearse una vez aceptado el diálogo, para recuperar el fichero seleccionado por el usuario.
- FileTitle. Cadena con el nombre, sin incluir la ruta, del fichero seleccionado.
- InitDir. Directorio inicial del cual se mostrará el contenido al visualizar el diálogo.
- Filter. Cadena con uno o más filtros para los ficheros. Un filtro especifica el tipo de ficheros que serán mostrados por el cuadro de diálogo. El formato de esta cadena es: descripción del tipo de fichero con la extensión entre paréntesis - separador - extensión usada por el cuadro de diálogo. Por ejemplo: Texto (*.TXT)|*.TXT. Esta cadena hace que el diálogo sólo muestre el fichero con la extensión .TXT.

Si fuera necesario incluir más de un tipo de ficheros, se deberá utilizar el separador ";" para delimitarlos: Texto (*.TXT)|*.TXT | Código (*.BAS)|*.BAS.

- FilterIndex. Valor numérico que indica cuál de los filtros especificados en la propiedad Filter, aparecerá como predeterminado al mostrar el cuadro. Si no se indica un valor, se utiliza 1.
- MaxFileSize. Número que informa sobre el tamaño máximo del nombre de archivo abierto.

Para las operaciones de apertura en el formulario, seleccionaremos la opción Archivo+Abrir, de su menú, que contiene el Código fuente 251.

```
Private Sub mnuArAbrir_Click()
' mostrar el cuadro de diálogo para
' apertura de ficheros
Dim lnManipFich As Integer
Dim lcTexto As String
lnManipFich = FreeFile ' tomar manipulador de fichero
' configurar cuadro de diálogo
Me.dlgOperaciones.DialogTitle = "Abrir fichero de texto"
Me.dlgOperaciones.InitDir = "c:\\"
```

```

Me.dlgOperaciones.Filter = "Control (*.log)|*.log|" & _
    "Texto (*.txt)|*.txt"
Me.dlgOperaciones.FilterIndex = 2
Me.dlgOperaciones.ShowOpen ' abrir cuadro
' si no se ha elegido un fichero, salir
If Len(Me.dlgOperaciones.FileName) = 0 Then
    Exit Sub
End If
' si se ha elegido fichero, abrirla
Open Me.dlgOperaciones.FileName For Input As lnManipFich
Do While Not EOF(lnManipFich)
    Line Input #lnManipFich, lcTexto
    Me.txtFichero.Text = Me.txtFichero.Text & lcTexto
Loop
Close #lnManipFich
End Sub

```

Código fuente 251

Grabación de ficheros

Para guardar ficheros utilizando el control CommonDialog, las propiedades a utilizar son las mismas que para abrir, existiendo alguna pequeña variación en las constantes para la propiedad Flags. En el programa de ejemplo, opción de menú Archivo+Guardar del formulario, podemos comprobar como se utiliza este control. En el Código fuente 252 tenemos el código de esta opción.

```

Private Sub mnuArGuardar_Click()
    ' mostrar el cuadro de diálogo para
    ' grabar ficheros
    Dim lnManipFich As Integer
    lnManipFich = FreeFile ' tomar manipulador de ficheros
    ' configurar cuadro de diálogo
    Me.dlgOperaciones.DialogTitle = "Guardar texto como"
    Me.dlgOperaciones.Filter = "Control (*.log)|*.log|" & _
        "Texto (*.txt)|*.txt"
    Me.dlgOperaciones.FilterIndex = 2
    Me.dlgOperaciones.InitDir = App.Path
    ' atención al operador Or para unir constantes
    ' de configuración del diálogo
    Me.dlgOperaciones.Flags = cdlOFNHideReadOnly Or cdlOFNHelpButton
    ' abrir cuadro de diálogo
    Me.dlgOperaciones.ShowSave
    ' si no hay texto para grabar, salir
    If Len(Me.txtFichero.Text) = 0 Then
        Exit Sub
    End If
    Open Me.dlgOperaciones.FileName For Output As lnManipFich
    Print #lnManipFich, Me.txtFichero.Text
    Close #lnManipFich
End Sub

```

Código fuente 252

Agregar controles complementarios

Los controles que se utilizarán a continuación, no se encuentran por defecto en el Cuadro de herramientas. Deberemos abrir la ventana Componentes (menú Proyecto+Componentes de VB), y

marcar la línea Microsoft Windows Common Controls 6.0, para que dichos controles se muestren disponibles.

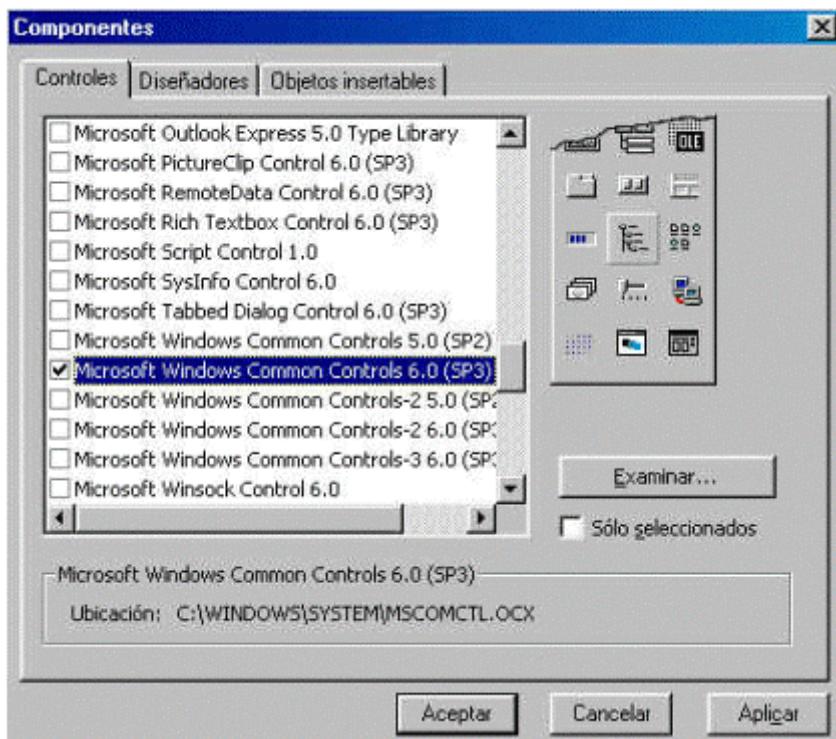


Figura 187. Selección del componente Common Controls 6.0.

Realizada esta acción, pasemos a mostrar el uso y configuración para alguno de estos controles.

El proyecto de ejemplo, [BarraHerram](#), utilizado como base, consiste en el formulario MDI mdiPrincipal; dos formularios secundarios MDI, frmDatos y frmInformes; y un control CommonDialog dlgOperaciones.

Ni los formularios secundarios MDI, ni el CommonDialog realizan operaciones efectivas, sólo se han incluido para mostrar como acceder a ellos mediante una barra de herramientas.



Figura 188. Formulario MDI al que se optimizará el interfaz.

ImageList

Este control se utiliza para almacenar un conjunto de imágenes que serán empleadas por otro control para visualizarlas. El acceso a las imágenes se realizará mediante una clave o índice.

En este caso, vamos a usar un ImageList para almacenar varias imágenes, que serán utilizadas como botones por un control de tipo barra de herramientas.



Figura 189. Control ImageList, en el Cuadro de herramientas.

Al igual que sucedía con CommonDialog, un control ImageList no tiene interfaz visual (no aparece en el formulario al ejecutar la aplicación), se muestra como un ícono en tiempo de diseño, y todos los valores han de establecerse a través de sus propiedades.

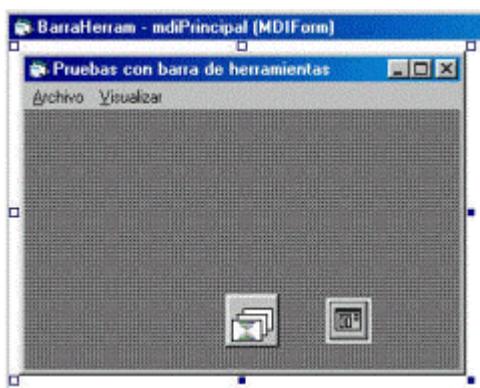


Figura 190. Formulario del ejemplo mostrando un control ImageList.

Una vez dibujado el control ImageList imlImagenes en el formulario, accederemos a su ventana Páginas de propiedades, mediante el menú contextual del control o pulsando en el botón que existe en la propiedad Personalizado de la ventana de propiedades, para configurar las propiedades que no están accesibles en la ventana de propiedades habitual.

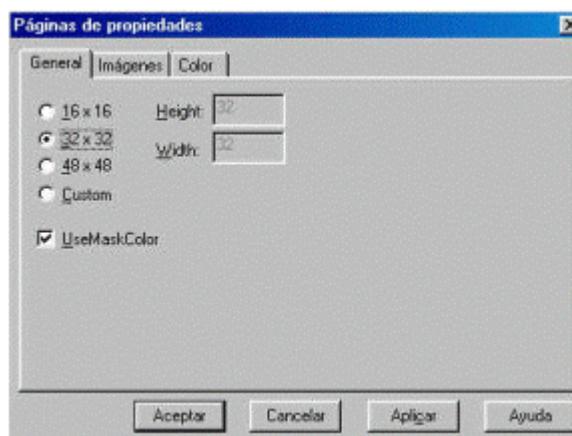


Figura 191. Pestaña General de la ventana Páginas de propiedades para un control ImageList.

En la pestaña General de esta ventana, estableceremos el tamaño de la imagen (Figura 191), que será mostrada por el control que haga uso del ImageList.

En la pestaña Imágenes (Figura 192), agregaremos los ficheros de imagen que van a formar parte de la lista. Usaremos el botón Insertar imagen para incluir nuevos ficheros en el control que serán visualizados en la lista Imágenes. El botón Quitar imagen, eliminará la imagen actualmente seleccionada en la lista.

La propiedad Index, contiene un número, que identifica de forma única a cada imagen. Si además, asignamos un valor a la propiedad Key, podremos referirnos a la imagen, mediante una cadena identificativa.

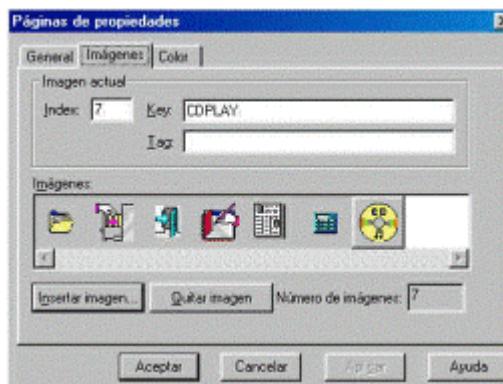


Figura 192. Control ImageList, mostrando la lista de imágenes disponible.

Toolbar

Este control contiene una colección de objetos Button, que por lo general, aunque no es obligatorio, corresponden a las opciones de menú del formulario que incluye la Toolbar.



Figura 193. Control Toolbar, en el Cuadro de herramientas.

Insertaremos en el formulario del ejemplo un control de este tipo, al que llamaremos tbrBarra.

Propiedades

- Align. Constante que indica la posición de la barra de herramientas dentro del formulario en la que está incluida. Los valores disponibles se muestran en la Tabla 41.

Constante	Valor	Descripción
vbAlignNone	0	(Predeterminado en formularios no MDI) Ninguno: el tamaño y la posición pueden establecerse en tiempo de diseño o en el código. Este valor se pasa

		por alto si el objeto está en un formulario MDI.
VbAlignTop	1	(Predeterminado en formularios MDI) Arriba: el objeto está en la parte superior del formulario y su ancho es igual al valor de la propiedad ScaleWidth del formulario.
vbAlignBottom	2	Abajo: el objeto está en la parte inferior del formulario y su ancho es igual al valor de la propiedad ScaleWidth del formulario.
VbAlignLeft	3	Izquierda: el objeto está en la parte izquierda del formulario y su ancho es igual al valor de la propiedad ScaleWidth Del formulario.
vbAlignRight	4	Derecha: el objeto está en la parte derecha del formulario y su ancho es igual al valor de la propiedad ScaleWidth del formulario.

Tabla 41. Constantes disponibles para la propiedad Align, de un control Toolbar.

- ToolTipText. Cadena informativa que se muestra al usuario cuando pasa el puntero del ratón sobre el área del control. En nuestro control no contendrá ningún valor, de esta manera evitaremos que aparezca junto con los ToolTipText de los botones de la barra, lo que podría confundir al usuario.
- ShowTips. Valor lógico que si es True, muestra el contenido de la propiedad ToolTipText de los botones visualizados en la Toolbar.

En la ventana Páginas de propiedades de este control, disponemos de algunas propiedades adicionales. Comenzaremos por la pestaña General, Figura 194.

- ImageList. Contiene el control ImageList, que guarda las imágenes utilizadas en los botones de la barra de herramientas.

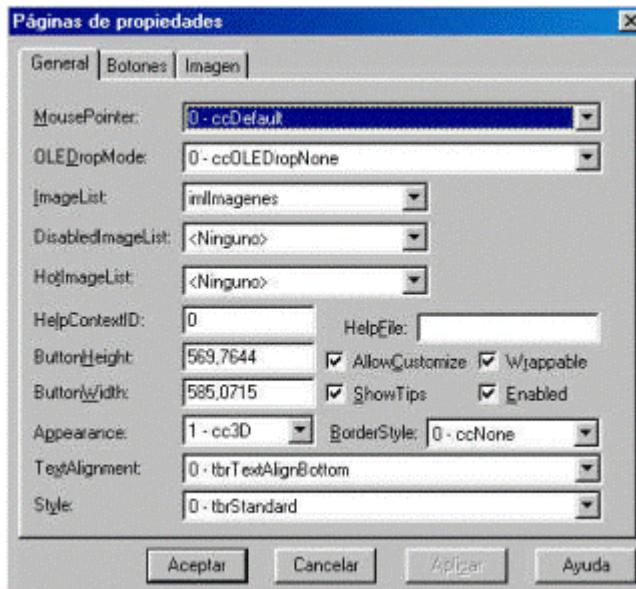


Figura 194. Página de propiedades General del control Toolbar.

En la pestaña Botones, usaremos los botones Insertar botón y Quitar botón para agregar o eliminar botones en la barra. Disponemos adicionalmente de una serie de propiedades para configurar cada botón de la barra, entre las que podemos destacar las siguientes.

- Index. Número de botón de la barra cuyas propiedades aparecen actualmente en la página de propiedades.
- Caption. Cadena que será mostrada dentro del botón, junto a la imagen .
- Key. Cadena para identificar de forma única el botón del resto de botones de la barra, que se corresponde con el valor del botón en el control ImageList asociado a la barra.
- Value. Estado del botón, según las siguientes constantes:
 - tbrUnpressed. El botón aparece sin pulsar, este es el valor por defecto.
 - tbrPressed. El botón se muestra pulsado.
- Style. Constante que indica la apariencia y comportamiento del botón. La Tabla 42 muestra los valores disponibles.

Constante	Valor	Descripción
TbrDefault	0	(Predeterminado) Botón de apariencia normal.
TbrCheck	1	Botón de estilo CheckBox
tbrButtonGroup	2	Grupo de botones entre los cuales siempre se encontrará uno pulsado. La pulsación de un nuevo botón, hará que dicho botón quede presionado, y el que hubiera pulsado vuelva a su estado normal.

TbrSeparator	3	Este tipo de botón no se muestra realmente como tal, sino que se trata de un espacio de 8 pixels que sirve para separar dos botones de la Toolbar.
TbrPlaceholder	4	Este tipo de botón, al igual que el separador, muestra un espacio vacío, pero a diferencia del separador, dispone de la propiedad Width, que puede ser establecida por el programador, para incluir en dicho espacio, un control de un tipo diferente al botón; un ComboBox por ejemplo.
TbrDropDown	5	MenuButton desplegable. Abre una lista de opciones.

Tabla 42. Constantes disponibles para los botones de un control Toolbar.

- ToolTipText. Cadena que informa al usuario de la función que desempeña el botón, mostrada al pasar el usuario el ratón sobre el botón.
- Image. Número de imagen del control ImageList conectado al Toolbar, que se mostrará en el botón.

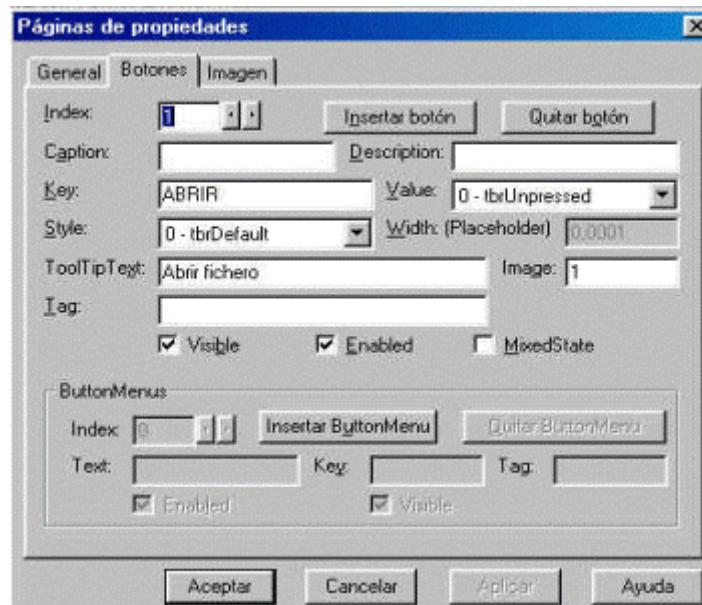


Figura 195. Página de propiedades Botones, para un control Toolbar.

Para crear un botón que despliegue opciones, debemos asignarle en la propiedad Style, la constante tbrDropDown. A continuación, pasaremos a los controles del Frame ButtonMenus, y pulsaremos el botón Insertar ButtonMenu, para crear una opción desplegable. En la propiedad Text, estableceremos la cadena a mostrar cuando se desplieguen las opciones, mientras que en la propiedad Key, asignaremos una cadena que identificará de forma única a la opción pulsada.

El evento a codificar en un control Toolbar, cuando es pulsado uno de sus botones es ButtonClick(). El Código fuente 253, muestra las operaciones a realizar para los botones pulsados en la barra de herramientas del ejemplo.

```
Private Sub tbrBarra_ButtonClick(ByVal Button As MSComctlLib.Button)
Select Case Button.Key
Case "ABRIR"
    mnuArAbrir_Click

Case "GUARDAR"
    mnuArGuardar_Click

Case "SALIR"
    mnuArSalir_Click

Case "DATOS"
    mnuViDatos_Click

Case "INFORMES"
    mnuViInformes_Click

End Select
End Sub
```

Código fuente 253

Este evento recibe como parámetro el botón de la barra de herramientas que ha sido pulsado. Por lo que consultando su propiedad Key, podemos averiguar cual ha sido y actuar en consecuencia.

Observe el lector, que al pulsar cada botón, estamos llamando a los eventos Click() de las correspondientes opciones del menú del formulario. Como se explicó anteriormente, esta es una acción no del todo correcta, ya que los eventos son llamados por las acciones del usuario o sistema, pero nunca directamente en el código.

El llamar a los eventos del menú es una cuestión práctica, ya que como suponemos que los botones de la barra de herramientas se corresponden con las opciones del menú del formulario en donde está dicha barra. Si cambiamos el código de una de las opciones, también deberíamos de cambiar el código del botón correspondiente. Al llamar a la opción de menú desde el botón de la barra, nos ahorraremos escribir dos veces el mismo código.

En este ejemplo, la barra de herramientas tiene algunos botones que se corresponden con las opciones del menú del formulario (el Código fuente 253), y también tiene el botón con el título Ejecutar, que despliega sus propias opciones. Las cuales deben de codificarse en un evento distinto ButtonMenuClick(), que se muestra seguidamente. De esta manera, abordamos todos los modos de codificación de la barra de herramientas, de forma que el lector compruebe todas sus posibilidades.

En el caso de botones desplegables, el evento recibe un objeto ButtonMenu, que se manipula de la misma forma que un botón normal de la barra.

```
Private Sub tbrBarra_ButtonMenuClick(ByVal ButtonMenu As MSComctlLib.ButtonMenu)
Select Case ButtonMenu.Key
Case "CALCULADORA"
    Shell "calc.exe", vbNormalFocus

Case "REPCD"
```

```

Shell "cdplayer.exe", vbNormalFocus

End Select
End Sub

```

Código fuente 254

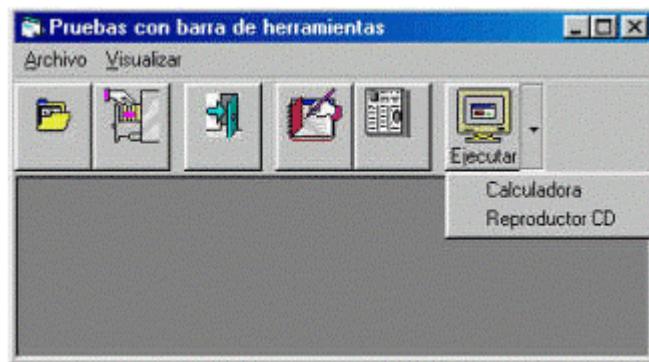


Figura 196. Formulario incorporando un control Toolbar.

StatusBar

Muestra una barra de estado, normalmente en la parte inferior del formulario, con información de utilidad diversa para el usuario, como pueda ser la fecha, hora, estado del teclado, etc.



Figura 197. Control StatusBar, en el Cuadro de herramientas.

El proyecto [BarraEstado](#), parte del proyecto anterior, completándolo en el formulario MDI con un control StatusBar.

Este control está formado por un conjunto de objetos Panel, cada uno de los cuales se encarga de mostrar una clase diferente de información en la barra de estado: fecha, hora, etc. Insertaremos un control de este tipo en el formulario mdiPrincipal, al que llamaremos sbrEstado, que pasamos a configurar seguidamente.

En cuanto a propiedades, además de la ventana de propiedades habitual, disponemos de una ventana de Páginas de propiedades; en la pestaña General, destacan las siguientes:

- Style. Constante que sirve para modificar el estilo de la barra, con dos valores:
 - sbrNormal. La barra muestra todos los paneles que se han definido.
 - sbrSimple. La barra muestra un único panel.
- SimpleText. Texto que se visualizará cuando el valor de la propiedad Style sea sbrSimple.

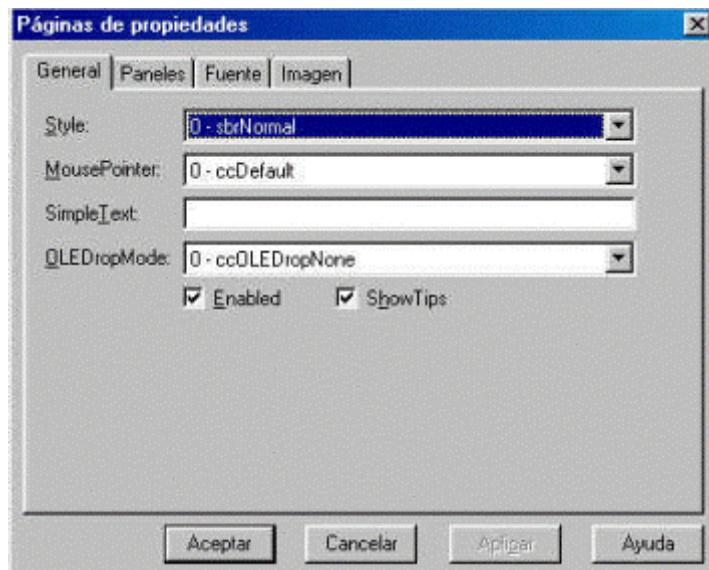


Figura 198. Pestaña General del control StatusBar.

En la pestaña Paneles, agregaremos o eliminaremos paneles mediante los botones Insertar panel y Quitar panel respectivamente. Algunas propiedades destacables son las siguientes:

- Index. Número de panel al que corresponden las propiedades actualmente visualizadas por la página.
- Text. Si el estilo del panel es de texto, se mostrará la cadena contenida en esta propiedad.
- Key. Cadena para identificar de forma única el panel del resto de paneles de la barra.
- Minimum Width. Si el panel es demasiado grande para la información que muestra, como puede ser el caso de los estados del teclado; con esta propiedad podemos ajustar el ancho del panel.
- Alignment. Constante que identifica el modo en que el texto del panel será alineado. La Tabla 43 muestra los valores disponibles.

Constante	Valor	Descripción
SbrLeft	0	(Predeterminado). El texto aparece alineado a la izquierda y a la derecha del mapa de bits.
SbrCenter	1	El texto aparece centrado y a la derecha de mapa de bits.
SbrRight	2	El texto aparece alineado a la derecha y a la izquierda del mapa de bits.

Tabla 43. Constantes disponibles para la propiedad Alignment de un objeto Panel.

- Style. Tipo de panel identificado por una de las constantes mostradas en la

Constante	Valor	Descripción
SbrText	0	(Predeterminado). Texto y mapa de bits. Establece el texto con la propiedad Text.
SbrCaps	1	Tecla BLOQ MAYÚS. Muestra las letras MAYÚS en negrita cuando BLOQ MAYÚS está activada y atenuada cuando está desactivada.
SbrNum	2	Tecla BLOQ NÚM. Muestra las letras NÚM en negrita cuando BLOQ NUM está activada y atenuadas cuando está desactivada.
SbrIns	3	Tecla INSERT. Muestra las letras INS en negrita cuando INSERT está activada y atenuadas cuando no lo está.
sbrScrl	4	Tecla BLOQ DESPL. Muestra las letras DESPL en negrita cuando BLOQ DESPL está activada y atenuadas cuando no lo está.
sbrTime	5	Hora. Muestra la hora actual con el formato del sistema.
sbrDate	6	Fecha. Muestra la fecha actual con el formato del sistema.
sbrKana	7	Kana. Muestra las letras KANA en negrita cuando BLOQ DESPL está activada y atenuada cuando está desactivada.

Tabla 44. Constantes disponibles para la propiedad Style, de un objeto Panel.

- Bevel. Configura el estilo de biselado para un objeto Panel. Los valores posibles se muestran en la Tabla 45.

Constante	Valor	Descripción
sbrNoBevel	0	Ninguno. El objeto Panel no presenta bisel y el texto aparece directamente sobre la barra de estado.
sbrInset	1	(Predeterminado). Hacia adentro. El objeto Panel aparece hundido en la barra de estado.
sbrRaised	2	Hacia afuera. El objeto Panel aparece levantado sobre la barra de estado.

Tabla 45. Constantes disponibles para la propiedad Bevel, de un objeto Panel.

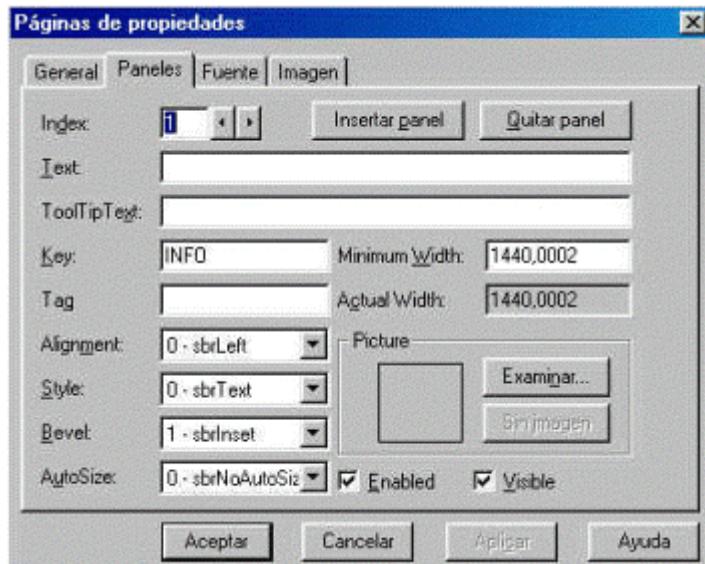


Figura 199. Pestaña Paneles del control StatusBar.

El lector habrá observado que también es posible asignar una imagen a un panel, sólo ha de seleccionar el fichero que contiene el gráfico, pulsando el botón Examinar del frame Picture.

Vistas las propiedades de este control, crearemos nuestro propio conjunto de paneles para el control sbrEstado y ejecutaremos la aplicación, para comprobar el resultado de los nuevos controles que acabamos de incorporar. La Figura 200 muestra el formulario en ejecución.

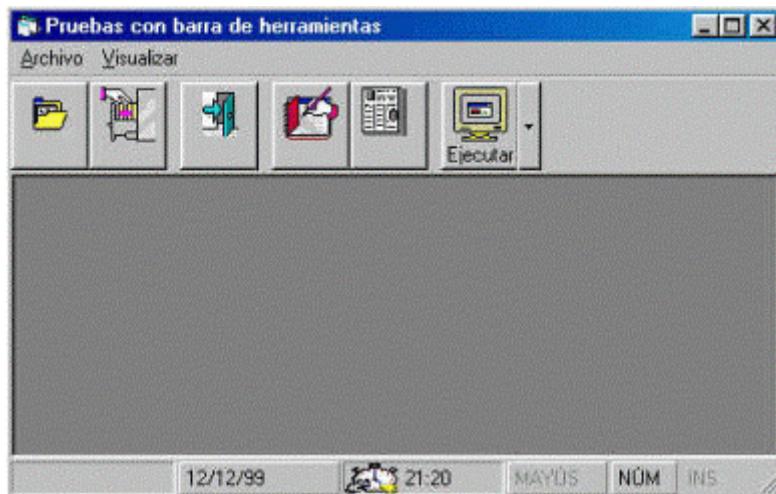


Figura 200. Formulario principal de la aplicación, mostrando los controles Toolbar y StatusBar.

Suponemos que el lector estará de acuerdo, en que el aspecto actual de este formulario MDI, resultará más atractivo al usuario, captará mejor su atención y le proporcionará más información con un simple vistazo.

En tiempo de ejecución, podemos manipular los paneles de un StatusBar como se muestra en el Código fuente 255, correspondiente a la opción de menú del formulario Archivo+Abrir, en la que se asigna una cadena al primer panel.

```
Private Sub mnuArAbrir_Click()
    Me.sbrEstado.Panels(1).Text = "Apertura de un fichero"
    Me.dlgOperaciones.ShowOpen
    Me.sbrEstado.Panels(1).Text = ""
End Sub
```

Código fuente 255

Crear una ventana Splash o de Inicio

Se denomina ventana Splash, a un formulario cuya zona de trabajo está ocupada habitualmente por una imagen con un logotipo, y que no dispone de ningún elemento para que el usuario pueda interaccionar con él: barra de título, menú de control, botones de maximizar, minimizar, cerrar, etc.

Este tipo de formulario, se visualiza durante unos segundos al inicio de la aplicación, proporcionando tiempo a la misma para que prepare su entorno de trabajo: apertura de ficheros, lectura de valores del Registro, etc. De esta forma, el usuario sabrá que el programa está trabajando internamente mientras se muestra este formulario, ya que de lo contrario, la aplicación podría dar la impresión de que se ha bloqueado.

Por las razones antes expuestas, si una vez terminado el desarrollo de una aplicación queremos darle ese toque de profesionalidad, al estilo de las más conocidas aplicaciones comerciales, es muy conveniente incluir un formulario de este tipo.

El proyecto [VentSplash](#), contiene una muestra de este tipo de formulario. Debido a que en este tema, estamos tratando de las mejoras a introducir progresivamente en un programa, este ejemplo parte del punto en donde finalizó el apartado anterior: un proyecto cuya ventana principal MDI contiene una barra de herramientas y una de estado. Como toque final, incorporaremos una ventana de inicio.

Creación

Para agregar una ventana de inicio, seleccionaremos la opción de VB Proyecto+Agregar formulario, que abrirá la ventana de selección del tipo de formulario a añadir. Entre los formularios predefinidos elegiremos Pantalla de inicio, como se muestra en la Figura 201.

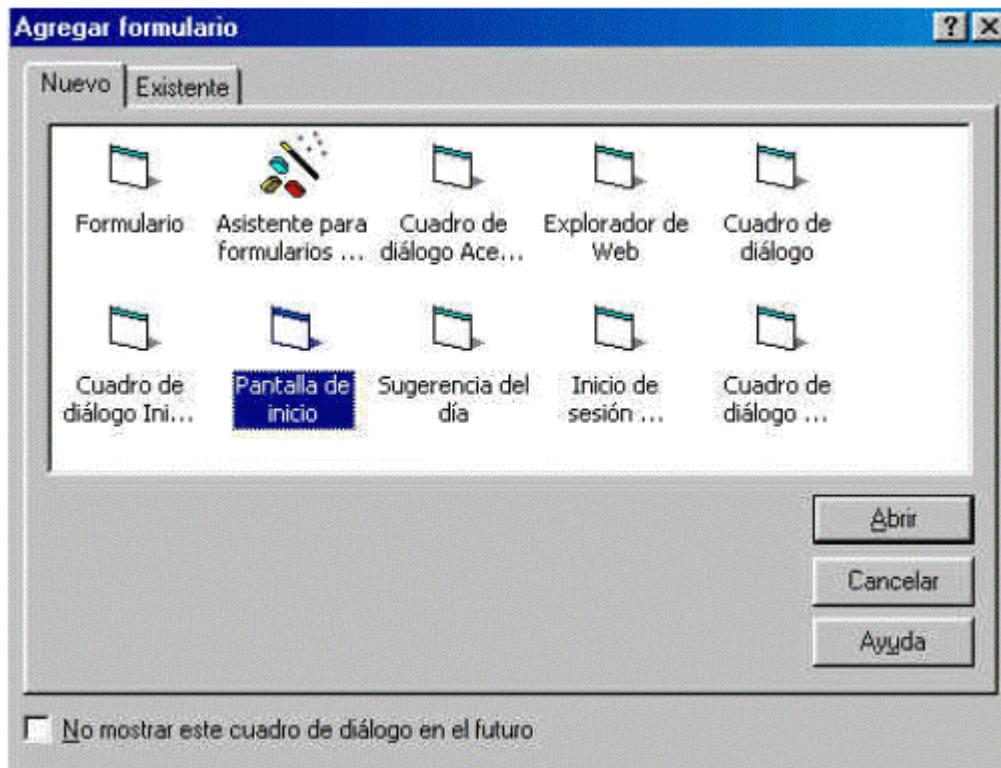


Figura 201. Ventana para agregar un formulario al proyecto.



Figura 202. Formulario Splash creado por VB.

Mediante esta acción, Visual Basic crea automáticamente una ventana con el nombre frmSplash, que podemos ver en la Figura 202.

Al ser un formulario meramente informativo, los controles que incluye son varios Label, un Image y un Frame que los agrupa a todos.

Configuración de propiedades

Debido a que tenemos que presentarlo como una especie de rótulo, ocultando la apariencia de formulario, las propiedades relacionadas con este particular son las siguientes:

- **BorderStyle.** Fixed Dialog.
- **Caption.** Cadena vacía.
- **ControlBox.** False.
- **MDIChild.** False.

Al haber creado VB el formulario, dichas propiedades ya vienen establecidas. En el caso de necesitar crear este tipo de ventana manualmente, las mencionadas propiedades son las que tendríamos que configurar.

Como siguiente paso, vamos a modificar la imagen del formulario y el texto de algunos Label, para darle un toque más personal, como se muestra en la Figura 203.



Figura 203. Formulario Splash con modificaciones respecto al original.

Escritura de código y visualización

El siguiente paso, consiste en agregar un módulo de código en el que escribiremos un procedimiento **Main()**, desde el que se iniciará la aplicación, y tendremos igualmente que configurar las propiedades del proyecto para que el objeto inicial sea este procedimiento.

Llegados a este punto, podemos escribir el Código fuente 256 para mostrar el formulario principal del proyecto y el de presentación.

```
Public Sub Main()
    Dim lmdiPrincipal As mdiPrincipal
    Dim lfrmSplash As frmSplash

    Set lmdiPrincipal = New mdiPrincipal
    lmdiPrincipal.Show
```

```
Set lfrmSplash = New frmSplash
lfrmSplash.Show

End Sub
```

Código fuente 256

La forma de cerrar el formulario frmSplash es presionando una tecla o haciendo clic sobre el control Frame que contiene. Los eventos que realizan estas acciones se muestran en el Código fuente 257.

```
Private Sub Form_KeyPress(KeyAscii As Integer)
    Unload Me
End Sub
Private Sub Frame1_Click()
    Unload Me
End Sub
```

Código fuente 257

Mediante esta técnica, tenemos la ventaja de que las líneas de código a escribir son muy pocas. Su gran inconveniente, sin embargo, radica en que una vez mostrados ambos formularios, hasta que el usuario no pulse o haga clic en el splash, este no desaparecerá; y lo más grave, si el usuario hace clic primero sobre el MDI, el splash no se descargará, quedará por debajo del MDI.

Supongamos además, que queremos iniciar el programa al estilo de las aplicaciones de Office: en primer lugar se muestra el splash, pasados unos instantes, aparece la ventana principal de la aplicación, pero el splash sigue visible, y finalmente, tras otro momento, el splash desaparece y queda la ventana principal de la aplicación.

Como debemos controlar determinados espacios de tiempo, hasta que se abra la ventana MDI y hasta cerrar el splash, lo queharemos será agregar a frmSplash un control Timer, con el nombre tmrTemporizador, que será el utilizado para abrir el MDI y cerrar el formulario de presentación. Asignaremos a su propiedad Interval el valor 2000, para que se ejecute cada 2 segundos.

Seguiremos declarando una variable a nivel de módulo para controlar el número de veces que se ejecuta el temporizador y escribiendo el código de su evento Timer(), en el que está la clave de esta técnica de inicio de la aplicación, y que podemos ver en el Código fuente 258.

```
Option Explicit

' con esta variable controlamos
' el número de veces que se llama
' al evento del control Timer
Private mnEventosTimer As Integer
'-----
Private Sub tmrTemporizador_Timer()
    ' asignar valor al contador de Timer
    mnEventosTimer = mnEventosTimer + 1
    ' la primera vez que se llama al
    ' temporizador...
If mnEventosTimer = 1 Then
    ' instanciar el formulario mdi de la aplicación
    Set gmdiPrincipal = New mdiPrincipal
    ' mostrarlo
```

```
gmdiPrincipal.Show
' volver a mostrar el formulario splash
' ya que al visualizar la ventana mdi
' queda oculto
Me.Show
Else
' en la siguiente llamada al temporizador
' cerrar la ventana splash
Unload Me
End If
End Sub
```

Código fuente 258

El procedimiento Main() también cambiará, ya que ahora sólo necesitaremos iniciar en él, el formulario frmSplash. Observe el lector, que el formulario MDI se ha declarado público a nivel de módulo, para que pueda ser accesible desde cualquier punto del proyecto.

```
Option Explicit

Public gmdiPrincipal As mdiPrincipal
'-----

Public Sub Main()

Dim lfrmSplash As frmSplash

Set lfrmSplash = New frmSplash
lfrmSplash.Show

End Sub
```

Código fuente 259

Hemos podido comprobar, como con unos mínimos cambios, podemos conseguir que el formulario de presentación se comporte de una forma más parecida a la de las aplicaciones más profesionales, ganando la aplicación en calidad.

Tratamiento de datos con ActiveX Data Objects (ADO)

Introducción a la programación con bases de datos

Una base de datos es una estructura diseñada para el manejo de información. Dicha información se clasifica dentro de la base de datos en tablas. Una tabla es un grupo de datos que hacen referencia a un tema determinado como pueda ser la información de clientes de una empresa, las facturas emitidas por la empresa, etc. Dentro de la tabla, los datos se organizan en filas o registros; si la tabla contiene la información de clientes, una fila hará referencia a la información de uno de los clientes. A su vez las filas están compuestas por columnas o campos, que hacen referencia a un aspecto concreto de la fila; siguiendo con el ejemplo de los clientes, un campo de una fila de clientes puede ser el nombre del cliente, la dirección, el teléfono, etc.

La Figura 204 nos muestra la representación gráfica de una tabla, en la que podemos ver sus filas y columnas.

Las bases de datos están contenidas en un *Sistema Gestor de Bases de Datos Relacionales (SGBDR)*, que es el que proporciona los mecanismos necesarios para realizar consultas de datos, mantener los niveles de integridad de la información, seguridad en el acceso, etc. A este conjunto de mecanismos se le denomina también *motor de datos*.

Según el motor de datos utilizado, el SGBDR proporcionará un conjunto de capacidades que en función de su número u optimización darán una mayor o menor potencia a la base de datos utilizada con respecto a otras.

CODIGO	NOMBRE	PRIMERPELLIDO	SECONPELLIDO	DOMICILIO	COD POSTAL	TELEFONO
11	ELENA	MESA	PERAL	HIERRO 4	46444	8889977
22		DIAZ	MANZANO	HIGUERAS 3	46111	5558899
32		ESCAMEZ	PORTA	MAR NEGRO 2	28124	6664589
44	ANGEL	GARCIA	SEGURA	PEZ 10	08111	5556677
55	VERONICA	DIAZ	FRUTOS	LLANOS 5	12457	4445811
60	JULIA	MESA	BARROS	HORNOS 85	17200	6664422
66	SOFIA	REY	LANZAS	BOSQUE 2	22001	9996677
70	ANTONIO	DIAZ	MERINO	CAMPOS 4	12001	3332214
75	Diego	DIAZ	MURILLO	RIOS 30	25002	2226644
88	LUIS	MARTOS	CANDAS	TRANSVERSAL 40	36212	2225577

Figura 204. Tabla de Clientes.

Diseño de la base de datos

Esta es la primera tarea que hemos de acometer en el desarrollo de una aplicación que manipule una base de datos, los pasos a dar en esta etapa del proyecto son los siguientes:

Establecer el modelo de la aplicación

En este paso, debemos analizar lo que la aplicación debe hacer, recabando información del futuro usuario/s de la aplicación. En función de sus necesidades, comenzaremos a tomar idea de las tareas que ha de afrontar el programa. Una vez que tenemos una visión global del trabajo a realizar, debemos separarlo en varios procesos para facilitar su programación.

Analizar que información necesita la aplicación

En este punto, y una vez averiguadas las tareas a realizar por el programa, hemos de determinar que información tenemos que incluir en la base de datos y cual no, de manera que no sobrecarguemos las tablas con datos innecesarios que podrían retardar la ejecución del programa.

Un ejemplo lo tendríamos en la típica tabla de clientes para un comercio; si se desea enviar a cada cliente una felicitación por su cumpleaños, es necesario incluir un campo con su fecha de nacimiento, en caso contrario, no se incluirá este campo evitando sobrecargar la tabla.

Establecer la información en tablas

Para organizar los datos en una aplicación, comenzaremos agrupándolos por temas. De esta manera los datos que tengan que ver con clientes formarán la tabla Clientes; los que tengan que ver con proveedores la tabla Proveedores, y así sucesivamente, por cada tema crearemos una tabla.

Hemos de tener cuidado en esta fase de diseño, puesto que por ejemplo, la información de las tablas Clientes y Proveedores (Código, Nombre, Dirección, etc.) es muy similar y podríamos pensar en ponerla en una única tabla, pero en el caso de los clientes, tendríamos que incluir un campo con la

dirección del almacén de entrega de la mercancía, mientras que como a los proveedores no vamos a realizarles entregas, dicho campo no sería necesario, lo que ocasionaría muchas filas en la tabla con este campo inutilizado cuando se tratara de proveedores.

Normalización de la base de datos

Una vez creadas las tablas, el proceso de normalización consiste en eliminar la información redundante existente entre las tablas. Pongamos como ejemplo una tabla que contiene facturas, y cada fila de la tabla incluye los datos del cliente, como se muestra en la Figura 205.

CabFacturas : Tabla								
	Nombre	Direccion	Poblacion	NIF	NumFac	Fecha	Articulo	Importe
	J.Torre	Alce 2	Madrid	1224466	35	10/05/98	Mesa	5000
►	I.Benitez	Cruces 5	Salamanca	5336678	36	5/06/98	Silla	2000
	M.Gutierrez	Arcipreste 12	Soria	3188775	37	20/06/98	Banqueta	1200
	E.Piquer	Valle 4	Zaragoza	5432311	38	22/06/98	Repisa	3000
*								0

Figura 205. Tabla de facturas con información sin normalizar.

Si en lugar de incluir todos los datos del cliente, creamos una tabla para los clientes con su información, agregando un campo de código, y en la tabla de cabecera de facturas, sustituimos los campos de clientes por el código de cliente, ahorraremos un importante espacio en la tabla de facturas y seguiremos manteniendo la referencia al cliente que hemos facturado, ya que el código de cliente nos llevará a sus datos dentro de la tabla Clientes, como se muestra en la Figura 206.



Clientes : Tabla				
	CodCli	Nombre	Direccion	Poblacion
	20	J.Torre	Alce 2	Madrid
►	35	I.Benitez	Cruces 5	Salamanca
	42	M.Gutierrez	Arcipreste 12	Soria
	12	E.Piquer	Valle 4	Zaragoza
*				

CabFacturas : Tabla				
	CodCli	NumFac	Fecha	Articulo
	20	35	10/05/98	Mesa
►	35	36	5/06/98	Silla
	42	37	20/06/98	Banqueta
	12	38	22/06/98	Repisa
*				0

Figura 206. Información normalizada entre las tablas Clientes y CabFacturas.

Un factor muy importante a efectos del rendimiento de la aplicación, es realizar un estudio del tamaño necesario para los campos de las tablas, establecer por ejemplo, un tamaño de 50 caracteres para un campo de NIF es estar desperdiciando claramente espacio en el disco duro por un lado, y por otro, si tenemos varios campos con un desfase parecido, cuando la tabla comience a crecer, todo ese conjunto

de pequeños detalles de diseño se volverá contra nosotros en forma de lentitud a la hora de hacer operaciones con la tabla.

Un elemento muy útil a la hora de aprovechar espacio lo constituyen las tablas de datos generales. Estas tablas contienen información que puede ser utilizada por todos los procesos de la aplicación, y sirven de apoyo a todas las tablas de la base de datos. Como ejemplo tenemos una tabla que guarde códigos postales, que puede ser utilizada tanto por una tabla de clientes como de proveedores, bancos, etc.

Relacionar las tablas

Eliminada toda información repetida o innecesaria de las tablas, necesitamos establecer relaciones entre las mismas. Una relación es un elemento por el cual dos tablas se vinculan mediante una clave, que pueden ser del siguiente tipo:

- Clave Primaria. Este tipo de clave consiste en que cada registro de la tabla es identificado de forma única mediante el valor de un campo o combinación de campos. Si intentamos dar de alta un registro con una clave ya existente, el motor de datos cancelará la operación provocando un error. Una clave primaria puede ser el código de un cliente, como vemos en la Figura 207, o un código postal.



The screenshot shows a Microsoft Access table window titled "Clientes : Tabla". The table has six columns: "CodCli", "Nombre", "Direccion", "Poblacion", and "NIF". The "CodCli" column is the primary key, indicated by a key icon in the header and bolded text in the cells. The data consists of five rows with the following values:

	CodCli	Nombre	Direccion	Poblacion	NIF
▶	12	E.Piquer	Valle 4	Zaragoza	5432311
	20	J.Torre	Alce 2	Madrid	1224466
	22	H.Mesa	Pino 2	Zaragoza	2211234
	42	M.Gutierrez	Arcipreste 12	Soria	3188775
*					

At the bottom of the window, there is a navigation bar labeled "Registro:" followed by a set of buttons for navigating between records: back, forward, first, last, and a special button for the current record.

Figura 207. Tabla Clientes con clave primaria establecida en el campo CodCli.

- Clave Ajena. Esta clave se forma mediante el valor de un campo o combinación de campos de una tabla, y tiene como propósito establecer una relación con la clave primaria de otra tabla, pudiendo existir varios registros de la clave ajena que se relacionen con un único registro en la clave primaria de la tabla relacionada. En la Figura 208, el campo "CodCli" de la tabla "CabFacturas" es clave ajena del campo del mismo nombre en la tabla "Clientes", existiendo varias filas de "CabFacturas" con el mismo valor, que se corresponden con una única fila de "Clientes".

Figura 208. La tabla CabFacturas tiene una clave ajena en el campo CodCli.

En las relaciones por otra parte, tenemos los siguientes tipos:

- Relación de uno a varios. Este es el tipo de relación más común; en él, una fila de una tabla se relaciona con una o más filas de una segunda tabla, pero cada fila de la segunda tabla se relaciona sólo con una de la primera. Un ejemplo claro de esta situación lo tenemos en la figura anterior, cada fila de la tabla Clientes puede relacionarse con una o varias filas de la tabla CabFacturas por el campo CodCli; pero cada fila de la tabla CabFacturas se relaciona sólo con una fila de la tabla Clientes.
- Relación de varios a varios. Esta relación se produce, cuando cada fila de una tabla se relaciona con cada fila de una segunda tabla y viceversa. En esta situación debemos definir una tabla intermedia que tenga relaciones de uno a varios con las otras dos tablas. Pongamos como ejemplo un almacén de productos cosméticos que distribuye a diferentes perfumerías; cada perfumería puede adquirir varios tipos de cosméticos, e igualmente, cada cosmético puede ser adquirido por varias perfumerías. Al trasladar esta información a una base de datos podríamos hacerlo como muestra la Figura 209.

Figura 209. Relación de varios a varios resuelto con una tabla intermedia.

Establecer la integridad referencial

Una vez que se establece una relación entre dos tablas de la base de datos, el administrador de la base de datos debe decidir, en el caso de que el motor de datos lo permita, si se debe gestionar la integridad referencial.

La integridad referencial es la operación por la cual se debe mantener la consistencia entre la información de las tablas de la base de datos. Pensemos en el ejemplo anterior de las tablas Clientes y CabFacturas; si hemos establecido una relación entre las dos tablas a través del campo CodCli de ambas, y eliminamos en la tabla Clientes la fila que contiene el código de cliente 42, nos encontraremos con un problema de inconsistencia de información, ya que cuando desde la tabla CabFacturas nos posicionemos en la fila que contiene el código de cliente 42 y vayamos a recuperar los datos de ese código de cliente a la tabla Clientes, nos encontraremos con que no existe ninguna fila con el código 42, en este caso, al registro no relacionado de la tabla CabFacturas se le denomina *registro huérfano*.

El tema de la integridad referencial es una decisión de diseño, puede exigirse o no; en el caso de que sea necesaria, será responsabilidad del programador mantenerla o del motor de datos en el caso de que este último la soporte. De cualquiera de las maneras, al exigir integridad referencial nos aseguramos que no quedarán registros huérfanos entre las tablas relacionadas, puesto que si intentamos modificar los valores que forman parte de la relación y no existe el nuevo valor introducido como parte de la clave, se producirá un error que no permitirá completar el proceso.

Adicionalmente se pueden establecer actualizaciones y eliminaciones en cascada, de forma que si se modifica o borra una fila en la primera tabla de la relación, también se modificarán o borrará los registros relacionados en la segunda tabla.

Definir índices para las tablas

En este punto hemos de analizar que tablas y campos de la base de datos son susceptibles de ser consultados un mayor número de veces, y definir un índice para ellos, ya que al ser información que necesitaremos consultar con más frecuencia que el resto, también será necesaria recuperarla más rápidamente.

Un índice ordena las filas de una tabla basándose en un criterio determinado. La clave del índice puede estar formada por uno o varios campos, de esta manera podemos encontrar rápidamente filas en la tabla, o mostrarla ordenada basándose en la clave del índice.

Definir validaciones en la inserción de datos

La validación consiste en comprobar que los datos que vamos a incluir en la fila de una tabla son correctos. Podemos definir validaciones a nivel de motor, de forma que sea el propio motor de datos el que compruebe la información y la rechace en caso de no ser válida. Otra manera de introducir validaciones es mediante el código del programa, lo cual veremos cuando manejemos datos usando código.

Acceso a información bajo cualquier formato

El concepto de datos y bases de datos es algo que está cambiando rápidamente. Tradicionalmente la información residía en bases de datos que la organizaban en tablas y la ofrecían al usuario mediante ciertas utilidades o programas que accedían a ellas.

Sin embargo, el uso de aplicaciones que en un principio no estaban pensadas para organizar datos al uso y manera de una base de datos habitual (procesadores de texto, hojas de cálculo, páginas web, etc.), y la continua expansión en la transferencia de información bajo estos formatos, sobre todo gracias a Internet, plantean el problema de convertir dicha información para que pueda ser leída desde una base de datos, o desarrollar un sistema que permita acceder y organizar dichos datos allá donde residan.

Hasta ahora, y debido a que el volumen de datos en este tipo de formato no era muy elevado, se incluían diferentes conversores en la base de datos para poder integrar el contenido de tales ficheros en tablas. Pero actualmente nos encontramos ante el problema del gran incremento de información a consultar bajo este tipo de ficheros, y la necesidad de realizar búsquedas y editar dicha información directamente en el formato en que reside. Entre las diferentes soluciones planteadas, el Acceso Universal a Datos propuesto por Microsoft, es una de las principales opciones a tener en cuenta.

Acceso Universal a Datos (UDA)

El Acceso Universal a Datos (Universal Data Access), es una especificación establecida por Microsoft para el acceso a datos de distinta naturaleza, que se basa en la manipulación de los datos directamente en el formato en el que han sido creados. ¿Por qué emplear esfuerzos en realizar conversiones entre formatos?, lo lógico y natural es manipular los datos directamente allá donde residan. Pues bien, mediante este nuevo paradigma, y empleando OLE DB y ADO, podremos acceder de una forma sencilla a un gran espectro de tipos de datos, como muestra el esquema de la Figura 210, sin perder un ápice de la potencia de que disponíamos usando DAO y RDO. Es más, mediante un modelo de objetos más simple y flexible, podremos obtener mejores resultados.

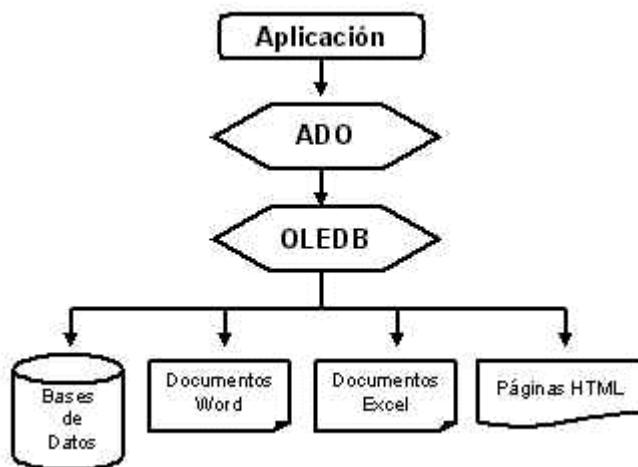


Figura 210. Acceso a datos bajo Universal Data Access.

OLE DB

OLE DB ha sido creado para sustituir progresivamente a ODBC. Los motivos principales son, por un lado, la ya comentada necesidad de acceder a información que reside en formatos no relacionales de datos. Por otra parte, cuando al navegar por Internet se realizan accesos a datos, la propia naturaleza de la Red no asegura que durante la consulta a una determinada página, la conexión a dichos datos vaya a mantenerse todo el tiempo que estamos conectados a dicha dirección. Finalmente está el objetivo de Microsoft, de que todo el acceso a objetos sea realizado mediante su especificación COM (Modelo de Objetos basado en Componentes).

ODBC está orientado a trabajar con información que reside en orígenes de datos tradicionales, y aunque se seguirá dando soporte a esta tecnología, los futuros objetivos van encaminados a utilizar OLE DB.

OLE DB se compone de un conjunto de interfaces de bajo nivel, que utilizan el modelo de componentes para proporcionar acceso a prácticamente cualquier tipo de formato de datos. Desde los relacionales, como puedan ser bases de datos Jet, SQL Server, Oracle; hasta no relacionales, como ficheros de correo electrónico, documentos Word, Excel, páginas web, etc.

La Figura 211, muestra el modelo de objetos de la jerarquía OLE DB, seguido de una breve descripción de cada uno.

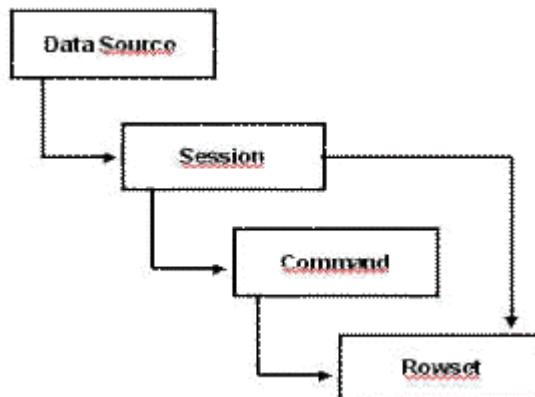


Figura 211. Modelo de objetos OLE DB.

- Data Source. Realiza las comprobaciones sobre el proveedor de datos a utilizar, los permisos de que dispone el usuario, e inicializa la conexión con la fuente de datos.
- Session. Contiene el código necesario para conectar con una fuente de datos.
- Command. Este objeto es el que permite definir los elementos que componen la base de datos y su manipulación, por ejemplo: la creación de tablas y ejecución de instrucciones; siempre y cuando el proveedor lo permita.
- Rowset. Los objetos de este tipo contienen un conjunto de filas, resultado de la ejecución de un objeto Command, o creados desde una sesión.

Al igual que en ODBC existen los conceptos de Cliente y Servidor, en OLE DB disponemos del Proveedor (Provider), o elemento que proporciona los datos; y el Consumidor (Consumer), o elemento

que utiliza los datos que le facilita el proveedor. Entre los proveedores incluidos con la versión de OLE DB que acompaña a Visual Basic 6, destacamos los que aparecen en la Tabla 46.

Proveedor	Tipo de dato
Microsoft Jet 3.51 OLE DB Provider.	Proporciona acceso a base de datos Jet.
Microsoft OLE DB Provider for ODBC Drivers.	Proporciona acceso a fuentes de datos ODBC.
Microsoft OLE DB Provider for Oracle.	Proporciona acceso a bases de datos Oracle.
Microsoft OLE DB Provider for SQL Server.	Proporciona acceso a bases de datos SQL Server.

Tabla 46. Proveedores OLE DB incluidos en Visual Basic.

Si nos encontramos ante el caso de tener que manipular un formato de datos aún no contemplado por ningún fabricante de proveedores OLE DB, mediante el OLE DB SDK, podemos desarrollar nuestros propios proveedores personalizados.

Debido a la complejidad de sus interfaces, la mejor forma de trabajar con OLE DB es utilizando objetos ADO, los cuales proporcionan un acceso simplificado a OLE DB, evitándonos el trabajo con los aspectos más complicados del mismo. La Figura 212, muestra el esquema de acceso a datos desde VB empleando OLE DB/ADO.

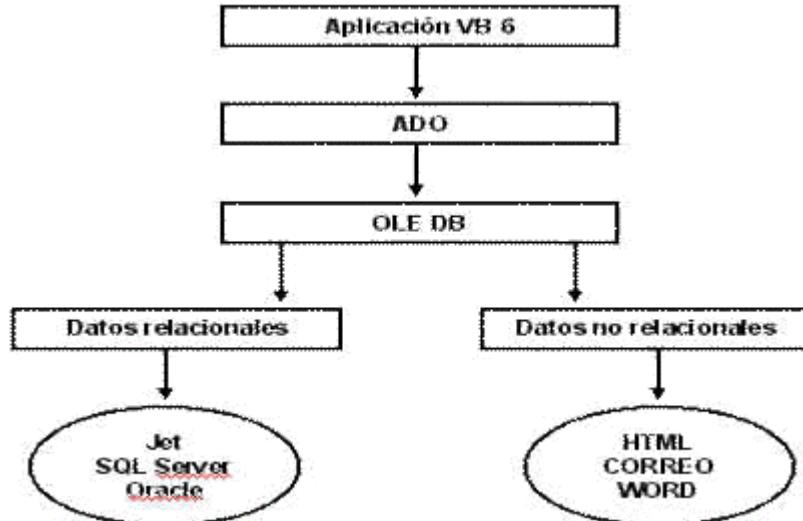


Figura 212. Modo de acceso a datos mediante OLE DB/ADO.

ActiveX Data Objects (ADO)

La jerarquía de objetos ADO, mediante un diseño más simple en comparación con las otras jerarquías de acceso a datos disponibles hasta la actualidad (DAO y RDO), y con un mayor nivel de rendimiento, se constituye a partir de Visual Basic 6, en el nuevo paradigma para la manipulación de datos facilitados por OLE DB. La propia Microsoft, recomienda encarecidamente emplear esta tecnología en

todos los desarrollos que se inicien con la nueva versión de VB. Entre sus diferentes características, podemos destacar las siguientes:

- Posibilidad de crear objetos del modelo ADO, sin necesidad de navegar a través de la jerarquía de objetos. De esta forma, sólo utilizaremos los objetos realmente necesarios para nuestros propósitos.
- Independencia del lenguaje.
- Mínimos requerimientos de memoria y de espacio en disco.
- Gran velocidad de ejecución.
- Es posible llamar a procedimientos almacenados con parámetros de entrada/salida.
- Amplia variedad de tipos de cursor.
- Devolución de conjuntos de resultados múltiples desde una única consulta.
- Ejecución de consultas síncronas, asíncronas o basadas en eventos.
- Permite trabajar de forma flexible tanto con las bases de datos existentes como con los proveedores de OLE DB.
- Manejo de errores optimizado.

La Figura 213 muestra el modelo de objetos ADO:

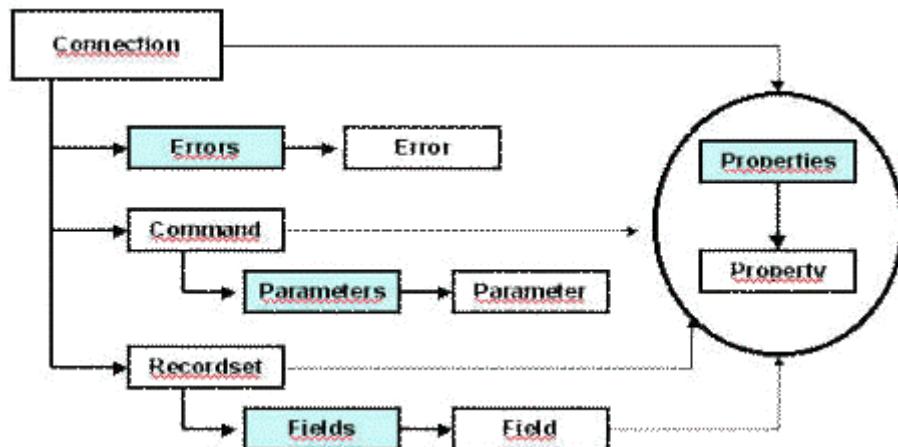


Figura 213. Jerarquía de objetos ADO.

En el caso de las aplicaciones existentes desarrolladas con DAO, y debido a que ADO tiene un carácter claramente más abierto en cuanto al tipo de datos a manejar, el aspecto de la migración de las aplicaciones creadas con DAO a ADO entraña ciertas dificultades que no permiten la simple recompilación del programa para convertirlo a esta nueva tecnología. Por esto se debe realizar un detallado análisis de la aplicación, para determinar si realmente es necesario o merece la pena el trabajo a dedicarle para convertir el código al nuevo modelo de objetos.

A continuación, se describen las principales características de cada uno de los objetos.

Connection

Un objeto Connection representa una conexión establecida con una base de datos. En función del tipo de fuente de datos al que nos conectemos, algunas propiedades o métodos de este objeto pueden no estar disponibles, por lo que recomendamos al lector que revise las especificaciones de la base de datos con la cual desea trabajar.

Propiedades

- Attributes. Constante de tipo Long, que indica las características de la conexión, según la Tabla 47.

Constante	Descripción
adXactCommitRetaining	Al efectuar una llamada a CommitTrans, se inicia una nueva transacción.
adXactAbortRetaining	Al efectuar una llamada a RollbackTrans, se inicia una nueva transacción.

Tabla 47. Constantes para la propiedad Attributes.

- CommandTimeout. Numérico de tipo Long, que indica el número de segundos que un objeto Command intentará procesar un comando antes de generar un error. El valor por defecto es 30, pero si es cero, no habrá límite en el tiempo de espera.
- ConnectionString. Cadena con la información que utilizará el objeto para establecer la conexión. Dentro de esta cadena, se pueden pasar varios parámetros con sus correspondientes valores, siempre que tengan el formato: parámetro = valor, separados por punto y coma.

Los parámetros empleados en esta cadena que pueden ser procesados por ADO se describen en la Tabla 48.

Parámetro	Descripción
Provider	Nombre del proveedor con el que se efectúa la conexión.
File Name	Fichero que contiene información sobre la conexión a establecer. Debido a que con este parámetro se carga el proveedor asociado, no se puede utilizar conjuntamente con Provider.
Remote Provider	Nombre del proveedor que se usa al abrir una conexión del lado del cliente (sólo Remote Data Service).
Remote Server	Ruta de acceso del servidor que se usa al abrir una conexión del lado del cliente (sólo Remote Data Service).

Tabla 48. Valores para ConnectionString.

Cualquier otro parámetro es enviado directamente al proveedor OLE DB correspondiente para ser procesado.

- ConnectionTimeout. Valor de tipo Long, que se utiliza para establecer el tiempo que se intentará establecer la conexión antes de generar un error. El valor por defecto es 15, en el caso de que se utilice cero, no existirá límite de tiempo en el intento de conexión.
- CursorLocation. Permite establecer la situación del cursor disponible por el proveedor, del lado del cliente o del lado del proveedor. Las constantes para esta propiedad se pueden ver en la Tabla 49.

Constante	Descripción
adUseNone	No se utilizan servicios de cursor. Este valor sólo se proporciona por compatibilidad con versiones anteriores.
adUseClient	Usa cursores del lado del cliente. También se puede utilizar la constante adUseClientBatch.
adUseServer	Predeterminado. Utiliza cursores del lado del servidor, suministrados por el controlador o por el proveedor de datos. Aunque este tipo de cursores suelen ser muy potentes y flexibles, puede que alguna característica no esté contemplada.

Tabla 49. Constantes para la propiedad CursorLocation.

- DefaultDatabase. Cadena con el nombre de la base de datos predeterminada para el objeto Connection.
- IsolationLevel. Constante que establece el nivel de aislamiento en las transacciones, según la Tabla 50

Constante	Descripción
adXactUnspecified	El proveedor está utilizando un nivel de aislamiento que no se puede determinar.
adXactChaos	Predeterminado. Impide sobrescribir cambios pendientes de transacciones con un nivel de aislamiento más alto.
adXactBrowse	Permite ver desde una transacción, cambios que no se han producido en otras transacciones.
adXactReadUncommitted	Realiza la misma función que adXactBrowse.
adXactCursorStability	Desde una transacción se podrán ver cambios en otras transacciones sólo después de que se hayan producido.
adXactReadCommitted	Igual que adXactCursorStability.
adXactRepeatableRead	Indica que desde una transacción no se pueden ver los cambios producidos en otras transacciones, pero que al

	volver a hacer una consulta se pueden obtener nuevos Recordsets.
adXactIsolated	Las transacciones se realizan aisladamente de otras transacciones.
adXactSerializable	Igual que adXactIsolated.

Tabla 50. Constantes para la propiedad IsolationLevel.

- Mode. Constante que indica los permisos de acceso a datos para una conexión, según la Tabla 51.

Constante	Descripción
adModeUnknown	Predeterminado. Indica que los permisos no se han establecido aún o que no se pueden determinar.
adModeRead	Indica que son permisos de sólo lectura.
adModeWrite	Indica que son permisos de sólo escritura.
adModeReadWrite	Indica que son permisos de lectura/escritura.
adModeShareDenyRead	Impide que otros abran una conexión con permisos de lectura.
adModeShareDenyWrite	Impide que otros abran una conexión con permisos de escritura.
adModeShareExclusive	Impide que otros abran una conexión.
adModeShareDenyNone	Impide que otros abran una conexión con cualquier tipo de permiso.

Tabla 51. Permisos disponibles para un objeto Connection.

- Provider. Cadena con el nombre del proveedor de datos.
- State. Indica el estado del objeto: abierto o cerrado.

Constante	Descripción
adStateClosed	Predeterminado. El objeto está cerrado.
adStateOpen	El objeto está abierto.

Tabla 52. Constantes de estado para un objeto Connection.

- Version. Retorna una cadena con el número de versión de ADO.

Métodos

- BeginTrans(), CommitTrans(), RollbackTrans(). Gestionan las transacciones dentro de un objeto Connection. BeginTrans inicia una transacción; CommitTrans guarda en la base de datos los cambios realizados durante la transacción; RollbackTrans cancela las operaciones efectuadas durante la transacción y finaliza la misma.

En el caso de BeginTrans, puede devolver un valor Long que indica el nivel de anidamiento de la transacción, utilizando la siguiente sintaxis:

```
lNivel = oConnection.BeginTrans()
```

Antes de intentar utilizar transacciones, es necesario comprobar que en la colección Properties del objeto, aparece la propiedad Transaction DDL, para asegurarnos que el proveedor las acepta.

- Cancel(). Cancela la ejecución de un método Execute u Open, cuando estos han sido llamados de forma asíncrona.
- Close(). Cierra un objeto Connection, liberando los recursos que estuviera utilizando. Este método no elimina el objeto de la memoria, ya que es posible modificar algunas de sus propiedades y volver a utilizarlo. Para eliminarlo por completo de la memoria es necesario asignarle Nothing.
- Execute(). Ejecuta una sentencia SQL, procedimiento almacenado, etc., sobre la base de datos que utiliza el objeto Connection.
 - Sintaxis para un objeto que no devuelva filas:


```
oConnection.Execute cComando, lRegistrosAfectados, lOpciones
```
 - Sintaxis para un objeto que devuelva filas. En este caso se obtiene un objeto Recordset de tipo Forward-only:


```
Set oRecordset = oConnection.Execute(cComando,
lRegistrosAfectados, lOpciones)
```
 - Parámetros:
 - cComando. Cadena con la sentencia SQL, nombre de la tabla, procedimiento almacenado, etc.
 - lRegistrosAfectados. Opcional. Valor Long en la que el proveedor deposita el número de registros afectados por la operación
 - lOpciones. Opcional. Constante que indica al proveedor como debe evaluar el parámetro cCommandText. La Tabla 53 muestra los valores disponibles.

Constante	Descripción
-----------	-------------

adCmdText	El parámetro cCommandText se evalúa como una instrucción SQL.
adCmdTable	ADO crea una consulta SQL que devuelve las filas de la tabla que se indica en cCommandText.
adCmdTableDirect	El resultado es una representación física de la tabla indicada en cCommandText.
adCmdStoredProc	El proveedor tiene que evaluar cCommandText como procedimiento almacenado.
adCmdUnknown	El tipo de comando en cCommandText es desconocido.
adExecuteAsync	Indica que el comando se tiene que ejecutar de forma asíncrona.
adFetchAsync	Indica que el resto de las filas siguientes a la cantidad inicial especificada en la propiedad CacheSize tiene que ser recuperada de forma asíncrona.

Tabla 53. Constantes utilizadas en el parámetro lOptions del método Execute.

Al finalizar la ejecución de este método, se genera un evento ExecuteComplete.

- Open(). Establece una conexión con una fuente de datos.
 - Sintaxis:


```
oConnection.Open cCadenaConexion, cIDUsuario, cPassword,
xOpcionesApertura
```
 - Parámetros:
 - cCadenaConexion. Opcional. Cadena con la información para establecer la conexión. El valor de este parámetro es igual que para la propiedad ConnectionString de este objeto.
 - cIDUsuario. Opcional. Cadena con el nombre de usuario que se va a conectar. Este valor tendrá prioridad en el caso de que también se utilice en cCadenaConexion.
 - cPassword. Opcional. Cadena con la clave de acceso del usuario. Este valor tendrá prioridad en el caso de que también se utilice en cCadenaConexion.
 - xOpcionesApertura. Opcional. Valor de tipo ConnectOptionEnum. Cuando este valor sea adConnectAsync, la conexión se abrirá de forma asíncrona.

Una vez se haya realizado la conexión, se generará un evento ConnectComplete.

- OpenSchema(). Proporciona información sobre el esquema o estructura interna de la base de datos sobre la que se ha establecido la conexión.

Dicha información se devuelve en forma de recordset estático de sólo lectura.

- Sintaxis:

```
Set oRecordset = oConnection.OpenSchema(xTipoConsulta,
cCriterio, IDEsquema)
```

- Parámetros:

- xTipoConsulta. Constante que indica el tipo de esquema a consultar.
- cCriterio. Según la constante utilizada en xTipoConsulta, este parámetro contendrá un array con los valores disponibles para cada tipo de consulta.

Los valores combinados de xTipoConsulta y cCriterio se muestran en la Tabla 54.

Valores de QueryType	Valores de Criterio
adSchemaAsserts	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaCatalogs	CATALOG_NAME
adSchemaCharacterSets	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME
adSchemaCheckConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
adSchemaCollations	COLLATION_CATALOG COLLATION_SCHEMA COLLATION_NAME
adSchemaColumnDomainUsage	DOMAIN_CATALOG DOMAIN_SCHEMA DOMAIN_NAME COLUMN_NAME
adSchemaColumnPrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME GRANTOR GRANTEE
adSchemaColumns	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME

	COLUMN_NAME
adSchemaConstraintColumnUsage	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaConstraintTableUsage	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
AdSchemaForeignKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME FK_TABLE_CATALOG FK_TABLE_SCHEMA FK_TABLE_NAME
adSchemaIndexes	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TYPE TABLE_NAME
adSchemaKeyColumnUsage	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
adSchemaPrimaryKeys	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME
adSchemaProcedureColumns	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME COLUMN_NAME
adSchemaProcedureParameters	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMTER_NAME
adSchemaProcedures	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMTER_TYPE
adSchemaProviderSpecific	Para proveedores no estándar.
adSchemaProviderTypes	DATA_TYPE BEST_MATCH

adSchemaReferentialConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
AdSchemaSchemata	CATALOG_NAME SCHEMA_NAME SCHEMA_OWNER
adSchemaSQLLanguages	<ninguno>
AdSchemaStatistics	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
adSchemaTableConstraints	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME CONSTRAINT_TYPE
adSchemaTablePrivileges	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME GRANTOR GRANTEE
adSchemaTables	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE
adSchemaTranslations	TRANSLATION_CATALOG TRANSLATION_SCHEMA TRANSLATION_NAME
adSchemaUsagePrivileges	OBJECT_CATALOG OBJECT_SCHEMA OBJECT_NAME OBJECT_TYPE GRANTOR GRANTEE
adSchemaViewColumnUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
adSchemaViewTableUsage	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
AdSchemaViews	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME

Tabla 54. Valores para xTipoConsulta y cCriterio.

- IDEsquema. Identificador global para un esquema de un proveedor no definido en la especificación OLE DB. Este parámetro es necesario cuando el valor de xTipoConsulta sea adSchemaProviderSpecific.

Command

Este objeto representa un comando emitido hacia una fuente de datos para consulta, edición de registros, etc.

Propiedades

- ActiveConnection. Contiene el objeto Connection al que pertenece el Command
- CommandText. Cadena con una sentencia SQL, tabla o procedimiento almacenado a ejecutar.
- CommandTimeout. Numérico de tipo Long, que indica el número de segundos que un objeto Connection intentará procesar un comando antes de generar un error. El valor por defecto es 30, pero si es cero, no habrá límite en el tiempo de espera.
- CommandType. Establece el tipo de comando a ejecutar, optimizando la evaluación de la propiedad CommandText. Dentro de la Tabla 55 se muestran las constantes disponibles para esta propiedad. Debemos ser cuidadosos a la hora de establecer esta propiedad, ya que si no guarda relación con el valor que contiene CommandText, se producirá un error al intentar ejecutar el objeto Command.

Constante	Descripción
AdCmdText	Evalúa CommandText como una instrucción SQL o una llamada a un procedimiento almacenado.
AdCmdTable	Crea una consulta SQL que devuelve las filas de la tabla que se indica en CommandText.
adCmdTableDirect	El resultado es una representación física de la tabla indicada en CommandText.
adCmdStoredProc	Evalúa CommandText como un procedimiento almacenado.
adCmdUnknown	Predeterminado. El tipo de comando de la propiedad CommandText es desconocido, por lo cual ADO debe realizar llamadas previas al proveedor para determinar el contenido de CommandText, y así, ejecutarlo adecuadamente. Por este motivo, el rendimiento en la ejecución se ve penalizado, razón por la que debemos evitar en lo posible el uso de este tipo de comando.

adCommandFile	Evalúa CommandText como el nombre de archivo de un valor Recordset persistente.
adExecuteNoRecords	Se utiliza cuando CommandText contiene una instrucción o procedimiento almacenado que no devuelve filas (como una modificación en el valor de una columna de una tabla). En el caso de recuperar alguna fila, no se devuelve. Siempre se combina con adCmdText o adCmdStoredProc. El uso de esta constante mejora el rendimiento en la ejecución del objeto Command, al procesarse de forma interna.

Tabla 55. Constantes para la propiedad CommandType.

- Prepared. Contiene un valor lógico que si es True, indica al proveedor que debe preparar/compilar la consulta contenida en el objeto antes de ejecutarla por primera vez. Esto causará una disminución de rendimiento en esa primera ejecución, pero al encontrarse compilado en las posteriores llamadas, se acelerará esa misma ejecución.

Cuando el valor de esta propiedad sea False, se ejecutará el comando sin realizar su compilación.

En el caso de que el proveedor no admita la compilación del comando, puede generar un error al intentar asignar el valor True a esta propiedad o simplemente no compilar el objeto.

- State. Indica el estado del objeto: abierto o cerrado. Consultar esta misma propiedad en el objeto Connection.

Métodos

- Cancel(). Cancela la ejecución de un método Execute, que ha sido llamado de forma asíncrona.
- CreateParameter(). Crea un objeto Parameter para un comando. Este método no agrega el nuevo parámetro a la colección Parameters del objeto Command. Para más información sobre el objeto Parameter, consulte el lector, el apartado dedicado a dicho objeto en este mismo tema.

- Sintaxis:

```
Set oParameter = oCommand.CreateParameter(cNombre, lTipo,
                                         lDireccion, lTamaño, lValor)
```

- Execute(). Ejecuta la consulta SQL o procedimiento almacenado contenido en la propiedad CommandText.

- Sintaxis para un objeto que devuelva filas:

```
Set oRecordset = oCommand.Execute(lRegistrosAfectados,
                                   aParametros, lOpciones)
```

- Sintaxis para un objeto que no devuelva filas:

```
oCommand.Execute lRegistrosAfectados, aParametros, lOpciones
```

- Parámetros:
 - lRegistrosAfectados. Opcional. Valor Long en la que el proveedor deposita el número de registros afectados por la operación.
 - aParametros. Opcional. Array de tipo Variant, con los valores de los parámetros utilizados por la instrucción SQL del comando.
 - lOpciones. La funcionalidad es la misma que en el objeto Connection, consulte el lector este mismo método en dicho objeto.

Una vez finalizada la ejecución de este método, se genera un evento ExecuteComplete.

Parameter

Este tipo de objeto, representa un parámetro asociado a un objeto Command de tipo consulta parametrizada o procedimiento almacenado, de manera que podamos ejecutar el mismo comando obteniendo diferentes resultados en función de los valores pasados en los parámetros.

Propiedades

- Attributes. Constante Long, que especifica las características del objeto según la Tabla 56.

Constante	Descripción
adParamSigned	Predeterminado. El parámetro acepta valores firmados.
adParamNullable	Indica que el parámetro acepta valores Null.
adParamLong	Indica que el parámetro acepta datos binarios largos.

Tabla 56. Constantes para la propiedad Attributes.

- Direction. Constante que indica la dirección del objeto Parameter, en función de las constantes de la Tabla 57.

Constante	Descripción
adParamUnknown	Indica que la dirección del parámetro es desconocida.
adParamInput	Predeterminado. El parámetro es de entrada.
adParamOutput	Indica un parámetro de salida.
adParamInputOutput	Indica un parámetro de entrada y otro de salida.

adParamReturnValue	Indica un valor devuelto.
--------------------	---------------------------

Tabla 57. Constantes para la propiedad Direction.

- Name. Cadena con el nombre del objeto.
- NumericScale. Número de tipo Byte que indica la cantidad de posiciones decimales para los parámetros con valores numéricos.
- Precision. Dato Byte que indica la cantidad de dígitos usados para representar los parámetros con valores numéricos.
- Size. Número Long que establece el tamaño en bytes o caracteres del parámetro. Si el tipo de datos es de longitud variable, esta propiedad debe ser establecida antes de añadir el objeto a la colección Parameters, o se producirá un error.
- Type. Indica el tipo de datos para el objeto. La Tabla 58 muestra las constantes para los tipos de datos disponibles en el modelo ADO. El tipo correspondiente para OLE DB se muestra entre paréntesis.

Constante	Descripción
adArray	Se une en una instrucción OR lógica con otro tipo para indicar que los datos son una matriz segura de ese tipo (DBTYPE_ARRAY).
adBigInt	Un entero con signo de 8 bytes (DBTYPE_I8).
adBinary	Un valor binario (DBTYPE_BYTES).
adBoolean	Un valor Boolean (DBTYPE_BOOL).
adByRef	Se une en una instrucción OR lógica con otro tipo para indicar que los datos son un puntero a los datos del otro tipo (DBTYPE_BYREF).
adBSTR	Una cadena de caracteres terminada en nulo (Unicode) (DBTYPE_BSTR).
AdChar	Un valor de tipo String (DBTYPE_STR).
adCurrency	Un valor de tipo Currency (DBTYPE_CY). Un valor Currency es un número de coma fija con cuatro dígitos a la derecha del signo decimal. Se almacena en un entero con signo de 8 bytes en escala de 10.000.
Hádate	Un valor de tipo Date (DBTYPE_DATE). Un valor Date se almacena como un valor de tipo Double; la parte entera es el número de días transcurridos desde el 30 de diciembre de 1899 y la parte fraccionaria es la

	fracción de un día.
adDBDate	Un valor de fecha (aaaammdd) (DBTYPE_DBDATE).
adDBTime	Un valor de hora (hhmmss) (DBTYPE_DBTIME).
adDBTimeStamp	Una marca de fecha y hora (aaaammddhhmmss más una fracción de miles de millones) (DBTYPE_DBTIMESTAMP).
adDecimal	Un valor numérico exacto con una precisión y una escala fijas (DBTYPE_DECIMAL).
adDouble	Un valor de coma flotante de doble precisión (DBTYPE_R8).
adEmpty	No se ha especificado ningún valor (DBTYPE_DBDATE)
AdError	Un código de error de 32 bits (DBTYPE_ERROR).
AdGUID	Un identificador único global (GUID) (DBTYPE_GUID).
adIDispatch	Un puntero a una interfaz Idispatch de un objeto OLE (DBTYPE_IDISPATCH).
adInteger	Un entero firmado de 4 bytes (DBTYPE_I4).
adIUnknown	Un puntero a una interfaz Iunknown de un objeto OLE (DBTYPE_IUNKNOWN).
adLongVarBinary	Un valor binario largo (sólo para el objeto Parameter).
adLongVarChar	Un valor largo de tipo String (sólo para el objeto Parameter).
adLongVarWChar	Un valor largo de tipo String terminado en nulo (sólo para el objeto Parameter).
adNumeric	Un valor numérico exacto con una precisión y una escala exactas (DBTYPE_NUMERIC).
adSingle	Un valor de coma flotante de simple precisión (DBTYPE_R4).
adSmallInt	Un entero con signo de 2 bytes (DBTYPE_I2).
adTinyInt	Un entero con signo de 1 byte (DBTYPE_I1).

adUnsignedBigInt	Un entero sin signo de 8 bytes (DBTYPE_UI8).
adUnsignedInt	Un entero sin signo de 4 bytes (DBTYPE_UI4).
adUnsignedSmallInt	Un entero sin signo de 2 bytes (DBTYPE_UI2).
adUnsignedTinyInt	Un entero sin signo de 1 byte (DBTYPE_UI1).
adUserDefined	Una variable definida por el usuario (DBTYPE_UDT).
adVarBinary	Un valor binario (sólo para el objeto Parameter).
adVarChar	Un valor de tipo String (sólo para el objeto Parameter).
adVariant	Un tipo Variant de automatización (DBTYPE_VARIANT).
adVector	Se une en una instrucción OR lógica con otro tipo para indicar que los datos son una estructura DBVECTOR, tal como está definida por OLE DB, que contiene un contador de elementos y un puntero a los datos del otro tipo (DBTYPE_VECTOR).
adVarWChar	Una cadena de caracteres Unicode terminada en nulo (sólo para el objeto Parameter).
adWChar	Una cadena de caracteres Unicode terminada en nulo (DBTYPE_WSTR).

Tabla 58. Tipos de datos de la jerarquía de objetos ADO.

- Value. Contiene un dato Variant con el valor asignado al objeto.

Métodos

- AppendChunk(). Permite añadir un texto de gran tamaño o datos binarios a un Parameter.
 - Sintaxis


```
oParameter.AppendChunk vntDatos
```
 - Parámetros
 - vntDatos. Valor Variant que contiene la información a añadir.

Si el bit adFldLong de la propiedad Attributes de un objeto Parameter tiene el valor True, se puede utilizar el método AppendChunk en dicho parámetro.

La primera vez que se llama a AppendChunk, se escriben los datos en el parámetro sobrescribiendo la información anterior. Las siguientes llamadas a este método, agregan los nuevos datos a los ya existentes. Si el valor pasado es nulo, se ignoran todos los datos del parámetro.

Recordset

Un objeto Recordset representa un conjunto de registros, bien puede ser la totalidad de registros de una tabla o un grupo más pequeño, resultado de la ejecución de una consulta SQL de selección, y que contenga registros de una o varias tablas.

Propiedades

- **AbsolutePage.** Informa de la página en la que está el registro actual. Las constantes que aparecen en la Tabla 59 nos ayudan en la identificación de la página.

Constante	Descripción
adPosUnknown	El objeto Recordset está vacío, la posición actual se desconoce o el proveedor no admite la propiedad AbsolutePage.
adPosBOF	El puntero del registro actual está al comienzo del archivo (es decir, la propiedad BOF tiene el valor True).
adPosEOF	El puntero del registro actual está al final del archivo (es decir, la propiedad EOF tiene el valor True).

Tabla 59. Constantes para la propiedad AbsolutePage.

- **AbsolutePosition.** Valor Long que informa sobre el número que tiene un registro dentro de un Recordset, o bien puede ser una de las constantes ya especificadas en la propiedad AbsolutePage. El uso de esta propiedad, aunque pueda ser de utilidad en casos muy determinados, no es en absoluto recomendable, ya los números de registro que maneja un objeto Recordset pueden cambiar bajo variadas circunstancias. Por este motivo, la forma más recomendable para guardar posiciones de registros siguen siendo los bookmarks.
- **ActiveConnection.** Contiene el objeto Connection al que pertenece el Recordset.
- **BOF - EOF.** Indican el comienzo y final del Recordset respectivamente.
- **Bookmark.** Valor que identifica un registro, de forma que podamos guardar su posición para volver a él en cualquier momento de la ejecución.
- **CacheSize.** Devuelve un número Long que indica la cantidad de registros que están en la memoria caché local.
- **CursorLocation.** Desempeña la misma función que en el objeto Connection. Consulte el lector el apartado sobre dicho objeto.

- **CursorType.** Establece el tipo de cursor para el Recordset, según las constantes que aparecen en la Tabla 60

Constante	Descripción
adOpenForwardOnly	Predeterminado. Cursor de tipo Forward-only. Idéntico a un cursor estático, excepto que sólo permite desplazarse hacia delante en los registros. Esto mejora el rendimiento en situaciones en las que sólo se quiere pasar una vez por cada registro.
AdOpenKeyset	Cursor de conjunto de claves. Igual que un cursor dinámico, excepto que no se pueden ver los registros que agregan otros usuarios, aunque los registros que otros usuarios eliminan son inaccesibles desde su conjunto de registros. Los cambios que otros usuarios hacen en los datos permanecen visibles.
adOpenDynamic	Cursor dinámico. Las incorporaciones, cambios y eliminaciones que hacen otros usuarios permanecen visibles, y se admiten todo tipo de movimientos entre registros, a excepción de los marcadores si el proveedor no los admite.
adOpenStatic	Cursor estático. Una copia estática de un conjunto de registros que se puede usar para buscar datos o generar informes. Las incorporaciones, cambios o eliminaciones que hacen otros usuarios no son visibles.

Tabla 60. Constantes disponibles en CursorType.

- **EditMode.** Devuelve un valor que indica el estado de edición del registro actual. Las constantes asociadas aparecen en la Tabla 61.

Constante	Descripción
adEditNone	Indica que no hay ninguna operación de modificación en ejecución.
adEditInProgress	Indica que los datos del registro actual se han modificado pero que no se han guardado aún.
adEditAdd	Indica que se ha invocado el método AddNew y que el registro situado actualmente en el búfer de copia es un nuevo registro que no se ha guardado en la base de datos.
adEditDelete	Indica que el registro actual se ha eliminado.

Tabla 61. Valores devueltos por la propiedad EditMode.

- Filter. Valor Variant, que establece un filtro para los registros de un Recordset. Puede estar compuesto por una o varias cadenas con el criterio de ordenación concatenadas mediante los operadores AND u OR; un array de Bookmarks; o una de las constantes que aparecen en la Tabla 62.

Constante	Descripción
AdFilterNone	Elimina el filtro actual y vuelve a poner todos los registros a la vista.
adFilterPendingRecords	Permite ver sólo los registros que han cambiado pero que no han sido enviados aún al servidor. Aplicable sólo para el modo de actualización por lotes.
adFilterAffectedRecords	Permite ver sólo los registros afectados por la última llamada a Delete, Resync, UpdateBatch o CancelBatch.
adFilterFetchedRecords	Permite ver los registros de la caché actual, es decir, los resultados de la última llamada para recuperar registros de la base de datos.
adFilterConflictingRecords	Permite ver los registros que fallaron en el último intento de actualización por lotes.

Tabla 62. Constantes a utilizar en la propiedad Filter.

- LockType. Indica el tipo de bloqueo para los registros durante su edición. Las constantes para esta propiedad se muestran en la Tabla 63.

Constante	Descripción
adLockReadOnly	Predeterminado. Sólo lectura.
adLockPessimistic	Bloqueo pesimista, registro a registro: el proveedor hace lo necesario para asegurar la modificación correcta de los registros, generalmente bloqueando registros en el origen de datos durante el proceso de modificación.
adLockOptimistic	Bloqueo optimista, registro a registro: el proveedor usa bloqueo optimista, bloqueando registros sólo cuando llama al método Update.
adLockBatchOptimistic	Actualizaciones optimistas por lotes: requerido para el modo de actualización por lotes como contraposición

	al modo de actualización inmediata.
--	-------------------------------------

Tabla 63. Tipos de bloqueo para un Recordset.

- MarshalOptions. Al utilizar recordsets del lado del cliente, esta propiedad mejora la comunicación entre esos tipos de recordset y el servidor, estableciendo que registros se devuelven al servidor (Tabla 64).

Constante	Descripción
adMarshalAll	Predeterminado. Todas las filas se devuelven al servidor.
adMarshalModifiedOnly	Sólo las filas modificadas se devuelven al servidor.

Tabla 64. Constantes para la propiedad MarshalOptions.

- MaxRecords. Establece el número máximo de registros que se devuelven a un Recordset.
- PageCount. Dato Long que informa sobre el número de páginas de un Recordset.
- PageSize. Valor Long que contiene el número de registros que constituyen una página de un objeto Recordset.
- RecordCount. Indica el número de registros que contiene un Recordset.
- Sort. Ordena uno o varios campos del Recordset en forma ascendente, descendente o combinando ambas. Para ello debemos asignar una cadena a esta propiedad con el nombre o nombres de los campos a ordenar, separados por comas e indicar el modo de ordenación mediante las palabras clave ASC o DESC.
- Source. Devuelve una cadena con el tipo de origen de datos de un Recordset.
- State. Informa sobre el estado del objeto. La Tabla 65 muestra las constantes que tiene esta propiedad.

Constante	Descripción
AdStateClosed	Predeterminado. Indica que el objeto está cerrado.
adStateOpen	Indica que el objeto está abierto.
adStateConnecting	Indica que el objeto Recordset se está conectando.
adStateExecuting	Indica que el objeto Recordset está ejecutando un comando.

adStateFetching	Indica que se está obteniendo el conjunto de filas del objeto Recordset.
-----------------	--

Tabla 65. Constantes para los estados del objeto Recordset.

- Status. Informa sobre el estado del registro actual. En la Tabla 66 se muestran los valores para esta propiedad.

Constante	Descripción
adRecOK	El registro se ha actualizado correctamente.
adRecNew	El registro es nuevo.
adRecModified	El registro se ha modificado.
adRecDeleted	El registro se ha eliminado.
adRecUnmodified	El registro no se ha modificado.
adRecInvalid	El registro no se ha guardado debido a que su marcador no es válido.
adRecMultipleChanges	El registro no se ha guardado debido a que habría afectado a múltiples registros.
adRecPendingChanges	El registro no se ha guardado debido a que hace referencia a una inserción pendiente.
adRecCanceled	El registro no se ha guardado debido a que la operación se ha cancelado.
adRecCantRelease	El nuevo registro no se ha guardado debido a los bloqueos de registro existentes.
AdRecConcurrencyViolation	El registro no se ha guardado debido a que la ejecución simultánea optimista estaba en uso.
adRecIntegrityViolation	El registro no se ha guardado debido a que el usuario ha infringido las restricciones de integridad.
adRecMaxChangesExceeded	El registro no se ha guardado debido a que había demasiados cambios pendientes.
adRecObjectOpen	El registro no se ha guardado debido a un conflicto con el objeto de almacenamiento abierto.
adRecOutOfMemory	El registro no se ha guardado debido a que el equipo se ha quedado sin memoria.

adRecPermissionDenied	El registro no se ha guardado debido a que los permisos del usuario eran insuficientes.
adRecSchemaViolation	El registro no se ha guardado debido a que infringe la estructura de la base de datos subyacente.
adRecDBDeleted	El registro ya se ha eliminado del origen de datos

Tabla 66. Valores para la propiedad Status.

Métodos

- AddNew(). Crea un nuevo registro en un Recordset.
 - Sintaxis


```
oRecordset.AddNew aListaCampos, aValores
```
 - Parámetros
 - aListaCampos. El nombre de un campo o un array con los nombres de los campos a asignar valor.
 - aValores. Un valor o un array con los valores a asignar a los campos de ListaCampos.
- Cancel(). Cancela la ejecución asíncrona del método Open().
- CancelBatch(). Cancela una actualización por lotes pendiente de procesar.
 - Sintaxis:


```
oRecordset.CancelBatch xRegistrosAfectados
```
 - Parámetros
 - xRegistrosAfectados. Opcional. Constante que indica los registros que resultarán afectados con este método. Lo vemos en la Tabla 67.

Constante	Descripción
adAffectCurrent	Sólo cancela las actualizaciones del registro actual.
adAffectGroup	Cancela las actualizaciones pendientes de los registros que cumplen el valor actual de la propiedad Filter. Para poder utilizar esta opción tiene que establecer la propiedad Filter a una de las constantes predefinidas válidas.

adAffectAll	Predeterminado. Cancela las actualizaciones pendientes de todos los registros del objeto Recordset, incluyendo los no visibles debido al valor actual de la propiedad Filter
-------------	--

Tabla 67. Constantes para los registros afectados en el método CancelBatch().

- CancelUpdate(). Cancela la edición de un registro nuevo o existente antes de llamar al método Update().
- Clone(). Crea un Recordset duplicado. Este método sólo está disponible sobre un Recordset que admite marcadores.

- Sintaxis:

```
Set rsDuplicado = rsOriginal.Clone(xTipoBloqueo)
```

- Parámetros

- rsDuplicado. Nuevo Recordset que se va a crear a partir de rsOriginal.
- rsOriginal. Recordset del cual se va a crear la copia.
- xTipoBloqueo. Opcional. Constante que especifica el tipo de bloqueo de rsOriginal. En la Tabla 68 vemos los valores para el tipo de bloqueo Clone()

Constante	Descripción
adLockUnspecified	Predeterminado. La copia se crea con el mismo tipo de bloqueo que el original.
adLockReadOnly	Se crea una copia de sólo lectura.

Tabla 68. Valores para el tipo de bloqueo en Clone().

El uso de Clone() es particularmente útil en situaciones en las que debamos disponer de información acerca de varios registros de un mismo Recordset simultáneamente, debido a que se obtiene un mayor rendimiento que abriendo diferentes Recordset con iguales características.

Las modificaciones efectuadas en uno de los objetos, serán visibles en el resto, excepto cuando se ejecute el método Requery() en el Recordset original. Es posible cerrar cualquiera de los Recordsets, incluido el original, sin que esto conlleve al cierre de los demás.

- Delete(). Borra el registro actual del Recordset.

- Sintaxis:

```
oRecordset.Delete xRegistrosAfectados
```

- Parámetros

- **xRegistrosAfectados.** Opcional. Constante que determina cuantos registros serán afectados por el borrado. Veamos la Tabla 69.

Constante	Descripción
adAffectCurrent	Predeterminado. Sólo elimina el registro actual.
adAffectGroup	Elimina los registros que cumplen el valor de la propiedad Filter. Tiene que establecer la propiedad Filter a una de las constantes predefinidas válidas para poder utilizar esta opción.

Tabla 69. Constantes para indicar los registros afectados por Delete().

- **Find().** Busca un registro en función de un criterio. Si el criterio se cumple, el registro localizado pasa a ser el actual, en caso contrario el Recordset se sitúa al final.

- **Sintaxis**

```
oRecordset.Find cCriterio, lModoComienzo, kDireccion,
vntPosComienzo
```

- **Parámetros:**

- **cCriterio.** Cadena que contiene la expresión de búsqueda incluyendo el campo en el que realizará la búsqueda, el operador y el valor a buscar como en la siguiente línea de ejemplo:

```
"Nombre >= 'Pedro'"
```

- **lModoComienzo.** Opcional. Valor Long que por defecto es cero e indica que la búsqueda se realizará desde el registro actual o desde un marcador.
- **kDireccion.** Opcional. Constante que indica si la búsqueda se realizará hacia adelante (adSearchForward) o hacia atrás (adSearchBackward).
- **vntPosComienzo.** Opcional. Marcador que establece la posición de comienzo de la búsqueda.

- **Move().** Desplaza la posición del registro actual en un Recordset a un nuevo registro.

- **Sintaxis**

```
oRecordset.Move lNumRegistros, vntComienzo
```

- **Parámetros**

- **lNumRegistros.** Número Long con la cantidad de registros a desplazar.
- **vntComienzo.** Cadena o Variant con un Bookmark a partir del que se realizará el desplazamiento. También puede ser una de las constantes que aparecen en la Tabla 70.

Constante	Descripción
adBookmarkCurrent	Predeterminado. Empieza en el registro actual.
adBookmarkFirst	Empieza en el primer registro.
adBookmarkLast	Empieza en el último registro.

Tabla 70. Valores para el desplazamiento con Move().

- MoveFirst(), MoveLast(), MoveNext(), MovePrevious(). Desplazan la posición del registro actual de un Recordset al primer, último, siguiente y anterior registro respectivamente. El Recordset debe admitir movimiento hacia atrás o marcadores para poder utilizar estos métodos.
- NextRecordset(). Cuando se emplee una instrucción que englobe varios comandos o un procedimiento almacenado que devuelva varios resultados, este método recupera el siguiente Recordset del conjunto de resultados.

- Sintaxis

```
Set rssiguiente =
  rsActual.NextRecordset (lRegistrosAfectados)
```

- Parámetros

- lRegistrosAfectados. Dato Long que utiliza el proveedor para informar del número de registros afectados por la ejecución del método.

Este método se emplea cuando se genera un Recordset al que se le pasa una cadena con una instrucción múltiple (por ejemplo: "SELECT * FROM Clientes; SELECT * FROM Cuentas")

- Open(). Abre un cursor.

- Sintaxis:

```
oRecordset.Open vntOrigen, vntConexionActiva, xTipoCursor,
  xTipoBloqueo, xOpciones
```

- Parámetros

- vntOrigen. Opcional. Variant que contiene un objeto Command, una sentencia SQL, el nombre de una tabla, procedimiento almacenado, o el nombre de fichero de un Recordset.
- vntConexionActiva. Opcional. Variant que contiene un objeto Conexion, o una cadena con parámetros para efectuar una conexión.
- xTipoCursor. Opcional. Constante que indica al proveedor el tipo de cursor a emplear. Consulte el lector la propiedad CursorType de este objeto.

- **xTipoBloqueo.** Opcional. Constante que determina el tipo de bloqueo. Consulte el lector la propiedad LockType de este objeto.
- **xOpciones.** Opcional. Constante que indica el modo de evaluación de vntOrigen. Consulte el lector la propiedad CommandType del objeto Command para un detalle sobre estas constantes.
- **Requery().** Vuelve a ejecutar la consulta del Recordset para actualizar completamente su contenido. Este método es equivalente a llamar a los métodos Close() y Open() sucesivamente.
- **Resync().** Actualiza la información de un Recordset desde la base de datos, refrescando los registros del Recordset. Este método no ejecuta la consulta en que está basado el Recordset, por lo que los nuevos registros no serán visibles.
 - **Sintaxis**
 - oRecordset.Resync **xRegistrosAfectados, xSobreescribirValores**
 - **Parámetros**
 - **xRegistrosAfectados.** Constante que indica a los registros que afectará este método, (Tabla 71).

Constante	Descripción
adAffectCurrent	Actualiza solamente el registro actual.
adAffectGroup	Actualiza los registros que cumplen el valor de la propiedad Filter actual. Debe establecer la propiedad Filter a una de las constantes válidas predefinidas para poder usar esta opción.
adAffectAll	Predeterminado. Actualiza todos los registros del objeto Recordset, incluyendo cualquier registro oculto por el valor de la propiedad Filter actual.

Tabla 71. Constantes para los registros afectados por Resync().

- **xSobreescribirValores.** Valor que indica si se sobrescribirán los valores subyacentes, en la tabla 72 se muestran las constantes para sobrescribir los registros en Resync().

Constante	Descripción
adResyncAllValues	Predeterminado. Se sobrescriben los datos y se cancelan las actualizaciones pendientes.
adResyncUnderlyingValues	No se sobrescriben los datos ni se cancelan las actualizaciones pendientes.

Tabla 72. Constantes para sobrescribir los registros en Resync().

- **Save()**. Guarda el Recordset en un fichero.

- **Sintaxis**

```
oRecordset.Save cFichero, xFormato
```

- **Parámetros**

- **cFichero**. Ruta y nombre del fichero en el que se guardará el Recordset.
- **xFormato**. Opcional. Formato en el que se guardará el objeto. El único valor disponible actualmente es **adPersistADTG**.

- **Supports()**. Informa de si un Recordset soporta cierta funcionalidad, retornando un valor lógico que indica si los modos de cursor pasados como parámetro son soportados.

- **Sintaxis**

```
bResultado = oRecordset.Supports(lModosCursor)
```

- **Parámetros**

- **lModosCursor**. Valor Long que puede especificarse mediante las constantes que aparecen en la Tabla 73.

Constante	Descripción
adAddNew	Puede usar el método AddNew para agregar nuevos registros.
adApproxPosition	Puede leer y establecer las propiedades AbsolutePosition y AbsolutePage.
adBookmark	Puede usar la propiedad Bookmark para tener acceso a registros específicos.
adDelete	Puede usar el método Delete para eliminar registros.
adHoldRecords	Puede recuperar más registros o cambiar la posición de recuperación siguiente sin efectuar todos los cambios pendientes.
adMovePrevious	Puede usar los métodos MoveFirst y MovePrevious, y los métodos Move o GetRows para desplazar hacia atrás la posición del registro actual sin que se requiera marcadores.
adResync	Puede actualizar el cursor con los datos visibles en la base de datos subyacente, usando el método Resync.
adUpdate	Puede usar el método Update para modificar datos existentes.
adUpdateBatch	Puede usar actualización por lotes (métodos UpdateBatch y CancelBatch) para transmitir cambios al proveedor en grupos.

Tabla 73. Modos de cursor para el método Supports().

- Update(). Guarda los cambios efectuados en el registro actual de un Recordset.

- Sintaxis

```
oRecordset.Update aListaCampos, aValores
```

- Parámetros

Consulte el lector los parámetros empleados en el método AddNew().

- UpdateBatch(). Guarda los cambios realizados al Recordset en la base de datos a la que pertenece mediante un proceso por lotes. El Recordset debe admitir este tipo de actualizaciones.

- Sintaxis

```
oRecordset.UpdateBatch xRegistrosAfectados
```

- Parámetros

- xRegistrosAfectados. Opcional. Constante que indica los registros que serán afectados al ejecutar este método.

Constante	Descripción
adAffectCurrent	Escribe solamente los cambios pendientes en el registro actual.
adAffectGroup	Escribe los cambios pendientes en los registros que cumplen el valor de la propiedad Filter actual. Para poder usar esta opción, debe establecer la propiedad Filter a una de las constantes válidas predefinidas.
adAffectAll	Predeterminado. Escribe los cambios pendientes en todos los registros del objeto Recordset, incluyendo aquellos que oculta el valor actual de la propiedad Filter.

Tabla 74. Constantes disponibles para los registros afectados en el método UpdateBatch().

Field

Objeto que representa a una columna de datos, en la colección Fields perteneciente a un objeto Recordset.

Propiedades

- ActualSize. Dato Long, que contiene la longitud actual del valor de un campo. No confundir con el tamaño definido para el campo. En este caso, si por ejemplo, un campo de una tabla se

ha definido como de tipo texto para 50 caracteres, y en el registro actual sólo tiene una cadena con 10 caracteres, ActualSize devolverá el valor 10.

- **Attributes.** Constante que especifica las características de un objeto, veámoslo en la Tabla 75.

Constante	Descripción
AdFldMayDefer	Indica que el campo se aplaza, es decir, los valores del campo no se recuperan del origen de datos con todo el registro, sino solamente cuando se tiene acceso explícito a los mismos.
adFldUpdatable	Indica que se puede escribir en el campo.
adFldUnknownUpdatable	Indica que el proveedor no puede determinar si se puede escribir en el campo.
AdFldFixed	Indica que el campo contiene datos de longitud fija.
adFldIsNullable	Indica que el campo acepta valores Null.
adFldMayBeNull	Indica que se pueden leer valores Null del campo.
AdFldLong	Indica que se trata de un campo binario largo. También indica que se pueden utilizar los métodos AppendChunk y GetChunk.
AdFldRowID	Indica que el campo contiene un identificador de fila persistente en el que no se puede escribir y que no tiene ningún valor significativo excepto la identificación de la fila (como por ejemplo un número de registro, un identificador único, etc.).
adFldRowVersion	Indica que el campo contiene algún tipo de marca de hora o de fecha que se utiliza para efectuar actualizaciones.
adFldCacheDeferred	Indica que el proveedor almacena los valores del campo en la memoria caché y que las lecturas siguientes se efectúan en dicha memoria .

Tabla 75. Atributos para un objeto Field.

- **DefinedSize.** Número Long, con el tamaño definido para un campo.
- **Name.** Cadena con el nombre del objeto.
- **NumericScale.** Número de tipo Byte que indica la cantidad de posiciones decimales para los campos con valores numéricos.

- OriginalValue. Variant que contiene el valor del campo antes de que se efectuaran cambios sobre él.
- Precision. Dato Byte que indica la cantidad de dígitos usados para representar los campos con valores numéricos.
- Type. Tipo de datos para el campo. Consulte el lector la propiedad del mismo nombre en el objeto Parameter para una descripción detallada de los tipos disponibles.
- UnderlyingValue. Dato Variant, que contiene el valor del campo de la base de datos, este es el valor visible para una transacción y puede ser el resultado de una actualización efectuada por otra transacción reciente.
- Value. Mediante esta propiedad recuperamos o asignamos valor a un objeto Field.

Métodos

- AppendChunk. Permite añadir un texto de gran tamaño o datos binarios a un campo.
- GetChunk. Devuelve todo el contenido o parte de un objeto Field, que contiene datos binarios o un texto de gran tamaño. Este método es útil en situaciones en las que la memoria del sistema es limitada, y no podemos manejar un gran volumen de datos a un tiempo.
 - Sintaxis

```
vntVariable = oField.GetChunk(lTamaño)
```

 - Parámetros
 - lTamaño. Número Long que establece la cantidad de información a recuperar.

Property

Este objeto representa una propiedad dinámica de un objeto ADO.

Las propiedades de los objetos ADO pueden ser intrínsecas y dinámicas. Las intrínsecas son aquellas que pueden ser manipulables mediante la sintaxis habitual Objeto.Propiedad. Las dinámicas vienen definidas por el proveedor de datos y son accesibles a través de la colección Properties del objeto ADO correspondiente.

Propiedades

- Attributes. Constante que especifica las características de un objeto, estos atributos aparecen en la Tabla 76.

Constante	Descripción
adPropNotSupported	Indica que el proveedor no admite la propiedad.

adPropRequired	Indica que el usuario debe especificar un valor para esta propiedad antes de que se inicialice el origen de datos.
adPropOptional	Indica que el usuario no necesita especificar un valor para esta propiedad antes de que se inicialice el origen de datos.
AdPropRead	Indica que el usuario puede leer la propiedad.
AdPropWrite	Indica que el usuario puede establecer la propiedad.

Tabla 76. Atributos para un objeto Property.

- Name. Cadena con el nombre del objeto.
- Type. Tipo de datos para el objeto Property. Consulte el lector la propiedad del mismo nombre en el objeto Parameter para una descripción detallada de los tipos disponibles.
- Value. Mediante esta propiedad recuperamos o asignamos valor a un objeto propiedad.

Error

Este objeto nos informa de los errores producidos durante las operaciones de acceso a datos. Cada vez que se produce un error de este tipo, se crea un objeto Error, que se añade a la colección Errors del objeto Connection.

Propiedades

- Description. Cadena informativa con un breve comentario del error.
- NativeError. Número Long con el código de error específico del proveedor de datos con el que estamos trabajando.
- Number. Valor Long con el número exclusivo perteneciente a un objeto Error.
- Source. Cadena que contiene el nombre del objeto o aplicación que generó el error.
- SQLState. Cadena de cinco caracteres con el estado de SQL para un error, que sigue el estándar ANSI SQL.

Una vez vistos en detalle los objetos que componen el modelo ADO, pasemos a continuación a la parte práctica, en la que a través de diversos ejemplos, comprobaremos el modo de trabajo con cada uno de ellos.

El ADO Data Control

El medio más fácil y rápido para acceder a la información de una base de datos en Visual Basic, pasa por el empleo del control ADO Data, y controles enlazados a este.

El control ADO Data nos conectará a la base de datos mediante un proveedor OLE DB, y visualizará el contenido de los registros en los controles enlazados.

Para aquellos lectores que hayan programado utilizando DAO o RDO, el aspecto y modo de funcionamiento es igual que en las versiones de este control para tales modelos de datos, variando el apartado de configuración para la conexión a la base de datos y modo de recuperación de información de la misma.

A lo largo de este tema encontraremos ejemplos que utilizan una ruta de datos prefijada. En estos casos, el lector puede optar por crear dicha ruta igual que se describe en el ejemplo o utilizar una establecida por él. En este último caso también deberá, si es necesario, alterar las propiedades de los controles que componen el ejemplo, y que hacen referencia a la ruta en la que reside la base de datos utilizada, para evitar posibles errores. Por otro lado, veremos también ejemplos, que utilizando el Registro de Windows y mediante un procedimiento de lectura y escritura de información en el mismo, nos permitirán indicar la ruta de datos del programa, sin estar obligados a emplear una predeterminada.

El proyecto de ejemplo [ADOCtrl](#), muestra como sin escribir ni una línea de código, podemos crear un mantenimiento para una tabla de una base de datos. Un mantenimiento limitado, eso sí, pero piense el lector por otra parte que todo se ha elaborado sin utilizar el editor de código, sólo empleando el editor de formularios, el cuadro de herramientas y la ventana de propiedades. Debemos tener en cuenta que la mayor potencia del control ADO Data reside en su sencillez de manejo. Gradualmente incorporaremos código para poder manejar los aspectos no accesibles directamente desde el control en modo de diseño.

En lo que respecta a las propiedades de los diferentes controles a emplear, asignaremos valor sólo a las que sea necesario, de forma que podamos comprobar cómo con un mínimo de configuración, podemos poner en marcha el programa.

En este mantenimiento abrimos la tabla Clientes del fichero DATOS.MDB, visualizando su contenido en los controles de un formulario, desde el que podremos modificar y añadir nuevos registros. Los pasos a seguir para desarrollar este proyecto son los siguientes:

Crear un nuevo proyecto de tipo EXE Estándar y asignarle el nombre ADOCtrl. Seleccionar la opción del menú de VB Proyecto+Referencias, para poder incorporar en este proyecto las referencias necesarias hacia las librerías de ADO: Microsoft ActiveX Data Objects 2.0 Library y Microsoft Data Binding Collection, que nos permitirán trabajar con objetos ADO y realizar el enlace de los mismos con controles. La primera de estas referencias será obligatoria en toda aplicación que vaya a manipular datos utilizando ADO. En la Figura 214 se muestran dichas referencias.

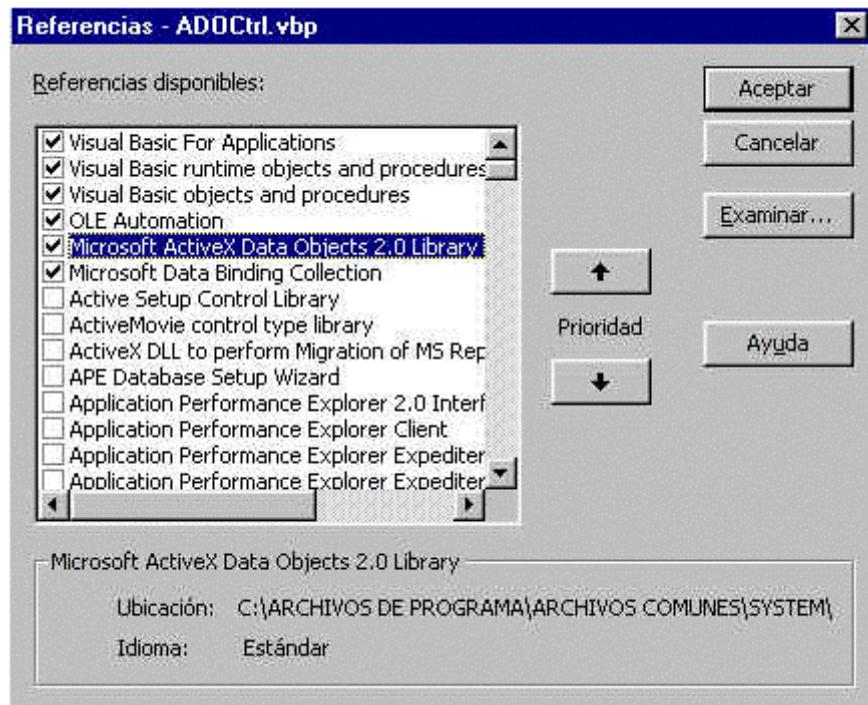


Figura 214. Establecimiento de referencias a las librerías de ADO en un proyecto.

Abrir el Cuadro de herramientas de VB, y pulsando sobre el mismo con el botón secundario del ratón, seleccionar de su menú contextual la opción Componentes, para agregar un nuevo control a este cuadro. Marcaremos el control Microsoft ADO Data Control 6.0 (OLEDB) tal y como muestra la Figura 215.

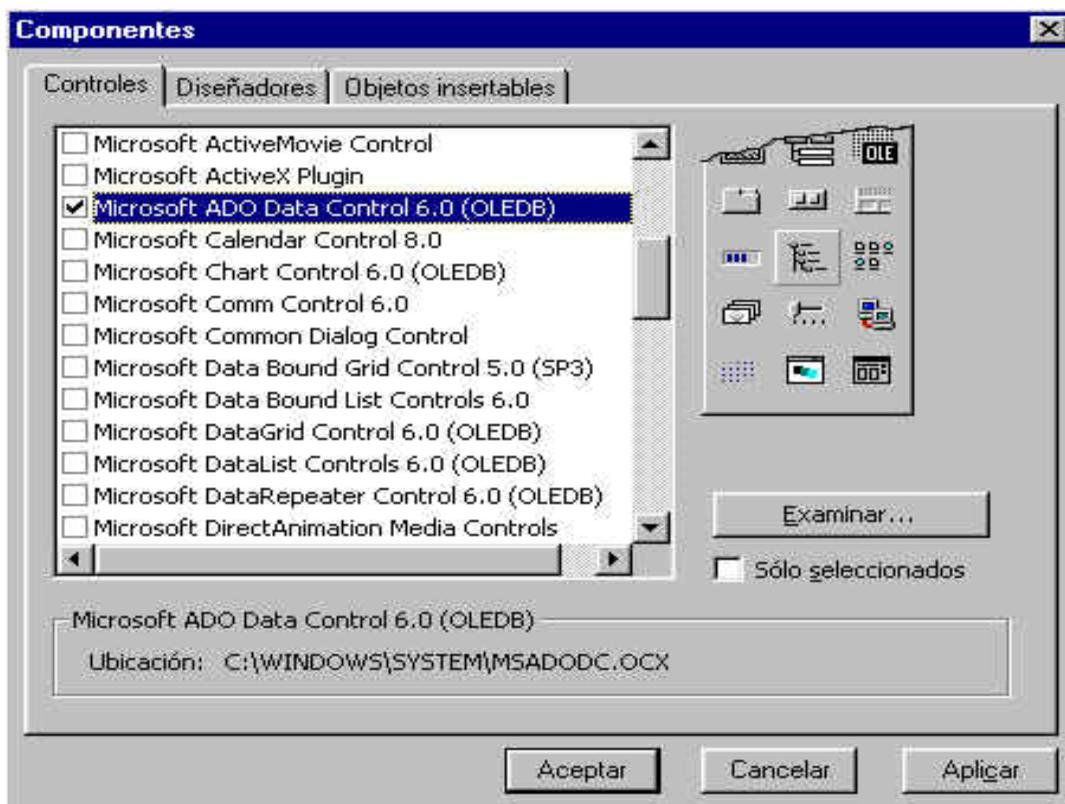


Figura 215. Selección del ADO Data Control para el Cuadro de herramientas.

Esta acción, incorporará el control en el Cuadro de herramientas, como se muestra en la Figura 216.



Figura 216. ADO Data Control incluido en el Cuadro de herramientas.

Insertar un control de este tipo en el formulario que viene por defecto en la aplicación o en un nuevo formulario que agreguemos. La Figura 217 muestra las operaciones de navegación por los registros efectuadas al pulsar los diferentes botones de este control.



Figura 217. Botones de navegación del control ADO Data.

Seleccionar el control insertado en el formulario, abrir su menú contextual y elegir la opción Propiedades de ADODC, que abrirá la ventana Páginas de propiedades para este control, como aparece en la Figura 218.

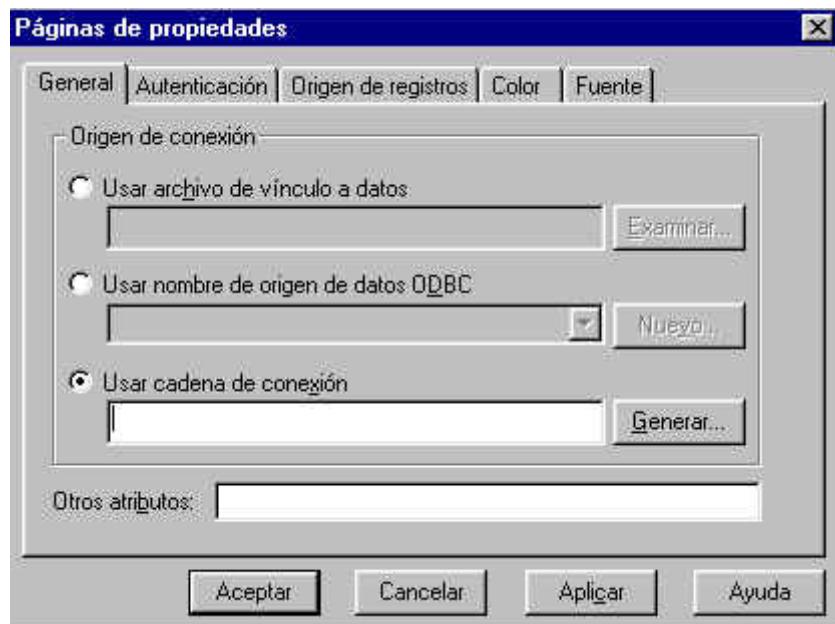


Figura 218. Páginas de propiedades del ADO Data Control.

En este punto debemos crear la conexión entre el control y la base de datos. Para ello, haremos clic sobre el OptionButton Usar cadena de conexión y pulsaremos el botón Generar para poder crear dicha cadena a través de un asistente que incorpora el control.

Se mostrará la ventana Propiedades de Data Link para crear la conexión. En su pestaña Proveedor deberemos seleccionar uno de los diferentes proveedores de OLE DB. En nuestro caso, ya que deseamos trabajar con una base de datos Access, elegiremos Microsoft Jet 3.51 OLE DB Provider, como se muestra en la Figura 219, y pulsaremos el botón Siguiente >>, para pasar a la pestaña Conexión.

Después del proveedor debemos introducir la ruta y nombre del fichero de base de datos con el que vamos a trabajar, en el primer TextBox de esta pestaña, o bien buscarlo pulsando el botón que se encuentra junto a este control. Adicionalmente, puede establecer un usuario y clave de acceso a la base de datos, pero en este caso no será necesario.

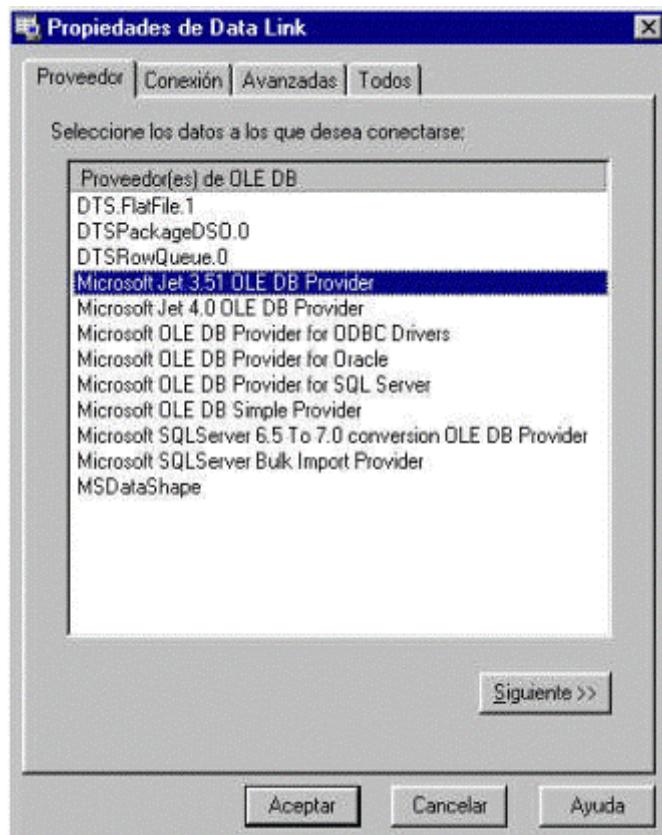


Figura 219. Selección del proveedor OLE DB para la conexión.

La ruta empleada en la Figura 220 es: C:\CursoVB6\Texto\DatosADO\ADOCtrl\Datos.mdb.

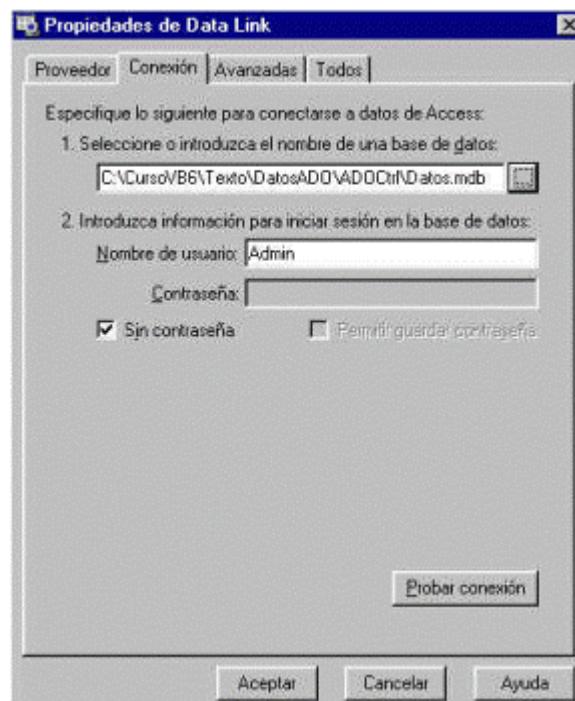


Figura 220. Ruta de la base de datos para la conexión.

Pulsando el botón Probar conexión, el lector podrá comprobar si la configuración de este paso es correcta y es posible establecer conexión con la base de datos, visualizando el resultado en una ventana de mensaje como la que aparece en la Figura 221.



Figura 221. Mensaje de comprobación de conexión desde el control ADO Data.

Activando la pestaña Avanzadas estableceremos entre otros valores los permisos de acceso a datos. Dejaremos el valor por defecto Share Deny None (como podemos ver en la Figura 222), para compartir los datos y permitir acceso a todos los usuarios.

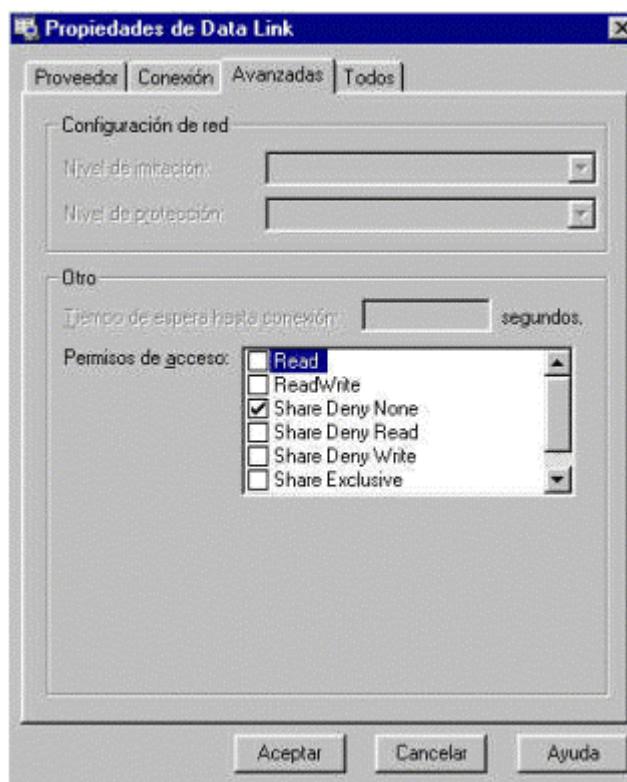


Figura 222. Permisos de acceso para la base de datos.

Finalmente, la pestaña Todos, como su nombre indica, nos permitirá visualizar al completo la totalidad de valores configurados para la conexión, modificando cualquiera de ellos al pulsar el botón Editar valor, como vemos en la Figura 223.

Pulsando Aceptar, finalizará la creación de la cadena de conexión que será depositada en la ventana de propiedades del control ADO Data. Esta cadena presentará variaciones dependiendo del proveedor de datos utilizado, para este caso será algo parecido a la siguiente:

```
Provider=Microsoft.Jet.OLEDB.3.51;Persist Security Info=False;Data
Source=C:\CursoVB6\Texto\DatosADO\ADOCtrl\Datos.mdb
```

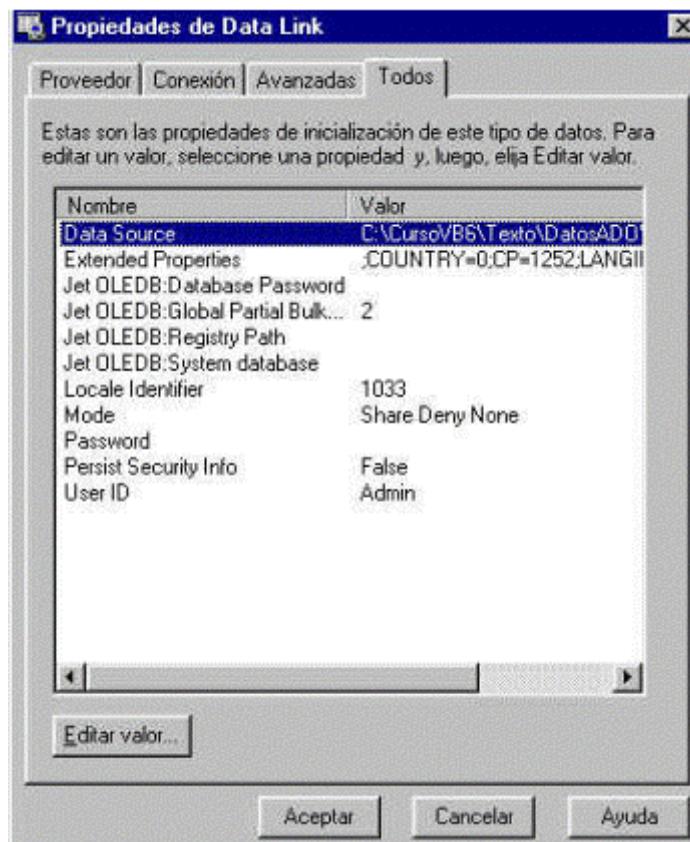


Figura 223. Valores al completo para la conexión con la base de datos.

Continuando con las páginas de propiedades de este control, la pestaña Autenticación contiene la información de claves de acceso a la base de datos, que aquí estará vacía debido a que no vamos a solicitar dichas claves. Pasaremos pues a la pestaña Origen de registros (Figura 224), en la cual estableceremos la forma y el contenido que vamos a solicitar de la base de datos.



Figura 224. Establecer origen de registros para el ADO Data.

En primer lugar, debemos seleccionar de la lista desplegable Tipo de comando, el modo en que vamos a recuperar los datos. Seleccionaremos adCmdTable, ya que vamos a acceder a una tabla. En la siguiente lista, elegiremos la tabla o procedimiento almacenado a consultar: Clientes. Si necesitáramos ejecutar una sentencia SQL contra una o varias tablas de la base, emplearíamos el TextBox Texto del comando (SQL), en esta ocasión está deshabilitado ya que el tipo de comando que vamos a ejecutar, se basa en una tabla física y no una consulta de selección. Pulsando Aceptar, finalizaremos la configuración de la conexión a la tabla de datos.

Un control ADO Data al llegar al último de los registros, permanece por defecto en él aunque sigamos pulsando su botón de avance. Para poder agregar nuevos registros a la tabla, modificaremos la propiedad EOFAction del control en la ventana de propiedades, asignándole el valor adDoAddNew, como muestra la Figura 225, que añadirá un nuevo registro a la tabla cuando al estar situados en el último registro, pulsemos de nuevo el botón de avance. Con esto finalizamos la configuración del control.

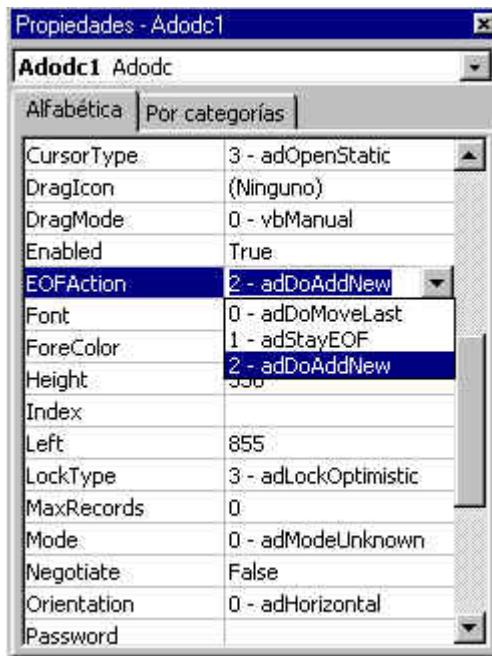


Figura 225. Valores disponibles para la propiedad EOFAction del ADO Data.

Ahora el control está listo para recuperar información de la tabla, pero por sí mismo no puede mostrar datos al usuario, por este motivo se hace necesario incluir controles que puedan enlazarse al ADO Data y mostrar la información que este contiene.

Incluiremos pues, desde el cuadro de herramientas, un control TextBox y un Label por cada campo de la tabla Clientes. En la propiedad Caption de los Label pondremos el nombre de cada campo y en los TextBox modificaremos las siguientes propiedades:

- Name. Un nombre identificativo del campo que va a mostrar el TextBox.
- DataSource. Esta propiedad abre una lista con todas las fuentes de datos definidas en el proyecto, en nuestro caso sólo disponemos de uno, que es el que seleccionamos.

- DataField. Esta propiedad abre una lista con los campos disponibles en la fuente de datos establecida en la propiedad DataSource del control enlazado. Lo que haremos aquí será seleccionar un campo diferente para cada TextBox.
- MaxLength. Aquí definimos el máximo número de caracteres que admite el TextBox, si no usáramos esta propiedad e intentáramos por ejemplo poner más de dos dígitos en el campo del código de cliente, VB nos avisaría que esto no es posible y no permitiría agregar el nuevo registro hasta que los datos fueran correctos.

Una vez hecho esto, ya hemos finalizado el trabajo, iniciamos la aplicación que tendrá un aspecto como el de la Figura 226.



Figura 226. Mantenimiento de datos usando el control ADO Data.

Ahora podemos utilizar el control de datos para navegar por los registros de la tabla, modificar el registro que aparece en el formulario o añadir nuevos registros desde el final de la tabla. Los controles TextBox serán llenados automáticamente con la información del registro actual, y todo esto se ha realizado utilizando básicamente el ratón.

Por supuesto que una aplicación real, siempre requerirá un tipo de validación a la hora de editar datos, que nos obligue a utilizar algo más que el simple control, pero puede darse el caso de que necesitemos desarrollar un programa en el que sea necesaria la introducción inmediata de datos, mientras se desarrollan los procesos definitivos de entrada de datos.

Podemos utilizar formularios de este estilo para que los usuarios puedan comenzar a introducir información, siempre que el diseño del programa y base de datos lo permitan.

Para finalizar con este apartado, supongamos que en lugar de acceder a la totalidad de la tabla, sólo necesitamos manejar una selección de sus registros. Esto lo conseguiremos, abriendo la ventana de páginas de propiedades del control ADO Data, y en la pestaña Origen de registros, como tipo de comando seleccionar adCmdText, con lo que se deshabilitará esta vez la lista dedicada a la tabla o procedimiento almacenado y se habilitará el TextBox para introducir una instrucción SQL, en el que podremos escribir un comando como el mostrado en la Figura 227, que además de seleccionar un rango de registros, los ordenará en función a un campo.

Efectúe el lector estos cambios en este proyecto, para comprobar como el control sólo nos ofrece el subconjunto de registros indicados en la sentencia SQL.



Figura 227. Seleccionar registros en el ADO Data mediante una instrucción SQL.

Validaciones a nivel de motor

Pongámonos en el caso de que en el anterior programa no se debe dar un valor inferior a "10" para el campo código de cliente, si queremos conseguirlo sin tener que escribir una rutina de validación en el propio programa, podemos emplear una regla de validación a nivel de motor, que consiste en que el propio motor de datos, en este caso Access, se encarga de comprobar los valores a introducir en un registro, si no se cumple el criterio de validación establecido, no se grabará el registro.

Abrimos en Access la base de datos Datos.MDB y la tabla Clientes en modo diseño, nos situamos en el campo CodCli, e introducimos la validación tal y como se muestra en la Figura 228.

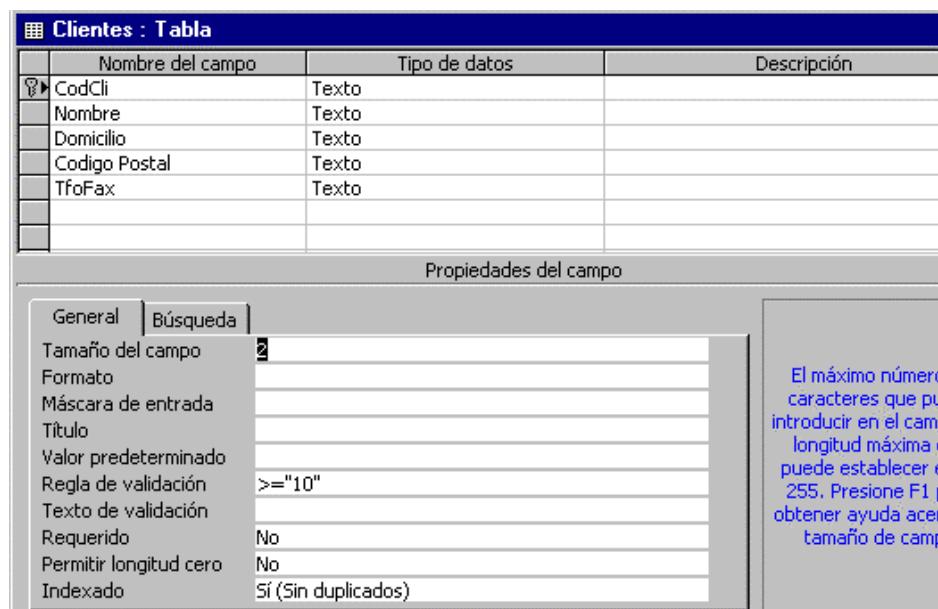


Figura 228. Regla de validación establecida en Access para un campo de una tabla.

Grabamos los cambios realizados en la tabla, cerramos Access y volvemos a ejecutar el ejemplo. Ahora si intentamos crear un nuevo registro, dando al campo de código el valor "06" por ejemplo, nos encontraremos con el aviso que muestra la Figura 229.



Figura 229. Aviso del motor de datos, por haber transgredido una regla de validación.

No podremos grabar el registro hasta que no cumplamos la regla de validación.

Es posible incluir reglas de validación para todos los campos de la tabla, y aunque la validación mediante una rutina de código siempre será más potente y flexible, este medio de controlar la grabación de los datos siempre puede sernos útil en validaciones sencillas.

El control DataGrid

El mantenimiento del ejemplo anterior tiene un defecto muy importante, y es que el usuario puede necesitar una visión global de los registros de la tabla y no tener que verlos uno a uno. Para solventar ese problema disponemos del control DataGrid o cuadrícula, que muestra los datos en forma de tabla, con lo cual podemos tener de un sólo vistazo, una mayor cantidad de información y navegar de forma más rápida por las filas.

La aplicación de ejemplo [CtlDataGrid](#), proporciona un mantenimiento similar al visto en el apartado dedicado al ADO Data, pero utilizando un DataGrid en lugar de los TextBox para editar la información de los campos de un registro.

La ruta para la base de datos utilizada en este ejemplo es:

C:\CursoVB6\Texto\DatosADO\CtlDataGrid, por lo que el lector deberá de crear una igual o modificar este aspecto en el control ADO Data del ejemplo, en la parte de las propiedades que genera la cadena de conexión con la base de datos.

En el caso de que este control no se encuentre en el cuadro de herramientas, deberemos incorporarlo mediante la ventana Componentes, al igual que hacímos en el apartado anterior con el control ADO Data, seleccionando en esta ocasión el control denominado *Microsoft DataGrid Control 6.0* (OLEDB), como podemos ver en la Figura 230.

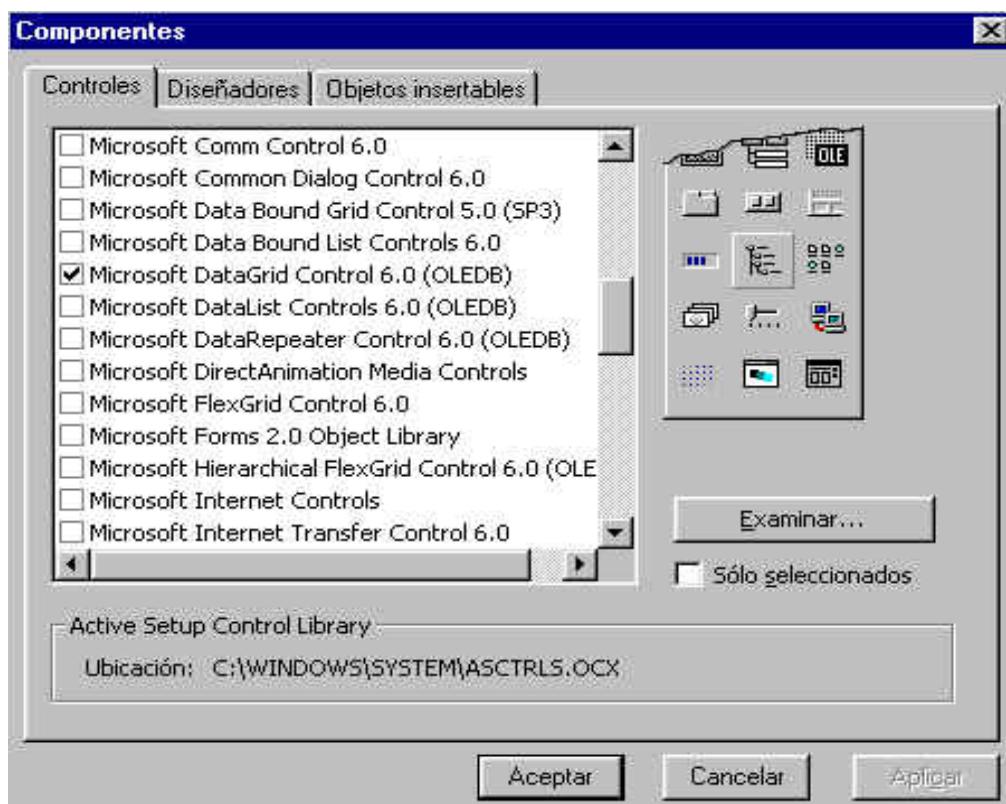


Figura 230. Selección del control DataGrid en la ventana Componentes.

Después de agregar este control, el cuadro de herramientas presentará un aspecto parecido al de la Figura 231.



Figura 231. Control DataGrid incluido en el cuadro de herramientas.

Los pasos para crear el formulario del programa y el control ADO Data, son iguales que en el anterior ejemplo, por lo que no los repetiremos aquí. Una vez llegados a este punto, insertaremos un control DataGrid de la misma forma que lo hacemos con otro control.

Para obtener resultados inmediatos, sólo tenemos que asignar a la propiedad **DataSource** del DataGrid, el nombre del control ADO Data y ejecutar la aplicación, de esta manera se visualizarán los registros en el control DataGrid, como se puede ver en la Figura 232.

Sin embargo, en muy pocas ocasiones nos bastará con mostrar este control con sus valores por defecto. Para alterar el aspecto de un DataGrid, podemos utilizar las ventanas de propiedades y páginas de propiedades de este control, o bien desde el código de la aplicación, manipular el aspecto del control antes de presentarlo al usuario. En este apartado lo haremos en modo de diseño.



Figura 232. Aplicación de mantenimiento de datos con DataGrid.

En primer lugar, seleccionaremos el control DataGrid del formulario y abriremos su menú contextual, eligiendo la opción *Recuperar campos*; de esta forma, la información de los campos de la tabla que contiene el control ADO Data pasará a la cuadrícula y podrán ser modificados.

Nuevamente abriremos el menú contextual del DataGrid, seleccionando la opción *Propiedades*, que mostrará la ventana de páginas de propiedades con varias pestañas para configurar este control. Entre el numeroso grupo de propiedades, vamos a comentar algunas de las utilizadas en este ejemplo, agrupadas por la página que las contiene.

- General.
 - Caption. Cadena para establecer un título en el DataGrid.
 - RowDividerStyle. Constante con las diferentes formas de división para las filas del control. En este caso no se visualizarán líneas divisorias entre las filas al emplear la constante `dbgNoDividers`.
 - AllowAddNew. Valor lógico para permitir al usuario añadir nuevos registros directamente en el DataGrid.
 - AllowUpdate. Valor lógico para permitir al usuario modificar los registros existentes en el control.
 - ColumnHeaders. Valor lógico para mostrar u ocultar los títulos de las columnas.
- Teclado.
 - AllowArrows. Valor lógico para poder efectuar la navegación por las celdas del DataGrid utilizando las teclas de movimiento del cursor.

- TabAction. Constante que establece el comportamiento de la tecla TAB cuando el DataGrid tiene el foco, para poder mover entre las celdas del control o pasar al resto de controles del formulario.
- Columnas.
 - Column. Columna del control a configurar.
 - Caption. Título de la columna.
 - DataField. Campo del control ADO Data que proporciona los datos y que se va a visualizar en el DataGrid.
- Diseño.
 - AllowSizing. Permite o impide variar el ancho de una columna.
 - Visible. Muestra u oculta una columna.
 - Button. Muestra un botón en la parte derecha de la celda, para columnas en las que cada celda necesite seleccionar entre un conjunto de valores.
 - Width. Establece el ancho de la columna.
- Divisiones.
 - AllowRowSizing. Permite modificar la altura de las filas.
 - RecordSelectors. Muestra la columna situada en primer lugar a la izquierda del control con los marcadores de fila.
 - MarqueeStyle. Permite establecer el aspecto de las filas normales y las que tienen el foco de edición.
- Formato.
 - Tipo de formato. Ofrece una lista de los formatos disponibles para aplicar a las diferentes columnas del control.

Después de algunos cambios en estas propiedades, ejecutaremos de nuevo el ejemplo, que nos mostrará de nuevo el DataGrid con un aspecto diferente de su configuración original (Figura 233).



Figura 233. Formulario con DataGrid modificado.

Asistente para crear aplicaciones

Visual Basic incorpora un estupendo asistente para generar aplicaciones, en el que podremos, a través de una serie de pasos, crear el esqueleto básico de un programa, permitiéndonos indicar el estilo del programa, formularios de datos, etc.

El proyecto [AsistApp](#), contiene el ejemplo de creación de un programa empleando esta utilidad, cuyos pasos reproducimos a continuación.

Al iniciar Visual Basic, o al seleccionar un nuevo proyecto, en el cuadro de diálogo de selección del tipo de proyecto, elegiremos Asistente para aplicaciones de VB, como muestra la Figura 234.

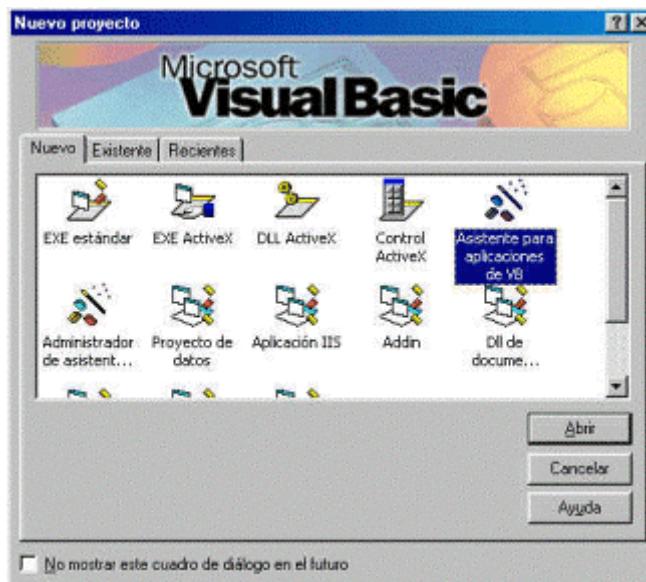


Figura 234. Seleccionar el asistente para aplicaciones.

Seguidamente se pondrá en marcha este asistente, que a través de los siguientes pasos, nos permitirá crear este programa.

En primer lugar aparece la parte introductoria del asistente. Cada vez que completemos una de estas fases, pulsaremos Siguiente para pasar a la próxima. Si en cualquier momento, necesitamos modificar algún aspecto de pasos anteriores, deberemos pulsar Anterior hasta llegar a dicho paso. Pulsando Cancelar, finalizaremos sin completar el asistente y el botón Ayuda nos llevará al sistema de ayuda de VB sobre ese paso en concreto.

Si hemos guardado en un fichero de perfiles (.RWP) una ejecución anterior de este asistente, podemos seleccionar en este paso dicho fichero, en el que estarán guardados los valores seleccionados en los pasos del asistente, ahorrándonos el volver a seleccionarlos.

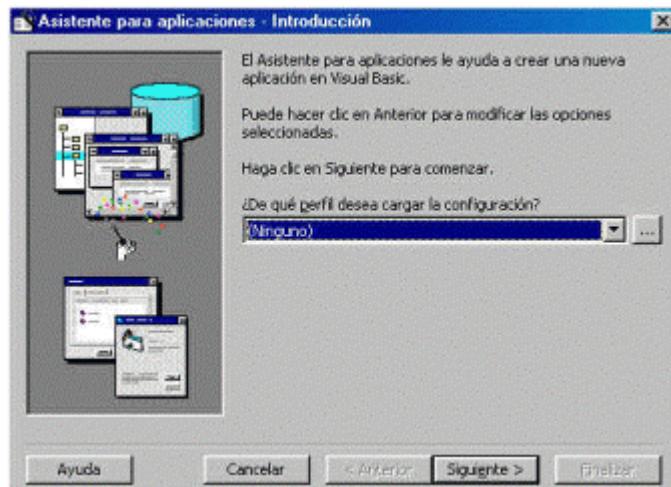


Figura 235. Introducción al asistente para aplicaciones.

A continuación se solicita el nombre de la aplicación y el tipo de interfaz que tendrá, como muestra la Figura 236.

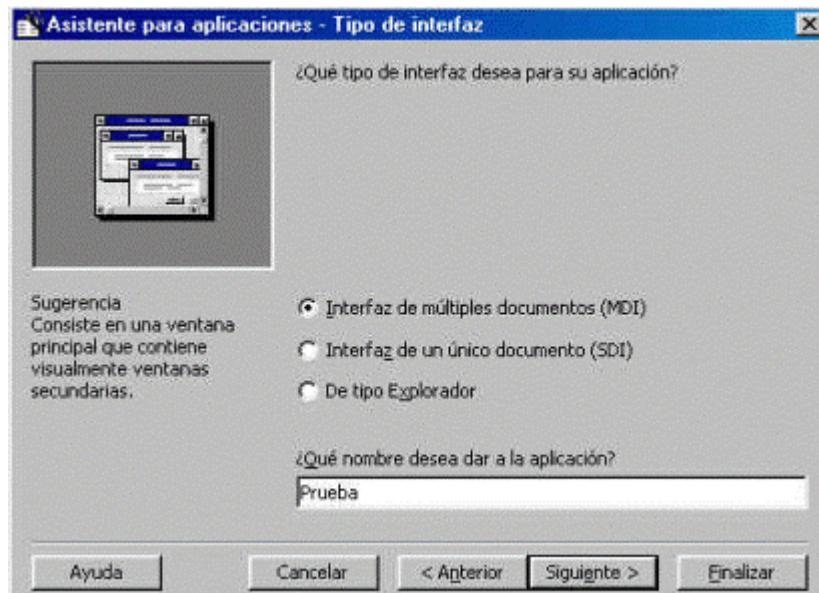


Figura 236. Nombre para la aplicación y tipo de interfaz.

Al elegir un interfaz MDI, dicha ventana MDI dispone de un grupo de menús y barra de herramientas predeterminados, en este paso, podemos alterar dichos valores por defecto, mostrando u ocultando los menús, y modificando también su orden.



Figura 237. Menús del programa.

El siguiente paso consiste en configurar la barra de herramientas de la ventana principal del programa. La lista izquierda muestra los botones disponibles, en la derecha aparecen los botones que forman la barra de herramientas. La fila de botones en la parte superior, es una representación de como se visualizarán los botones en ejecución. Si pulsamos alguno de estos botones, podemos cambiar el nombre que mostrarán y su imagen asociada.

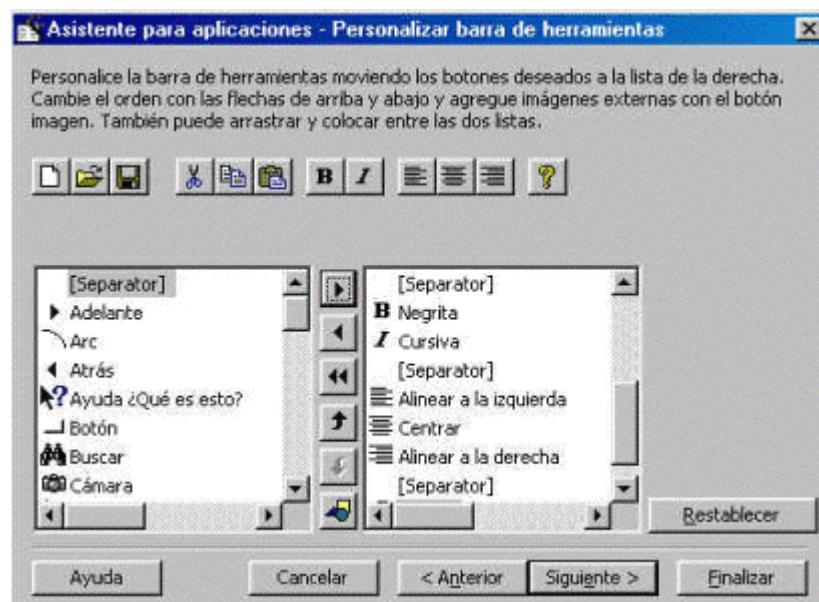


Figura 238. Configurar barra de herramientas del programa.

Si pensamos distribuir el programa en más de un idioma, debemos utilizar un fichero de recursos, que podemos especificar en este paso. Por defecto no utilizaremos este tipo de fichero.

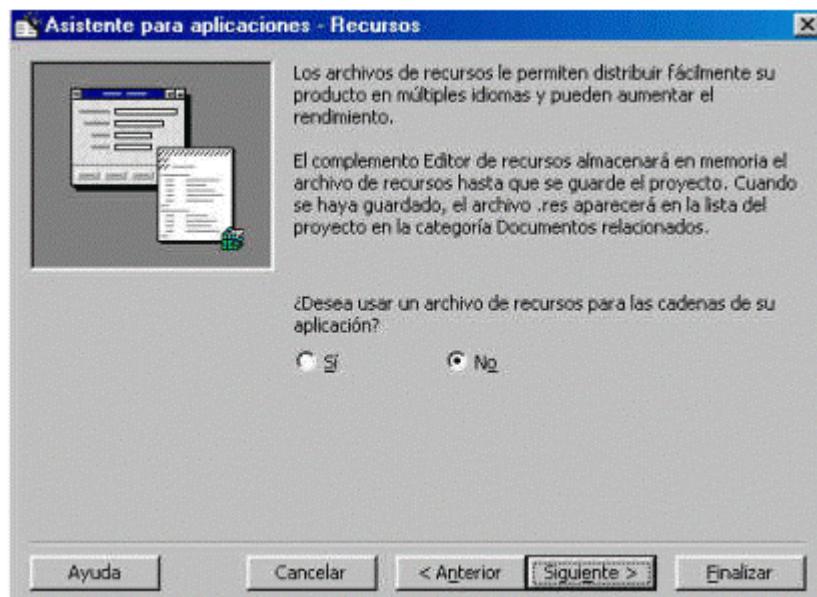


Figura 239. Adjuntar fichero de recursos a la aplicación.

Desde nuestro programa, podemos facilitar el uso de un navegador de Internet, que enlace con una página determinada. Estas opciones las podemos establecer en el siguiente paso del asistente.

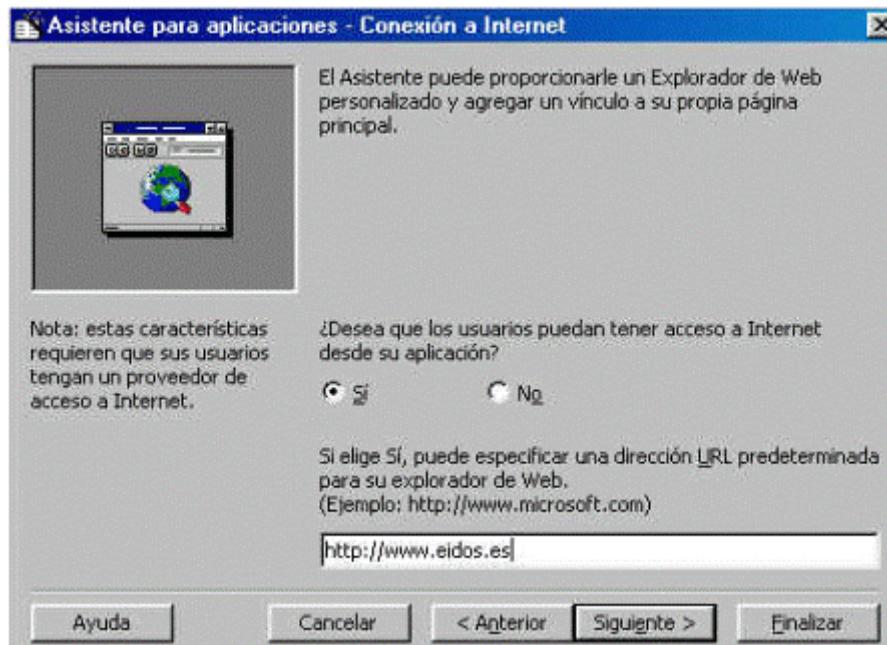


Figura 240. Opciones de Internet para el programa.

En el siguiente paso, indicaremos los formularios de tipo estándar que creará el asistente. Para este programa se creará un formulario de inicio y uno de tipo Acerca de...

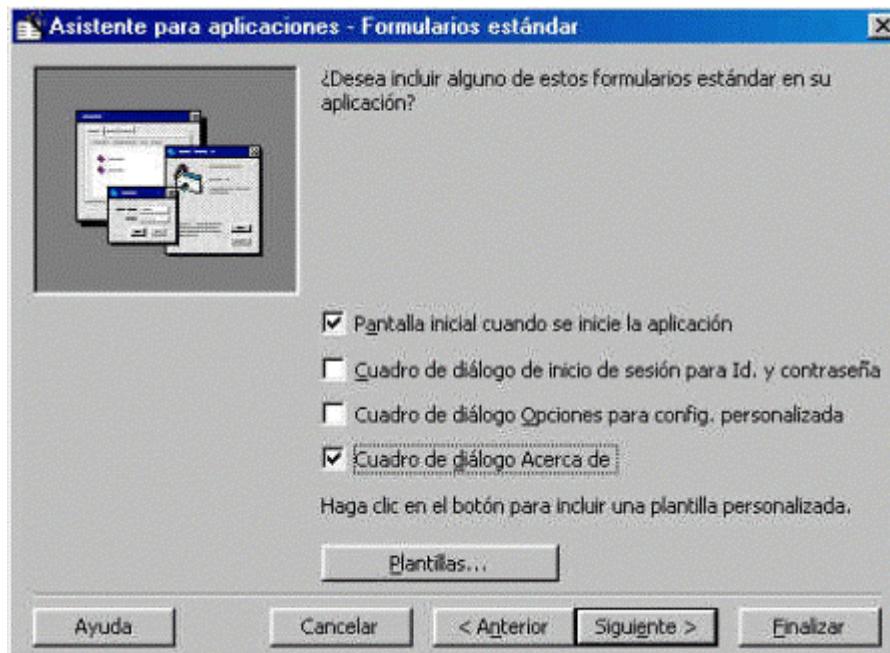


Figura 241. Establecer ventanas estándar para el programa.

En el caso de que necesitemos algún tipo de mantenimiento de datos, en el siguiente paso pulsaremos el botón Crear nuevo formulario, que iniciará el asistente para formularios de datos, otra utilidad que nos permite crear a través de una serie de sencillos pasos, formularios para el mantenimiento de una tabla de una base de datos.



Figura 242. Acceso al asistente para formularios de datos.



Figura 243. Pantalla inicial del asistente para formularios de datos.

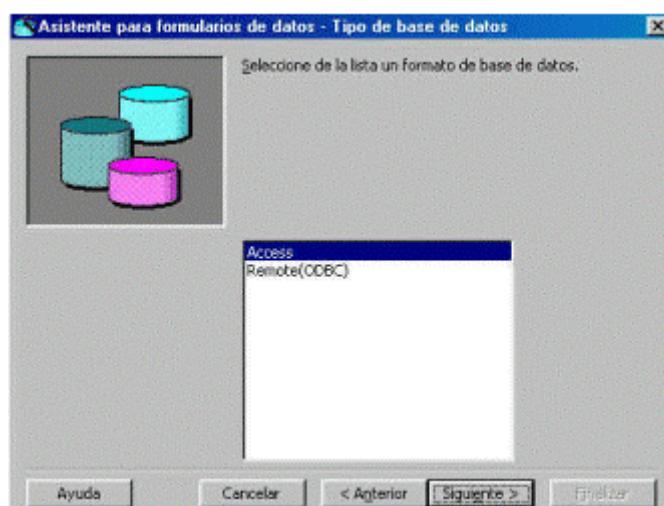


Figura 244. Selección del formato de base de datos.

Después de entrar en este asistente, debemos seleccionar el formato de base de datos a utilizar, como aparece en la Figura 244

A continuación escribiremos la ruta y el nombre de la base de datos a utilizar, o pulsaremos Examinar para localizarla.

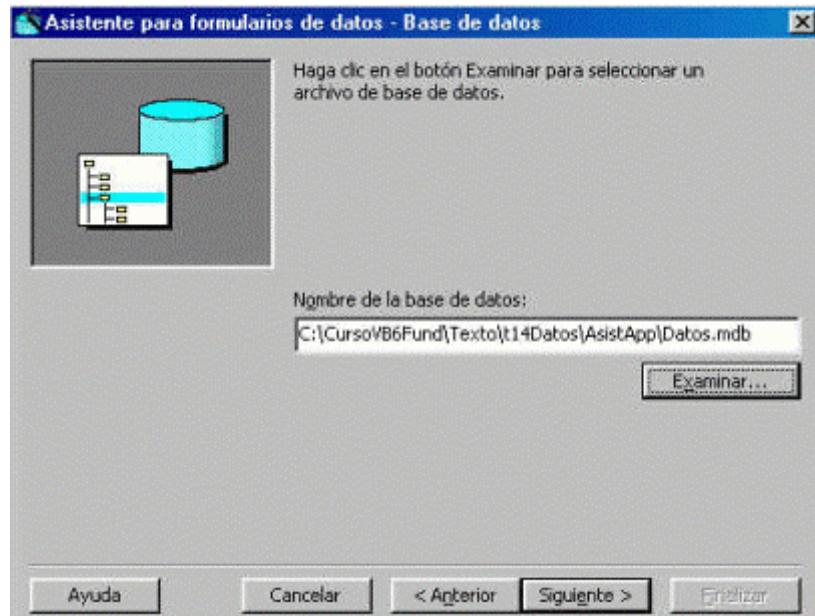


Figura 245. Selección del fichero que contiene la base de datos.

En el siguiente paso, escribiremos el nombre que tendrá el formulario en el que realizaremos el mantenimiento, seleccionando además, el modo de visualización de los datos: registro a registro, en cuadrícula, etc., y el modo en el que se realizará el enlace con los datos: control de datos, código, etc.

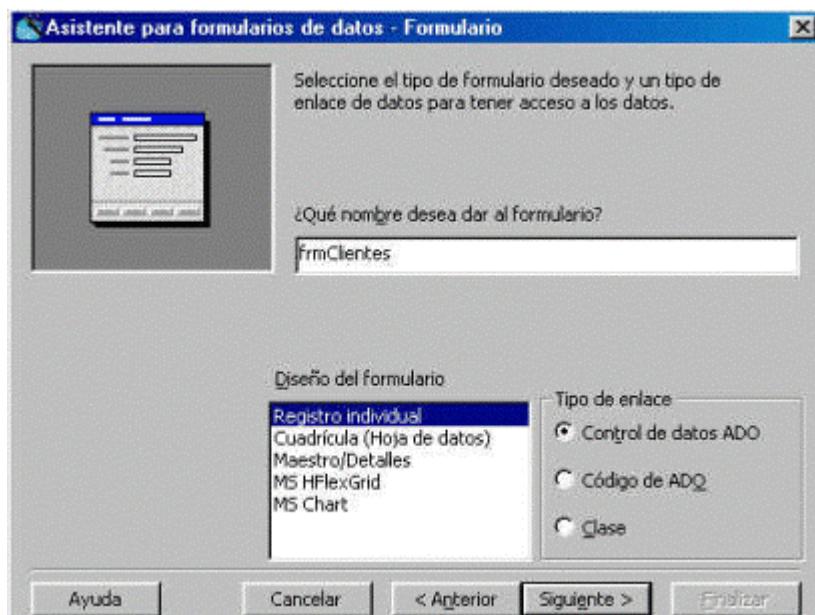


Figura 246. Información del formulario.

Después de la base de datos, tenemos que seleccionar qué tabla vamos a manipular y los campos correspondientes.

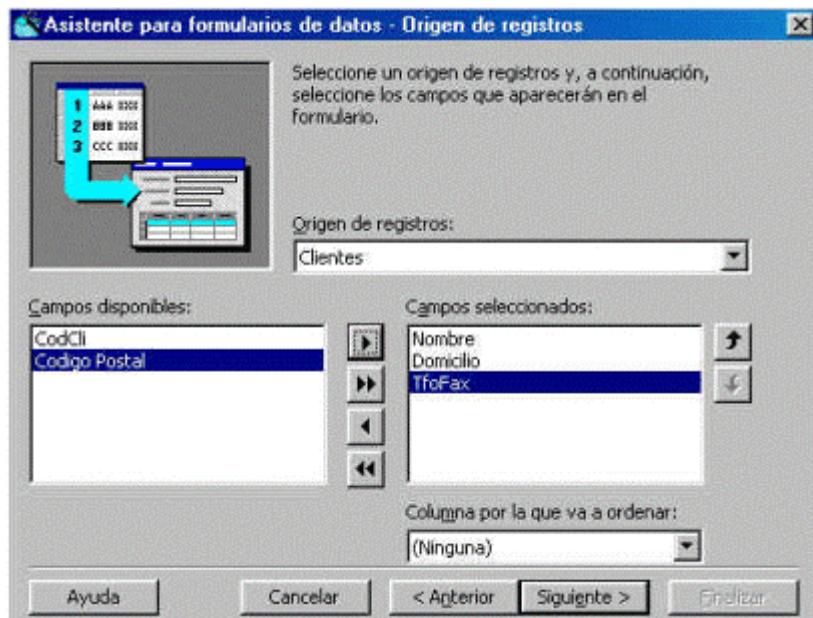


Figura 247. Tabla y campos a editar.

En el próximo paso, estableceremos las operaciones a realizar con los registros de un conjunto disponible. En nuestro caso, sólo vamos a permitir añadir y eliminar registros, como se observa en la Figura 248.

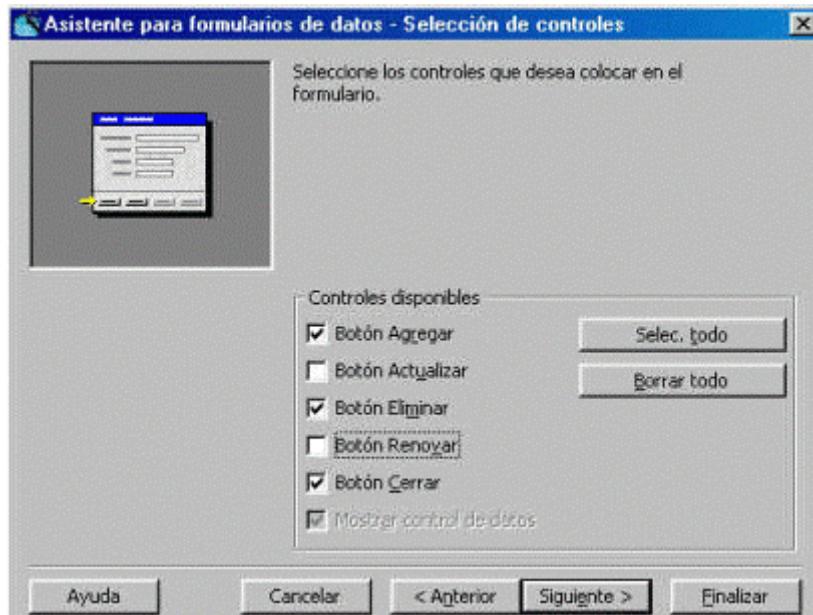


Figura 248. Establecer operaciones con los registros.

Finalizamos este asistente en el siguiente paso, en el que podemos grabar los pasos dados en un fichero de perfil (Figura 249).

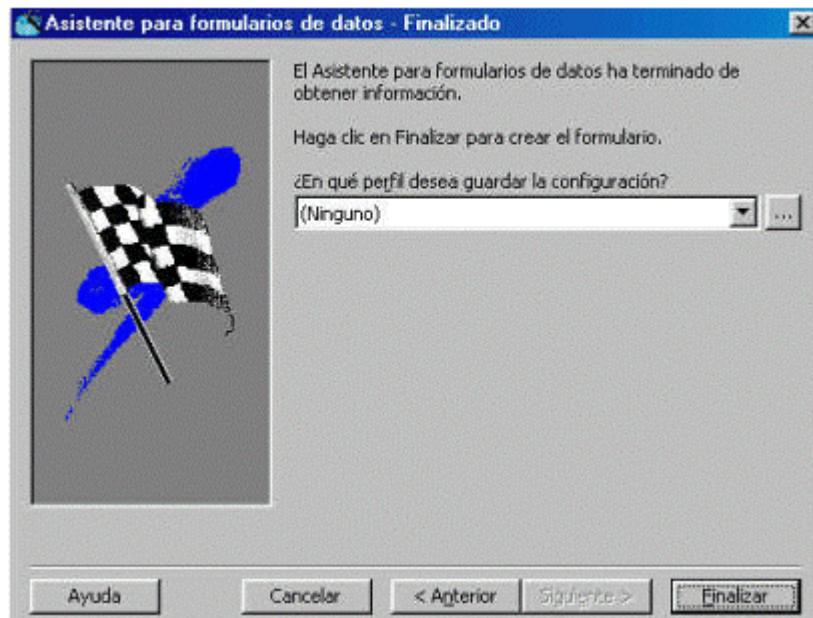


Figura 249. Final del asistente de creación de formularios de datos.

Al pulsar el botón Finalizar, se creará el formulario en el proyecto, preguntándonos el asistente si deseamos crear un nuevo formulario, a lo que responderemos No, volviendo al asistente de aplicaciones.

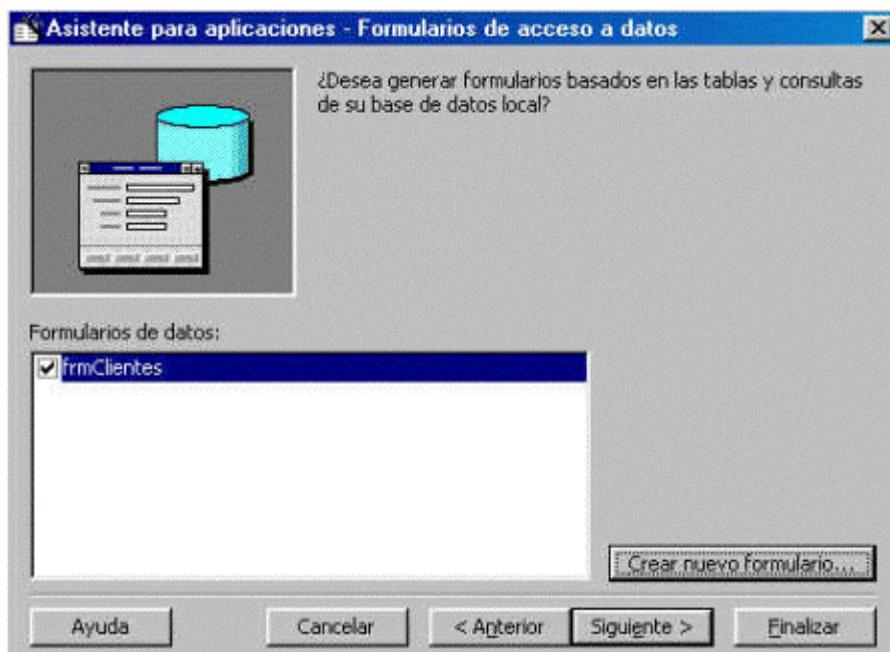


Figura 250. Asistente para aplicaciones con el formulario de datos creado.

En el siguiente paso de este asistente, finalizamos la creación del proyecto. Podemos guardar los pasos realizados en un fichero de perfil, y ver un informe de resumen. Al pulsar Finalizar, se crearán todos los elementos de la aplicación.



Figura 251. Final del asistente para aplicaciones.

Terminado el asistente, la aplicación ya está en condiciones de ser usada, o bien podemos añadirle nuevos elementos: formularios, controles y código, para completarla allá donde el asistente no ha llegado.

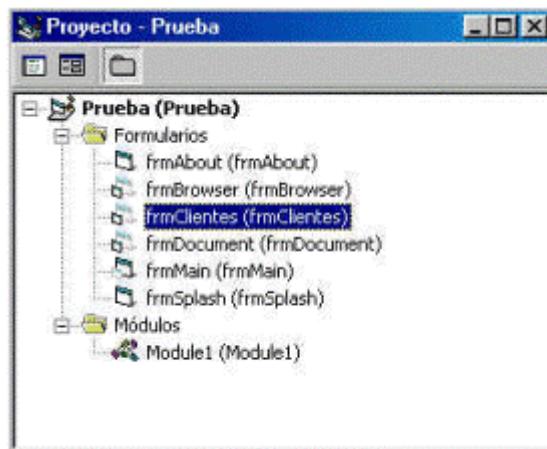


Figura 252. Explorador de proyectos de la aplicación creada mediante el asistente.

Para el lector que desarrolle este ejemplo, debemos comentar que hemos detectado un problema al intentar ejecutar la aplicación creada con el asistente, que consiste en un error en la referencia hacia la librería de ADO, puesto que el asistente no la incorpora automáticamente. Para ello, el lector debe abrir la ventana de Referencias (menú Proyecto+Referencias), y establecer la referencia hacia la librería Microsoft ActiveX Data Objects 2.0 Library, de modo que se pueda ejecutar el programa sin errores.

En el caso de que el lector sólo esté interesado en ejecutar el asistente para formularios de datos, dicho asistente no está accesible por defecto al instalar Visual Basic. Para hacerlo disponible, debemos seleccionar la opción de menú Complementos+Administrador de complementos, que nos mostrará la relación de asistentes y utilidades disponibles para la herramienta.

En dicha ventana, deberemos seleccionar la línea con el nombre VB6 Data Form Wizard, y en el apartado Comportamiento de carga, marcar la opción Cargado/Descargado. Esto hará que se cargue el asistente y esté disponible en el menú Complementos de VB.

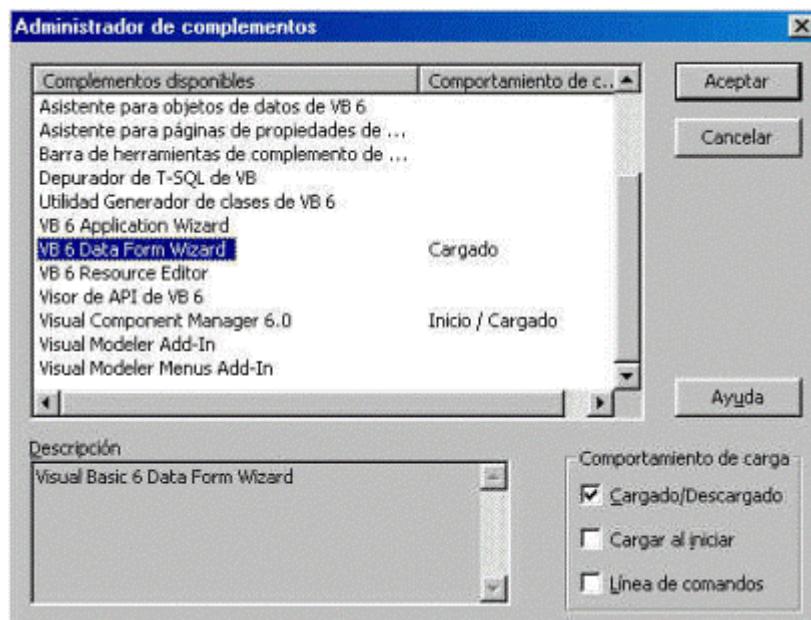


Figura 253. Selección del asistente para formularios de datos en el administrador de complementos.

Glosario de términos

Normativa de codificación

Bases para la escritura de código fuente y denominación de controles y objetos en los desarrollos.

- **Controles.** Comenzarán por el tipo de control, según la Tabla 77

Prefijo	Control
adc	Data (ADO)
Ani	Animación
bed	Pen Bedit
cbo	Combobox y dropdown Listbox
chk	CheckBox
clp	Picture clip
cmd	CommandButton
cll	Collection

col	Column
com	Comunicaciones
ctr	Control Genérico (parámetro)
dat	Data (DAO)
db	Base de datos ODBC
dir	Dir ListBox
dlg	CommonDialog
drv	Drive Listbox
Ds	Dynaset ODBC
flb	File Listbox
frm	Form
fra	Frame
gau	Barra gauge
gpb	Group Pushbutton
grd	Grid
hed	Pen Hedit
hsb	Horizontal Scrollbar
img	Imagen
iml	ImageList
ink	Pen Lnk
key	Keyboard key status
lbl	Label
lin	Line
lst	Listbox
mpm	MAPI message
mps	MAPI session
mci	MCI

mnu	Menu
opt	OptionButton
ole	OLE Client
pic	Picture
pnl	Panel 3D
shp	Shape
sbr	StatusBar
spn	Spin
tbr	ToolBar
txt	TextBox
tmr	Timer
vsb	Vertical Scrollbar

Tabla 77. Controles

Ejemplos:

- CommandButton: cmdAceptar
- TextBox: txtNombre
- CheckBox: chkRevisado
- **Menús.** Se usará el prefijo *mnu* seguido de un literal descriptivo de la opción, dependiendo del nivel en que se encuentre la opción.

Tomemos una estructura de menú como la siguiente:

Archivo
 Fichero
 Consultas -> Clientes
 Proveedores
 Salir

Los nombres a asignar a cada elemento del menú quedarían de la siguiente forma:

MnuArchivo
 mnuArFichero
 mnuArConsultas -> mnuArCoClientes
 mnuArCoProveedores
 mnuArSalir

Según lo visto, las opciones de menú situadas en el menú principal o primer nivel utilizarán el nombre completo de la opción. Las opciones en los niveles inferiores utilizarán las dos primeras letras del nivel superior seguidas del nombre de su opción.

- **Variables y funciones.** La estructura para denominar variables y funciones será la siguiente:

<ámbito><tipo><cuerpo>

- Valores para ámbito:

g	Global
L	Local
S	Static
m	Módulo
v	Variable pasada por valor
r	Variable pasada por referencia

Tabla 78. Valores para ámbito

- Valores para tipo:

b	Boolean
by	Byte
m	Currency 64 bit
d	Double 64 bit
db	Database
dt	Date+Time
f	Float/Single 32 bit
h	Handle
l	Long
n	Integer
c	String
u	Unsigned
ul	Unsigned Long

vnt	Variant
w	Word
a	Array
o	Objeto

Tabla 79. Valores para tipo

- Valores para tipos de datos:

- *RDO*

EN	Environment
CN	Connection
QY	Query
RSF	Resulset forward only
RSS	Resulset static
RSD	Resulset dynamic
RSK	Resulset keyset
RSB	Resulset client batch
RSN	Resulset no cursor

Tabla 80. Valores para tipo RDO

- *DAO*

WS	Workspace
RCF	Recordset forward only
RCS	Recordset snapshot
RCD	Recordset dynaset
RCT	Recordset table

Tabla 81. Valores para tipo DAO

- *ADO*

acn	Connection
acm	Command
ars	Recordset
apr	Parameter
afl	Field

Tabla 82. Valores para tipo ADO

- Para el cuerpo de la variable se utilizará WordMixing.

Ejemplos:

- Variable local, integer, para guardar un número de referencia.

`LnNumReferencia`

- Variable global, cadena de caracteres, que contiene el código de usuario de la aplicación.

`GcUsuarioApp`

- Variable a nivel de módulo, cadena de caracteres, que contiene el nombre de la aplicación.

`McAppNombre`

- Constantes.** Seguirán las mismas reglas que las variables para el ámbito y tipo. El cuerpo de la constante deberá ir en mayúsculas, y en vez de utilizar WordMixing, se utilizará el guión bajo "_" para separar los distintos componentes del cuerpo.

Ejemplo:

`gnNUMERO_LINEA`

Constante NUMERO_LINEA, global, de tipo integer



Si quiere ver más textos en este formato, visítenos en: <http://www.lalibreriadigital.com>. Este libro tiene soporte de formación virtual a través de Internet, con un profesor a su disposición, tutorías, exámenes y un completo plan formativo con otros textos. Si desea inscribirse en alguno de nuestros cursos o más información visite nuestro campus virtual en: <http://www.almagesto.com>.

Si quiere información más precisa de las nuevas técnicas de programación puede suscribirse gratuitamente a nuestra revista **Algoritmo** en: <http://www.algoritmodigital.com>. No deje de visitar nuestra revista **Alquimia** en <http://www.eidos.es/alquimia> donde podrá encontrar artículos sobre tecnologías de la sociedad del conocimiento.

Si quiere hacer algún comentario, sugerencia, o tiene cualquier tipo de problema, envíelo a la dirección de correo electrónico lalibreriadigital@eidos.es.