

Parte 1: Introducción al Desarrollo de Aplicaciones con Visual Basic

¿Qué es Visual Basic?

Visual Basic es un ambiente gráfico de desarrollo de aplicaciones para el sistema operativo Microsoft Windows. Las aplicaciones creadas con Visual Basic están basadas en objetos y son manejadas por eventos. **Visual Basic** se deriva del lenguaje Basic, el cual es un lenguaje de programación estructurado. Sin embargo, **Visual Basic** emplea un modelo de programación manejada por eventos.

Las Aplicaciones Procedurales

En las aplicaciones tradicionales o procedurales, es la aplicación quien controla que porciones de código se ejecuta, y la secuencia en que este se ejecuta. La ejecución de la aplicación se inicia con la primera línea de código, y sigue una ruta predefinida a través de la aplicación, llamando procedimientos según sea necesario.

Las Aplicaciones Manejadas por Eventos

En las aplicaciones manejadas por eventos, la ejecución no sigue una ruta predefinida. En vez de esto, se ejecutan diferentes secciones de código en respuesta a eventos. Los eventos se desencadenan por acciones del usuario, por mensajes del sistema o de otras aplicaciones. La secuencia de eventos determina la secuencia en que el código se ejecuta. Es por esto que la ruta que sigue el código de la aplicación es diferente cada vez que se ejecuta el programa.

Una parte esencial de la programación manejada por eventos es el escribir código que responda a los posibles eventos que pueden ocurrir en una aplicación. Visual Basic facilita la implementación del modelo de programación manejada por eventos.

¿Qué es un objeto?

Cada formulario (ventana), menú o control que se crea con Visual Basic es un módulo autocontenido llamado **objeto**. Los bloques básicos de construcción de una aplicación con Visual Basic son los objetos. Cada objeto tiene un conjunto de características y un comportamiento definido (**propiedades, métodos y eventos**) que lo diferencian de otros tipos de objeto. En otras palabras, un objeto formulario ha sido diseñado para cumplir determinada función en una aplicación, y no es lo mismo que un objeto menú.

Propiedades

El conjunto de datos que describen las características de un objeto se le conoce como sus **propiedades**. Para un formulario tenemos por ejemplo, las propiedades **BackColor** (color de fondo), **Height** (altura).

Algunas propiedades no solo determinan el aspecto que tiene el objeto, sino que además pueden determinar su comportamiento; por ejemplo, la propiedad **MaxButton** establece si el formulario tendrá o no el botón *Maximizar*. La presencia o ausencia de este botón determinará si el formulario se puede o no maximizar.

Métodos

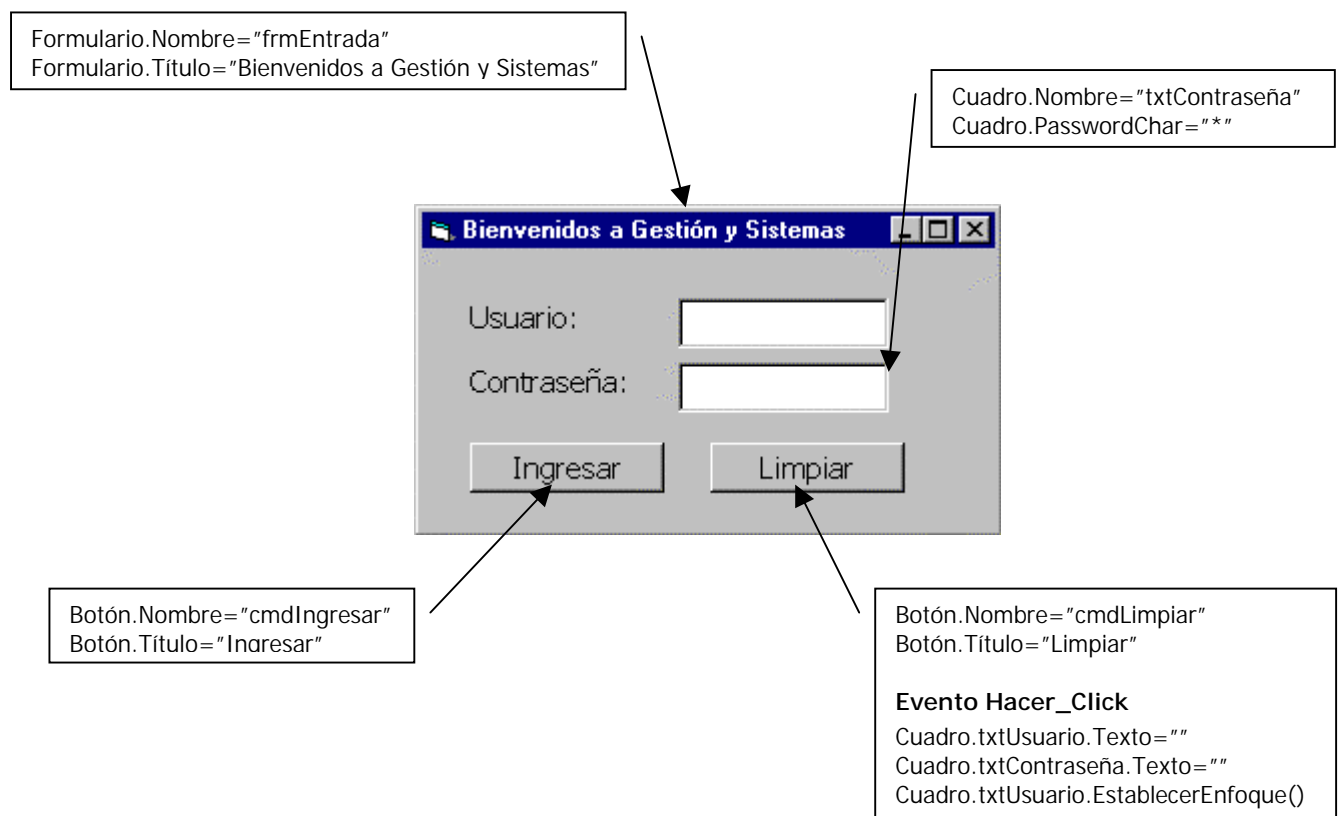
Los métodos son un conjunto de procedimientos que permiten que un objeto ejecute una acción o tarea sobre sí mismo. Por ejemplo, para un formulario tenemos el método **Hide** que hará que el formulario se oculte; o el método **Show** que hará que el formulario se vuelva a mostrar.

Eventos

Un **evento** es una acción que es reconocida por el objeto. Un evento ocurre (se dispara) como resultado de la interacción del usuario con el objeto. También puede dispararse debido a la ejecución de código (sentencias) o como resultado de la interacción de otro objeto con el objeto de poseedor del evento. Para un formulario tenemos por ejemplo; el evento **Load** que se dispara cuando se carga el formulario; o el evento **Click** para un botón de comando, se dispara cuando se hace clic sobre él.

¿Qué papel cumplen las propiedades, métodos y eventos?

Toda aplicación necesita una interfaz de usuario, la parte visual a través de la cual el usuario interactúa con la aplicación. Los bloques básicos de construcción de una interfaz de usuario son los formularios y los controles. Visual Basic utiliza técnicas de **programación visual** para diseñar las aplicaciones.



Para diseñar esta ventana (Formulario), del conjunto de objetos de Visual Basic seleccionamos un objeto tipo **Formulario** (Form). Luego a la propiedad **Nombre** (Name) le asignamos el valor **frmEntrada**; a la propiedad **Título** le asignamos el valor **Bienvenidos a Gestión y Sistemas**.

Dentro del formulario se colocan los controles. Para que el usuario pueda ingresar un dato (por ejemplo, la contraseña) colocamos en la ventana un control tipo **Cuadro de Texto** (TextBox); a continuación establecemos su propiedad **Nombre** en **txtContraseña**, y su propiedad **PasswordChar** es un *

(asterisco) para que el dato ingresado sea reemplazado por asteriscos sólo en la pantalla, para que no se pueda visualizar.

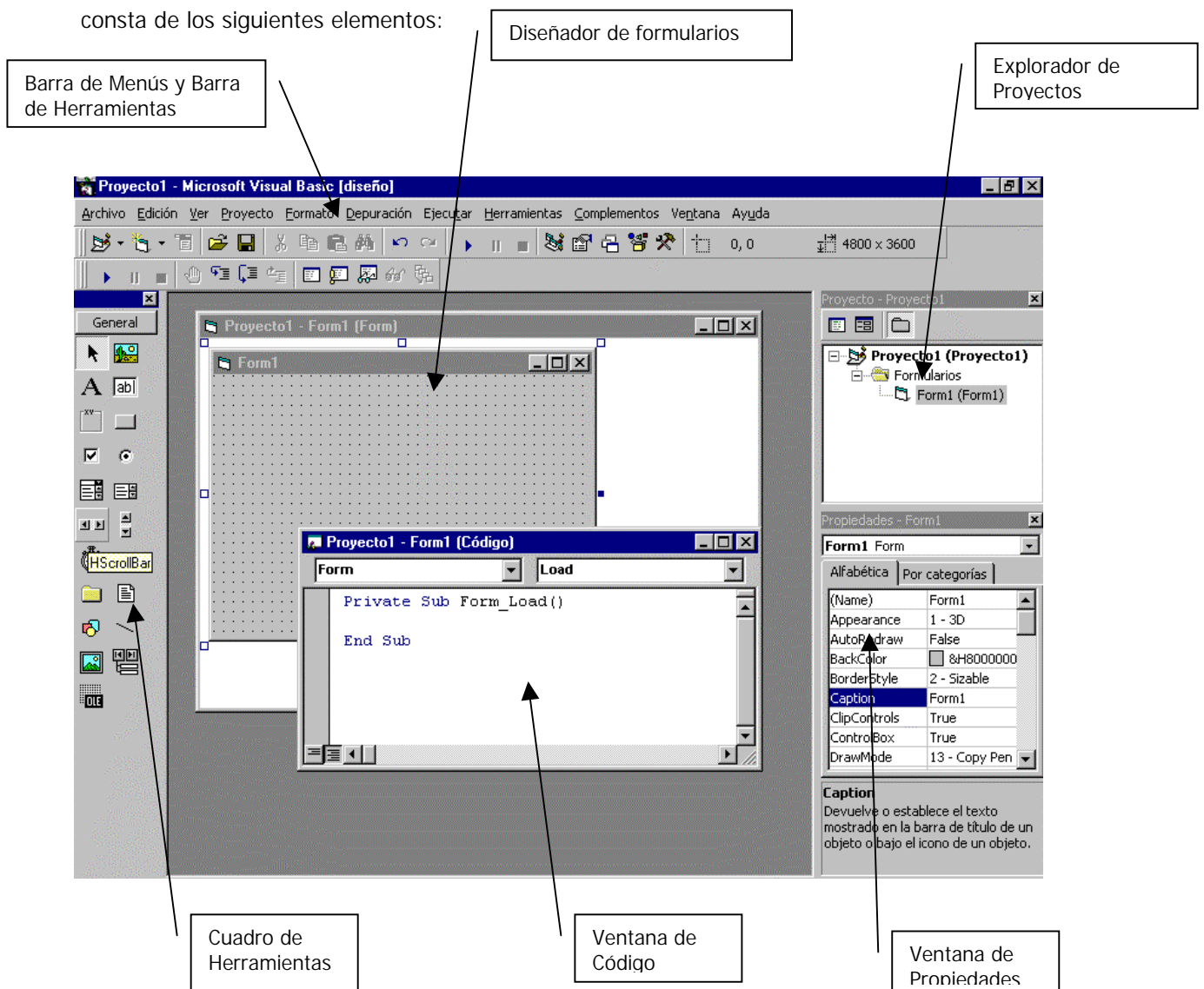
Para obtener el botón *Ingresar* seleccionamos un control tipo **Botón de Comando** (CommandButton), y lo colocamos en la ventana. Luego cambiamos sus propiedades; a la propiedad **Nombre** le asignamos **cmdIngresar**, y a la propiedad **Título** le asignamos **Ingresar**.

Se desea que cuando el usuario haga clic en el botón **Limpiar**, se borre cualquier dato que el usuario haya ingresado en los **Cuadros de Texto**, y que el punto de inserción se ubique en el cuadro **txtUsuario**. Para que esto ocurra debemos programar el evento **Hacer_Click** del botón de comando **cmdLimpiar**.

El evento debe ejecutar dos sentencias para cambiar la propiedad **Texto** de cada uno de los cuadros de texto, y luego invocar al método **EstablecerEnfoque()** del cuadro de texto **txtUsuario**.

El Entorno Integrado de Desarrollo (IDE)

Cuando se inicia Visual Basic, se crea un proyecto nuevo con un formulario. El IDE de Visual Basic consta de los siguientes elementos:



Barra de Menús

Presenta los comandos que se usan para trabajar con Visual Basic. Además de los menús estándar **Archivo**, **Edición**, **Ver**, **Ventana** y **Ayuda**, contiene otros menús para tener acceso a funciones específicas de programación, como **Proyecto**, **Formato** o **Depuración**.

Barra de Herramientas

Permite un acceso directo (solo un clic) a muchas de las operaciones más frecuentes utilizadas durante el desarrollo de aplicaciones.

Cuadro de Herramientas

Contiene todos los objetos y controles que se pueden añadir a los formularios para crear aplicaciones.

Diseñador de Formularios

Funciona como una ventana en la que se puede personalizar el diseño de la interfaz de usuario (ventana) de una aplicación.

Explorador de Proyectos

Lista de los archivos (formularios, módulos, etc.) del proyecto actual. Un **Proyecto** es una colección de archivos que utiliza para construir una aplicación.

Ventana de Propiedades

Lista los valores de las propiedades del formulario o control seleccionado que pueden ser modificados durante el diseño del formulario o control.

Ventana de Código

Funciona como un editor para escribir el código (sentencias) de la aplicación.

Obtención de Ayuda del Sistema

Visual Basic proporciona una variedad de recursos para ayudarle a encontrar la información que necesite cuando se encuentre trabajando dentro del entorno de desarrollo.

Ayuda en línea

Visual Basic proporciona una amplia ayuda en línea. El archivo de Ayuda contiene mucho código de ejemplo que se puede copiar directamente a una aplicación.

La ayuda de Visual Basic es sensible al contexto. Para emplear la ayuda sensible al contexto en la ventana de código, escriba la palabra para la cual desea información, y luego presione **F1**. Por ejemplo, si desea información acerca de la sentencia **Open**, escriba **Open** y presione **F1**.

Libros en Pantalla

Además de la ayuda sensible al contexto, el CD-ROM de Visual Basic incluye una versión en línea de la documentación impresa para Visual Basic. Para acceder a los Libros en Pantalla, haga clic en **Libros en Pantalla** dentro del menú **Ayuda** de Visual Basic.

La Ventana de Código

La ventana o editor de código de Visual Basic proporciona de manera automática información relevante a medida que se ingresa código. Por ejemplo, si se escribe el nombre de un control, seguido de un

punto, las propiedades y métodos para ese control serán mostrados automáticamente en un cuadro de lista. Luego se puede escoger la propiedad o método deseado para completar la sentencia. Cuando se ingresa el nombre de una función en la ventana de código, Visual Basic automáticamente proporciona el formato o sintaxis de la función.

¿Cómo se añaden controles al formulario?

Para añadir controles a un formulario tenemos dos métodos:

Método 1

Teniendo el cuadro de Herramientas o un Formulario visible, haga doble clic en el control que desea añadir en el *Cuadro de Herramientas*, los controles se ubican en el centro del formulario, uno encima de otro, luego hay que moverlos a la posición deseada dentro del formulario.

Método 2

1. Haga clic sobre el control en el Cuadro de Herramientas.
2. Ubique el puntero del Mouse (una cruz) sobre el formulario en la esquina superior izquierda donde desea colocar el control.
3. Realice un clic sostenido mientras arrastra el puntero a la esquina superior derecha donde colocará el control.
4. Suelte el botón del Mouse.

Estos cuatro pasos se repiten con cada control que desea añadir al formulario.

Terminología de Visual Basic

Conforme trabaje con Visual Basic, necesitará estar familiarizado con los siguientes términos:

Término	Definición
Tiempo de diseño	Es el momento en el que se construye la aplicación.
Tiempo de ejecución	Es el momento en el cual ejecutamos e interactuamos con la aplicación como lo haría el usuario.
Formulario	Un formulario sirve como una ventana que puede personalizar como la interfaz de su aplicación o como un cuadro de diálogo que usa para obtener información del usuario. Un formulario puede existir individualmente o puede servir como un documento dentro de una interfaz de documento múltiple (MDI)

Término	Definición
Controles	Representación gráfica de objetos tales como botones, cuadros de lista, cuadros de edición, etc., con los que el usuario interactúa para proporcionar información a la aplicación.
Objetos	Un término general usado para describir todos los formularios y controles que forman parte de la aplicación.
Propiedades	Los valores de un objeto, tales como tamaño, título, color, etc.
Métodos	Las acciones que un objeto puede realizar sobre sí mismo.
Eventos	Son acciones reconocidas por un formulario o control. Los eventos ocurren a medida que el usuario interactúa con los objetos de la aplicación.
Programación controlada por eventos	Cuando un programa es <i>controlado por eventos</i> , usted escribe código que se ejecuta en respuesta a eventos invocados por el usuario. Difiere de la <i>programación procedural</i> , en la cual el programa comienza en la primera línea de código y sigue un flujo definido llamando procedimientos cuando es necesario. La programación controlada por eventos es la esencia de las interfaces gráficas de usuario; el usuario acciona y el código responde.

¿Qué es un proyecto?

Cuando desarrolla una aplicación, Visual Basic crea un archivo especial llamado **Archivo de Proyecto** para administrar todos los demás archivos de la aplicación.

El **Archivo de Proyecto** es simplemente una lista de todos los archivos y objetos asociados con el proyecto, así como información sobre las opciones del entorno. Esta información se actualiza cada vez que se guarda el proyecto. Todos los archivos y objetos también se pueden compartir con otros proyectos. Un proyecto está compuesto por los siguientes archivos:

Tipo de archivo	Extensión	Descripción
Proyecto	.vbp	Realiza el seguimiento de todos los componentes de la aplicación.
Formulario	.frm .frx	Incluye el formulario, los objetos sobre el formulario y el código que se ejecuta cuando ocurre un evento en el formulario.
Módulo estándar	.bas	Contiene procedimientos Sub y Function que pueden ser invocados por cualquier formulario u objeto sobre el formulario. (<i>opcional</i>)

Tipo de archivo	Extensión	Descripción
Controles Personalizados	.ocx	Controles adicionales a los controles estándar proporcionados por Microsoft u otras empresas. (<i>opcional</i>)
Módulo de clase	.cls	Contiene la definición de clase, métodos y propiedades de un nuevo tipo de objeto. (<i>opcional</i>)
Recursos	.res	Contiene información binaria usada por la aplicación. Son usados generalmente cuando se crean programas para múltiples lenguajes. (<i>opcional</i>)

Cuando ha completado todos los archivos del proyecto puede convertir el proyecto en un archivo ejecutable (.exe).

Nota: Con las ediciones Profesional y Empresarial de Visual Basic también puede crear otro tipo de archivos ejecutables, como archivos **.ocx** y **.dll**.

Pasos para crear una aplicación

El proceso de creación de una aplicación Visual Basic puede descomponer en una serie de siete pasos.

1. Crear la interfaz de usuario

Usted crea una interfaz dibujando controles y objetos sobre un formulario. A fin de hacer que su código sea más fácil de leer y depurar, debe luego asignar nombres a los objetos usando convenciones de nombres estándar.

2. Establecer las propiedades de los objetos de la interfaz

Luego de añadir objetos al formulario, se establece las propiedades de los objetos. Puede establecer valores iniciales ya sea usando la ventana de propiedades en tiempo de diseño o escribiendo código para modificar las propiedades en tiempo de ejecución.

3. Escribir código para los eventos

Luego de establecer las propiedades iniciales para el formulario y cada objeto, añada el código que se ejecutará en respuesta a los eventos. Los eventos ocurren cuando diferentes acciones ocurren sobre un control u objeto. Por ejemplo, clic es un evento que puede ocurrir para un botón de comando.

4. Guardar el proyecto

Cuando crea el proyecto, asegúrese de darle un nombre usando el comando **Guardar Proyecto como** del menú **Archivo**. Guarde su proyecto frecuentemente conforme añada código. Al guardar un proyecto se guardan cada formulario y módulo de código en el proyecto.

5. Probar y depurar la aplicación

Conforme añada código al proyecto, puede usar el comando **Iniciar** en la Barra de Herramientas para ejecutar su aplicación y ver su comportamiento. También puede usar las herramientas de depuración para verificar errores y modificar código.

6. Crear un archivo ejecutable

Al completar su proyecto, crear un archivo ejecutable usando el comando **Generar XXXXXXXX.exe** del menú **Archivo**.

7. Crear una aplicación de instalación

Debido a que su archivo ejecutable depende de otros archivos, tales como el archivo en tiempo de ejecución de Visual Basic (Vbrun50032.dll), algunos archivos OCX y archivos DLL adicionales requeridos por la aplicación o por los controles ActiveX.

Convenciones para los nombres de los objetos

Los objetos deben llevar nombres con un prefijo coherente que facilite la identificación del tipo de objeto. A continuación se ofrece una lista de convenciones recomendadas para algunos de los objetos permitidos por Visual Basic.

Tipo de control	Prefijo	Ejemplo
Panel 3D	pnl	pnlGrupo
Botón animado	ani	aniBuzón
Casilla de verificación	chk	chkSóloLectura
Cuadro combinado, cuadro lista desplegable	cbo	cboInglés
Botón de comando	cmd	cmdSalir
Diálogo común	dlg	dlgArchivoAbrir
Comunicaciones	com	comFax
Control de datos	dat	datBiblio
Cuadro combinado enlazado a datos	dbcbo	dbcboLenguaje
Cuadrícula enlazada a datos	dbgrd	dbgrdResultadoConsulta
Cuadro de lista enlazado a datos	dblst	dblstTipoTarea
Cuadro de lista de directorios	dir	dirOrigen
Cuadro de lista de unidades	drv	drvDestino
Cuadro de lista de archivos	fil	filOrigen
Formulario	frm	frmEntrada
Marco	fra	fraLenguaje
Medidor	gau	gauEstado
Gráfico	gra	graIngresos
Cuadrícula	grd	grdPrecios
Barra de desplazamiento horizontal	hsb	hsbVolumen
Imagen (Image)	img	imgIcono
Estado de tecla	key	keyMayúsculas
Etiqueta	lbl	lblMsjAyuda
Línea	lin	linVertical

Tipo de control	Prefijo	Ejemplo
Cuadro de lista	lst	lstCódigoDePolítica
Mensaje MAPI	mpm	mpmEnviarMsj
Sesión MAPI	mps	mpsSesión
MCI	mci	mciVideo
Formulario MDI secundario	mdi	mdiNota
Menú	mnu	mnuArchivoAbrir
MS Flex Grid	msg	msgClientes
MS Tab	mst	mstPrimero
ActiveX	ole	oleHojaDeTrabajo
Esquema	out	outDiagramaDeOrg
Pen Bedit	bed	bedNombre
Pen Hedit	hed	hedFirma
Trazo de pluma	ink	inkMapa
Imagen (Picture)	pic	picVGA
Clip de imagen	clp	clpBarraDeHerramientas
Informe	rpt	rptGananciasTrimestre1
Forma	shp	shpCírculo
Cuadro de número	spn	spnPáginas
Cuadro de texto	txt	txtApellido
Cronómetro	tmr	tmrAlarma
Arriba-abajo	upd	updDirección
Barra de desplazamiento vertical	vsb	vsbVelocidad
Control deslizante	sld	sldEscala
Lista de imágenes	ils	ilsTodoslosIconos
Vista de árbol	tre	treOrganización
Barra de herramientas	tlb	tlbAcciones
TabStrip	tab	tabOpciones
Barra de estado	sta	staFechaHora
Lista	lvw	lvwEncabezados
Barra de progreso	prg	prgCargarArchivo
RichTextBox	rtf	rtfInforme

Prefijos sugeridos para menús

Las aplicaciones suelen usar muchos controles de menú, lo que hace útil tener un conjunto único de convenciones de nombres para estos controles. Los prefijos de controles de menús se deben extender más allá de la etiqueta inicial "mnu", agregando un prefijo adicional para cada nivel de anidamiento, con el título del menú final en la última posición de cada nombre. En la tabla siguiente hay algunos ejemplos.

Secuencia del título del menú	Nombre del controlador del menú
Archivo Abrir	mnuArchivadorAbrir
Archivo Enviar correo	mnuArchivoEnviarCorreo

Secuencia del título del menú**Nombre del controlador del menú**

Archivo Enviar fax

mnuArchivoEnviarFax

Formato Carácter

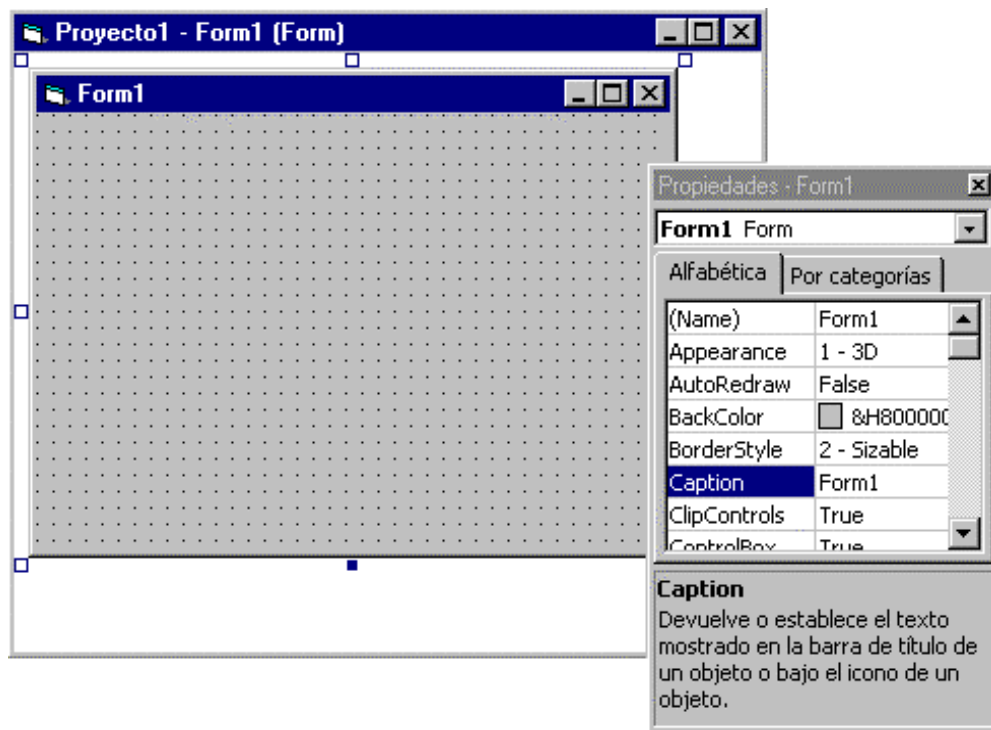
mnuFormatoCarácter

Ayuda Contenido

mnuAyudaContenido

Cuando se usa esta convención de nombres, todos los miembros de un grupo de menús determinado se muestran uno junto a otro en la ventana Propiedades de Visual Basic. Además, los nombres del control de menú documentan claramente los elementos del menú a los que están adjuntos.

Formularios



El formulario es el principal medio de comunicación entre el usuario y la aplicación. Los usuarios interactúan con los controles sobre el formulario para ingresarle datos y obtener resultados.

Propiedades

BackColor	Color de fondo del formulario.
BorderStyle	Estilo del borde del formulario.
Caption	Texto en la barra de título del formulario.
ControlBox	True/False. Determina si tiene o no el cuadro de control.
Enabled	True/False. Determina si está habilitado para responder a las acciones del usuario.
Icon	Icono que se muestra cuando el formulario está minimizado.
Left y Top	Ubicación del formulario.
MaxButton	True/False. Determina si tiene o no el botón <i>Maximizar</i> .

MinButton	True/False. Determina si tiene o no el botón <i>Minimizar</i> .
Name	Nombre del formulario.
WindowState	Estado inicial del formulario (normal, maximizado o minimizado)

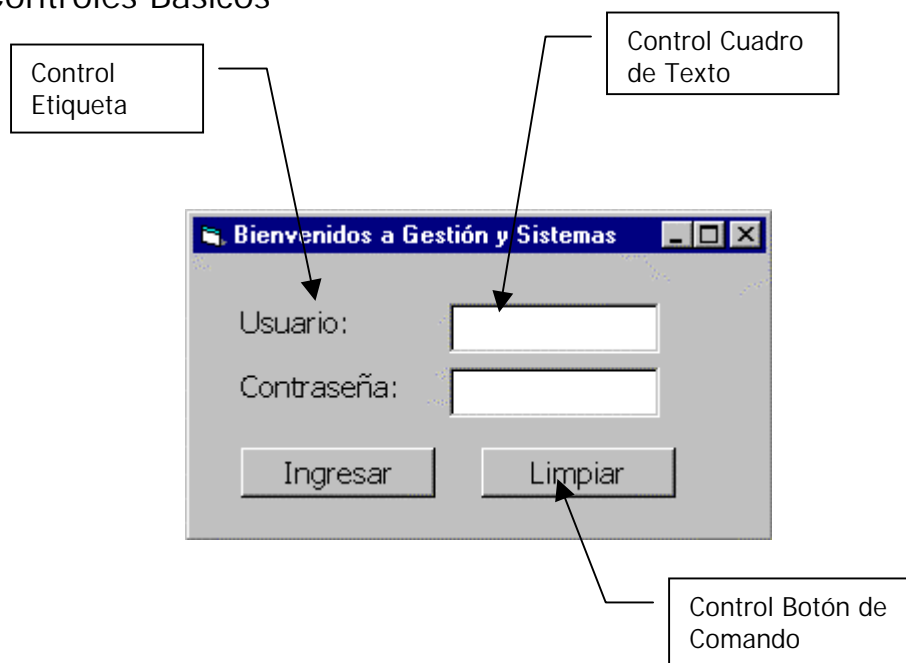
Eventos

Activate	Ocurre cuando el formulario se convierte en la ventana activa.
Click	Ocurre cuando hace clic sobre el formulario.
Deactivate	Ocurre cuando el formulario deja de ser la ventana activa.
Load	Ocurre cuando se carga un formulario.
Unload	Ocurre cuando un formulario está a punto de descargarse.

Métodos

Hide	Oculta el formulario.
Refresh	Actualiza el contenido del formulario.
SetFocus	Le entrega el enfoque al formulario.
Show	Hace visible el formulario.

Controles Básicos



Con los controles, los usuarios pueden operar y obtener los resultados de una aplicación. Puede añadir controles a un formulario seleccionando la herramienta adecuada del **Cuadro de Herramientas**. Entre los controles más comunes a utilizar en una aplicación tenemos: Etiqueta (Label), Cuadro de Texto (TextBox) y Botón de Comando (CommandButton).

Control Etiqueta (Label)



Se utiliza para mostrar texto que el usuario no puede modificar. Generalmente para identificar otros controles en el formulario o para mostrar instrucciones al usuario.

Propiedades

Alignment	Alineación del texto dentro del control.
AutoSize	True/False. Determina si el tamaño del control se ajusta automáticamente al texto que contiene.
Caption	Texto que muestra el control.
Name	Nombre del control.
Font	Establece la fuente, estilo y tamaño para el texto del control.

Control Cuadro de Texto (Textbox)



Se utiliza para que el usuario le proporcione datos a la aplicación o para que la aplicación le devuelva la información al usuario. El texto que se muestra en el control puede ser cambiado por el usuario.

Propiedades

Enabled	True/False. Establece un valor que determina si el control puede responder a eventos generados por el usuario.
Font	Establece la fuentes, estilo y tamaño para el texto del control.
Locked	True/False. Determina si es posible modificar el texto en el control.
MaxLength	Establece la longitud máxima permitida para el texto en el control.
MultiLine	Establece si el control puede aceptar múltiples líneas de texto.
Name	Nombre del control.
PasswordChar	Carácter utilizado para ocultar el texto que realmente contiene el control.
Text	Texto que realmente contiene y muestra el control.
Visible	Establece si el control será visible para el usuario.

Eventos

Change	Ocurre cuando cambia el texto que contiene el control.
GotFocus	Ocurre cuando el control recibe el enfoque.
KeyDown	Ocurre cuando el usuario presiona una tecla mientras el control tiene el enfoque.
LostFocus	Ocurre cuando el control pierde el enfoque.

Métodos

Refresh	Actualiza el texto del control.
SetFocus	Mueve el enfoque al control.

Control Botón de Comando (Commandbutton)



Permite que la aplicación inicie, interrumpa o termine un proceso.

Propiedades

Cancel	True/False. Establece si el botón se comportará como el botón cancelar en el formulario y se invocará su evento Click cada vez que se presione la tecla ESC .
Caption	Establece el texto que muestra el botón.
Default	True/False. Establece si el botón se comportará como el botón predeterminado en el formulario.
Font	Establece la fuente, estilo y tamaño para el texto del control.
Name	Nombre del botón.
Visible	True/False. Establece si el botón será visible para el usuario.

Eventos

Click	Ocurre cuando se hace clic sobre el botón.
-------	--------------------------------------------

Métodos

SetFocus	Mueve el enfoque al botón.
----------	----------------------------

Estableciendo Propiedades

Al diseñar la interfase de usuario de una aplicación Visual Basic, se deben establecer la propiedades para los controles (objetos) creados.

Estableciendo Propiedades en Tiempo de Diseño

Algunas propiedades pueden ser establecidas en tiempo de diseño. Para establecer estas propiedades se emplea la ventana de propiedades. Para acceder a la ventana de propiedades, oprima en botón secundario del ratón sobre un objeto, y luego haga clic en **Propiedades**. También se puede obtener el mismo resultado seleccionando el objeto y luego presionando **F4**.

Si selecciona varios objetos a la vez y accede a la ventana de propiedades, sólo se mostrarán las propiedades que son comunes para todos los controles seleccionados. Cualquier cambio que se haga a una propiedad será aplicada a todos los controles.

Estableciendo Propiedades en Tiempo de Ejecución

En tiempo de ejecución, se puede escribir código para establecer u obtener el valor de una propiedad. La siguiente línea de código establece a negrita la fuente de un cuadro de texto llamado txtData.

```
txtData.Font.Bold = True ' Establece el texto a negrita
```

Este código establece la propiedad **Text** del cuadro de texto txtData

```
txtData.Text = "Hola mundo" 'Establece el valor del texto
```

Si se omite el nombre de la propiedad, se establece la propiedad predeterminada del control. La propiedad predeterminada de un cuadro de texto es la propiedad **Text**. La propiedad predeterminada de una etiqueta es la propiedad **Caption**. Las siguientes líneas de código establecen las propiedades predeterminadas text y caption de un cuadro de texto y de una etiqueta.

```
txtData = "Se establece la propiedad Text del cuadro de texto"  
lblData = "Se establece la propiedad Caption de la etiqueta"
```

Obteniendo Propiedades en Tiempo de Ejecución

Puede emplear el siguiente código para obtener el valor de una propiedad en tiempo de ejecución.

```
Dim sNombre as String  
sNombre = txtName.Text
```

Procedimientos de Evento

Visual Basic invoca automáticamente procedimientos de evento en respuesta a acciones del teclado, del ratón o del sistema. Por ejemplo, los botones de comando tienen un procedimiento de evento Click. El código que se escriba en el procedimiento de evento Click es ejecutado cuando el usuario haga clic en un botón de comando.

Para abrir la ventana de código, haga doble clic en el control o formulario, haga clic en la orden **Código** del menú **Ver**.

Cada control tiene un conjunto fijo de procedimientos de evento. Los procedimientos de evento para cada control son mostrados en un cuadro de lista despegable en la ventana de código. El siguiente código muestra el procedimiento de evento Click para un botón de comando llamado cmdOK.

```
Private Sub cmdOK_Click()  
    MsgBox "Hola"  
End Sub
```

Orden de Tabulación de los Controles

El **orden de tabulación** es el orden en que un usuario se mueve de un control a otro pulsando la tecla Tab. Por omisión, el orden de tabulación es igual al orden en que se han colocado los controles en el formulario.

Para cambiar el orden de tabulación de los controles de un formulario, establezca el valor de la propiedad **TabIndex** de cada control. El valor de esta propiedad va desde 0 hasta n-1, siendo n el número de controles que tiene el formulario.

Quitar un control del orden de tabulación

Normalmente, presionando la tecla Tab en tiempo de ejecución pasamos de un control a otro en el orden de tabulación establecido. Podemos quitar un control del orden de tabulación si establecemos su propiedad **TabStop** en **False**. Un control cuya propiedad **TabStop** se ha establecido en **False** sigue manteniendo su posición en el orden de tabulación actual, aunque el control es saltado al ir de un control a otro con la tecla Tab.

Nota: Los controles que no pueden obtener enfoque, al igual que los controles desactivados o invisibles, no tienen la propiedad **TabIndex** y no están incluidos en el orden de tabulación. Cuando el usuario presiona Tab, estos controles son ignorados.

Tecla de Acceso Rapido a un Control

Si el control tiene la propiedad **Caption**, se le puede asignar una tecla de acceso rápido para seleccionar el control y de esta manera el control recibirá el enfoque cada vez que se oprima **ALT + TecladeAcceso**. Para especificar la tecla de acceso rápido debe resaltar un carácter de la cadena en la propiedad **Caption** colocando delante de este el símbolo "&". Por ejemplo si el valor de la propiedad **Caption** es la cadena **Usuario**, y se desea definir la tecla **u** como la de acceso rápido, deberá establecer la propiedad **Caption** de la siguiente manera: **U&uario**. El texto del control se verá así **Uuario**.

Algunos controles, tales como el control **Cuadro de Texto**, no tiene la propiedad **Caption**. Para crear una tecla de acceso para esos controles:

1. Coloque un control **Etiqueta** cerca de otro control.
2. Establezca la propiedad **Caption** del control **Etiqueta** para contener la tecla de acceso apropiada.
3. Establezca para el control **Etiqueta** un valor de orden de tabulación menor en una unidad que el de otro control.

Cuando pulse **ALT + TecladeAcceso** del control **Etiqueta**, el enfoque se moverá hacia el otro control debido a que el control **Etiqueta** no puede recibir el enfoque.

Parte 2: Manejo de Formularios

Modulos de Formulario

Cada formulario en su aplicación tiene un módulo de formulario asociado, estos son guardados con una extensión de archivo FRM y contienen:

- Los valores de las propiedades para el formulario y sus controles.
- Declaración de variables en el ámbito del formulario.
- Procedimientos de evento y procedimientos generales en el ámbito del formulario.

Nota: Las descripciones gráficas de un formulario y los controles sobre el formulario son almacenados en formato binario en un archivo con extensión FRX.

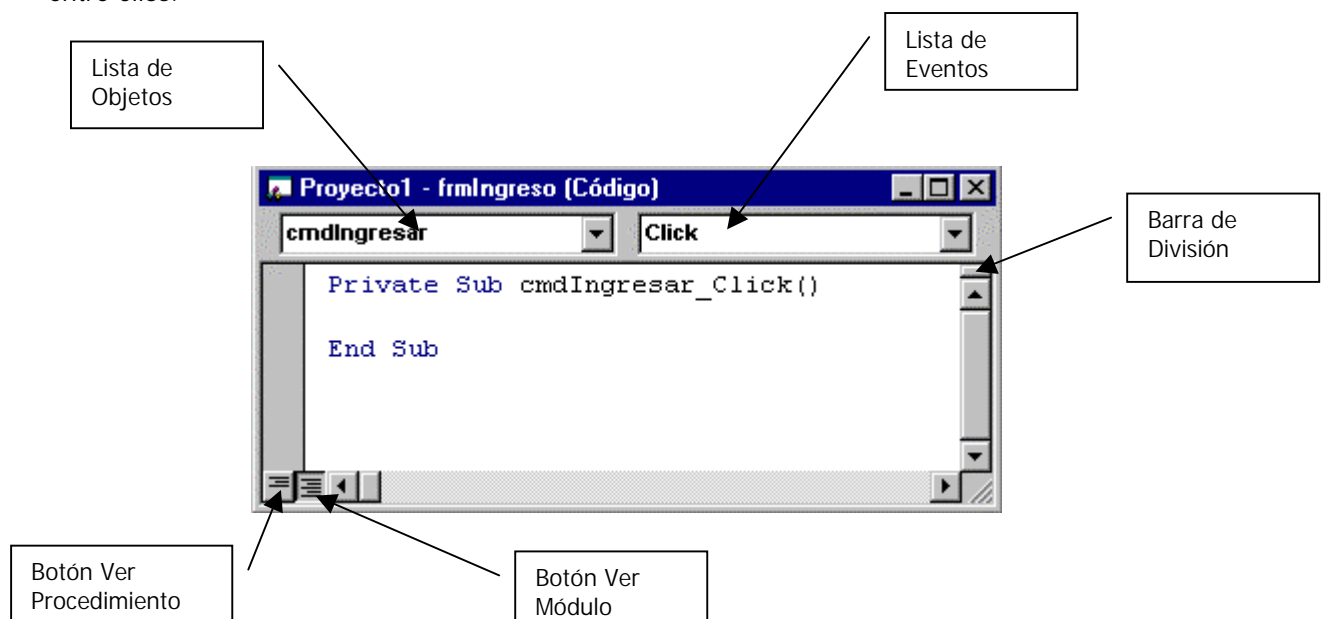
Modulos Estándar

Los módulos estándar pueden contener código que es común a varios formularios en su aplicación. Este código es por omisión público, lo cual significa que fácilmente compartido con otros módulos de código, tales como un módulo de formulario. Estos módulos contienen declaraciones de procedimientos, tipos y variables. No pueden almacenar procedimientos de evento por que no contienen objetos.

La Ventana de Código

Código es un término general para todas las sentencias Visual Basic que usted escribe en una aplicación: procedimientos de evento y procedimientos generales. El código en Visual Basic se escribe en la ventana de Código. El editor de texto es solo un editor ASCII con colores para diferenciar las palabras claves en el código que escribe.

La *Ventana de Código* se usa para escribir, mostrar y editar el código de su aplicación. Puede abrir una ventana de código por cada módulo de su aplicación, de modo que puede fácilmente copiar y pegar entre ellos.



La **Ventana de Código** contiene:

El Cuadro Lista de Objetos

Muestra el nombre del objeto seleccionado. Haga clic en la flecha a la derecha del cuadro Objeto para mostrar una lista de todos los objetos asociados con el formulario.

El Cuadro Lista de Eventos

Muestra todos los eventos reconocidos para el formulario o control mostrado en el cuadro Objeto. Cuando seleccionamos un evento, en la ventana de código se muestra el procedimiento de evento asociado con ese evento.

La Barra de División

Desde el menú *Ventana* puede ejecutar el comando *Dividir* para dividir la ventana de código en dos partes, cada una de las cuales se desplaza separadamente. Puede entonces ver diferentes partes de su código al mismo tiempo. La información que aparece en el cuadro Objeto y Procedimiento se refiere al código en la parte que tiene el enfoque. El mismo comando utilizado para dividir la ventana puede utilizarlo para cerrar una de sus partes o también lo puede hacer arrastrando la barra de división hacia la parte superior o inferior de la ventana.

El Botón Ver Procedimiento

Establece que en la ventana de código se edite un procedimiento a la vez.

El Botón Ver Módulo Completo

Establece que en la ventana de código se tenga acceso a todos los procedimientos, separados por una línea separadora uno de otro.

Editando Código

Use las características de edición de Visual Basic para que su código sea más fácil de leer.

Sangría

Use la sangría para diferenciar partes de su código, tales como estructuras repetitivas y condicionales. Para aplicar sangría a una sección de sentencias de un código use la tecla **Tab** o el comando **Aplicar sangría** del menú **Edición**. Se forma similar, **Shift + Tab** o el comando **Anular sangría** del menú **Edición** quitará una sangría a las líneas seleccionadas. Veamos el siguiente ejemplo:

```
Private Sub cmdIngresar_Click()  
    If Len(Trim(txtUsuario))=0 Then  
        txtUsuario.SetFocus  
    ElseIf Len(Trim(txtContraseña))=0 Then  
        txtContraseña.SetFocus  
    ElseIf txtContraseña = "AGPS" Then  
        MsgBox "La clave ingresada es correcta"  
        Unload Me  
    Else  
        MsgBox "La clave ingresada no es válida"
```

```
txtContraseña.SelStart=0
txtContraseña.SelLength= Len(Trim(txtContraseña))
txtContraseña.SetFocus
End If
End Sub
```

Carácter de Continuación de Línea

El carácter subrayado (_) es el carácter de continuación de línea, y se usa para dividir una sentencia en múltiples líneas. Esto hace que la sentencia sea más fácil de leer porque está contenida totalmente dentro de la ventana de código en lugar de extenderse mas allá de sus límites. El carácter de continuación de línea se coloca luego de un espacio de la sentencia, como se muestra en el siguiente ejemplo:

```
MsgBox "La clave ingresada no es válida", _
vbOKOnly + vbExclamation, _
"Mensaje"
```

Comentarios

El añadir documentación y comentarios a su código permite comprender mejor lo que hace el código. Esto también ayuda a comprender el código si necesita volver a revisarlo en alguna fecha posterior. Un comentario se inicia con el carácter apóstrofe ('), de modo que todo el texto que continúe a este carácter será ignorado en la ejecución de la aplicación. Veamos el siguiente ejemplo:

```
Private Sub cmdLimpiar_Click()
    'Este procedimiento limpia la ventana de identificación
    txtUsuario.Text = "" 'Limpia el cuadro de texto Usuario
    txtContraseña.Text = "" 'Limpia el cuadro de texto Contraseña
    txtUsuario.SetFocus ' Mueve el enfoque al cuadro de texto Usuario
End Sub
```

Cuadro de Mensaje y de Entrada

Una de las formas más simples de obtener información para y desde el usuario es utilizando las funciones **MsgBox** e **InputBox** respectivamente.

Función MsgBox()

Los cuadros de mensaje ofrecen un modo simple y rápido de consultar a los usuarios por información simple o para permitirles tomar decisiones sobre el camino que su programa debe tomar. Puede usar esta función para mostrar diferentes tipos de mensaje y botones con los cuales el usuario da una respuesta.

```
Rpta = MsgBox("¿Está seguro de eliminar a este cliente?", _
vbQuestion + vbYesNo, "Confirmación")
```



Formato

`MsgBox(prompt [, buttons] [, title] [, helpfile, context])`

EL formato de la función MsgBox consta de los siguientes argumentos:

<i>Parte</i>	<i>Descripción</i>
Prompt	Requerido. Expresión de cadena que representa el mensaje en el cuadro de diálogo. La longitud máxima de prompt es de aproximadamente 1024 de caracteres, según el ancho de los caracteres utilizados. Si prompt consta de más de una línea, puede separarlos utilizando un carácter de retorno de carro (Chr(13)) o un carácter de avance de línea (Chr(10)), o una combinación de caracteres de retorno de carro - avance de línea (Chr(13 y Chr(10)) entre cada línea y la siguiente.
Buttons	Opcional. Expresión numérica que corresponde a la suma de los valores que especifican el número y el tipo de los botones que se pretenden mostrar, el estilo de icono que se va a utilizar, la identidad del botón predeterminado y la modalidad del cuadro de mensajes. Si se omite este argumento, el valor predeterminado para buttons es 0.
Title	Opcional. Expresión de cadena que se muestra en la barra de título del cuadro de diálogo. Si se omite title, en la barra de título se coloca el nombre de la aplicación.
Helpfile	Opcional. Expresión de cadena que identifica el archivo de Ayuda que se utiliza para proporcionar ayuda interactiva en el cuadro de diálogo. Si se especifica helpfile, también se debe especificar context.
Context	Opcional. Expresión numérica que es igual al número de contexto de Ayuda asignado por el autor al tema de Ayuda correspondiente. Si se especifica context, también se debe especificar helpfile.

Valores

El argumento buttons puede asumir los siguientes valores:

<i>Constante</i>	<i>Valor</i>	<i>Descripción</i>
VbOKOnly	0	Muestra solamente el botón Aceptar.
VbOKCancel	1	Muestra los botones Aceptar y Cancelar.
VbAbortRetryIgnore	2	Muestra los botones Anular, Reintentar e Ignorar.
VbYesNoCancel	3	Muestra los botones Sí, No y Cancelar.
VbYesNo	4	Muestra los botones Sí y No.
VbRetryCancel	5	Muestra los botones Reintentar y Cancelar.
VbCritical	16	Muestra el icono de mensaje crítico.
VbQuestion	32	Muestra el icono de pregunta de advertencia.
<i>Constante</i>	<i>Valor</i>	<i>Descripción</i>
VbExclamation	48	Muestra el icono de mensaje de advertencia.
VbInformation	64	Muestra el icono de mensaje de información.

VbDefaultButton1	0	El primer botón es el predeterminado.
VbDefaultButton2	256	El segundo botón es el predeterminado.
VbDefaultButton3	512	El tercer botón es el predeterminado.
VbDefaultButton4	768	El cuarto botón es el predeterminado.
VbApplicationModal	0	Aplicación modal; el usuario debe responder al cuadro de mensajes antes de poder seguir trabajando en la aplicación actual.
VbSystemModal	4096	Sistema modal; se suspenden todas las aplicaciones hasta que el usuario responda al cuadro de mensajes.

El primer grupo de valores (0 a 5) describe el número y el tipo de los botones mostrados en el cuadro de diálogo; el segundo grupo (16, 32, 48, 64) describe el estilo del icono, el tercer grupo (0, 256, 512, 768) determina el botón predeterminado y el cuarto grupo (0, 4096) determina la modalidad del cuadro de mensajes. Cuando se suman números para obtener el valor final del argumento buttons, se utiliza solamente un número de cada grupo.

Nota: Estas constantes las especifica Visual Basic. Por tanto, el nombre de las mismas puede utilizarse en cualquier lugar del código en vez de sus valores reales.

Valores devueltos

Constante	Valor	Descripción
-----------	-------	-------------

VbOk	1	Aceptar
VbCancel	2	Cancelar
VbAbort	3	Anular
VbRetry	4	Reintentar
VbIgnore	5	Ignorar
VbYes	6	Sí
VbNo	7	No

Nota: Si desea omitir algún argumento, debe incluir el delimitador de coma correspondiente o utilizar argumentos con nombre.

Ejemplos

1

```
StrMsg = "¿Desea continuar?"
Estilo = vbYesNo + vbExclamation + vbDefaultButton2
StrTitulo = "Responda"
Rpta = MsgBox(StrMsg, Estilo, StrTitulo)
If Rpta = vbYes Then
    -----
Else
    -----
End If
```

2

```

StrMsg = "¿Desea continuar?"
iEstilo = vbYesNo + vbExclamation + vbDefaultButton2
StrTitulo = "Responda"
iRpta = MsgBox( Prompt:=strMsg, Title:= StrTitulo, Buttons:= iEstilo )
If iRpta= vbYes Then
    -----
    -----
Else
    -----
    -----
End If

```

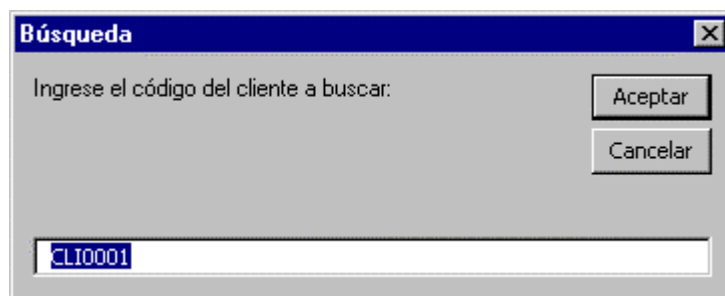
Función InputBox()

La función InputBox muestra un mensaje en un cuadro de diálogo, espera que el usuario escriba un texto o haga clic en un botón y devuelve un tipo String con el contenido del cuadro de texto.

```

strCodigo = InputBox("Ingrese el código del cliente a buscar:", _
"Búsqueda", "CLI0001")

```



Formato

```
InputBox( prompt [, title] [, default] [, xpos] [,ypos] [, helpfile, context] )
```

El formato de la función InputBox consta de los siguientes argumentos con nombre:

<i>Parte</i>	<i>Descripción</i>
Prompt	Requerido. Expresión de cadena que se muestra como mensaje en el cuadro de diálogo. La longitud máxima de prompt es de aproximadamente de 1024 caracteres, según el ancho de los caracteres utilizados. Si prompt consta de más de una línea, puede separarlos utilizando un carácter de retorno de carro (Chr(13)), un carácter de avance de línea (Chr(10)) o una combinación de los caracteres de retorno de carro – avance de línea (Chr(13) y (Chr(10)) entre cada línea y la siguiente.

Title	Opcional. Expresión de cadena que se muestra en la barra de título del cuadro de diálogo. Si omite title, en la barra de título se coloca el nombre de la aplicación.
Default	Opcional. Expresión de cadena que se muestra en el cuadro de texto como respuesta predeterminada. Si omite default, se muestra el cuadro de texto vacío.
Xpos	Opcional. Expresión numérica que especifica, la distancia en sentido horizontal entre el borde izquierdo del cuadro de diálogo y el borde izquierdo de la pantalla. Si se omite xpos, el cuadro de diálogo se centra horizontalmente.
Ypos	Opcional. Expresión numérica que especifica, la distancia en sentido horizontal entre el borde izquierdo del cuadro de diálogo y el borde izquierdo de la pantalla. Si se omite ypos, el cuadro de diálogo se coloca aproximadamente un tercio de la altura de la pantalla, desde el borde superior de la misma.
Helpfile	Opcional. Expresión de cadena que identifica el archivo de Ayuda que se utilizará para proporcionar ayuda interactiva para el cuadro de diálogo. Si se especifica helpfile, también deberá especificar context.
Context	Opcional. Expresión numérica que es el número de contexto de Ayuda asignado por el autor al tema de Ayuda correspondiente. Si se especifica context, también deberá especificarse helpfile.

Comentarios

Si el usuario hace clic en **Cancelar**, la función devuelve una cadena de caracteres de longitud cero ("").

Nota: Si desea omitir algunos argumentos, debe incluir el delimitador de coma correspondiente o utilizar argumentos con nombre.

Constantes Predefinidas

Visual Basic reconoce cierta cantidad de constantes predefinidas que pueden ser usadas en cualquier parte de su código en lugar de valores numéricos. Puede hacer que su código sea más fácil de leer y escribir mediante el uso de estas constantes. Además, los valores de estas constantes pueden cambiar en versiones posteriores de Visual Basic, su uso permitirá que su código sea compatible. Por ejemplo, la propiedad **WindowState** de un formulario puede aceptar las siguientes constantes:

<u>Constante</u>	<u>Valor</u>	<u>Descripción</u>
VbNormal	0	Normal
VbMinimized	1	Minimizado
VbMaximized	2	Maximizado

Por ejemplo, para establecer el estado del formulario **frmEntrada** en maximizado, la sentencia sería:

```
frmEntrada.WindowState = vbMaximized
```

Manejo de Formularios

Normalmente la interfaz de una aplicación está compuesta por varios formularios. Cuando Visual Basic inicia la aplicación, automáticamente se muestra el formulario de arranque, mientras que los otros

formularios deben ser mostrados y ocultados a través de código. El método o función usado depende de lo que deseamos hacer.

Tarea	Método o Instrucción
Cargar un formulario en memoria, pero sin mostrarlo	Use la sentencia Load , o haga referencia a una propiedad o control sobre el formulario.
Cargar o mostrar el formulario.	Use el método Show .
Mostrar un formulario cargado.	Use el método Show .
Ocultar u formulario	Use el método Hide .
Ocultar un formulario y descargarlo de memoria.	Use la sentencia Unload .

Método Show

Muestra un formulario. Si el formulario no está cargado al momento de ejecutar el método Show, Visual Basic lo cargará automáticamente.

Formato

```
NombreDelFormulario.Show
```

Ejemplo

```
FrmEntrada.Show
```

Sentencia Load

Carga un formulario a la memoria, pero no lo muestra.

Formato

```
Load NombreDelFormulario
```

Ejemplo

```
Load FrmEntrada
```

Evento Load

El evento **Load** ocurre cuando el formulario es cargado en la memoria. Esto sucede cuando se usa la sentencia **Load**, o cuando se invoca el método **Show** y el formulario aún no está cargado en memoria. Normalmente, este evento se utiliza para establecer algunas propiedades del formulario, los controles que se encuentran en él, o variables a nivel del formulario.

Ejemplos:

1

```
Private Sub Form_Load()  
    frmIngreso.Left = (Screen.Width - frmIngreso.Width) / 2
```

```
        frmIngreso.Top = (Screen.Height - frmIngreso.Height) / 2
    End Sub
```

2

```
Private Sub Form_Load()
    txtUsuario.Text=""
    txtContraseña.Text=""
    cmdIngresar.Enabled=False
End Sub
```

Método Hide

Oculta un formulario, pero no lo descarga de memoria.

Formato

```
NombreDelFormulario.Hide
```

Ejemplo:

```
frmIngreso.Hide
```

Sentencia Unload

Descarga un formulario de memoria

Formato

```
Unload NombreDelFormulario
```

Comentarios

La descarga de un formulario puede ser necesario o conveniente en aquellos casos en los que la memoria utilizada se necesite para alguna otra tarea o cuando sea necesario restablecer las propiedades a sus valores originales.

Antes de descargar un formulario se ejecuta el evento **Unload (Form_Unload)**. Si establece el argumento **Cancelar** a **True** en este evento, no se descargará el formulario.

Sugerencia

Use la palabra **Me** para referirse al formulario actual.

Ejemplo

```
' Descarga el formulario actual
Private Sub cmdCerrar-Click ( )
    Unload Me
End Sub
```

Evento Unload

Ocurre cuando un formulario está a punto de descargarse. Este evento se desencadena porque un usuario cierra el formulario mediante el comando **Cerrar** del menú **Control** o una sentencia **Unload**.

El parámetro **Cancelar**, es un entero que determina si el formulario es descargado. Si **Cancelar** es 0, el formulario se descarga. Establecer **Cancelar** a cualquier valor distinto de cero impide que el formulario sea descargado.

Ejemplo

```
Private Sub Form_Unload(Cancel As Integer)
    Dim iRpta As Integer
    iRpta = MsgBox("¿Esta seguro de cerrar la ventana?", _
        VbYesNo + vbQuestion, "Mensaje")
    If iRpta = vbNo Then
        Cancel = True
    End If
End Sub
```

Formularios Modales y No Modales

Un formulario que se abre como **Modal**, no permite que el usuario interactúe con otros formularios de la misma aplicación hasta que no sea cerrado.

Un formulario que se abre como **No Modal**, permite al usuario cambiar a otros formularios de la misma aplicación sin que los primeros sean cerrados.

El argumento **Estilo** del método **Show** determina si el formulario se abre como **Modal** o **No Modal**.

Ejemplos

1 ' Cargar el formulario frmIngreso _
como Modal
frmEntrada.Show vbModal

2 'Cargar el formulario _
frmIngreso como No Modal
frmEntrada.Show vbModalless

Finalización de una Aplicación

Se puede finalizar la ejecución de la aplicación descargando el último formulario de la aplicación o usando la sentencia **End**. La sentencia **End** termina la ejecución de la aplicación y descarga todos los

formularios. Cuando finaliza su aplicación con la sentencia **End** no se desencadena ningún evento de los formularios.

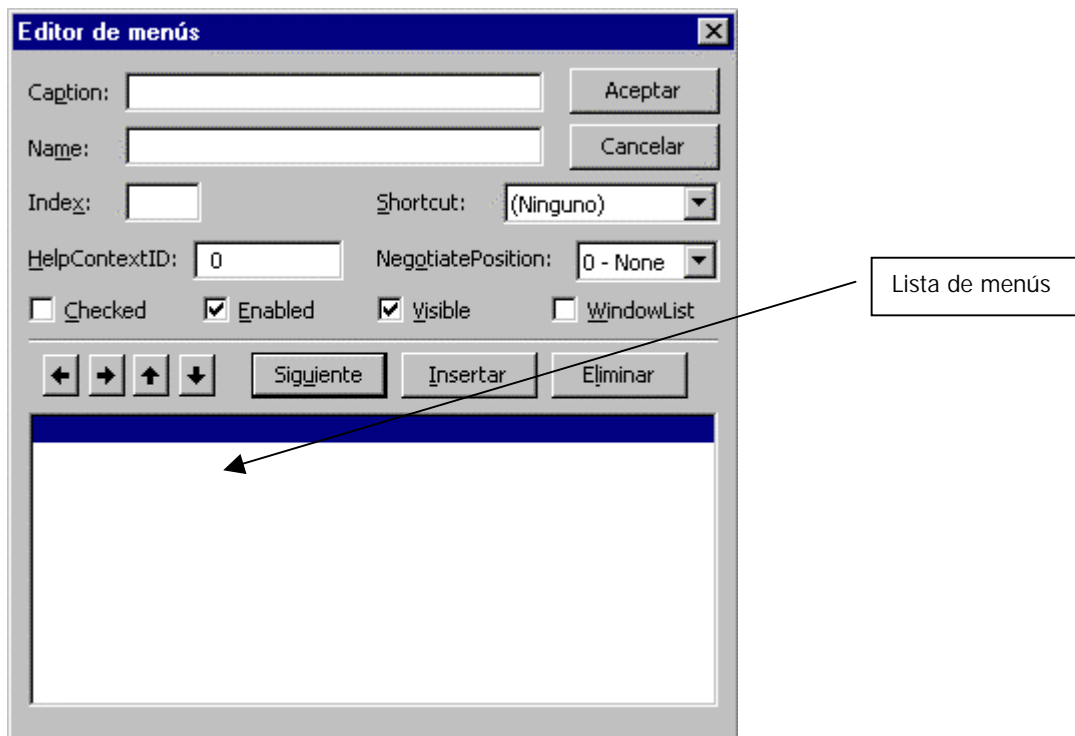
Ejemplos

1 ' Descarga el formulario actual
Private Sub cmdCerrar-Click ()
 Unload Me
End Sub

2 ' Finaliza la aplicación
Private Sub cmdFinalizar-Click ()
 End
End Sub

El Editor de menús





Esta herramienta permite crear menús personalizados para la aplicación y definir sus propiedades. Para ingresar, estando en tiempo de diseño, haga clic en el menú **Herramientas** y luego en la orden **Editor de Menús** o en el botón equivalente de la barra de herramientas estándar.



Nota: Para poder ingresar el editor de menús no debe encontrarse en la ventana de código.

Opciones del cuadro de diálogo

Parte	Descripción
Caption	Le permite escribir el nombre del comando o del menú que desea que aparezca en la barra de menús o en un menú. Si desea crear una barra separadora en el menú, escriba un único guión (-) en el cuadro Caption.

<i>Parte</i>	<i>Descripción</i>
	Para dar al usuario acceso a través del teclado a un elemento del menú, escriba el signo & delante de una letra. En tiempo de ejecución, esta letra aparecerá subrayada (el signo & no será visible) y el usuario tendrá acceso al menú o al comando si presiona las teclas ALT y la correspondiente a la letra subrayada. Si necesita que aparezca en el menú el signo &, deberá escribir dos signos & consecutivos.
Name	Le permite escribir un nombre de control para el elemento del menú. El nombre de control es un identificador que sólo se utiliza para tener acceso al elemento del menú en el código, no aparece en ningún menú.
Index	Le permite asignar un valor numérico que determina la posición del control en una matriz de controles. Esta posición no tiene ninguna relación con la ubicación del control en la pantalla.
Shortcut	Le permite seleccionar una tecla de método abreviado para cada comando.
HelpContextID	Le permite asignar un valor numérico único para el Id. de contexto. Este valor se utiliza para encontrar el tema de Ayuda apropiado en el archivo de Ayuda identificado mediante la propiedad HelpFile.
NegotiatePosition	Le permite seleccionar la propiedad NegotiatePosition del menú. Esta propiedad determina si el menú aparece o no en un formulario contenedor y cómo aparece.
Checked	Le permite hacer que aparezca inicialmente una marca de verificación a la izquierda de un elemento del menú. Se utiliza normalmente para indicar si una opción de alternar está activada o no.
Enabled	Le permite seleccionar si el elemento del menú debe responder a eventos. Desactive esta opción si desea que el elemento del menú no esté disponible y aparezca atenuado.
Visible	Le permite hacer que un elemento aparezca en el menú.
WindowList	Determina si el control del menú contiene una lista de formularios secundarios MDI abiertos en una aplicación MDI.
Flecha a la derecha	 <p>Pasa el menú seleccionado a un nivel inferior cada vez que hace clic en el botón. Puede crear hasta cuatro niveles de submenús.</p>
Flecha a la izquierda	 <p>Pasa el menú seleccionado a un nivel superior cada vez que hace clic en el botón. Puede crear hasta cuatro niveles de submenús.</p>
Flecha arriba	 <p>Cada vez que se hace clic en este botón, el elemento seleccionado del menú se mueve hacia arriba una posición dentro del mismo nivel de menú.</p>
Flecha abajo	 <p>Cada vez que se hace clic en este botón, el elemento seleccionado del menú se mueve hacia abajo una posición dentro del mismo nivel de menú.</p>
Lista Menús	Es un cuadro de lista que muestra en orden jerárquico todos los elementos del menú. Los elementos de los submenús aparecen indentados para indicar su ubicación o su nivel en la jerarquía.
Siguiente	Selecciona la línea siguiente.
Insertar	Inserta una línea en el cuadro de lista, inmediatamente encima de la línea actualmente

<i>Parte</i>	<i>Descripción</i>
	seleccionada.
Eliminar	Elimina Borra la línea actualmente seleccionada.
Aceptar	Cierra el Editor de menús y aplica todos los cambios efectuados en el último formulario seleccionado. El menú está disponible en tiempo de diseño, pero si selecciona un menú en la fase de diseño, se abre la ventana Código para el evento Clic de ese menú en lugar de ejecutarse el código del evento.
Cancelar	Cierra el Editor de menús y cancela todos los cambios.

Añadiendo Controles Adicionales al Cuadro de Herramientas

El Cuadro de Herramientas contiene los controles intrínsecos o estándar de Visual Basic. Es posible ampliar el cuadro de herramientas añadiendo controles ActiveX. La edición profesional de Visual Basic proporciona controles ActiveX adicionales. También es posible comprar controles ActiveX a terceras partes.

Para añadir un control ActiveX al cuadro de herramientas haga lo siguiente:

1. En el menú **Proyecto**, haga clic en **Componentes**. Visual Basic mostrará el cuadro de diálogo **Componentes**.
2. En la ficha **Controles**, haga clic en el control que desea incluir, y luego haga clic en **Aceptar**. Visual Basic añadirá el control al cuadro de herramientas.

El control *ToolBar*



Un control **ToolBar** contiene una colección de objetos **Button** utilizados para crear una barra de herramientas asociada a una aplicación.

Comentarios

Normalmente, una barra de herramientas contiene botones que corresponden a elementos de menú de una aplicación, proporcionando una interfaz gráfica al usuario que le permite tener acceso a las funciones y comandos empleados con más frecuencia en esa aplicación.

El control **ToolBar** le permite crear barras de herramientas agregando objetos **Button** a una colección **Buttons**; cada objeto **Button** puede tener texto opcional o una imagen, proporcionados por un control **ImageList** asociado. Puede mostrar una imagen en un botón con la propiedad **Image** o mostrar texto con la propiedad **Caption**, o ambos, para cada objeto **Button**. En tiempo de diseño puede agregar objetos **Button** al control utilizando la **Página de propiedades** del control **ToolBar**. En tiempo de ejecución, puede agregar o quitar botones de la colección **Buttons** mediante los métodos **Add** y **Remove**.

Para programar el control **ToolBar**, agregue código al evento **ButtonClick** para que responda al botón seleccionado. También puede determinar el comportamiento y la apariencia de cada objeto **Button** mediante la propiedad **Style**. Por ejemplo, si a cuatro botones se les asigna el estilo **ButtonGroup**, sólo se podrá presionar uno de ellos y al menos uno estará siempre presionado.

La facilidad de uso se mejora considerablemente programando descripciones **ToolTipText** de cada objeto **Button**. Para mostrar información sobre herramientas, la propiedad **ShowTips** debe establecerse a **True**.

El Control ImageList



Un control **ImageList** contiene una colección de objetos **ListImage**, a cada uno de los cuales se puede hacer referencia mediante su índice o su clave. El control **ImageList** no está concebido para utilizarlo en solitario, sino como punto de almacenamiento central para proporcionar cómodamente imágenes a otros controles.

Comentarios

Puede usar el control **ImageList** con cualquier control que asigne un objeto **Picture** a una propiedad **Picture**.

Es posible agregar imágenes de diferentes tamaños al control **ImageList**, pero todas se ajustan al mismo tamaño. El tamaño de los objetos de **ListImage** está determinado por uno de los siguientes valores:

- El valor de las propiedades **ImageWidth** y **ImageHeight** antes de agregar alguna imagen.
- Las dimensiones de la primera imagen agregada.

No hay ninguna limitación en cuanto al tamaño de la imagen, pero el número total de imágenes que se puede cargar está limitado por la cantidad de memoria disponible.

Durante el diseño del programa puede agregar imágenes mediante la ficha **General** del cuadro de diálogo **Propiedades** del control **ImageList**. En tiempo de ejecución puede agregar imágenes mediante el método **Add** para la colección **ListImages**.

Nota: Los controles **Toolbar** e **ImageList** forman parte del grupo de controles ActiveX adicionales Microsoft Windows Common Controls 5.0 (archivo COMCTL32.OCX). Para usarlos en su aplicación debe agregar el archivo COMCTL32.OCX al proyecto. Cuando distribuya su aplicación, instale el archivo COMCTL32.OCX en la carpeta System o System32 (en plataformas con Windows NT) de Microsoft Windows del usuario.

Interfaz de Múltiples Documentos (MDI – Multiple Document Interfase)

El Objeto MDIForm

Un formulario MDI es una ventana que actúa como fondo de una aplicación y es el contenedor de formularios que tienen su propiedad MDIChild establecida a True. Para crear un objeto MDIForm, elija **Agregar formulario MDI** en el menú **Proyecto**, luego de agregarlo tenga en cuenta lo siguiente:

- Una aplicación sólo puede tener un objeto MDIForm, pero varios formularios secundarios MDI.
- Si un formulario secundario MDI tiene menús, la barra de menús del formulario secundario reemplazará automáticamente a la barra de menús del objeto MDIForm cuando el formulario secundario MDI esté activo.
- Un formulario secundario MDI minimizado se mostrará como un icono en el MDIForm.
- Un objeto MDIForm sólo puede contener controles Menu y PictureBox, y controles personalizados que tengan una propiedad Align. Para colocar otros controles en un MDIForm, puede dibujar un cuadro de imagen en el formulario y después dibujar otros controles dentro del cuadro de imagen. Puede utilizar el método Print para mostrar texto en un cuadro de imagen de un MDIForm, pero no puede usar este método para mostrar texto en el MDIForm propiamente dicho.
- Un objeto MDIForm no puede ser modal.

- Los formularios secundarios MDI se diseñan de forma independiente del MDIForm, pero siempre están contenidos en el MDIForm en tiempo de ejecución.

Formulario secundario MDI

Un formulario contenido dentro de un formulario MDI en una aplicación con interfaz de múltiples documentos (MDI). Para crear un formulario secundario, establezca su propiedad MDIChild a True.

Propiedad MDIChild

Devuelve o establece un valor que indica si un formulario debe mostrarse como formulario secundario MDI dentro de un formulario MDI. Es de sólo lectura en tiempo de ejecución. Los valores admitidos para la propiedad MDIChild son:

Valor	Descripción
-------	-------------

True	El formulario es MDI secundario y se mostrará dentro del formulario MDI primario.
False	(Predeterminado) El formulario no es MDI secundario.

Comentarios

Utilice esta propiedad al crear una aplicación con interfaz de múltiples documentos (MDI). En tiempo de ejecución, los formularios que tengan establecida a True esta propiedad se mostrarán dentro de un formulario MDI. Los formularios MDI secundarios pueden maximizarse, minimizarse y desplazarse, siempre dentro del formulario MDI primario. Cuando trabaje con formularios MDI secundarios, tenga en cuenta lo siguiente:

- En tiempo de ejecución, cuando un formulario MDI secundario se maximiza, su título se combina con el del formulario MDI primario.
- En tiempo de diseño, los formularios MDI secundarios se muestran de la misma forma que el resto y sólo se mostrarán dentro del formulario primario en tiempo de ejecución. El icono de un formulario MDI secundario en la ventana Proyecto es distinto de los iconos de otros tipos de formularios.
- Los formularios MDI secundarios no pueden ser modales.
- El entorno operativo Microsoft Windows controla el tamaño y la posición iniciales de los formularios MDI secundarios, a menos que los establezca específicamente en el procedimiento de evento Load.
- Si se hace referencia a un formulario MDI secundario antes de cargar el formulario MDI primario, éste se cargará automáticamente. Sin embargo, si se hace referencia al formulario MDI primario antes de cargar un formulario MDI secundario, el formulario MDI secundario no se cargará.

Parte 3: Tipos De Datos, Constantes y Variables

Tipos de Datos

Un tipo de dato determina la naturaleza del dominio de valores que puede tomar una variable, las operaciones en que puede participar y el espacio de memoria que necesita. La tabla siguiente muestra los tipos de datos, incluyendo el tamaño de almacenamiento y el intervalo.

Tipo de Dato	Tamaño de Almacenamiento	Rango
Byte	1 byte	0 a 255
Boolean	2 bytes	True o False
Integer	2 bytes	-32.768 a 32.767
Long (entero largo)	4 bytes	-2.147.483.648 a 2.147.483.647
Single (coma flotante/ precisión simple)	4 bytes	-3,402823E38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
Double (coma flotante/ precisión doble)	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
Currency (entero a escala)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
Decimal	14 bytes	+/- 79.228.162.514.264.337.593.543.950.335 sin punto decimal; +/- 7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/- 0,00000000000000000000000000000001
Date	8 bytes	1 de enero de 100 a 31 de Diciembre de 9999
Object	4 bytes	Cualquier referencia a tipo Object
String (longitud variable)	10 bytes + longitud de la cadena	Desde 0 a 2.000 millones
String (longitud fija)	Longitud de la cadena	Desde 1 a 65.400 aproximadamente
Variant (con números)	16 bytes	Cualquier valor numérico hasta el intervalo de un tipo Double
Variant (con caracteres)	22 bytes + longitud de la cadena	El mismo intervalo para un tipo String de longitud variable.
Definido por el usuario (utilizando Type)	Número requerido por los elementos	El intervalo de cada elemento es el mismo que el intervalo de su tipo de datos

Nota: Las matrices de cualquier tipo de datos requieren 20 bytes de memoria más cuatro bytes para cada dimensión de matriz, más el número de bytes que ocupan los propios datos. Por ejemplo, los datos de una matriz unidimensional que constan de cuatro elementos de datos tipo **Integer** de dos bytes cada uno, ocupan ocho bytes. Los ocho bytes que requieren los datos más los 24 bytes necesarios para la matriz suman un requisito total de memoria de 32 bytes para dicha matriz. Un tipo **Variant** que contiene una matriz requiere 12 bytes más que la matriz por sí sola.

Convertir tipos de datos

Visual Basic proporciona varias funciones de conversión que puede utilizar para convertir valores de tipos de datos específicos. Por ejemplo, para convertir un valor a **Currency**, utilice la función **Ccur**:

```
PagoPorSemana = Ccur (horas * PagoPorHora)
```

La siguiente tabla muestra las funciones de conversión:

Funciones de Conversión	Convierten una expresión en
Cbool	Boolean
Cbyte	Byte
Ccur	Currency
Cdate	Date
CDbl	Double
Cint	Integer
CLng	Long
CSng	Single
CStr	String
Cvar	Variant
CVErr	Error

Nota: Los valores que se pasan a una función de conversión deben ser válidos para el tipo de dato de destino o se producirá un error. Por ejemplo, si intenta convertir un tipo **Long** en un **Integer**, el tipo de **Long** debe de estar en el intervalo válido del tipo de dato **Integer**.

Variables

Las variables se utilizan para almacenar valores temporalmente durante la ejecución de una aplicación. Las variables tienen un nombre (la palabra que utiliza para referirse al valor que contiene la variable) y un tipo de dato (que determina la clase de dato que la variable puede almacenar).

Almacenamiento y recuperación de datos en variables

Utilice una sentencia de asignación para realizar cálculos y asignar el resultado a una variable:

```
ManzanaVendidas = 10 ' Se pasa el valor 10 a la variable
ManzanasVendidas = ManzanasVendidas + 1
```


Observe que el signo igual del ejemplo es un operador de asignación, no un operador de igualdad; el valor **10** se asigna a la variable **ManzanasVendidas**.

Declaración de Variables

Declarar una variable es decirle al programa algo de antemano. Se declara una variable mediante la instrucción **Dim**, proporcionando un nombre a la variable, según la siguiente sintaxis:

Dim nombre-variable [**As** tipo]

Las variables que se declaran en un procedimiento mediante la sentencia **Dim** sólo existen mientras se ejecuta el procedimiento. Cuando termina el procedimiento, desaparece el valor de la variable. Además, el valor de una variable de un procedimiento es **local** a dicho procedimiento; es decir, no puede tener acceso a una variable de un procedimiento desde otro procedimiento. Estas características le permiten utilizar los mismos nombres de variables en distintos procedimientos sin preocuparse por posibles conflictos o modificaciones accidentales.

El nombre de una variable debe cumplir con los siguientes requisitos:

- Debe comenzar con una letra.
- No puede incluir un punto o un carácter de declaración de tipo (\$,&!,%,#,@).
- No debe exceder de 255 caracteres.
- Debe ser única en el mismo alcance, que es el intervalo desde el que se puede hacer referencia a la variable: un procedimiento, formulario, etc.

La cláusula opcional **As** tipo de la sentencia **Dim** le permite definir el tipo de dato o de objeto de la variable que va a declarar. Los tipos de datos definen el tipo de información que almacena la variable. Algunos ejemplos de tipos de datos son **String**, **Integer** y **Currency**. Las variables también pueden contener objetos de Visual Basic u otras aplicaciones. Algunos ejemplos de tipos de objeto de Visual Basic, o clases, son **Object**, **Form1** y **TextBox**.

Hay otras formas de declarar variables:

- Declarar una variable en la sección Declaraciones de un módulo de formulario, estándar o de clase, en vez de un procedimiento, hace que la variable esté disponible para todos los procedimientos del módulo.
- Declarar una variable mediante la palabra clave **Public** hace que esté accesible para toda la aplicación.
- Declarar una variable local mediante la palabra clave **Static** conserva su valor aunque termine el procedimiento.

Declaración Implícita

No tiene por qué declarar una variable antes de utilizarla. Por ejemplo, podría escribir una función donde no hiciera falta declarar **TempVal** antes de utilizarla:

```
Function Raíz (num)
    TempVal = Abs (num)
    Raíz = Sqr(TempVal)
End Function
```

Visual Basic crea automáticamente una variable con ese nombre, que puede utilizar como si la hubiera declarado explícitamente. Aunque es cómodo, puede provocar errores sutiles en el código si se equivoca de nombre de variable. Por ejemplo, suponga que ha escrito esta función:

```
Function Raíz (num)
    TempVal = Abs (num)
    Raíz = Sqr (TempVal)
End Function
```

A primera vista, parece igual. Pero como se ha escrito erróneamente la variable **TempVal** en la tercera línea, la función devolverá siempre cero. Cuando Visual Basic encuentra un nombre nuevo, no puede averiguar si realmente desea declarar una variable nueva o simplemente ha escrito de forma errónea una variable existente, por lo que crea una variable nueva con ese nombre.

Declaración Explícita

Para evitar problemas al equivocarse de nombre en las variables, puede configurar Visual Basic para que le avise siempre que encuentre un nombre que no se haya declarado explícitamente como una variable.

Para declarar variables de forma explícita:

- Incluya esta sentencia en la sección *Declaraciones Generales* del módulo de clase, de formulario o estándar:

```
Option Explicit
```

- o bien -

En el menú **Herramientas**, elija **Opciones**, haga clic en la ficha **Editor** y active la opción **Declaración de variables requerida**. Esto inserta automáticamente la sentencia **Option Explicit** en los módulos nuevos, pero no en los ya creados, por lo que tendrá que agregar manualmente **Option Explicit** a los módulos existentes en el proyecto.

Si hubiera tenido efecto dicha instrucción en el módulo de formulario o módulo estándar que contiene la función **Raíz**, Visual Basic habría reconocido TempVal y TemVal como variables no declaradas y habría generado errores para ambas. Debería, por tanto, declarar explícitamente TempVal:

```
Function Raíz (num)
    Dim TempVal
    TempVal = Abs (num)
    Raíz = Sqr (TempVal)
End Function
```

Alcance de las Variables

El alcance de una variable define en qué partes del código son reconocidas. Cuando declara una variable en un procedimiento, sólo el código de dicho procedimiento puede tener acceso o modificar el valor de la variable; tiene un alcance **local** al procedimiento. A veces, sin embargo, se necesita utilizar

una variable con un alcance más general, como aquella cuyo valor está disponible para todos los procedimientos del mismo módulo o incluso para todos los procedimientos de toda la aplicación. Visual Basic le permite especificar el alcance de una variable cuando la declara.

Establecido el alcance de las variables

Dependiendo de cómo se declara, una variable tiene como alcance un procedimiento (local) o un módulo.

Alcance	Privado	Público
Nivel de procedimiento	Las variables son privadas al procedimiento donde se declaran.	No es aplicable. No puede declarar variables públicas dentro de un procedimiento.
Nivel de módulo	Las variables son privadas al módulo donde se declaran.	Las variables están disponibles para todos los módulos.

Variables utilizadas en un procedimiento

Las variables al nivel de procedimiento sólo se reconocen en el procedimiento en el que se han declarado. Se las conoce también como variables locales. Se declaran mediante las palabras clave **Dim** o **Static**. Por ejemplo:

```
Dim intTemp As Integer
-      o bien      -
Static intContador As Integer
```

Los valores de variables locales declaradas con **Static** existen mientras se ejecuta la aplicación, mientras que las variables declaradas con **Dim** sólo existen mientras se ejecuta el procedimiento.

Variables utilizadas en un módulo

De forma predeterminada, una variable al nivel de módulo está disponible para todos los procedimientos del módulo, pero no para el código de otros módulos. Cree variables al nivel de módulo declarándolas con la palabra clave **Private** en la sección **Declaraciones Generales** al principio del módulo. Por ejemplo:

```
Private intTemp As Integer
```

Al nivel de módulo, no hay diferencia entre **Private** y **Dim**, pero es preferible usar **Private** porque contrasta con **Public** y hace que el código sea más fácil de comprender.

Variables utilizadas por todos los módulos

Para hacer que una variable al nivel de módulo esté disponible para otros módulos, utilice la palabra clave **Public** para declararlas. Los valores de las variables públicas están disponibles para todos los procedimientos de la aplicación. Al igual que todas las variables al nivel del módulo, las variables públicas se declaran en la sección **Declaraciones Generales** al principio del módulo. Por ejemplo:

```
Public intTemp As Integer
```

Nota: No puede declarar variables públicas en un procedimiento, sólo en la sección **Declaraciones Generales** de un módulo.

Constantes

A menudo verá que el código contiene valores constantes que reaparecen una y otra vez. O puede que el código dependa de ciertos números que resulten difíciles de recordar (números que, por sí mismos, no tienen un significado obvio).

En estos casos, puede mejorar mucho la legibilidad del código y facilitar su mantenimiento si utiliza constantes. Una **constante** es un nombre significativo que sustituye a un número o una cadena que no varía. Aunque una constante recuerda ligeramente a una variable, no puede modificar una constante o asignarle un valor nuevo como ocurre con una variable. Hay dos orígenes para las constantes:

- Constantes **intrínsecas** o **definidas por el sistema** proporcionadas por Visual Basic.
- Las constantes **simbólicas** o **definidas por el usuario** se declaran mediante la instrucción **Const**.

Creación de sus propias constantes

La sintaxis para declarar una constante es la siguiente:

[Public | Private] Const nombre_constante **[As tipo]** = expresión

El argumento *nombre_constante* es un nombre simbólico válido (las reglas son las mismas que para crear nombres de variable) y *expresión* está compuesta por constantes y operadores de cadena o numéricos; sin embargo, no puede utilizar llamadas a funciones en *expresión*. Una instrucción **Const** puede representar una cantidad matemática o de fecha y hora:

```
Const conPi = 3.14159265358979
Public Const conMaxPlanetas As Integer = 9
Const conFechaSalida = #1/1/95#
```

Se puede utilizar también la instrucción **Const** para definir constantes de cadena:

```
Public Const conVersion = " 07.10.A"
Const conNombreClave = "Enigma"
```

Puede colocar más de una declaración de constante en una única línea si las separa con comas:

```
Public Const conPi=3.14, conMaxPlanetas=9, conPobMundial=6E+09
```

A menudo, la expresión del lado derecho del signo igual (=) es un número o cadena literal, pero también puede ser una expresión que dé como resultado un número o una cadena (aunque la *expresión* no puede contener llamadas a funciones). Puede incluso definir constantes en términos de constantes previamente definidas:

```
Const conPi2 = conPi * 2
```

Una vez que defina las constantes, puede colocarlas en el código para hacerlo más legible. Por ejemplo:

```
Static SistemaSolar (1 To conMaxPlanetas)  
If numPersonas > conPopMundial Then Exit Sub
```

Alcance de las constantes definidas por el usuario

Una instrucción **Const** tiene igual alcance que una declaración de variable y se le aplican las mismas reglas:

- Para crear una constante que sólo exista en un procedimiento, declárela dentro del procedimiento.
- Para crear una constante disponible para todos los procedimientos de un módulo, pero no para el código que está fuera del módulo, declárela en la sección **Declaraciones Generales** del módulo.
- Para crear una constante disponible en toda la aplicación, declare la constante en la sección **Declaraciones Generales** de un módulo estándar y coloque delante de **Const** la palabra clave **Public**. No se pueden declarar las constantes públicas en un módulo de clase o de formulario.

Evitar referencias circulares

Como es posible definir constantes en términos de otras constantes, deberá tener cuidado para no establecer un *ciclo* o referencia circular entre dos o más constantes. Se produce un ciclo cuando se tienen dos o más constantes públicas, cada una de las cuales está definida en función de la otra. Por ejemplo:

```
'En el Módulo 1:  
Public Const conA = conB * 2 ' Disponible en toda la aplicación  
  
'En el Módulo 2:  
Public Const conB = conA / 2 ' Disponible en toda la aplicación
```

Si se produce un ciclo, Visual Basic generará un error cuando intente ejecutar la aplicación. No puede ejecutar el código hasta que resuelva la referencia circular. Para evitar la creación de un ciclo, restrinja todas las constantes públicas a un único módulo o, al menos, al menor número posible de módulos.

Convenciones para Nombres de Constantes y Variables

Las variables se deben definir siempre con el menor alcance posible. Las variables globales (públicas) pueden hacer lógica de una aplicación muy difícil de entender. Las variables globales también hacen más difícil mantener y volver a usar el código.

En Visual Basic las variables pueden tener el alcance siguiente:

Alcance	Declaración	Visible en
Nivel de procedimiento	Dim o Static en el Procedimiento, Subprocedimiento o Función	El procedimiento en el que está declarada

Alcance	Declaración	Visible en
Nivel de módulo	Private en la sección <i>Declaraciones Generales</i> de un módulo de formulario o de código (.frm, .bas)	Todos los procedimientos del módulo de formulario o de código
Global	Public en la sección <i>Declaraciones Generales</i> de un módulo de código (.bas)	En toda de aplicación

En una aplicación de Visual Basic, las variables globales se deben usar sólo cuando no exista ninguna otra forma cómoda de compartir datos entre formularios. Cuando haya que usar variables globales, es conveniente declararlas todas en un único módulo agrupadas por funciones y dar al módulo un nombre significativo que indique su finalidad, como Públicas.

Una práctica de codificación correcta es escribir código modular siempre que sea posible. Por ejemplo, si la aplicación muestra un cuadro de diálogo, coloque todos los controles y el código necesario para ejecutar la tarea del diálogo en un único formulario. Esto ayuda a tener código de la aplicación organizado en componentes útiles y minimiza la sobrecarga en tiempo de ejecución.

A excepción de las variables globales (que no se deberían pasar), los procedimientos y funciones deben operar sólo sobre los objetos que se les pasan.

Prefijos de alcance de variables

A medida que aumenta el tamaño del proyecto, también aumenta la utilidad de reconocer rápidamente el alcance de las variables. Esto se consigue escribiendo un prefijo de alcance de una letra delante del prefijo de tipo, sin aumentar demasiado la longitud del nombre de las variables.

Alcance	Prefijo	Ejemplo
Global	g	gstrNombreUsuario
Nivel de módulo	m	mbInProgresoDelCálculo
Local del Procedimiento	Ninguno	dblVelocidad

Una variable tiene alcance *global* si se declara como **Public** en un módulo estándar o en un módulo de formulario. Una variable tiene alcance de *nivel de módulo* si se declara como **Private** en un módulo estándar o en un módulo de formulario, respectivamente.

Nota: La coherencia es crucial para usar esta técnica de forma productiva; el corrector de sintaxis de Visual Basic no interceptará las variables de nivel de módulo que comience con "p".

Constantes

El nombre de las constantes se debe escribir en mayúsculas y minúsculas, con la letra inicial de cada palabra en mayúsculas. Aunque las constantes estándar de Visual Basic no incluyen información de tipo

de datos y el alcance de una constante. Para los nombres de constantes, se deben seguir las mismas normas que para las variables. Por ejemplo:

```

mintMáxListaUsuario    ' Límite de entradas máximas para la
                        ' lista de usuarios
                        ' (valor entero, local del módulo)
gstrNuevaLínea          ' Carácter de nueva línea
                        ' (cadena, global de la aplicación)

```

Variables

Declarar todas las variables ahorra tiempo de programación porque reduce el número de errores debidos a nombres de variables errados (por ejemplo, aNombreUsuarioTmp frente a sNombreUsuarioTmp frente a sNombreUsuarioTemp). En la ficha **Editor** del cuadro de diálogo **Opciones**, active la opción **Declaración de variables requerida**. La instrucción **Option Explicit** requiere que declare todas las variables del programa de Visual Basic.

Las variables deben llevar un prefijo para indicar su tipo de datos. Opcionalmente, y en especial para programas largos, el prefijo se puede ampliar para indicar el alcance de la variable.

Tipos de datos de variables

Use los siguientes prefijos para indicar el tipo de datos de una variable.

Tipo de datos	Prefijo	Ejemplo
Boolean	bln	blnEncontrado
Byte	byt	bytDatosImagen
Objeto Collection	col	colWidgets
Currency	cur	curIngresos
Date (Time)	dtm	dtmInicio
Double	dbl	dblTolerancia
Error	err	errNúmDeOrden
Integer	int	intCantidad
Long	lng	lngDistancia
Object	obj	objActivo
Single	sng	sngMedia
String	str	strNombreF
Definido por el usuario	udt	udtEmpleado
Variant	vnt	vntChecksum

Nombres descriptivos de variables y procedimientos

El cuerpo de un nombre de variable o procedimiento se debe escribir en mayúsculas y minúsculas y debe tener la longitud necesaria para describir su funcionalidad. Además, los nombres de funciones deben empezar con un verbo, como IniciarNombreMatriz o CerrarDiálogo.

Para nombres que se usen con frecuencia o para términos largos, se recomienda usar abreviaturas estándar para que los nombres tengan una longitud razonable. En general, los nombres de variables con más de 32 caracteres pueden ser difíciles de leer en pantalla VGA.

Cuando se usen abreviaturas, hay que asegurarse de que sean coherentes en toda la aplicación. Alterar aleatoriamente entre Cnt y Contar dentro de un proyecto provoca una confusión innecesaria.

Tipos definidos por el usuario

En un proyecto grande con muchos tipos definidos por el usuario, suele ser útil dar a cada uno de estos tipos un prefijo de tres caracteres. Si estos prefijos comienzan con "u", será fácil reconocerlos cuando se esté trabajando con tipos definidos por el usuario. Por ejemplo, "ucli" se podría usar como prefijo para las variables de un tipo Cliente definido por el usuario.

Tipos Adicionales de Variables

Registros o Estructuras

Son tipos de datos definidos por el usuario. Es básicamente un conjunto de varios datos de tipos elementales agrupados bajo una denominación común. Debe declararse en la sección **Declaraciones Generales** de un módulo. Se usa la palabra reservada **Type**.

Sintaxis

```
Type NombreDelNuevoTipo
    NombreDelElemento1 As TipoDato
    NombreDelElemento2 As TipoDato
    NombreDelElemento3 As TipoDato
    ...
    ...
End Type
```

Ejemplo

```
Type RegEmpleado
    EmpCódigo As Integer
    EmpNombre As String * 40
    EmpCargo As String * 15
End Type
```

Arreglo de Variables

Un arreglo es una colección de elementos del mismo tipo con un nombre común. Los elementos son identificados por el nombre común y un índice.

Sintaxis

***Dim** NombreDelArreglo(Dimensión1, Dimensión2, ...) As TipoDeDato*

Visual Basic soporta hasta 60 dimensiones. Al declarar las dimensiones se puede indicar un solo número, en cuyo caso se entiende que dicha dimensión va de cero hasta el número indicado. También es posible indicar explícitamente el inicio y término de la dimensión.

Ejemplo

```
Dim Lista1 (20) As Integer           ' 21 elementos, del 0 al 20
Dim Lista2 (1 to 20) As Integer      ' 20 elementos, del 1 al 20
Dim Tabla (1 to 10, 1 to 20) As String ' Tabla de 10 x 20
```

Arreglos Dinámicos

Hay situaciones en las cuales se desea usar un arreglo, pero al momento del diseño no se sabe sus dimensiones. Para este tipo de situaciones Visual Basic permite declaraciones de arreglos del siguiente modo:

```
Dim x ( ) As Integer
...
...
Redim x (lstLista.ListCount)
```

Cuando se redimensiona un arreglo, los valores almacenados anteriormente se pierden, porque cada elemento es reinicializado con cero o null dependiendo del tipo de dato del elemento. Si se desea preservar los valores debemos usar la palabra clave **Preserve** en la sentencia de redimensionamiento.

```
Redim Preserve x ( intNúmeroDeElementos )
```

OPERADORES

Aritméticos

^	Exponenciación
*	Multiplicación
/	División
	División entera
Mod	Residuo entero (Ejm: A Mod B)
+	Suma
-	Resta
&	Concatenación de cadenas

Comparación

=	Igual
<>	Distinto

>	Menor que
<=	Menor o igual
>=	Mayor o igual
Like	Compara dos cadenas
*	Cero o más caracteres (Ejm: cad Like "ma*")
?	Cualquier carácter
#	Cualquier dígito (0-9)
[lista]	cualquier carácter en lista
[!lista]	cualquier carácter que no esta en lista
Is	Usado para comparar dos variables de referencia a objetos

Lógicos

And	"Y" lógico
Or	"O" lógico
Xor	"O" Exclusivo
Not	Negación

Parte 4: Estructuras de Control

Estructuras de Control

Las estructuras de control le permiten controlar el flujo de ejecución del programa. Tenemos dos tipos de estructuras de control:

- Estructuras de decisión
- Estructuras de bucle

Estructuras de Decisión

Los procedimientos de Visual Basic pueden probar condiciones y, dependiendo de los resultados, realizar diferentes operaciones. Entre las estructuras de decisión que acepta Visual Basic se incluyen las siguientes:

- *If...Then*
- *If...Then...Else*
- *Select Case*

If...Then

Use la estructura **If...Then** para ejecutar una o más instrucciones basadas en una condición. Puede utilizar la sintaxis de una línea o un bloque de varias líneas:

- *If condición Then Sentencias*
- *If condición Then*
 Sentencias
End If

Condición normalmente es una comparación, pero puede ser cualquier expresión que dé como resultado un valor numérico. Visual Basic interpreta este valor como **True** o **False**; un valor numérico cero es **False** y se considera **True** cualquier valor numérico distinto de cero. Si **condición** es **True**, Visual Basic ejecuta todas las *sentencias* que siguen a la palabra clave **Then**. Puede utilizar sintaxis de una línea o de varias líneas para ejecutar una sentencia basada en una condición, los siguientes dos ejemplos son equivalentes:

1 `If cualquierFecha < Now Then CualquierFecha = Now`

2 `If cualquierFecha < Now Then
 CualquierFecha = Now
End If`

Observe que el formato de una única línea de **If...Then** no utiliza la instrucción **End If**. Si se desea ejecutar más de una línea de código cuando **condición** sea **True**, debe utilizar la sintaxis de bloque de varias líneas **If...Then...End If**.

```

1 If cualquierFecha < Now Then
        CualquierFecha = Now
        Timer1.Enabled = False ' Desactiva el control Timer.
End If

```

```

2 If chkAlumnoUNI.Value=1 Then
        txtCosto = Format (txtCosto*0.70,"Fixed")
        txtCódigo.Enabled = True
End If

```

If...Then...Else

Utilice un bloque **If...Then...Else** para definir varios bloques de sentencias, uno de los cuales se ejecutará:

```

If condición1 Then
    [bloque de sentencias 1]
[ElseIf] condición2 Then
    [bloque de sentencias 2] ...
[Else]
    [bloque de sentencias n]
End If

```

Visual Basic evalúa primero **condición1**. Si es **False**, Visual Basic procede a evaluar **condición2** y así sucesivamente, hasta que encuentre una condición **True**. Cuando encuentra una condición **True**, Visual Basic ejecuta el bloque de instrucciones correspondientes y después ejecuta el código que sigue a **End If**. Opcionalmente, puede incluir un bloque de instrucciones **Else**, que Visual Basic ejecutará si ninguna de las condiciones es **True**.

If...Then...ElseIf es un caso especial de **If...Then...Else**. Observe que puede tener cualquier número de cláusula **ElseIf** o ninguna. Puede incluir una cláusula **Else** sin tener en cuenta si tiene o no cláusula **ElseIf**.

Por ejemplo, la aplicación podría realizar distintas acciones dependiendo del control en que se haya hecho clic de una matriz de controles de menú:

```

1 Private Sub mnuCut_Click (Index As Integer)
        If Index = 0 Then ' Comando Cortar
            CopyActiveControl ' Llama a procedimientos generales
            ClearActiveControl
        ElseIf Index = 1 Then ' Comando Copiar
            CopyActiveControl
        ElseIf Index = 2 Then ' Comando Borrar
            ClearActiveControl
        Else ' Comando Pegar
            PasteActiveControl
        End If
End Sub

```

2

```

If ClaveUsuario="DSI" Then
    ' Permite al usuario entrar al sistema
    ...
Else
    ' Mostrar un mensaje advirtiendo error en la clave
    ...
End If

```

3

```

Private Sub DeterminaCondición ( )
    If Val (txtPromedio) >=13 Then
        txtCondición = "Aprobado"
    ElseIf Val (txtPromedio) >= 10 Then
        txtCondición = "Asistente"
    Else
        txtCondición = "Desaprobado"
    End If
End Sub

```

Observe que siempre puede agregar más cláusulas **ElseIf** a la estructura **If...Then**. Sin embargo, esta sintaxis puede resultar tediosa de escribir cuando cada **ElseIf** compara la misma expresión con un valor distinto. Para estas situaciones, puede utilizar la estructura de decisión **Select Case**.

Select Case

Visual Basic proporciona la estructura **Select Case** como alternativa a **If...Then...Else** para ejecutar selectivamente un bloque de sentencias entre varios bloques. La sentencia **Select Case** ofrece posibilidades similares a la instrucción **If...Then...Else**, pero hace que el código sea más legible cuando hay varias opciones.

La estructura **Select Case** funciona con una única expresión de prueba que se evalúa una vez solamente, al principio de la estructura. Visual Basic compara el resultado de esta expresión con los valores de cada **Case** de la estructura. Si hay una coincidencia, ejecuta el bloque de sentencias asociado a ese **Case**:

```

Select Case expresión_prueba
    [Case lista_expresiones1
        [bloque de sentencias 1]]
    [Case lista_expresiones2
        [bloque de sentencias 2]]
    .
    .
    .
    [Case Else
        [bloque de sentencias n]]
End Select

```

Cada **lista_expresiones** es una lista de uno a más valores. Si hay más de un valor en una lista, se separan los valores con comas. Cada **bloque de sentencias** contiene cero o más instrucciones. Si más de un **Case** coincide con la expresión de prueba, sólo se ejecutará el bloque de instrucciones asociado con la primera coincidencia. Visual Basic ejecuta las instrucciones de la cláusula (opcional) **Case Else** si ningún valor de la lista de expresiones coincide con la expresión de prueba.

Por ejemplo, suponga que agrega otro comando al menú **Edición** en el ejemplo **If...Then...Else**. Podría agregar otra cláusula **Elseif** o podría escribir la función con **Select Case**:

```

1 Private Sub mnuCut_Click (Index As Integer)
    Select Case Index
    Case 0          ' Comando Cortar
        CopyActiveControl ' Llama a procedimientos generales
        ClearActiveControl
    Case 1          ' Comando copiar.
        CopyActiveControl
    Case 2          ' Comando borrar.
        ClearActiveControl
    Case 3          ' Comando Pegar.
        PasteActiveControl
    Case Else
        frmFind.Show ' Muestra el cuadro de
                     ' diálogo Buscar.
    End Select
End Sub

```

```

2 Select Case TipoUsuario
    Case "Supervisor"
        ' Proporciona al usuario privilegios de Supervisor
        ...
        ...
    Case "Usuario"
        ' Proporciona al usuario privilegios de Usuario
        ...
        ...
    Case Else
        ' Proporciona al usuario privilegio de invitado
        ...
        ...
End Select

```

Observe que la estructura **Select Case** evalúa una expresión cada vez que al principio de la estructura. Por el contrario, la estructura **If...Then...Else** puede evaluar una expresión diferente en cada sentencia **Elseif**. Sólo puede sustituir una estructura **If...Then...Else** con una estructura **Select Case** si la instrucción **If** y cada instrucción **Elseif** evalúa la misma expresión.

Otros Ejemplos

```
1 If Ventas > 100000 Then
    strDscto = Format (0.10, "Fixed")
ElseIf Ventas > 50000 Then
    strDscto = Format (0.05, "Fixed")
Else
    strDscto = Format (0.02, "Fixed")
End If
```

```
2 Select Case Cantidad
    Case 1
        sngDscto = 0.0
    Case 2, 3
        sngDscto = 0.05
    Case 4 To 6
        sngDscto = 0.10
    Case Else
        sngDscto = 0.20
End Select
```

```
3 intRpta = MsgBox ("Guarda cambios antes de salir" , vbYesNo)
Select Case intRpta
    Case vbYes
        GuardarCambios
        Unload Me
    Case vbNo
        Unload Me
End Select
```

Estructuras de Repetición

Las estructuras de repetición o bucle le permiten ejecutar una o más líneas de código repetidamente. Las estructuras de repetición que acepta Visual Basic son:

- Do...Loop
- For...Next
- For Each...Next

Do...Loop

Utilice el bucle **Do** para ejecutar un bloque de sentencias un número indefinido de veces. Hay algunas variantes en la sentencia **Do...Loop**, pero cada una evalúa una condición numérica para determinar si

continúa la ejecución. Como ocurre con **If...Then**, la **condición** debe ser un valor o una expresión que dé como resultado **False** (cero) o **True** (distinto de cero).

En el siguiente ejemplo de **Do...Loop**, las **sentencias** se ejecutan siempre y cuando **condición** sea **True**:

Do While *condición*

Sentencias

Loop

Cuando Visual Basic ejecuta este bucle **Do**, primero evalúa **condición**. Si **condición** es **False** (cero), se salta todas las **sentencias**. Si es **True** (distinto de cero) Visual Basic ejecuta las **sentencias**, vuelve a la instrucción **Do While** y prueba la condición de nuevo.

Por tanto, el bucle se puede ejecutar cualquier número de veces, siempre y cuando **condición** sea distinta de cero o **True**. Nunca se ejecutan las **sentencias** si **condición** es **False** inicialmente. Por ejemplo, este procedimiento cuenta las veces que se repite una cadena destino dentro de otra cadena repitiendo el bucle tantas veces como se encuentre la cadena de destino:

```
Function ContarCadenas (cadenalarga, destino)
    Dim posición, contador
    posición = 1
    Do While InStr (posición, cadenalarga, destino)
        posición = InStr (posición, cadenalarga, destino)+1
        contador = contador + 1
    Loop
    ContarCadenas = contador
End Function
```

Si la cadena destino no está en la otra cadena, **InStr** devuelve 0 y no se ejecuta el bucle.

Otra variante de la instrucción **Do...Loop** ejecuta las **sentencias** primero y prueba la **condición** después de cada ejecución. Esta variación garantiza al menos una ejecución de **sentencias**:

Do

Sentencias

Loop While *condición*

Hay otras dos variantes análogas a las dos anteriores, excepto en que repiten el bucle siempre y cuando **condición** sea **False** en vez de **True**.

Hace el bucle cero o más veces

Do Until *condición*

Sentencias

Loop

Hace el bucle al menos una vez

Do

Sentencias

Loop Until *condición*

For...Next

Los bucles **Do** funcionan bien cuando no se sabe cuántas veces se necesitará ejecutar las **sentencias** del bucle. Sin embargo, cuando se sabe que se va a ejecutar las **sentencias** un número determinado de veces, es mejor elegir el bucle **For...Next**. A diferencia del bucle **Do**, el bucle **For** utiliza una variable llamada **contador** que incrementa o reduce su valor en cada repetición del bucle. La sintaxis es la siguiente:

```
For contador = iniciar To finalizar [Step incremento]
    Sentencias
Next [contador]
```

Los argumentos **contador**, **iniciar**, **finalizar** e **incremento** son todos numéricos.

Nota: El argumento **incremento** puede ser positivo o negativo. Si **incremento** es positivo, **iniciar** debe ser menor o igual que **finalizar** o no se ejecutarán las sentencias del bucle. Si **incremento** es negativo, **iniciar** debe ser mayor o igual que **finalizar** para que se ejecute el cuerpo del bucle. Si no se establece **Step**, el valor predeterminado de **incremento** es 1. Al ejecutar el bucle **For**, Visual Basic:

1. Establece contador al mismo valor que **iniciar**.
2. Comprueba si **contador** es mayor que **finalizar**. Si lo es, Visual Basic sale del bucle. (Si **incremento** es negativo, Visual Basic comprueba si **contador** es menor que **finalizar**.)
3. Ejecuta las **sentencias**.
4. Incrementa **contador** en 1 o en **incremento**, si se especificó.
5. Repite los pasos 2 a 4.

Este código imprime los nombres de todas las fuentes de pantalla disponibles:

```
Private Sub Form-Click ( )
    Dim I As Integer
    For i = 0 To Screen.FontCount
        Print Screen.Fonts (i)
    Next
End Sub
```

For Each...Next

El bucle **For Each...Next** es similar al bucle **For...Next**, pero repite un grupo de sentencia por cada elemento de una colección de objetos o de una matriz en vez de repetir las sentencias un número especificado de veces. Esto resulta especialmente útil si no se sabe cuántos elementos hay en la colección. He aquí la sintaxis del bucle **For Each...Next**:

```
For Each elemento In grupo
    Sentencias
Next elemento
```

Por ejemplo, el siguiente procedimiento **Sub** abre la base de datos Biblio.mdb y agrega el nombre de cada tabla a un cuadro de lista.

```
Sub ListTableDefs ( )
    Dim objDb As Database
    Set objDb = OpenDatabase("c:/Archivos de programa/Devstudio/" & _
        "vb/biblio.mdb", True, False)
    For Each TableDef In objDb.TableDefs ( )
        List1.AddItem TableDef.Name
    Next TableDef
End Sub
```

Tenga en cuenta las restricciones siguientes cuando utilice **For Each...Next**:

- Para las colecciones, **elemento** sólo puede ser una variable **Variant**, una variable **Object** genérica o un objeto mostrado en el Examinador de objetos.
- Para las matrices, **elemento** sólo puede ser una variable **Variant**.
- No puede utilizar **For Each...Next** con una matriz de tipos definidos por el usuario porque un **Variant** no puede contener un tipo definido por el usuario.

El siguiente ejemplo habilita todos los **Cuadro de Texto** del formulario:

```
Private Sub ModoEdición ( )
    Dim control
    For Each control In form1.Controls
        If TypeOf control Is TextBox Then
            Control.Enabled = True
        End If
    Next control
End Sub
```

Salida de una Estructura de Control

La instrucción **Exit** le permite salir directamente de un bucle **For** o de un bucle **Do**. La sintaxis de la sentencia **Exit** es sencilla: **Exit For** puede aparecer tantas veces como sea necesario dentro de un bucle **For** y **Exit Do** puede aparecer tantas veces como sea necesario dentro de un bucle **Do**:

```
For contador = iniciar To finalizar [Step incremento]
    [bloque sentencias]
    [Exit For]
    [bloque sentencias]
Next [contador]
```

```
Do [{While / Until} condición]
    [bloque de sentencias]
    [Exit Do]
    [bloque de sentencias]
Loop
```

```
Do
    [bloque de sentencias]
    [Exit Do]
    [bloque de sentencias]
Loop [{While / Until} condición]
```

Exit For y **Exit Do** son muy útiles ya que, algunas veces, resulta apropiado salir inmediatamente de un bucle sin realizar más iteraciones o sentencias dentro del bucle.

Cuando utilice la instrucción **Exit** para salir de un bucle, el valor de la variable contador difiere, dependiendo de cómo haya salido del bucle:

- Cuando termina un bucle, la variable **contador** contiene el valor del límite superior más el paso.
- Cuando sale de un bucle prematuramente, la variable **contador** conserva su valor según las reglas usuales del alcance.
- Cuando sale antes del final de una colección, la variable **contador** contiene **Nothing** si se trata de un tipo de dato **Object** y **Empty** si es un tipo de dato **Variant**.

Parte 5: Controles Estándar Adicionales y Procedimientos

Control Marco (Frame)



Este control permite agrupar otros controles para darle mayor funcionalidad a la interfaz. Los controles **Botones de Opción** necesariamente tienen que estar agrupados por el control **Marco**. Para agrupar controles, dibuje primero el control Marco y, a continuación, dibuje los controles dentro de Marco.

Propiedades

Caption	Título de marco.
Enabled	Determina si está habilitado para responder a las acciones del usuario.
Name	Nombre del control.
Visible	Determina si el Marco y los controles que contiene están visibles o no.

Control Casilla de Verificación (CheckBox)



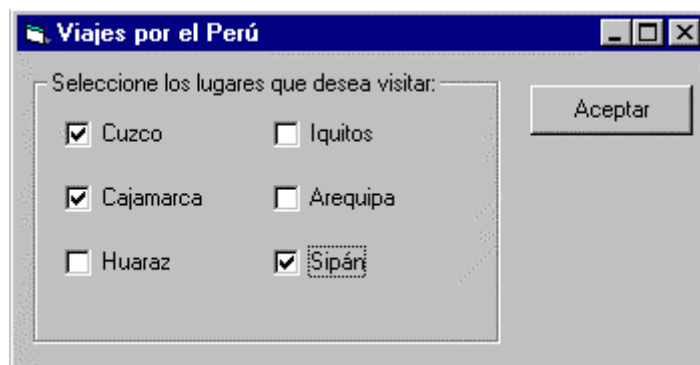
Las casillas de verificación se utilizan para proporcionar al usuario opciones de tipo Si/No o Verdadero/Falso. Cuando el usuario selecciona una opción (activa la casilla), aparece una marca de verificación (✓) dentro de la casilla.

Propiedades

Caption	Descripción que acompaña a la casilla.
Enabled	True/False. Determina si está habilitado para responder a las acciones del usuario.
Name	Nombre del control.
Value	0 – Unchecked (Vacío, no marcado) 1 – Checked (Marcado) 2 – Grayed (Gris, Indefinido)
Visible	Determina si la casilla está visible o no.

Eventos

Click	Ocurre cuando el usuario hace clic sobre la casilla.
-------	------------------------------------------------------



Control Botón de Opción (OptionButton)



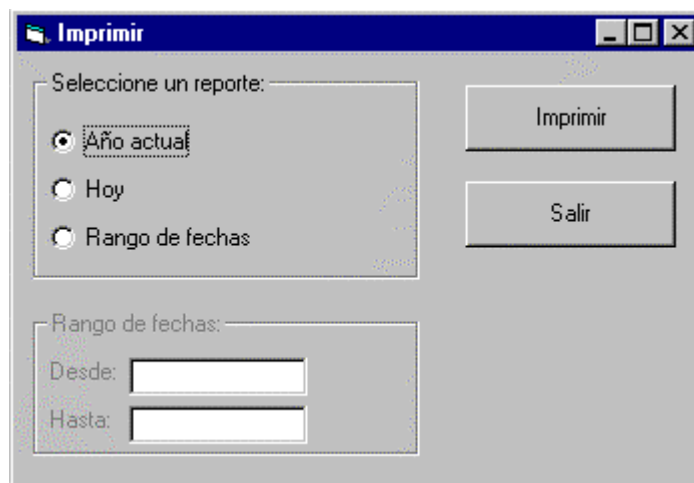
Estos controles se utilizan para que el usuario seleccione una opción de un grupo opciones. La opción seleccionada tiene un punto en el centro.

Propiedades

Caption	Descripción que acompaña a la opción.
Enabled	True/False. Determina si está habilitado para responder a las acciones del usuario.
Name	Nombre del control.
Value	True/False, marcado o no marcado.
Visible	True/False. Determina si el botón está visible o no.

Eventos

Click Ocurre cuando el usuario hace clic sobre el botón.



Arreglo de Controles

Cuando creamos un arreglo de controles, todos los controles que forman el arreglo deben tener el mismo nombre (Propiedad Name), la propiedad **Index** establece el índice de cada control en el arreglo, esta propiedad comienza desde 0.

Un arreglo de controles es un grupo de controles que comparten el mismo:

- Tipo de objeto
- Nombre del control
- Procedimientos de evento

Código más fácil de escribir y mantener

Los arreglos de controles hace que el código sea más fácil de escribir y mantener debido a que solo escribe un solo procedimiento de evento para todos los controles pertenecientes al arreglo, el parámetro **Index** del procedimiento de evento determina desde que control ocurrió el evento.

Código más eficiente

Los arreglos de controles hacen que el código sea más eficiente y mejoran el rendimiento de la aplicación debido a que usa menos recursos del sistema que los controles individuales.

¿Cómo se crea un arreglo de controles?

Luego de crear y establecer el nombre del primer control, tiene dos alternativas:

Método 1

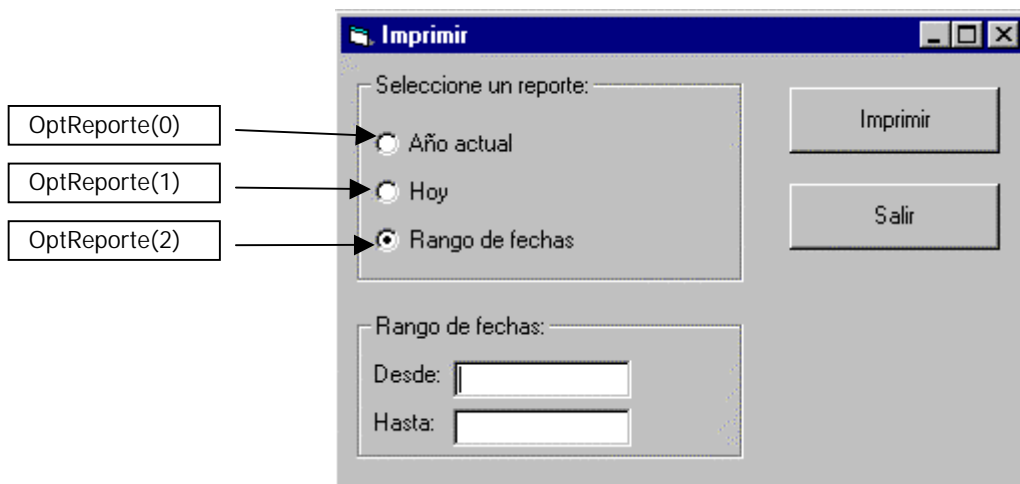
1. Copiar el primer control al portapapeles.
2. Luego pegar el control de portapapeles en el formulario. Visual Basic le preguntará si desea crear un arreglo de controles.
3. Repita el paso 2 hasta completar el arreglo.

Método 2

1. Dibuje el siguiente control y establezca su nombre igual al del primer control. Visual Basic le preguntará si desea crear un arreglo de controles.
2. Repita el paso 1 hasta completar el arreglo.

Ejemplo

La interfaz mostrada tiene un arreglo de controles para seleccionar el reporte, el procedimiento de evento es el mismo para cualquier elemento del arreglo.



El código para el evento clic es:

```
Private Sub optReporte_Click(Index As Integer)
    Select Case Index
        Case 0, 1
            fraRango.Enabled = False
            lblDesde.Enabled = False
            lblHasta.Enabled = False
            txtDesde.Enabled = False
            txtHasta.Enabled = False
        Case 2
```

```

        fraRango.Enabled = True
        lblDesde.Enabled = True
        lblHasta.Enabled = True
        txtDesde.Enabled = True
        txtHasta.Enabled = True
        txtDesde.SetFocus
    End Select
End Sub

```

Control Barra de Desplazamiento



Barra de Desplazamiento Horizontal (HScrollBar)



Barra de Desplazamiento Vertical (VScrollBar)

Son usados con frecuencia para permitir rápidos desplazamientos a través de una lista grande de items. Por ejemplo: archivos, indicadores de posición actual de una escala de valores, indicadores de volumen en un sistema de audio.

Propiedades

LargeChange	De 1 a 32767. Cantidad de cambio cuando el usuario hace clic en el desplazamiento largo.
Max	Máximo valor de desplazamiento cuando el botón se encuentra en la posición más alta. Valor predeterminado: 32767.
Min	Mínimo valor del desplazamiento cuando el botón se encuentra en la posición más baja. Valor predeterminado: 0.
Name	Nombre del control.
SmallChange	De 1 a 32767. Cantidad de cambio cuando el usuario hace clic en la flecha de desplazamiento.
Value	Valor actual en la escala de valores.

Evento

Change	Ocurre cuando el usuario se desplaza o se cambia el valor de la propiedad Value.
--------	----------------------------------------------------------------------------------

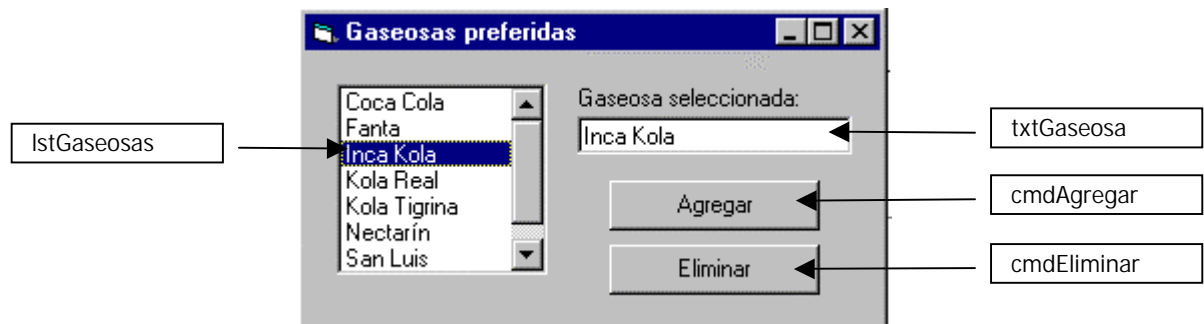
Control Cuadro de Lista (ListBox)



Un control **ListBox** muestra una lista de elementos entre los cuales el usuario puede seleccionar uno o más elementos. Si el número de elementos supera el número que puede mostrarse, se agregará automáticamente una barra de desplazamiento al control **ListBox**.

La propiedad **List** es un arreglo que contiene los elementos de la lista, y comienza con índice 0. La propiedad **ListCount** establece el número total de elementos de la lista. La propiedad **ListIndex** contiene el índice del elemento seleccionado, el cual es un número entre 0 (primer elemento) y el número total de elementos en la lista -1 (**ListCount - 1**). Si no se selecciona ningún elemento, el valor de la propiedad **ListIndex** será -1.

La propiedad **NewIndex** contiene el índice del último elemento añadido a la lista. Esto puede ser útil si desea hacer algo con el elemento añadido, por ejemplo, que sea el elemento actualmente seleccionado.



Propiedades

Enabled	True/False. Determina si el control responde a las acciones del usuario.
List	Arreglo con los elementos de la lista.
ListCount	Número de elementos de la lista.
ListIndex	Elemento seleccionado.
MultiSelect	Establece si es posible seleccionar varios elementos o uno solo.
Name	Nombre del control.
NewIndex	Índice del último elemento añadido al Cuadro de Lista.
Selected	Arreglo de valores lógicos paralelo y del mismo tamaño al arreglo list , indica que elementos han sido seleccionados (True) de la lista. Se utiliza en lugar de ListIndex cuando establecemos la propiedad MultiSelect en 1 ó 2.
Sorted	True/False. Establece los elementos se ordenan alfabéticamente.
Style	Establece el comportamiento del control.
Text	Devuelve el elemento seleccionado en el cuadro de lista; el valor de retorno es siempre equivalente al que devuelve la expresión List(ListIndex). Es de sólo lectura en tiempo de diseño y es de sólo lectura en tiempo de ejecución.

Métodos

AddItem	Permite añadir nuevos elementos a la lista.
RemoveItem	Permite eliminar elementos de la lista.

Eventos

Click	Ocurre cuando el usuario interactúa con el control.
-------	-----------------------------------------------------

Ejemplos

Muestra en el Cuadro de Texto txtGaseosa el elemento seleccionado

```
Private Sub lstGaseosas_Click()  
    txtGaseosa.Text = lstGaseosas.Text  
End Sub
```


Añade un nuevo elemento al Cuadro de Lista lstGaseosas

```

Private Sub cmdAgregar_Click()
    Dim strNuevoElemento As String
    strNuevoElemento = InputBox("Ingrese una nueva gaseosa:", _
        "Nueva gaseosa")
    If Trim(strNuevoElemento) <> "" Then
        lstGaseosas.AddItem strNuevoElemento
    End If
End Sub

```

Elimina el elemento actual del Cuadro de Lista lstGaseosas

```

Private Sub cmdEliminar_Click()
    If lstGaseosas.ListIndex <> -1 Then
        lstGaseosas.RemoveItem lstGaseosas.ListIndex
    End If
End Sub

```

Control Cuadro Combinado (ComboBox)

Un control **ComboBox** combina las características de un control **TextBox** y un control **ListBox**; los usuarios pueden introducir información en la parte del cuadro de texto o seleccionar un elemento en la parte de cuadro de lista del control.

Para agregar o eliminar elementos en un control **ComboBox**, se usa el método **AddItem** o **RemoveItem**. Establezca las propiedades **List**, **ListCount** y **ListIndex** para permitir a un usuario tener acceso a los elementos de un control **ComboBox**. Como alternativa, puede agregar elementos a la lista mediante la propiedad **List** en tiempo de diseño.

Propiedades

Enabled	True/False. Determina si el control responde a las acciones del usuario.
List	Arreglo con los elementos de la lista.
ListCount	Número de elementos de la lista
ListIndex	Elemento seleccionado.
Name	Nombre del control.
NewIndex	Índice del último elemento añadido al cuadro de lista.
Sorted	True/False. Establece si los elementos se ordenan alfabéticamente.
Style	Establece el comportamiento del control.
Text	Texto que contiene el control.

Métodos

AddItem	Permite añadir nuevos elementos a la lista.
RemoveItem	Permite eliminar elementos de la lista.

Eventos

Click	Ocurre cuando el usuario interactúa con la lista del control.
Change	Ocurre cuando el valor de la propiedad Text es modificado.

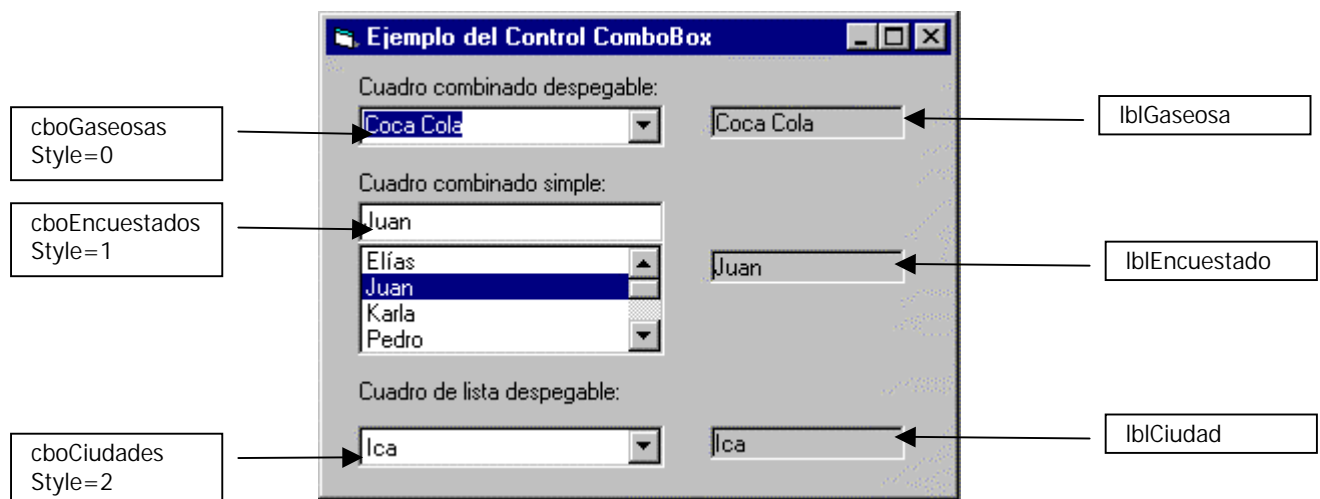
Propiedad Style

Esta propiedad establece el comportamiento del control **ComboBox**, y puede tomar los siguientes valores:

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 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. Incrementa la propiedad Height para mostrar más elementos de la lista.
vbComboDropDownList	2	Lista desplegable. Este estilo sólo permite la selección desde la lista desplegable.

Ejemplo

En la siguiente interfaz se ilustra el uso del control **ComboBox** y la propiedad **Style**.



Muestra la gaseosa seleccionada por el usuario en la etiqueta lblGaseosa

```
Private Sub cboGaseosas_Click()  
    lblGaseosa.Caption = cboGaseosas.Text  
End Sub
```

Actualiza la etiqueta lblGaseosa cuando el usuario modifica el control cboGaseosas

```
Private Sub cboGaseosas_Change()  
    lblGaseosa.Caption = cboGaseosas.Text  
End Sub
```

Muestra el encuestado seleccionado por el usuario en la etiqueta lblEncuestado

```
Private Sub cboEncuestados_Click()  
    lblEncuestado.Caption = cboEncuestados.Text  
End Sub
```

Muestra en la etiqueta lblCiudad el elemento seleccionado del control cboCiudades

```
Private Sub cboCiudades_Click()  
    lblCiudad.Caption = cboCiudades.Text  
End Sub
```

Procedimientos

Existen dos tipos de procedimientos con los que se trabaja en Visual Basic: los procedimientos de evento y los procedimientos generales.

Procedimientos de Evento

Visual Basic invoca automáticamente procedimientos de evento en respuesta a acciones del teclado, del ratón o del sistema. Por ejemplo, los botones de comando tienen un procedimiento de evento Click. El código que se escriba en el procedimiento de evento Click es ejecutado cuando el usuario haga clic en un botón de comando.

Cada control tiene un conjunto fijo de procedimientos de evento. Los procedimientos de evento para cada control son mostrados en un cuadro de lista despegable en la ventana de código.

Procedimientos Generales

Los procedimientos generales son procedimientos **Sub** o **Function** que son creados para que lleven a cabo tareas específicas, estos deben ser invocados de manera explícita.

Para crear un procedimiento general, se debe abrir la ventana de código y hacer clic en la orden **Agregar procedimiento** del menú **Herramientas**. También se puede crear un nuevo procedimiento escribiendo el encabezado de procedimiento **Sub**, seguido por el nombre del procedimiento, en una línea en blanco dentro de la ventana de código.

Si se tiene código duplicado en varios procedimientos de evento, se puede colocar el código en un procedimiento general y luego invocar al procedimiento general desde los procedimientos de evento.

Procedimientos Sub

Los procedimientos **Sub** no retornan valores. Por ejemplo:

```
Public Sub Seleccionar(Cuadro As TextBox)  
    Cuadro.SelStart = 0
```

```
Cuadro.SelLength = Len(Cuadro.Text)
End Sub
```

Los procedimientos **Sub** son invocados especificando sólo el nombre del procedimiento, o empleando la instrucción **Call** con el nombre del procedimiento. Por ejemplo:

```
Call Seleccionar(Text1)
```

Si se emplea la instrucción **Call**, se debe encerrar la lista de argumentos entre paréntesis. Si se omite **Call**, también se deben omitir los paréntesis alrededor de la lista de argumentos.

Procedimientos Function

Los procedimientos **Function** devuelven valores. En el siguiente ejemplo, el procedimiento **Function** recibe un número y devuelve ese número al cuadrado.

```
Public Function Cuadrado(N As Integer) As Integer
    Cuadrado = N * N
End Function
```

Si se desea guardar el valor devuelto, se debe usar paréntesis cuando se invoque a la función, como se muestra a continuación:

```
Resultado = Cuadrado (5)
```

Si se omiten los paréntesis, se puede ignorar el valor devuelto y no guardarlo en una variable. Esto puede ser útil si se quiere ejecutar una función y no se desea el valor devuelto. Por ejemplo:

```
Cuadrado 5
```

Alcance del Código

Además de poder declarar código en un módulo de formulario, también se puede declarar procedimientos en un módulo de código estándar.

Los módulos de código estándar sólo contienen código de Visual Basic, y son un buen lugar para almacenar código que no es específico para un solo formulario. Los procedimientos pueden ser declarados como **Private** (privados) o **Public** (públicos).

Los procedimientos declarados como **Private** pueden ser llamados o invocados sólo por otros procedimientos localizados en ese formulario, módulo o clase.

Los procedimientos declarados como **Public** en un formulario se convierten en métodos del formulario. El procedimiento puede ser llamado desde cualquier lugar de la aplicación especificando los nombres del formulario y del procedimiento.

Los procedimientos declarados como **Public** en un módulo están disponibles para toda la aplicación, y pueden ser llamados especificando el nombre del procedimiento.

El siguiente código declara un procedimiento **Public**.

```
Public Sub MiProc()  
  
End Sub
```

Si se declara el procedimiento en un módulo de formulario, puede ser llamado con el siguiente código.

```
Form1.MiProc
```

Si se declara el procedimiento en un módulo estándar, puede ser llamado con el siguiente código.

```
MiProc
```

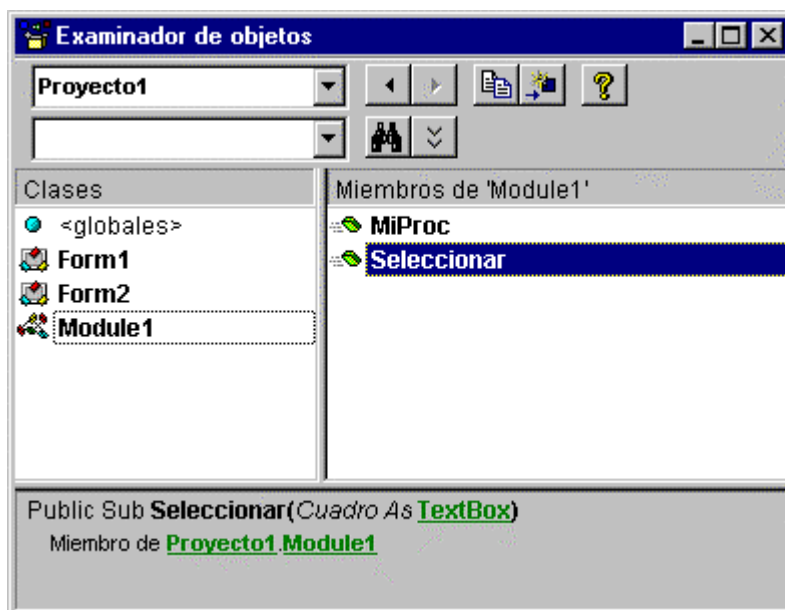
Si se declara un procedimiento con el mismo nombre en dos módulos estándar, se debe especificar el nombre del módulo como se muestra en el siguiente código.

```
Module1.MiProc
```

Uso del Examinador de Objetos para ver Procedimientos y Librerías de Objetos

Una vez que se ha añadido código a una aplicación se puede emplear el **Examinador de objetos** para listar a todos los procedimientos que han sido creados, pasar rápidamente a un procedimiento específico, o pegar una llamada a un procedimiento dentro de una ventana de código.

El **Examinador de objetos** visualiza las clases que están disponibles desde cualquier librería de objetos a la que se ha establecido una referencia. Para visualizar el **Examinador de objetos** presione la tecla de función **F2**.



Parte 6: Depuración, Validación de Datos y Manipulación de Errores

Cuando desarrolle aplicaciones en Visual Basic, es importante depurar el código escrito y manipular los probables errores que puedan ocurrir. También es importante prevenir en lo posible estos errores validando los ingresos de datos a la aplicación.

Herramientas de Depuración

Visual Basic proporciona herramientas interactivas para localizar errores en tiempo de ejecución y errores en la lógica del programa. Se puede acceder a todas las herramientas de depuración empleando el menú **Depuración** o la barra de herramientas **Depuración**. Las herramientas de depuración en Visual Basic incluyen:

Puntos de interrupción y expresiones de interrupción

Establece un punto de interrupción para detener un programa en ejecución. Se puede establecer un punto de interrupción en tiempo de diseño o en tiempo de ejecución mientras se esté en modo de interrupción.

Expresiones de inspección

Emplee las expresiones de inspección para examinar una variable o expresión en particular. El valor de cada expresión de inspección es actualizado en los puntos de interrupción.

Opciones paso a paso

Use las opciones paso a paso para ejecutar porciones de código ya sea una instrucción o procedimiento a la vez.

Pila de llamadas

Emplee Pila de llamadas para visualizar todas las llamadas a procedimientos activas y rastrear la ejecución de una serie de procedimientos anidados.

La ventana inmediato

En modo de interrupción, se puede probar una sentencia ejecutable escribiéndola en la ventana inmediato. Visual Basic ejecuta la sentencia inmediatamente de modo que se pueda evaluar el código.

La ventana locales

Esta ventana automáticamente visualiza todas las variables declaradas en el procedimiento actual, junto con sus valores.

Validación de Datos

Se puede prevenir algunos errores en el ingreso de datos y mejorar el uso de una aplicación validando información mientras es ingresada a los campos de la aplicación.

Restricción de Opciones con Controles

Una manera de asegurar ingresos válidos es restringiendo el número de opciones que un usuario puede escoger. Por ejemplo, se puede emplear un cuadro de lista para permitir a los usuarios seleccionar un nombre de un producto en un formulario. Debido a que los usuarios deben escoger un producto de una lista predefinida, no podrán ingresar un nombre de producto no válido.

También se pueden usar botones de opción para un número pequeño de opciones, o casillas de verificación para opciones de tipo booleanas.

Uso de la propiedad MaxLength

La propiedad **MaxLength** determina la longitud máxima de una cadena en un cuadro de texto. El sistema emitirá un sonido (beep) cuando el usuario trate de escribir una cadena que exceda la longitud máxima. Si se desea visualizar un mensaje de error, se necesita interceptar la tecla oprimida en el evento `KeyPress`.

Uso de la propiedad Locked

La propiedad **Locked** determina si los usuarios pueden modificar los datos en un cuadro de texto. Si la propiedad **Locked** es establecida a **True**, los usuarios sólo pueden ver y copiar los datos de un cuadro de texto.

Uso del Evento KeyPress para Validar Datos

Se pueden emplear los eventos `KeyPress`, `KeyDown` y `KeyUp` para validar datos mientras el usuario escribe. Se puede prevenir que el usuario ingrese ciertos caracteres (por ejemplo, puede limitar el ingreso de datos a valores numéricos). Se puede también modificar los datos mientras son ingresados (por ejemplo, puede convertir todos los caracteres a mayúsculas).

El evento `KeyPress` tiene lugar cuando el usuario ingresa un carácter ASCII estándar. Esto no incluye la mayoría de la teclas especiales, tales como las teclas de función, las teclas direccionales, o la tecla `DELETE`. Para responder a estas teclas se debe emplear los eventos `KeyDown` y `KeyUp`. El siguiente código cambia los caracteres a mayúsculas mientras el usuario escribe.

```
Private Sub Text1_KeyPress(KeyAscii as Integer)
    KeyAscii = Asc(Ucase(Chr(KeyAscii)))
End Sub
```

El siguiente código previene que los usuarios ingresen sólo valores numéricos en un cuadro de texto.

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii <> 8 Then ' Permite el uso de la tecla BACKSPACE
        If KeyAscii < 48 Or KeyAscii > 57 Then
            KeyAscii = 0 ' Establece el caracter a nulo si está _
                          fuera del rango
        End If
    End If
End Sub
```

Para visualizar una lista de los valores ASCII, busque **ASCII** en la ayuda de Visual Basic.

Validación de Información a Nivel de Formulario

Además de emplear técnicas a nivel de campo para validar los datos mientras son ingresados, se puede escribir código que valide los datos en todos los campos de un formulario al mismo tiempo. En esta parte del curso se verán las técnicas de validación a nivel de formulario soportadas por Visual Basic.

Habilitando el Botón Aceptar

Una forma de validar la información del formulario sería la de asegurar que un usuario ha ingresado los datos en todos los campos en un formulario antes de que se le permita continuar. Esto puede llevarse a cabo deshabilitando el botón Aceptar en un formulario hasta que el usuario haya llenado todos los campos, como se muestra a continuación.



Para comprobar cada tecla oprimida en un formulario, se debe establecer la propiedad **KeyPreview** de un formulario a **True**. El formulario recibe primero el evento del teclado y luego lo recibe el control.

El siguiente código habilita el botón Aceptar luego de ingresar datos en todos los campos.

```
Private Sub Form_Load()  
    Me.KeyPreview = True  
    cmdAceptar.Enabled = False  
End Sub  
  
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)  
    ' Se habilita el botón Aceptar sólo si todos los cuadros de texto _  
    tienen datos  
    Dim ControlAct As Control  
    For Each ControlAct In Controls  
        If TypeOf ControlAct Is TextBox Then  
            If ControlAct.Text = "" Then ' El cuadro de texto está vacío  
                cmdAceptar.Enabled = False  
                Exit Sub  
            End If  
            ' Se podrían verificar otro tipo de controles si _  
            fuera necesario  
        End If  
    Next  
    cmdAceptar.Enabled = True 'Todos los cuadros de texto tienen datos  
End Sub
```


Validación de Todos los Campos de un Formulario

Una manera sencilla de validar todos los campos de un formulario al mismo tiempo es poniendo el código de validación en el evento Click del botón **Aceptar**. En esta caso, la aplicación le permite al usuario completar el ingreso de datos en todos los campos del formulario y luego validarlos. La aplicación establece el enfoque en el primer campo que contiene los datos incorrectos.

El siguiente código valida todos los campos numéricos del formulario mostrado anteriormente.

```
Private Sub cmdAceptar_Click()  
    If Not (IsNumeric(txtEdad.Text)) Then  
        MsgBox "Ingrese la edad correctamente"  
        txtEdad.SetFocus  
    ElseIf Not (IsNumeric(txtSueldo.Text)) Then  
        MsgBox "Ingrese el sueldo correctamente"  
        txtSueldo.SetFocus  
    Else  
        MsgBox "Datos correctos"  
        Unload Me  
    End If  
End Sub
```

Uso del Evento QueryUnload

El evento QueryUnload tiene lugar justo antes del evento Unload cuando el formulario es descargado. El evento QueryUnload permite determinar como fue iniciado el evento Unload y cancelar este evento. Es útil cuando un usuario no ha completado el ingreso de datos a un formulario, o cuando se quisiera preguntar al usuario si desea grabar cambios realizados antes cerrar el formulario.

El evento QueryUnload tiene los siguientes argumentos:

- El argumento **Cancel** cancela el evento Unload. Si **Cancel** se establece a **True** la aplicación permanece como estaba antes de que se intentara la descarga.
- El argumento **UnloadMode** indica como fue iniciado el evento Unload.

Valores que se pueden obtener

El argumento **UnloadMode** (*modo_descarga*) devuelve los siguientes valores:

Constante	Valor	Descripción
VbFormControlMenu	0	El usuario eligió el comando Cerrar del menú Control del formulario.
VbFormCode	1	Se invocó la instrucción Unload desde el código.
VbAppWindows	2	La sesión actual del entorno operativo Microsoft Windows está finalizando.
VbAppTaskManager	3	El Administrador de tareas de Microsoft Windows está cerrando la aplicación.
VbFormMDIForm	4	Un formulario MDI secundario se está cerrando porque el formulario MDI también se está cerrando.

El siguiente código cancela el evento Unload y pregunta al usuario antes de cerrar el formulario.

```
Private Sub Form_QueryUnload(Cancel as Integer, UnloadMode as Integer)  
    Dim iRespuesta Integer
```

```

iRespuesta = MsgBox ("¿Realmente desea salir?", vbYesNo)
If iRespuesta = vbNo Then
    Cancel=True
End If
End Sub

```

Notas:

- Cuando se cierra un objeto **MDIForm**, el evento *QueryUnload* ocurre primero para el formulario MDI principal y después en todos los formularios secundarios MDI. Si ningún formulario cancela el evento *QueryUnload*, ocurre el evento *Unload* primero en todos los demás formularios y después en un formulario MDI principal. Cuando un formulario secundario o un objeto **Form** se cierra, el evento *QueryUnload* de ese formulario ocurre antes que el evento *Unload* del formulario.
- Cuando una aplicación se cierra, puede utilizar los procedimientos de evento *QueryUnload* o *Unload* para establecer la propiedad **Cancel** a **True**, deteniendo el proceso de cierre. Sin embargo, el evento *QueryUnload* ocurre en todos los formularios antes de que se descargue ninguno de ellos y el evento *Unload* ocurre conforme se descarga cada formulario.

Manipulación de Errores en Tiempo de Ejecución

No importa lo bien que se diseñe una aplicación, los errores en tiempo de ejecución siempre ocurrirán. Los usuarios olvidan poner los discos en las unidades, sistemas se ejecutan con poca memoria, y archivos no se encuentran donde se supone deberían estar. Añadiendo código de manipulación de errores efectivo a una aplicación, se crean aplicaciones más robustas.

Entendiendo el Proceso de Manipulación de Errores

El proceso de manipulación de errores involucra los siguientes pasos.

1. Habilitar la interceptación de errores que especifica hacia donde se bifurcará la ejecución cuando ocurra un error.
2. Escribir el código de manipulación de errores.
3. Salir del código de manipulación de errores.

La instrucción **On Error GoTo** habilita la interceptación de errores y especifica hacia donde saltará la ejecución cuando ocurra un error. Si ocurriera un error en tiempo de ejecución, la ejecución saltará hacia la etiqueta especificada por la instrucción **On Error GoTo**. El manipulador de errores ejecuta el código de manipulación de errores seguido por una instrucción **Resume** que indica donde deberá continuar el proceso.

El siguiente código muestra como emplear las instrucciones **On Error GoTo** y **Resume**.

```

Private Sub Command1_Click()
    Dim NombreAplic As String
    On Error GoTo ManipulaErr
    NombreAplic = InputBox("Ingrese el nombre de la aplicación:")
    Shell NombreAplic, vbNormalFocus
    Exit Sub

ManipulaErr:

```

```
If Err.Number = 53 Then
    MsgBox "No se encontro la aplicación"
    If MsgBox("¿Desea intentarlo nuevamente?", vbYesNo) _
        = vbYes Then
        NombreAplic = InputBox _
            ("Ingrese el nombre de la aplicación:")
        Resume 'Intentar nuevamente
    Else
        Resume Next 'Ejecutar la siguiente instrucción
    End If
Else
    MsgBox "Error desconocido"
End If
End Sub
```

En el código o rutina de manipulación de errores, se emplean las propiedades y métodos del objeto **Err** para verificar que error ocurrió, borrar un valor de error o desencadenar un error.

Las Propiedades del Objeto Err

La propiedad **Number** es un entero que indica el último error que tuvo lugar. Para determinar que error ha ocurrido, se verifica el valor de **Err.Number**. En algunos casos, se puede corregir un error y permitir continuar el proceso sin interrumpir el usuario. En otros, se deberá notificar al usuario de un error, y tomar alguna acción basada en la respuesta del usuario.

La propiedad **Description** es una cadena que contiene una descripción del error.

La propiedad **Source** contiene el nombre del objeto aplicación que generó el error. Es útil cuando se emplea Automatización. Por ejemplo, si se trabaja con Microsoft Excel y genera un error, Microsoft Excel establecerá **Err.Number** al código de error apropiado y establecerá **Err.Source** a **Excel.Application**.

Los Métodos del Objeto Err

El método **Clear** establece el valor de **Err.Number** a cero. Básicamente, el método **Clear** se emplea borrar explícitamente el objeto **Err** después de controlar un error.

El método **Raise** genera un error en tiempo de ejecución. Por ejemplo, se podría emplear este método para probar el código de manipulación de errores.

```
Err.Raise 53 'Archivo no encontrado
```

Opciones de las Instrucción Resume

La instrucción **Resume** se emplea para especificar donde continuará el proceso de una aplicación luego de manipular un error. La siguiente tabla lista los tres tipos de instrucciones **Resume** disponibles en Visual Basic.

Instrucción	Descripción
Resume	Regresa a la instrucción que generó el error. Emplee Resume para repetir una operación luego de corregir el error.

Instrucción	Descripción
Resume Next	Regresa a la instrucción inmediatamente siguiente a la que generó el error.
Resume línea	La ejecución continúa en la línea especificada en el argumento obligatorio línea.

Nota:

- Si utiliza una instrucción **Resume** en otro sitio que no sea una rutina de manipulación de errores, ocurre un error.

Deshabilitando la Manipulación de Errores

Puede ser difícil depurar código que tiene habilitada la manipulación de errores. Visual Basic puede ejecutar el código de manipulación de errores cuando se quiera ingresar al modo de Interrupción y depurar la aplicación.

Visual Basic proporciona opciones para deshabilitar la manipulación de errores en el momento de la depuración.

- Para cambiar como serán manipulados los errores:
 1. En el menú **Herramientas**, haga clic en **Opciones**.
 2. En la ficha **General**, debajo de **Interceptación de errores**, haga clic en la opción deseada y luego en **Aceptar**.

La siguiente tabla describe las opciones de manipulación de errores disponibles debajo de **Interceptación de errores** en la ficha **General**.

Opción	Descripción
Modo de interrupción en todos	Si se selecciona esta opción, Visual Basic ignora cualquiera de las instrucciones On Error GoTo e ingresa en modo de Interrupción si tiene lugar algún error en tiempo de ejecución.
Interrupción en módulos de clase	Si se selecciona esta opción, Visual Basic ejecutará las instrucciones On Error GoTo para manipular errores en tiempo de ejecución.
Interrupción en errores no controlados	Visual Basic ingresa a modo de Interrupción en cualquier error para el cual no se haya establecido una rutina de manipulación.

Compilación de una Aplicación

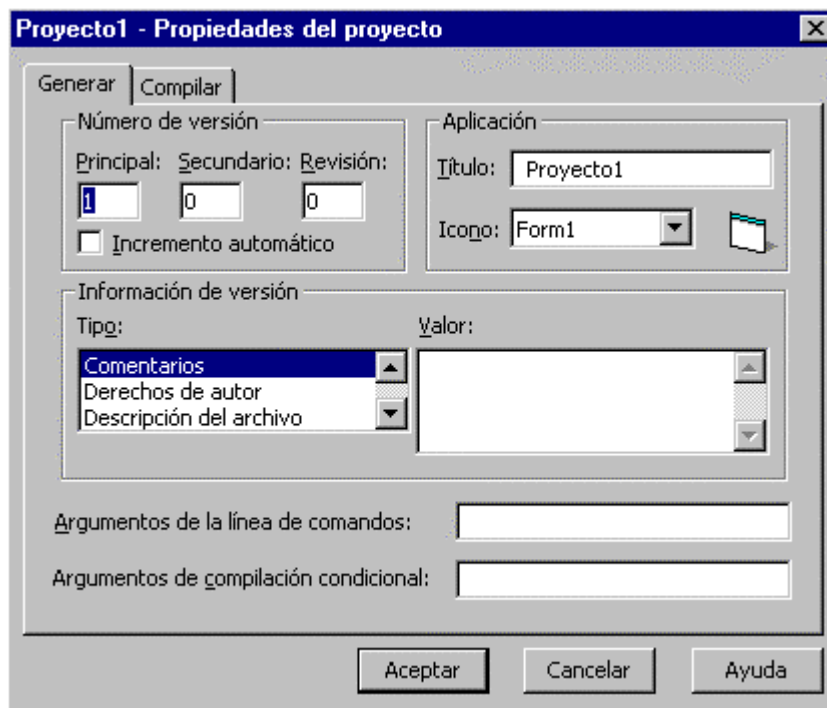
Una vez finalizada la creación de una aplicación, se puede crear el archivo ejecutable para los usuarios. Esta parte del curso describe como compilar un proyecto.

Creación de un Archivo Ejecutable

Crear un archivo ejecutable en Visual Basic es un proceso sencillo.

1. En el menú **Archivo**, haga clic en **Generar <NombreProyecto>.exe**
2. Ingrese el nombre para el archivo ejecutable.
3. Para añadir información específica de versión:

En el cuadro de diálogo **Generar proyecto**, haga clic en el botón **Opciones**. En la ficha **Generar** escriba los números de versión y el texto de información de la versión , y luego haga clic en **Aceptar**.



Además del archivo ejecutable, se debe de proporcionar diversas DLL's y otros archivos a los usuarios. Se debería crear un programa de instalación que instale la aplicación en la computadora del usuario.

El Asistente para instalar aplicaciones de Visual Basic hace sencilla la tarea de crear discos o una carpeta de distribución para una aplicación. Los usuarios pueden luego ejecutar el programa de instalación en sus computadoras para instalar y registrar los archivos apropiados.

Parte 7: Controles Estándar Avanzados y Acceso a Datos

Control Cuadro de Lista de Unidades (DriveListBox)



Un control DriveListBox permite al usuario seleccionar una unidad de disco válida en tiempo de ejecución. Utilice este control para mostrar una lista de todas las unidades válidas del sistema de un usuario. Puede crear cuadros de diálogo que permitan al usuario abrir un archivo de una lista de un disco en cualquier unidad disponible.

Propiedades

Drive	Devuelve o establece la unidad seleccionada en tiempo de ejecución. No está disponible en tiempo de diseño.
List	Contiene la lista de conexiones de unidad efectivas.
ListCount	Devuelve el número de conexiones con unidades de disco.
ListIndex	Devuelve o establece el índice del elemento seleccionado actualmente en el control. No está disponible en tiempo de diseño.

Eventos

Change	Ocurre cuando el usuario selecciona una nueva unidad o cuando se cambia la configuración de la propiedad Drive mediante código.
--------	---------------------------------------------------------------------------------------------------------------------------------

Control Cuadro de Lista de Directorios (DirListBox)



Un control DirListBox muestra directorios y rutas de acceso en tiempo de ejecución. Utilice este control para mostrar una lista jerárquica de directorios. Puede crear cuadros de diálogo que, por ejemplo, permitan a un usuario abrir un archivo desde una lista de archivos de todos los directorios disponibles.

Propiedades

List	Contiene una lista de todos los directorios.
ListCount	Devuelve el número de subdirectorios del directorio actual.
ListIndex	Indica el índice de la ruta de acceso actual.
Path	El valor de la propiedad Path es una cadena que indica una ruta de acceso, como C:\Ob o C:\Windows\System.

Eventos

Change	Ocurre cuando el usuario hace doble clic en un nuevo directorio o cuando se cambia la configuración de la propiedad Path mediante código.
--------	-------------------------------------------------------------------------------------------------------------------------------------------

Control Cuadro de Lista de Archivos (FileListBox)



El control FileListBox encuentra y muestra los archivos del directorio especificado por la propiedad Path en tiempo de ejecución. Utilice este control para mostrar una lista de los archivos seleccionados por

tipo. Puede crear cuadros de diálogo en la aplicación que, por ejemplo, permitan al usuario seleccionar un archivo o un grupo de archivos.

Propiedades

FileName	Devuelve o establece el nombre de un archivo seleccionado.
List	Contiene una lista con los archivos del directorio expandido actualmente que coinciden con la propiedad Pattern. No se incluye la ruta de acceso.
ListCount	Devuelve el número de archivos del directorio actual que coinciden con el valor de la propiedad Pattern.
ListIndex	Devuelve o establece el índice del elemento seleccionado actualmente.
Path	El valor de la propiedad Path es una cadena que indica una ruta de acceso, como C:\Ob o C:\Windows\System.
Pattern	Devuelve o establece un valor que indica los nombres de archivo mostrados en un control FileListBox en tiempo de ejecución.

Eventos

Click	Ocurre cuando el usuario presiona y suelta un botón del ratón en un control FileListBox.
-------	------------------------------------------------------------------------------------------



Control Imagen (Image)

El control Image se utiliza para mostrar un gráfico. Un control Image puede mostrar un gráfico desde un mapa de bits, un icono o un metarchivo, así como un metarchivo mejorado, un archivo JPEG o archivos GIF.

Propiedades

Picture	Devuelve o establece un gráfico que se mostrará en el control. También se le puede asignar un gráfico devuelto por la función LoadPicture.
Stretch	True/False. Devuelve o establece un valor que indica si un gráfico cambia su tamaño para ajustarse al de un control Image.

Comentarios

*El control **Image** utiliza menos recursos del sistema y actualiza con más rapidez que un control **PictureBox**, pero sólo admite un subconjunto de las propiedades, los eventos y los métodos de **PictureBox**. Use la propiedad **Stretch** para determinar si el gráfico se escala para ajustarse al control o viceversa. Aunque puede colocar un control **Image** dentro de un contenedor, un control **Image** no puede actuar como contenedor.*

Función LoadPicture()

Carga un gráfico en un objeto Picture, un control PictureBox o un control Image.

Formato:

LoadPicture(*NombreDeArchivoGráfico*)

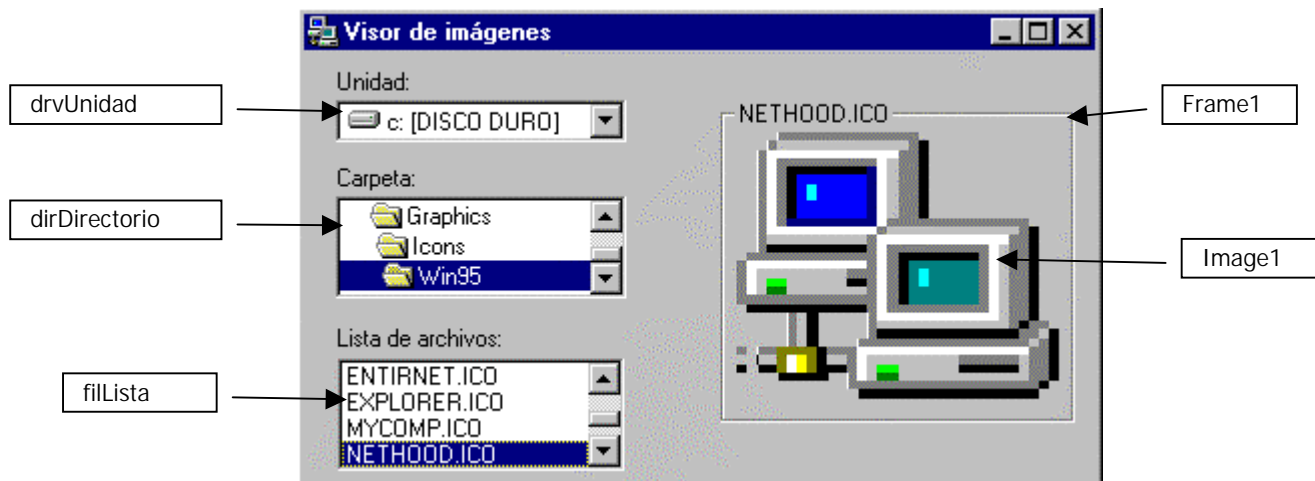
También se puede usar la función **LoadPicture()** para asignar un icono a un formulario o al puntero del ratón mostrado en pantalla.

Ejemplos de la función LoadPicture():

```
'Establece el icono del formulario
Set Form1.Icon = LoadPicture("MIICONO.ICO")

'Establece el puntero del ratón
Screen.MouseIcon = LoadPicture("MIICONO.ICO")
Screen.MousePointer = 99
```

Creación de un visor de imágenes



El código para los controles es el siguiente:

```
Private Sub drvUnidad_Change()
    dirDirectorio.Path = drvUnidad.Drive
End Sub

Private Sub dirDirectorio_Change()
    filLista.Path = dirDirectorio.Path
End Sub

Private Sub filLista_Click()
    Image1.Picture = LoadPicture(dirDirectorio.Path & _
        "\" & filLista.filename)
    Frame1.Caption = filLista.filename
    Form1.Icon = LoadPicture(dirDirectorio.Path & _
        "\" & filLista.filename)
End Sub
```


Control Temporizador (Timer)



Un control **Timer** puede ejecutar código a intervalos periódicos produciendo un evento **Timer**. El control **Timer**, invisible para el usuario, resulta útil para el procesamiento de fondo. No existe ningún límite práctico en cuanto al número de controles **Timer** activos que se puede tener en Visual Basic 5.0 ejecutándose en Windows 95 o en Windows NT.

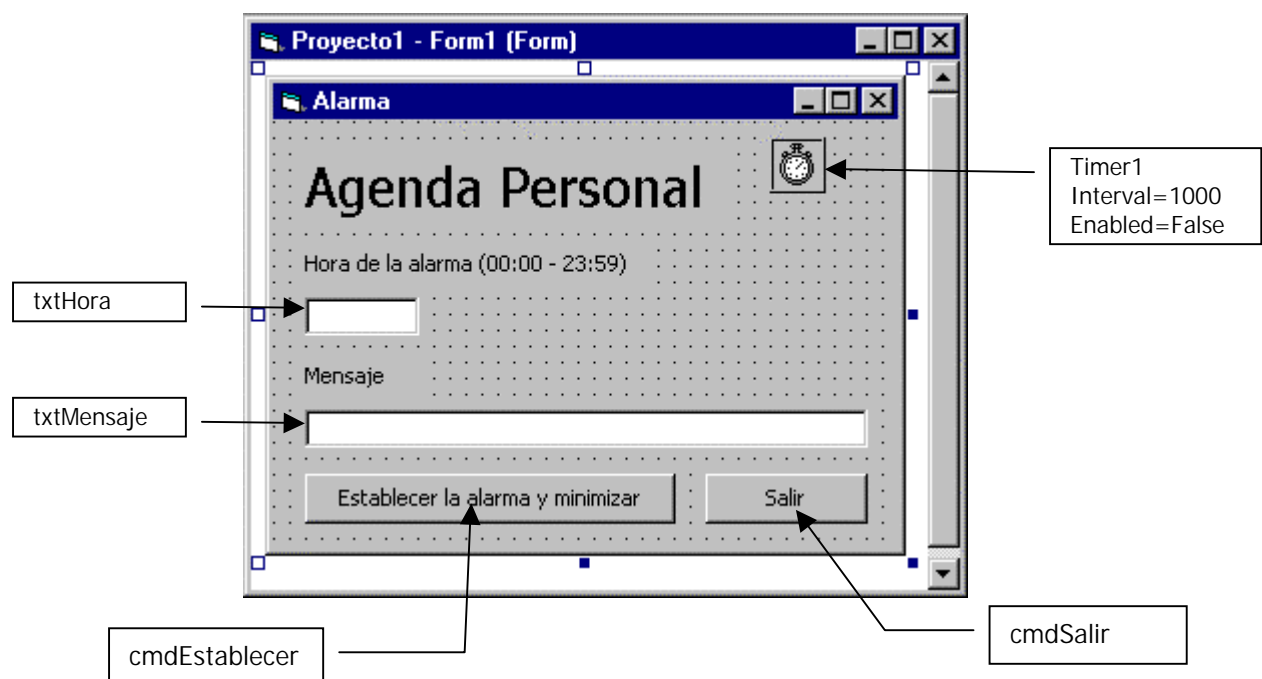
Propiedades

- Enabled** True/False. Activa o desactiva el control. Si se desactiva el control **Timer**, estableciendo **Enabled** a **False**, se cancelará la cuenta atrás establecida por la propiedad **Interval** del control.
- Interval** Devuelve o establece el número de milisegundos entre las llamadas al evento **Timer** de un control **Timer**. Los valores admitidos para milisegundos son 0, que desactiva el control **Timer**, o de 1 a 65535. El máximo, 65535 milisegundos, equivale a poco más de un minuto.

Eventos

- Timer** Ocurre cuando ha transcurrido un intervalo preestablecido para el control **Timer**.

Ejemplo de uso del control Timer



Minimiza el formulario y activa el control temporizador Timer1

```
Private Sub cmdEstablecer_Click()  
    Form1.WindowState = 1  
    Timer1.Enabled = True  
End Sub
```

Termina la ejecución de la aplicación

```
Private Sub cmdSalir_Click()  
    End  
End Sub
```

Compara la hora actual con la hora ingresada en el cuadro txtHora cada segundo y muestra el mensaje establecido en caso de que sean iguales, desactiva el control temporizador y restaura la ventana a su tamaño normal.

```
Private Sub Timer1_Timer()  
    Dim hora As String  
    hora = Format(Time, "hh:mm")  
    If hora = txtHora.Text Then  
        Beep  
        MsgBox txtMensaje.Text, vbOKOnly + vbInformation, "Mensaje"  
        Timer1.Enabled = False  
        Form1.WindowState = 0  
    End If  
End Sub
```

Uso del control Data



En Visual Basic puede utilizar el control **Data** para crear aplicaciones de bases de datos para una gran variedad de formatos de base de datos. El control **Data** interactúa con el motor de base de datos Microsoft Jet y permite crear aplicaciones preparadas para datos con la mínima cantidad de código posible.

Vista General del Acceso a Datos

Antes de trabajar con la funcionalidad de base de datos desde Visual Basic, se debe entender las capacidades de acceso a datos, así como, la terminología empleada.

Opciones de Acceso a Datos en Visual Basic

Esta parte del curso enumera las diversas opciones que Visual Basic proporciona para acceder a datos.

Uso del Motor de Base de Datos Microsoft Jet

Los objetos de acceso a datos – Data Access Objects (**DAO**) y el control **Data** usan el motor de base de datos Microsoft Jet para acceder a bases de datos. El motor de base de datos Jet puede acceder a los siguientes 3 tipos de bases de datos.

- **Bases de datos Jet**
Estas bases de datos son creadas y manipuladas directamente por el motor Jet. Microsoft Access y Visual Basic emplean el mismo motor de base de datos Jet.
- **Bases de datos de Método de Acceso Secuencial Indexado (ISAM)**
Los formatos de estas bases de datos incluyen Btrieve, dBase, Microsoft Visual FoxPro, y Paradox.
- **Bases de datos compatibles con ODBC (Open DataBase Connectivity - Conectividad Abierta de Base de Datos)**
Estas bases de datos incluyen las bases de datos cliente/servidor que conforman el estándar ODBC, tal como Microsoft SQL Server. La mayoría de bases de datos que soportan ODBC pueden ser accedidas empleando Visual Basic.

Otros métodos de Acceso a Datos

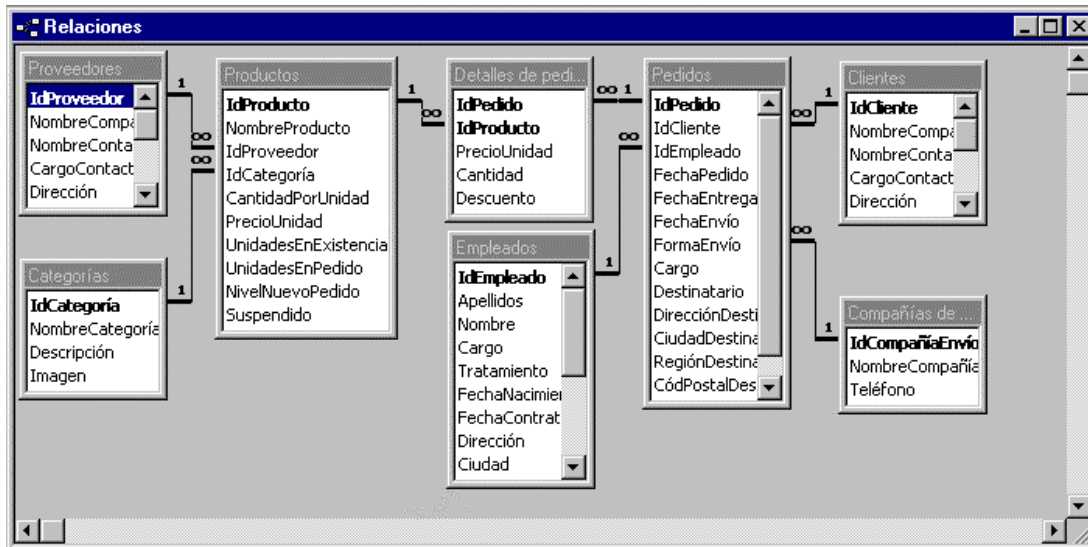
Los otros métodos de acceso a datos soportados por Visual Basic incluyen:

- **El control Origen de Datos Remoto (Remote Data Source)**
Este es un control que emplea ODBC para acceder a bases de datos ODBC tales como Microsoft SQL Server y Oracle. El control **Origen de Datos Remoto** sólo está disponible en la edición Empresarial de Visual Basic.
- **Librerías ODBC**
Estas librerías permiten que se llame a la interface de programación de aplicaciones (API) de ODBC directamente y están disponibles como un producto separado.
- **Librerías SQL de Visual Basic**
Estas librerías proporcionan un enlace directo a Microsoft SQL Server, y están disponibles como un producto separado.

Entendiendo los Conceptos Básicos acerca de las Bases de Datos

La mayoría de sistemas de bases de datos emplean el modelo de base de datos relacional. Este modelo presenta los datos como una colección de tablas. Una tabla es un grupo lógico de información relacionada. Por ejemplo, la base de datos Neptuno contiene una tabla que almacena a los empleados, y otra almacena los pedidos de una compañía ficticia.

La base de datos Neptuno.mdb es una base de datos de ejemplo incluida con Microsoft Access.



Elementos de una Tabla

La base de datos Neptuno contiene un número de tablas que agrupan la información. Estas tablas incluyen Pedidos, Clientes, y Empleados.

En una base de datos Jet, las filas de la tabla son denominadas registros, y las columnas campos.

Empleados : Tabla

Nombre del campo	Tipo de datos	Descripción
IdEmpleado	Autonumérico	Número asignado a un empleado nuevo
Apellidos	Texto	
Nombre	Texto	
Cargo	Texto	Cargo del empleado.
Tratamiento	Texto	Tratamiento usado en los saludos.
FechaNacimiento	Fecha/Hora	

Propiedades del campo

General	Búsqueda
Tamaño del campo	Entero largo
Nuevos valores	Incrementalmente
Formato	
Título	Id. de empleado
Indexado	Sí (Sin duplicados)

Un nombre de campo puede tener hasta 64 caracteres de longitud, incluyendo espacios. Presione F1 para obtener ayuda acerca de los nombres de campo.

La Clave Principal

Cada tabla debe de tener una clave principal, que es un campo (o una combinación de campos) que es único para cada registro en la tabla. Por ejemplo, el campo `IdEmpleado` es la clave principal para la tabla `Empleados`.

Una tabla puede también tener claves foráneas, que son campos que hacen referencia a una clave principal de otra tabla. Por ejemplo, en la base de datos `Neptuno`, la tabla `Pedidos` tiene un campo llamado `IdCliente`. Este campo es una clave foránea porque hace referencia a un cliente de la tabla `Cientes`. En vez de duplicar toda la información del cliente por cada pedido, sólo se ingresa la clave principal del cliente a quien pertenece el pedido, como la clave principal es única por cada cliente, hay un solo cliente por cada pedido, y un cliente puede tener muchos pedidos. En términos de una base de datos, la relación entre la tabla `Cientes` y la tabla `Pedidos` es una relación del tipo uno-a-varios.

Registros

Un registro contiene información acerca de un solo ingreso en una tabla. Generalmente, no se desea que dos registros en una tabla tengan los mismos datos. Por ejemplo, un registro en la tabla `Empleados` contiene información acerca de un único empleado.

Campos

Cada campo en una tabla contiene una parte de la información. Por ejemplo, la tabla `Empleados` incluye campos para el `Id` del empleado, `Apellidos`, `Nombre`, etc.

Indices

Los índices de una tabla de una base de datos son listas ordenadas que son más rápidas para las búsquedas que las tablas en sí. Para habilitar un acceso más rápido a una base de datos, la mayoría de bases de datos emplean uno o más índices. Por ejemplo, la tabla `Empleados` tiene un índice para la columna `IdEmpleado`.

Trabajando con el Control Data

El control **Data** de Visual Basic permite escribir aplicaciones de bases de datos muy eficaces con muy poco código.

En esta parte se aprenderá a generar aplicaciones de bases de datos con el control **Data** y el objeto **Recordset** asociado. También se aprenderá como el Asistente para Formularios de Datos puede construir una aplicación que incluye el control **Data**.

Tener Acceso a Datos con el Control Data

El control **Data** implementa el acceso a datos mediante el motor de base de datos Microsoft Jet. Esta tecnología proporciona acceso a muchos formatos de base de datos y permite crear aplicaciones que manejan datos sin necesidad de escribir código.

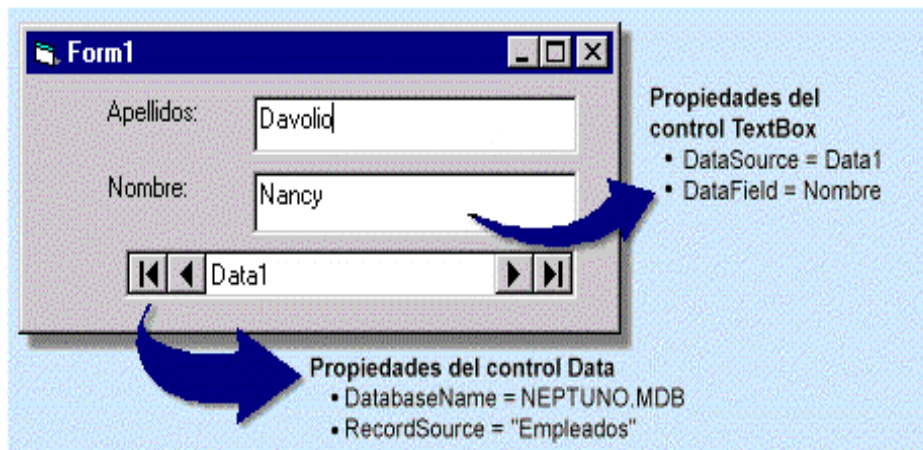
Para crear una aplicación de base de datos que emplee el control **Data**, se siguen los siguientes pasos:

1. Añadir el control **Data** al formulario y establecer las propiedades para especificar la base de datos y la tabla desde la cual se obtendrán los datos.
2. Añadir controles enlazados a datos al formulario, y establecer las propiedades para enlazar los controles al control **Data** para que los datos puedan ser mostrados.

Uso de Controles Enlazados a Datos

Cuando un control que ha colocado en un formulario se enlaza a datos, se muestran automáticamente los datos de la base de datos en el control enlazado. Si un usuario cambia los datos de un control enlazado, dichos cambios se actualizarán automáticamente en la base de datos en cuanto el usuario se desplace a otro registro. Muchos controles intrínsecos o estándar de Visual Basic permiten ser enlazados a datos, como es el caso de los controles **CheckBox**, **Image**, **Label**, **PictureBox**, **TextBox**, **ListBox**, **ComboBox** y los **contenedores OLE**.

La siguiente ilustración es un ejemplo de un formulario que contiene un control **Data** y dos controles enlazados.



Establecer las Propiedades del Control Data

Los siguientes pasos describen como conectar un control **Data** a una base de datos.

1. Especificar la base de datos a la cual se quiere acceder estableciendo la propiedad **DatabaseName** al nombre de la base de datos.
2. Para especificar que registros recuperar, establecer la propiedad **RecordSource** al nombre de la tabla dentro de la base de datos, o a una cadena SQL.

Nota: Para acceder a una base de datos dBase, Paradox, o Btrieve, se debe establecer la propiedad **DatabaseName** a la carpeta que contiene los archivos de la base de datos, y la propiedad **Connect** al tipo apropiado de base de datos.

Enlazar Controles

Después de establecer los valores de las propiedades para el control **Data**, es necesario enlazar al control **Data** controles individuales y después especificar qué campo de la tabla mostrará cada control.

1. En tiempo de diseño, establecer la propiedad **DataSource** del control enlazado a datos al control **Data**.
2. En tiempo de diseño o en tiempo de ejecución, especificar que campo se desea mostrar en el control enlazado estableciendo la propiedad **DataField**.

La propiedad **DataField** puede ser establecida en tiempo de diseño o en tiempo de ejecución.

Usar las Propiedades y Métodos del Control Data

Para especificar los datos que se desean recuperar, se debe establecer las propiedades **DatabaseName** y **RecordSource** de un control **Data**. Además, se pueden establecer las siguientes propiedades y métodos.

La Propiedad Connect

Esta propiedad especifica el tipo de base de datos a abrir. Puede incluir argumentos tales como un nombre de usuario (user ID) y una contraseña.

La Propiedad Exclusive

La propiedad **Exclusive** determina si se tiene o no un uso exclusivo de la base de datos. Si esta propiedad se establece a **True**, y luego se abre sin problemas la base de datos, ninguna otra aplicación podrá abrirla hasta que sea cerrada.

La Propiedad ReadOnly

Esta propiedad determina si se puede o no actualizar o grabar cambios en la base de datos. Si no se tiene planeado hacer cambios en la base de datos, es más eficiente establecer la propiedad **ReadOnly** a **True**.

La Propiedad Recordset

La propiedad **Recordset** es un objeto que contiene el conjunto de registros devueltos por el control **Data**. Esta propiedad contiene a su vez propiedades y métodos que pueden ser usados para trabajar con los registros devueltos.

Las Propiedades BOFAction y EOFAction

Estas propiedades determinan que acción tomará el control **Data** cuando las propiedades **BOF** o **EOF** del recordset tomen como valor **True**.

Por ejemplo, si la propiedad **EOFAction** del control **Data** es establecida a `vbActionAddNew`, y se emplea el control **Data** para desplazarse, una vez que se pase el último registro, el control **Data** automáticamente ejecutará el método **AddNew** de modo que se pueda añadir un nuevo registro.

El Método Refresh

El método **Refresh** renueva el objeto **Recordset**. Si se cambia la propiedad **RecordSource** en tiempo de ejecución, se debe invocar al método **Refresh** para renovar el recordset.

El siguiente código muestra como emplear el método **Refresh**.

```
Data1.RecordSource = "SELECT * FROM Empleados " & _  
    "WHERE [IdEmpleado] = " & txtIdEmp.Text  
Data1.Refresh
```

El Objeto Recordset

En una aplicación de base de datos, los usuarios trabajan con el control **Data** para desplazarse entre registros dentro de la base de datos. Los usuarios pueden hacer clic en los botones del control **Data** para avanzar o retroceder registro a registro o para ir directamente al primer o al último registro.

¿Qué es un Recordset?

Todo el conjunto de registros al que hace referencia un control **Data** se denomina conjunto de registros o **Recordset**. El **Recordset** se almacena en la memoria, transfiriéndose al disco si es necesario.

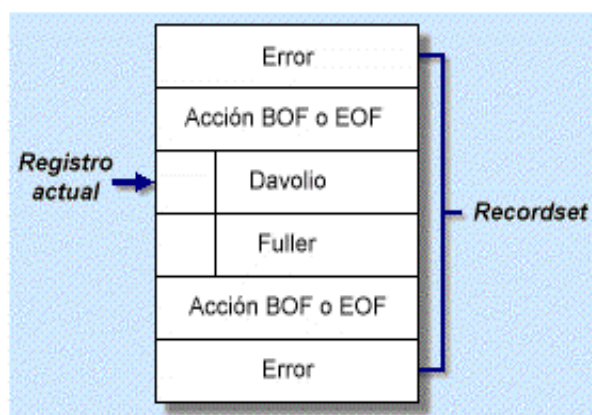
Para manipular el **Recordset**, se emplea la propiedad **Recordset** del control **Data**. El **Recordset** contiene el registro actual. La información del registro actual es mostrada por los controles enlazados. Se puede cambiar la posición del registro actual haciendo clic en el control **Data** o escribiendo código que emplee métodos del objeto **Recordset**.

Determinar los límites de un Recordset

Si utiliza código para cambiar la posición del registro actual, debe comprobar las propiedades **EOF** y **BOF** del objeto **Recordset** para determinar el inicio y el final del mismo. Cuando se desplace al registro **EOF** o al **BOF**, se ejecutará la acción indicada por el valor de la propiedad **BOFAction** o **EOFAction**.

Por ejemplo, se puede establecer la propiedad **EOFAction** para añadir un nuevo registro automáticamente. Si se establece la propiedad **EOFAction** a **EOF**, ninguna acción será tomada cuando se desplace al registro **EOF**. Cuando se desplace pasando un registro los registros **BOF** o **EOF**, se producirá un error en tiempo de ejecución.

En la siguiente ilustración se muestra cómo las propiedades **BOF** y **EOF** determinan los límites del objeto **Recordset**.



Para emplear el objeto **Recordset** de un control **Data** determinado, se debe especificar la propiedad **Recordset** del control **Data**, como se muestra en el siguiente código.

```
Datal.Recordset.MoveNext 'Mueve el registro actual al siguiente
If Datal.Recordset.EOF Then
    Datal.Recordset.MoveLast
```

Uso de las Propiedades y Métodos de un Recordset

Utilice los métodos y las propiedades del objeto **Recordset** para recuperar información del conjunto de registros, desplazarse por los registros y agregar, actualizar o eliminar registros.

Las Propiedades BOF y EOF

Las propiedades **BOF** y **EOF** del objeto **Recordset** indican si la posición del registro actual es antes del primer registro o después del último registro dentro del conjunto de registros. Si no hay registros en el recordset, entonces el valor de las propiedades **BOF** y **EOF** es **True**.

El Método AddNew del Recordset

Para agregar un nuevo registro a un recordset, se emplea el método **AddNew**. Cuando se ejecuta el método **AddNew**, Visual Basic limpia los controles enlazados y establece la propiedad **EditMode** del control **Data** a **dbEditAdd**.

El nuevo registro no será añadido a la base de datos hasta que sea ejecutado explícitamente un método **UpdateRecord** o **Update**, o hasta que el usuario se mueva a otro registro.

El siguiente código muestra como agregar un nuevo registro a un recordset.

```
Private Sub cmdAgregar_Click()  
    Data1.Recordset.AddNew
```

El Método UpdateRecord del Control Data

El método **UpdateRecord** se emplea para grabar el registro actual a una base de datos. El siguiente código muestra como grabar el registro actual y actualizar la base de datos.

```
Private Sub cmdGrabar_Click()  
    Data1.UpdateRecord
```

El Método CancelUpdate del Control Data

El método **CancelUpdate** se emplea para cancelar un método **AddNew** o **Edit** y renovar o refrescar los controles enlazados con datos del recordset.

Por ejemplo, si un usuario ha modificado los campos de un formulario, pero todavía no los ha actualizado, el método **CancelUpdate** refrescará los campos con los datos originales del recordset.

Si un usuario selecciona un botón **Agregar** de un formulario, y luego decide no agregar el registro, el método **CancelUpdate** cancelará la operación y visualizará el registro actual.

El siguiente código muestra como cancelar la agregación o edición de un registro.

```
Private Sub cmdCancelar_Click()  
    Data1.CancelUpdate
```

El Método Delete del Recordset

Para eliminar un registro de una base de datos, se emplea el método **Delete**. El registro eliminado permanecerá como el registro actual hasta que el usuario se mueva a un registro diferente, como se muestra en el siguiente código.

```
Private Sub cmdEliminar_Click()  
    Data1.Recordset.Delete  
    Data1.Recordset.MoveNext  
    If Data1.Recordset.EOF Then  
        Data1.Recordset.MoveLast  
    End If  
End Sub
```

Nota: La base de datos Neptuno tiene definidas reglas de integridad referencial que impiden eliminar registros relacionados.

Uso del Asistente para Formularios de Datos

El Asistente para Formularios de Datos es una utilidad de Visual Basic que genera formularios de datos simples que emplean el control **Data**. Se puede emplear el Asistente para Formularios de Datos para construir rápidamente formularios para una aplicación de base de datos.

Para cargar el Asistente para Formularios de Datos, se emplea la orden **Administrador de Complementos** del menú **Complementos**.

Uso de los Eventos del Control Data

El control **Data** proporciona tres eventos que puede utilizar para mejorar la aplicación de base de datos: Validate, Error y Reposition. Dichos eventos permiten omitir parte del comportamiento predeterminado del control **Data**.

Usar el Evento Validate

Utilice el evento Validate para comprobar los datos antes de guardar un registro en la base de datos. Este evento se produce justo antes de que Visual Basic escriba en la base de datos los cambios procedentes de los controles enlazados y de que vuelva a colocar el puntero del registro actual en otro registro de la base de datos. Puede utilizar el evento Validate para pedir al usuario que confirme los cambios realizados.

Sintaxis

El evento Validate tiene la siguiente sintaxis:

Private Sub Data1_Validate (*index As Integer, action As Integer, save As Integer*)

El Argumento Action

El argumento *action* indica la operación que provocó el evento Validate. El evento Validate se produce como resultado de realizar las operaciones siguientes:

- **MoveFirst, MovePrevious, MoveNext, MoveLast**
- **AddNew**
- **Update**
- **Delete**
- **Find**
- Establecer la propiedad **Bookmark**
- Cerrar la base de datos
- Descargar el formulario

Para cancelar cualquiera de estas acciones, asigne al argumento *action* el valor `vbDataActionCancel`.

El Argumento Save

El argumento *save* indica si va a guardarse o no el registro. Si *save* es **True**, los datos enlazados han cambiado. Para cancelar la acción de guardar puede asignar a *save* el valor **False**.

El siguiente código pide al usuario que confirme los cambios hechos a la base de datos. Si el usuario responde No, los cambios serán cancelados.

```
Private Sub Data1_Validate (Action As Integer, Save As Integer)
    Dim iRespuesta As Integer
    If Save = True Then
        iRespuesta = MsgBox ("¿Desea guardar los cambios?" , vbYesNo)
        If iRespuesta = vbNo Then
            Save = False
            Data1.UpdateControls 'Actualiza los campos
        End If
    End If
End Sub
```

Usar el Evento Reposition

Utilice el evento Reposition para modificar la apariencia de un formulario o realizar una acción necesaria cuando se desplace a un nuevo registro.

Este evento tiene lugar cuando Visual Basic desplaza el puntero del registro actual a otro registro de la base de datos. También se produce la primera vez que se abre la base de datos.

Modificar la apariencia de un formulario

Para cambiar la manera en que un formulario muestra información basándose en el registro seleccionado en ese momento hay que utilizar el evento Reposition. Por ejemplo, puede modificar el título del control **Data** de forma que se muestre el registro número *n*.

Para ver el número del registro actual, utilice la propiedad **AbsolutePosition** del objeto **Recordset**. El número de registro es relativo a cero, por lo que el primer registro es el 0.

El siguiente código muestra como visualizar el número del registro actual.

```
Private Sub Data1_Reposition()
    Data1.Caption = "Registro N°" & Data1.Recordset.AbsolutePosition + 1
End Sub
```

Tratar los cambios al desplazarse a un nuevo registro

Cuando un usuario se desplaza a un nuevo registro mediante el control **Data**, puede que los datos del formulario tengan que presentarse de una forma distinta en el caso del nuevo registro.

Por ejemplo, es posible que en un formulario que muestre registros de empleados haya distintas opciones dependiendo de si los empleados son fijos, temporales o si trabajan por horas. Cada registro mostrará la información sobre un empleado distinto, cuyas opciones pueden no ser las mismas para todos los registros. Para que se seleccione la opción correcta para cada registro, escriba código en el evento Reposition.

El código siguiente utiliza el evento Reposition para modificar la apariencia de un formulario.

```
Private Sub Data1_Reposition()
    Data1.Caption=Data1.Recordset.AbsolutePosition
    If Data1.Recordset("IdEmpleado") > 5 Then
        optSenior.Value = True
    Else
        optJunior.Value = True
    End If
End Sub
```

```
Data1.Caption = "Registro N°" & Data1.Recordset.AbsolutePosition + 1
End Sub
```

Esta ilustración muestra el formulario basado en el código anterior.

Usar el Evento Error

El evento Error tiene lugar cuando un usuario interactúa con el control **Data** y se produce un error de acceso a datos. Utilice el evento Error para agregar tratamiento de errores personalizado al control **Data**.

Por ejemplo, si un usuario modifica un campo y después hace clic en el control **Data** para desplazarse al siguiente registro, el control **Data** actualizará el registro actual. Si se produce un error de acceso a datos durante la actualización, se producirá el evento Error.

Los valores de los campos enlazados no cambian tras producirse un error. El usuario puede corregir los valores y hacer clic en el control **Data** para tratar de actualizar nuevamente el registro.

Mostrar un mensaje de error personalizado

Si no coloca código para tratar errores en el evento Error y se produce un error cuando un usuario interactúa con el control **Data**, Visual Basic muestra el mensaje de error y el programa continúa ejecutándose.

Si no desea que se muestre el mensaje de error estándar, puede asignar al argumento Response el valor 0 y mostrar un mensaje de error personalizado como se muestra en el siguiente código.

```
Private Sub Data1_Error (DataErr As Integer, Response As Integer)
    If DataErr = 3022 Then 'Error de clave principal duplicada
        MsgBox "Ingrese un Id de Empleado único"
        txtIdEmp.SetFocus
        Response = 0
    Else
        Response = 1 'Muestra el mensaje de error estándar
    End If
End Sub
```

Uso de Controles Enlazados a Datos ActiveX

Además de los controles enlazados intrínsecos, Visual Basic ofrece varios **controles ActiveX** enlazados a datos. En esta parte del curso se describen algunos controles ActiveX enlazados a datos avanzados.

Usar el Control DBGrid



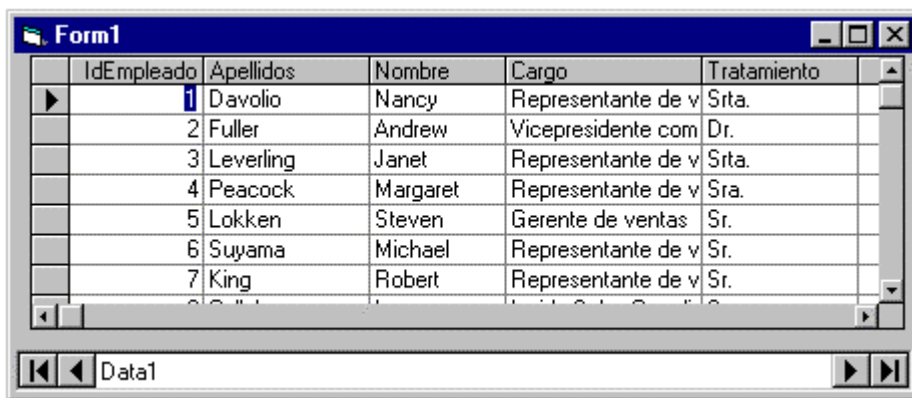
El control de cuadrícula enlazada a datos (control **DBGrid**) permite que los usuarios de su aplicación de base de datos trabajen con varios registros a la vez.

Mostrar múltiples registros

DBGrid es un control ActiveX que muestra una serie de filas y columnas que representan registros y campos de un objeto **Recordset**. Cuando asigne a la propiedad **DataSource** del control **DBGrid** un control **Data**, el control **DBGrid** se llenará automáticamente de datos y se establecerán automáticamente sus encabezados de columna a partir del conjunto de registros del control **Data**.

Al contrario que la mayoría de los controles enlazados a datos, el control **DBGrid** le permite ver y modificar varios registros simultáneamente.

En la siguiente ilustración se muestra un formulario que utiliza un control **DBGrid** para presentar registros procedentes de la base de datos Neptuno.



Para añadir el control DBGrid a un proyecto

1. Hacer clic en la orden **Componentes** del menú **Proyecto**.
2. En la ficha **Controles**, seleccionar **Microsoft Data Bound Grid Control**, y luego hacer clic en **Aceptar**.

El control **DBGrid** tiene varias propiedades que especifican cómo se comporta el control. Por ejemplo, si asigna a la propiedad **AllowUpdate** el valor **True**, un usuario puede modificar los datos del control. También puede establecer propiedades para columnas individuales del control **DBGrid**.

Puede cambiar el título de la columna, cambiar el campo de datos al que se enlaza la columna, agregar valores predeterminados, etc.

Para establecer propiedades de columnas

1. Oprima el botón secundario del ratón sobre el control **DBGrid**.
2. Haga clic en la orden **Propiedades**.
3. Haga clic en **Columnas**.

Puede cambiar el título de la columna, cambiar el campo de datos al que se enlaza la columna, agregar valores predeterminados, etc.

Para establecer los encabezados de las columnas del control **DBGrid**, luego de haberlo enlazado a un control **Data**, oprima el botón secundario del ratón sobre el control en tiempo de diseño, y luego haga clic en la orden **Recuperar campos**.

Obtener y establecer el texto de la celda actual

Utilice la colección **Columns** del control **DBGrid** para recuperar el texto de la celda seleccionada actualmente en tiempo de ejecución. Por ejemplo:

```
MsgBox DBGrid1.Columns(DBGrid1.Col).Text
```

Para cambiar la información del control **DBGrid**, cambie el objeto **Recordset** asociado. Por ejemplo, si el control **DBGrid** está enlazado al recordset **Data1**, se ejecutaría el siguiente código:

```
Data1.Recordset.Edit  
Data1.Recordset.Fields("NombreProducto") = "Disco duro"  
Data1.Recordset.Update
```

Usar el evento BeforeUpdate

El evento **BeforeUpdate** tiene lugar antes de que se muevan datos desde un control **DBGrid** al búfer de copia del control **Data**. Puede validar los datos y cancelar la actualización si es necesario.

Usar el Control MSFlexGrid

El control **MSFlexGrid** proporciona características avanzadas para la presentación de datos en una cuadrícula. Es similar al control **DBGrid** aunque, cuando se enlaza a un control **Data**, el control **MSFlexGrid** muestra datos de sólo lectura. Puede utilizar el control **MSFlexGrid** para combinar filas o columnas de información y así agrupar la información relacionada.

En la siguiente ilustración se muestran registros agrupados por Id. de pedido en la tabla Detalles de pedidos.

IdPedido	IdProducto	PrecioUnidad	Cantidad	Descuento
10248	11	14	12	0
	42	9,8	10	0
	72	34,8	5	0
10249	14	18,6	10	0
	51	42,4	40	0
10250	41	7,7	10	0
	51	42,4	35	0.15
	65	16,8	15	0.15

- **Para combinar celdas**

1. Establezca la propiedad **MergeCells** a un valor diferente a cero.
2. Establezca las propiedades arreglo **MergeRow()** y **MergeCol()** a **True** para las filas y columnas que desee combinar.

Por ejemplo, para combinar las celdas en la tabla Detalles de pedidos, añada el siguiente código al formulario donde se encuentra el control **MSFlexGrid**.

```
Private Sub Form_Load()  
    MSFlexGrid1.MergeCells = flexMergeFree
```

```
MSFlexGrid1.MergeCol(0) = True
End Sub
```

Usar el Control DBCombo



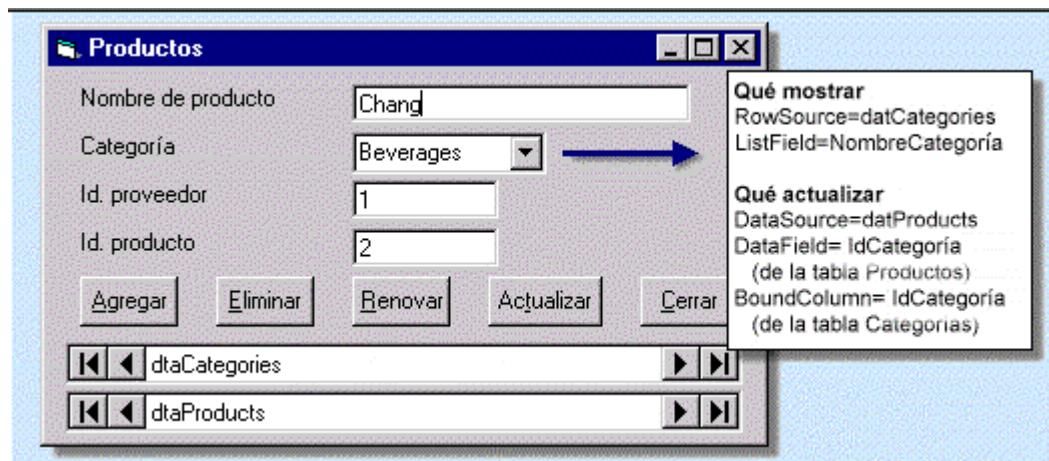
Puede utilizar el control de cuadro de lista enlazado a datos (**DBList**) o el cuadro combinado enlazado a datos (**DBCombo**) para presentar automáticamente una lista de valores de un Recordset. Esto resulta útil para proporcionar valores válidos al usuario.

Obtener información de una tabla de búsqueda

También puede utilizar estos controles en aplicaciones de "tabla de búsqueda".

Por ejemplo, puede presentar una lista de nombres de categoría válidos (en vez de Id.) y utilizar el Id. correspondiente cuando el usuario agregue o modifique datos.

La ilustración siguiente muestra un formulario que utiliza el control **DBCombo** para presentar nombres de categorías de la tabla Productos.



Establecer propiedades del control DBCombo

Para determinar el valor que se presenta en el control **DBCombo**, asigne a la propiedad **RowSource** un nombre de control **Data** y a la propiedad **ListField** un nombre de campo.

El cuadro combinado enlazado a datos contiene todos los valores de ese campo. Para determinar qué campo de la base de datos se actualiza cuando un usuario cambia un valor, establezca las propiedades **DataSource** y **DataField**. Para establecer la relación entre la tabla que contiene los valores de búsqueda y la tabla que se está modificando realmente, establezca la propiedad **BoundColumn**.