

2013

# Curso Básico de Visual Basic 2010

## Teoría

En este documento se les provee la teoría de las clases dictadas a lo largo de todo el curso quedan a su disposición nuestros correos por cualquier consulta: Cuzziol J. Jose(jcuzziol@hotmail.com) Iglesias Dario(darigles@hotmail.com) Pelozo Victor Facundo(facun2\_plz@hotmail.com)

Cuzziol J. Jose Iglesias Dario Pelozo Victor Facundo

U.N.N.E

25/02/2013



|  |    |
|--|----|
| <b>1- Conceptos Basicos y Definiciones de .NET</b> | 2  |
| ¿Qué es Visual Basic .NET?                         | 2  |
| Entorno de Desarrollor                             | 2  |
| .NET Framework                                     | 51 |
| ¿Que es un Namespace (o espacio de nombres)?       | 51 |
| ¿Que es un assembly (o ensamblado)?                | 4  |
| .NET   | 53 |
| <b>2- Introduccion a la herramienta de trabajo</b> | 5  |
| Creacion de Nuevo Proyecto                         | 5  |
| Programación Orientada a Eventos                   | 5  |
| Manejador de Eventos                               | 6  |
| Procedimientos                                     | 6  |
| Ámbito de las variables                            | 6  |
| Módulos  | 6  |
| Funciones  | 7  |
| <b>3- Manejo de las herramientas basicas</b>       | 7  |
| Formulario   | 7  |
| Boton  | 11 |
| TextBox  | 14 |
| Label  | 22 |
| LinkLabel  | 23 |
| CheckBox   | 23 |
| RadioButton  | 25 |
| PictureBox   | 27 |
| ComboBox   | 28 |
| DateTimePicker                                     | 30 |
| MenuStrip  | 31 |
| MsgBox   | 33 |
| <b>4. Control de Eventos</b>                       | 36 |
| Manejo de Eventos                                  | 36 |
| Evento Mas Comunes                                 | 37 |
| Metodos Mas Comunes                                | 38 |
| <b>5-Manejo de Tablas</b>                          | 38 |
| DataRepeter  | 38 |
| DataGridView                                       | 41 |
| <b>7-Base de datos</b>                             | 51 |
| Conceptos Base de Datos                            | 51 |
| Coneccion a Base de datos                          | 54 |

# 1- Conceptos Básicos y Definiciones .NET

## ¿Qué es Visual Basic .NET?

Es un lenguaje orientado a objetos y eventos que soporta encapsulación, herencia y polimorfismo.

Es una mejora a Visual Basic formando parte de Visual Studio y compartiendo el entorno de desarrollo con Microsoft Visual C++ .NET, Microsoft Visual C# .NET, etc.

## Entorno de Desarrollo

El Entorno de Desarrollo recibe el nombre de Entorno de Desarrollo de Microsoft Visual Studio .NET. Este entorno es personalizable y contiene todas las herramientas necesarias para construir programas para Microsoft Windows.

El Entorno de Desarrollo contiene múltiples ventanas y múltiples funcionalidades y es por consecuencia llamado un entorno de desarrollo integrado (integrated development environment IDE).

La ventana central es la ventana de diseño (Designer Window), la cual contiene el formulario a desarrollar.

La caja de herramientas (ToolBox) se localiza de lado izquierdo. En el extremo derecho tenemos la ventana de explorador de soluciones (Solution Explorer).

La ventana de propiedades (Properties window) contiene tres partes:

- La parte superior contiene un combo box que muestra el nombre y la clase del objeto seleccionado.
- La parte media contiene la lista de propiedades del objeto seleccionado, de lado derecho contiene un conjunto de cajas para ver y editar el valor de la propiedad seleccionada.
- La parte inferior es un cuadro descriptivo que proporciona una breve descripción de la propiedad seleccionada.

Es necesario tener instalado el Visual Studio .NET, al ejecutarlo se presenta una página de inicio, en caso de no presentarse entonces de clic en Help/Show Start Page. En esta página será posible establecer su perfil, por ejemplo identificarse como Desarrollador Visual Studio o más específico como Desarrollador Visual Basic con lo cual Visual Studio configura de inmediato el entorno de desarrollo para programar en Visual Basic.

Para iniciar un nuevo proyecto, de clic en la opción Projects y clic en el botón [New Project], esta acción abre una ventana donde se indicará el archivo a abrir, los proyectos Visual Basic .NET tiene la extensión .vbproj. Una vez que abre el proyecto si la página de inicio estaba visible continuará así y en el Explorador de Soluciones (Solution Explorer) se cargan los archivos correspondientes al proyecto.

En Visual Basic .NET existen dos archivos:

- Un archivo de proyecto .vbproj, el cual contiene información específica para una determinada tarea de programación.
- Un archivo de solución .sln, el cual contiene información relacionada con uno o más proyectos. Este tipo de archivo puede administrar varios proyectos relacionados entre sí y son similares a los archivos de grupos de proyecto (.vbg) en Visual Basic 6

Si la solución tiene un único proyecto, abrir el archivo de proyecto .vbproj o el archivo de solución .sln tiene el mismo resultado, pero si la solución es multiproyecto entonces deberá abrir el archivo de solución.

Best Practices: Procure siempre abrir el archivo de solución .sln.

Los formularios en Visual Basic .NET tienen la extensión .vb. Se mostraran a manera de pestañas la página de inicio, la vista de diseño y el código del formulario.

Para evitar el acoplamiento de ventanas, mientras arrastre la ventana pulse la tecla [Ctrl], si desea integrar la ventana como pestaña entonces arrastre la ventana sobre otras pestañas y libere.

El control Imagen desaparece en Visual Studio.

Ya no tendrá que utilizar el tabulador para adentrar su código.

### **.NET Framework**

Visual Studio .NET tiene una nueva herramienta que comparte con Visual Basic, Visual C++, Visual C#, etc. llamada .NET Framework que además es una interfaz subyacente que forma parte del propio sistema operativo Windows.

La estructura de .NET Framework es por Clases mismas que puede incorporar a sus proyectos a través de la instrucción Imports, por ejemplo una de sus Clases es System.Math la cual soporta los siguientes métodos

| Método  | Descripción   |
|---------|---|
| Abs(n)  | Calcula el valor absoluto de n  |
| Atan(n) | Calcula el arcotangente de n en radianes                                    |
| Cos(n)  | Calcula el coseno del ángulo n expresado en radianes                        |
| Exp(n)  | Calcula el constante de e elevada a n                                       |
| Sign(n) | Regresa -1 si n es menor que cero, 0 si n es cero y +1 si n es mayor a cero |
| Sin(n)  | Calcula el seno del ángulo n expresado en radianes                          |
| Sqr(n)  | Calcula la raíz cuadrada de n.  |
| Tan(n)  | Calcula la tangente del ángulo n expresado en radianes                      |

### **¿Qué es un Namespace (o espacio de nombres)?**

"Un espacio de nombres es un esquema lógico de nombres para tipos en el que un nombre de tipo simple, como MiTipo, aparece precedido por un nombre jerárquico separado por puntos. [...]"

Así es como lo definen en el eBook de .NET Framework que mencioné al principio.

Para que nos entendamos, un Namespace, (prefiero usar el nombre en inglés, ya que así es como aparecerá en el código), es una forma de agrupar clases, funciones, tipos de datos, etc. que están relacionadas entre sí. Por ejemplo, entre los Namespaces que

Podemos encontrar en el .NET Framework encontramos uno con funciones relacionadas con Visual Basic: Microsoft.VisualBasic. Si te fijas, Microsoft y Visual BASIC están separados por un punto, esto significa que Microsoft a su vez es un Namespace que contiene otros "espacios de nombres", tales como el mencionado Visual BASIC, CSharp y Win32 con el cual podemos acceder a eventos o manipular el registro del sistema...

Para saber qué es lo que contiene un Namespace, simplemente escribe el nombre con un punto y te mostrará una lista desplegable con los miembros que pertenecen a dicho espacio de nombres.

Por regla general se deberían agrupar en un Namespace funciones o clases que estén relacionadas entre sí. De esta forma, será más fácil saber que estamos trabajando con funciones relacionadas entre sí.

Pero el que distintos espacios de nombres pertenezcan a un mismo Namespace, (viene bien esto de usar la traducción castellana e inglesa de una palabra, para no ser redundante), no significa que todos estén dentro de la misma librería o assembly. Un Namespace puede estar repartido en varios assemblies o librerías. Por otro lado, un assembly, (o ensamblado), puede contener varios Namespaces.

Pero de esto no debes preocuparte, ya que el IDE de Visual Studio .NET se encarga de "saber" en que assembly está el Namespace que necesitamos.

### ***¿Qué es un assembly (o ensamblado)?***

"Un ensamblado es el bloque constructivo primario de una aplicación de .NET Framework. Se trata de una recopilación de funcionalidad que se construye, versiona e instala como una única unidad de implementación (como uno o más archivos). [...]"

Para que nos entendamos, podríamos decir que un assembly es una librería dinámica (DLL) en la cual pueden existir distintos espacios de nombres. Aunque esto es simplificar mucho, por ahora nos vale.

Un ensamblado o assembly puede estar formado por varios ficheros DLLs y EXEs, pero lo más importante es que todos los ensamblados contienen un manifiesto (o manifest), gracias al cual se evitan muchos de los quebraderos de cabeza a los que Windows nos tiene acostumbrados, al menos en lo referente a las distintas versiones de las librerías y ejecutables, seguramente habrás oído hablar de las DLL Hell (o librerías del demonio) expresión que se usa cuando hay incompatibilidad de versiones entre varias librerías que están relacionadas entre sí. Por ejemplo, supongamos que tenemos una librería DLL que en su primera versión contenía X funciones. Al tiempo, se crea la segunda versión de dicha librería en la que se cambian algunas funciones y se añaden otras nuevas, para mejorar el rendimiento de las funciones contenidas en esa librería se usa otra DLL que es usada por algunas de las funciones contenidas en esa segunda versión. Esa otra librería puede ser una librería del sistema, la cual a su vez se actualiza con nueva funcionalidad y puede que dicha funcionalidad dependa a su vez de una tercera librería.

Resulta que instalamos un programa que usa las últimas versiones de todas estas librerías.

### ***¿Qué es el .NET Framework?***

".NET Framework es un entorno para construir, instalar y ejecutar servicios Web y otras aplicaciones.

Se compone de tres partes principales: el Common Language Runtime, las clases Framework y ASP.NET"

"El .NET Framework es un entorno multi-lenguaje para la construcción, distribución y ejecución de Servicios Webs y aplicaciones."

"El .NET Framework es una nueva plataforma diseñada para simplificar el desarrollo de aplicaciones en el entorno distribuido de Internet."

"El .NET Framework consta de dos componentes principales: el Common Language Runtime y la librería de clases .NET Framework."

Tercer intento, aclarando las cosas, para que se te "queden" grabadas:

El .NET Framework es el corazón de .NET, cualquier cosa que queramos hacer en cualquier lenguaje .NET debe pasar por el filtro cualquiera de las partes integrantes del .NET Framework.

El Common Language Runtime (CLR) es una serie de librerías dinámicas (DLLs), también llamadas assemblies, que hacen las veces de las DLLs del API de Windows así como las librerías runtime de Visual Basic o C++. Como sabrás, cualquier ejecutable depende de una forma u otra de una serie de librerías, ya sea en tiempo de ejecución como a la hora de la compilación. Pues el CLR es eso, una serie de librerías usadas en tiempo de ejecución para que nuestros ejecutables o cualquiera basado en .NET puedan funcionar. Se acabó eso de que existan dos tipos de ejecutables: los que son autosuficientes y no dependen de librerías externas o los que necesitan de librerías en tiempo de ejecución para poder funcionar, tal es el caso de las versiones anteriores de Visual Basic.

Por otro lado, la librería de clases de .NET Framework proporciona una jerarquía de clases orientadas a objeto disponibles para cualquiera de los lenguajes basados en .NET, incluido el Visual Basic.

### ***.NET***

Es un lenguaje de programación orientado a objetos que se puede considerar una evolución de Visual Basic implementada sobre el framework .NET. Su introducción resultó muy controvertida, ya que debido a cambios significativos en el lenguaje VB.NET no es compatible

hacia atrás con Visual Basic, pero el manejo de las instrucciones es similar a versiones anteriores de Visual Basic, facilitando así el desarrollo de aplicaciones más avanzadas con herramientas modernas.

La gran mayoría de programadores de VB.NET utilizan el entorno de desarrollo integrado Microsoft Visual Studio en alguna de sus versiones (desde el primer Visual Studio .NET hasta Visual Studio .NET 2012, que es la última versión de Visual Studio para la plataforma .NET), aunque existen otras alternativas, como SharpDevelop (que además es libre).

Al igual que con todos los lenguajes de programación basados en .NET, los programas escritos en VB .NET requieren el Framework .NET o Mono para ejecutarse.

### ***¿Qué es un objeto?***

Cada formulario (ventana), menú o control que se crea con Visual Basic es un módulo auto contenido 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.

## **2- Introducción a la herramienta de trabajo**

### ***Creación de Nuevo Proyecto***

De clic en el botón [New Project] o File/New/Project, como tipo de proyecto seleccione Visual Basic Project, como plantilla seleccione Windows Application, por último indique la ubicación donde desea almacenar su proyecto. Al dar clic Visual Studio configura el entorno de desarrollo y crea un directorio con el mismo nombre que especifico para la aplicación.

### ***Programación Orientada a Eventos***

Visual Basic .NET soporta la Programación Orientada a Eventos en la cual las aplicaciones reconocen y responden a eventos.

Un Evento es una acción o acontecimiento reconocido por algunos objetos para los cuales es necesario escribir el código para responder a dicho evento. Los eventos pueden ocurrir como

resultado de una acción del usuario (onClick), por invocación a través de código o disparados por el sistema (Timer Tick Event).

### ***Manejador de Eventos***

Un Manejador de Eventos contiene código que responde a eventos particulares. Un desarrollador diseña cuidadosamente sus aplicaciones determinando los controles disponibles para el usuario y los eventos apropiados asociados a estos controles, entonces, el desarrollador escribe el código para integrar los eventos consistentes con el diseño de la aplicación.

### ***Procedimientos***

Un procedimiento es un conjunto de sentencias que realizan una acción lógica. Existen tres tipos de procedimientos en Visual Basic .NET:

Event procedures/Event handler, procedimiento que contiene código que es ejecutado en respuesta a un evento. Cuando el evento es disparado el código dentro del manejador de eventos es ejecutado.

Visual Basic .NET para los manejadores de eventos utiliza una convención estándar la cual combina el nombre del objeto seguido de un guión bajo y el nombre del evento.

```
Private|Public Sub objeto_Evento(parámetros) handles Objeto.Evento
    sentencias
End Sub
```

Cada manejador de eventos provee dos parámetros, el primer parámetro llamado sender provee una referencia al objeto que dispara el evento, el segundo parámetro es un objeto cuyo tipo de dato depende del evento que es manejado. Ambos parámetros son pasados por valor.

Si un parámetro es declarado por referencia ByRef el parámetro apunta al argumento actual. Por default los argumentos se pasan por valor ByVal el parámetro es una copia local del argumento.

Sub procedures, contiene código que el desarrollador crea para realizar una acción lógica.

Function procedures, contiene código que el desarrollador crea para realizar una acción lógica y regresa un valor, el valor que una función envía de regreso al programa que lo invoca es llamado valor de regreso. Para regresar un valor se utiliza la sentencia Return.

### ***Ámbito de las variables***

Cuando es declarada una variable también se define su ámbito, el ámbito de una variable es la región de código en la cual la variable se referencia directamente. Existen dos tipos de ámbitos de las variables:

- Local, es una variable declarada dentro de un procedimiento y se destruye cuando el procedimiento termina de ejecutarse.
- Modular, es una variable declarada a nivel módulo fuera de cualquier procedimiento y son declaradas en la parte superior del Editor de Código arriba del primer procedimiento, este espacio es llamado Sección de Declaraciones Generales (General Declaration Section).

### ***Módulos***

Los miembros (constantes, variables, sub y functions) declarados dentro de un módulo son de manera predeterminada accesibles públicamente y se puede obtener acceso a ellos mediante cualquier código que tenga acceso al módulo. Esto significa que las variables en un módulo

estándar son de hecho variables globales porque son visibles desde cualquier parte del proyecto y existen durante toda la vida útil del programa.

### ***Funciones***

Una función en visual basic net es un módulo de un programa separado del cuerpo principal, que realiza una tarea específica y que puede regresar un valor a la parte principal del programa u otra función o procedimiento que la invoque.

La forma general de una función es:

```
Function Nom_fun(parametros)
```

```
instrucciones
```

```
nomfun = cargarlo porque es quien regresa el dato
```

```
End Function
```

La lista de parámetros formales es una lista de variables separadas por comas (,) que almacenaran los valores que reciba la función estas variables actúan como locales dentro del cuerpo de la función.

Aunque no se ocupen parámetros los paréntesis son requeridos.

Dentro del cuerpo de la función deber haber una instrucción que cargue el NOMFUNCION para regresar el valor, de esta manera se regresan los datos.

Sin embargo es de considerar que NOMFUNCION puede regresar un dato, una variable o una expresión algebraica (no ecuación o formula) como lo muestran los siguientes ejemplos;

a) FUNCION1 = 3.1416

b) FUNCION1 = area

c) FUNCION1 = x + 15 / 2

Recordar además:

a) Una función no se llama usando CALL

b) Cuando se llame a una funcion debera haber una variable que reciba el valor que regresara la función, es decir generalmente se llama una función mediante una sentencia de asignación, por ejemplo resultado = funcion(5, 3.1416)

## **3- Manejo de las herramientas básicas**

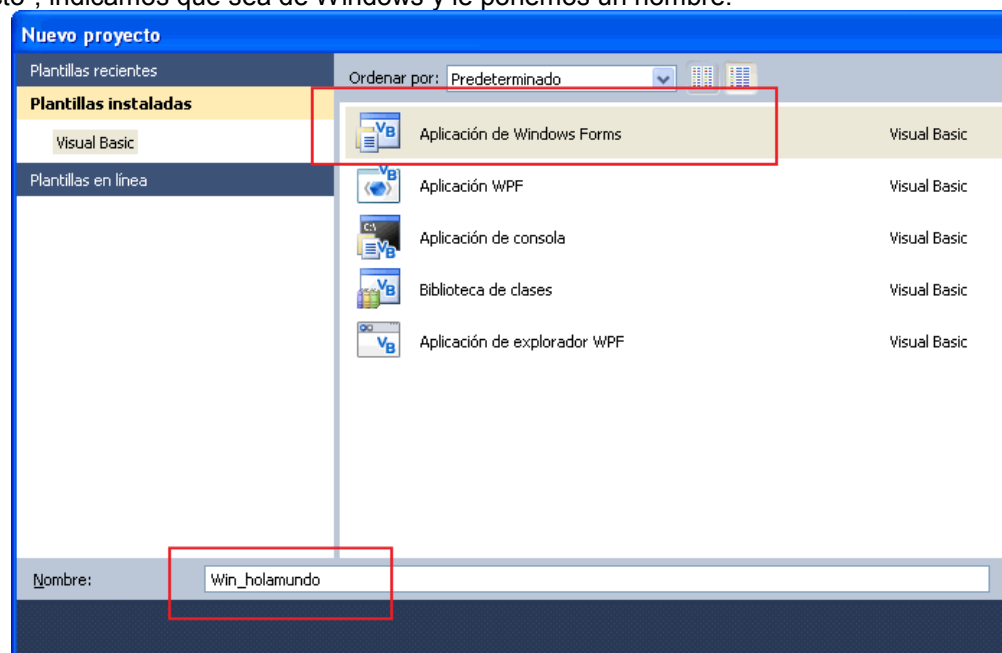
### ***Formularios***

Los formularios son las ventanas que contienen a otros controles y en ellos podemos incluir controles de comando, como botones, cajas de texto, controles para bases de datos, y por supuesto el código necesario de nuestros programas. También se dijo, los controles e inclusive los Formularios tienen sus propias características, como las propiedades (un ejemplo el color del fondo o propiedad Backcolor), sus eventos (el evento load que se ejecuta cuando cargamos el formulario en memoria), y sus métodos (por ejemplo el método show que es para mostrar el

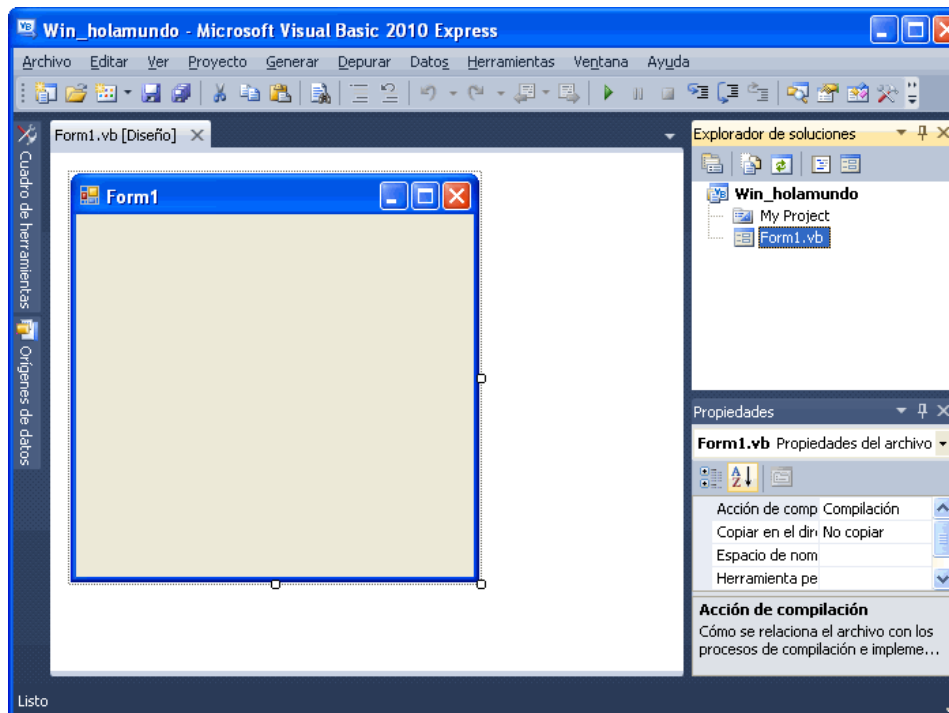


formulario y el método hide que es para ocultarlo, en el caso de un formulario. Acá hay que hacer una cosa muy importante. Cuando se comienza a programar, suele ser muy casual confundir métodos y eventos y no son lo mismo. **Los métodos** son funciones propias de cada objeto, por ejemplo el método hide de un formulario no lo posee un command button. Con **los eventos** suele ocurrir lo mismo. Un formulario posee el evento load y un command button no lo posee (a no ser que pertenezca un arreglo de controles pero ese es otro tema que se verá en otra ocasión). Pero otros eventos si los comparten ambos. Siguiendo el ejemplo del formulario y el botón, los dos tienen un evento llamado click, y todas las instrucciones de código que escribamos en la rutina de código o procedimiento que estén bajo este evento, se ejecutarán cuando hagamos un click con el mouse sobre el objeto. El evento Click lo poseen la mayoría de los controles, pero no tiene por qué ser así. Para seguir con el tema de los métodos y eventos, vamos a hacer un ejemplo simple de ejercicio. Este ejemplo, consistirá en un formulario que contendrá un botón y, al presionarlo nos mostrará otro formulario.

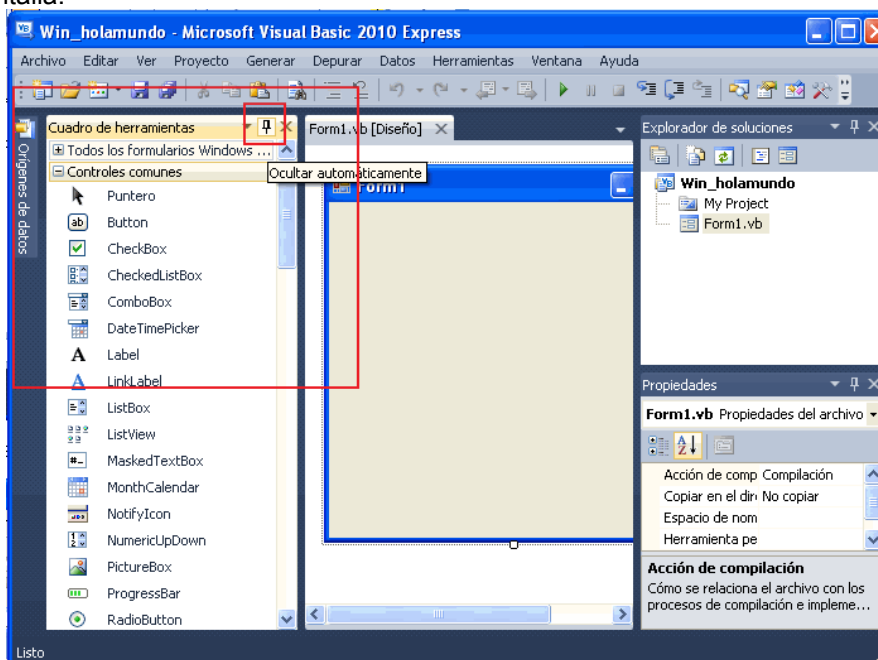
Ahora vamos a elegir crear una aplicación Windows. Volvemos a indicar que queremos un "Nuevo proyecto", indicamos que sea de Windows y le ponemos un nombre:



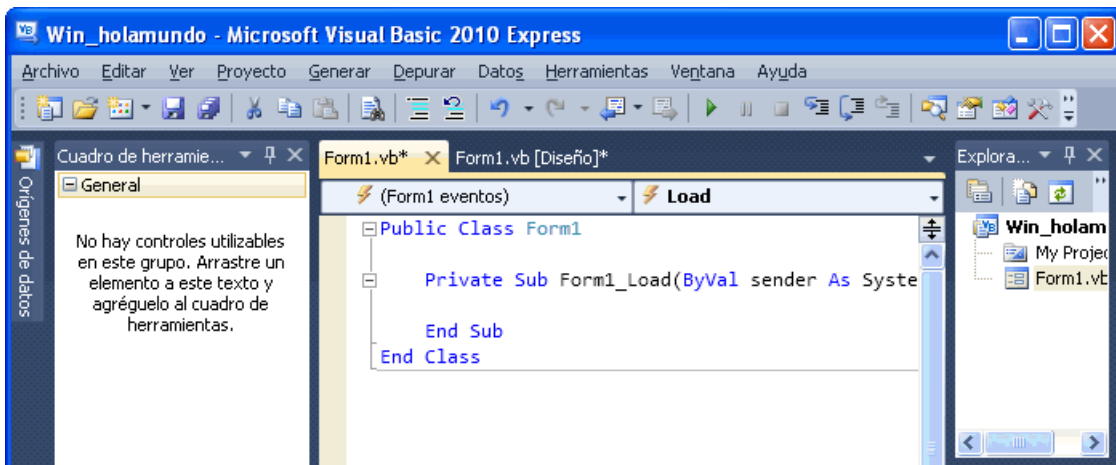
El aspecto del IDE ahora será distinto ya que ahora partimos de un formulario Windows:



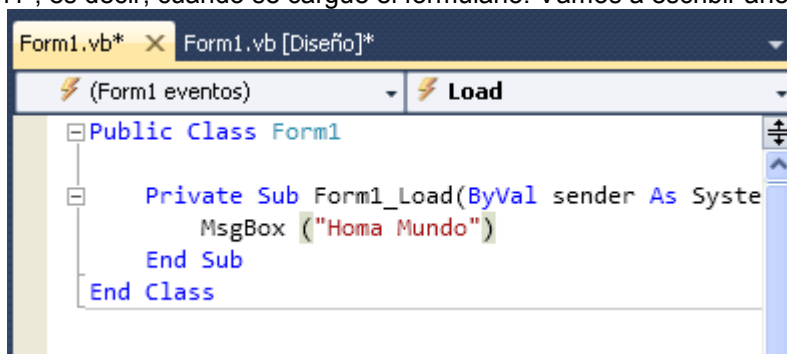
En la parte de la izquierda tenemos dos solapas verticales: "Cuadro de herramientas" y "Orígenes de datos". La primera contiene los controles de Windows que podremos utilizar en nuestras aplicaciones Windows. Haz clic en ella y luego en la "chincheta" que aparece para que se quede fija en la pantalla:



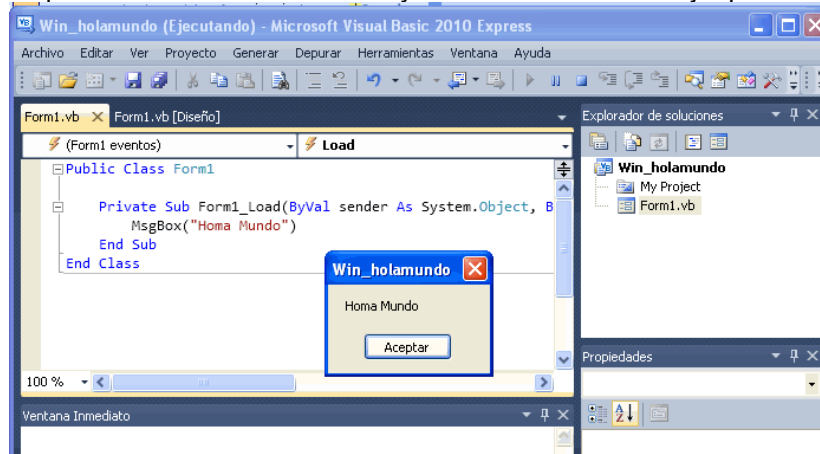
En nuestro entorno de desarrollo para la aplicación Windows tenemos una serie de controles a la izquierda, en el centro un formulario para nuestra aplicación y la derecha dos ventanas: el explorador de soluciones y la ventana de propiedades. Luego veremos con detalle todas estas partes, sigamos con nuestro ejemplo. Vamos a hacer doble clic en el centro del formulario que tiene de título "Form1" en el centro de la ventana. Nos mostrará una ventana con este aspecto:



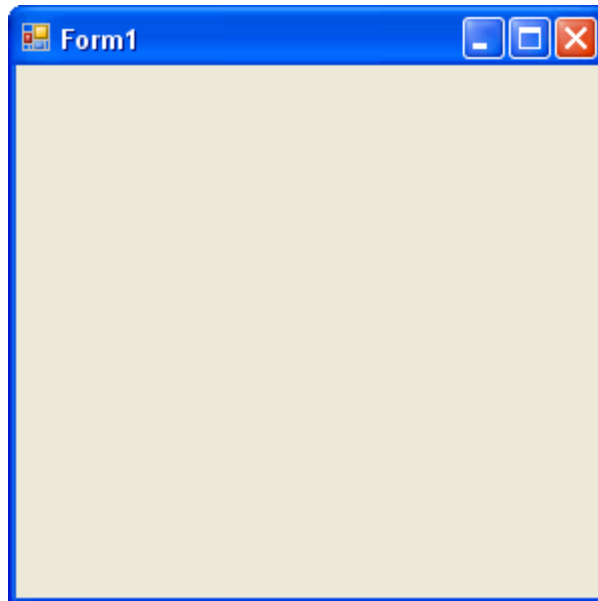
Nos está indicando que es ahí donde debemos introducir el código cuando se produzca el evento "Load" de "Form1", es decir, cuando se cargue el formulario. Vamos a escribir ahora este código:



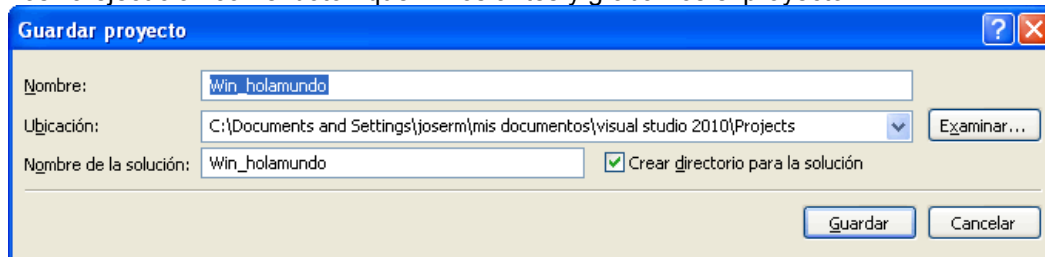
Ejecutaremos la aplicación como hicimos antes y nos mostrará el mensaje que le hemos indicado:



Y si pulsamos en "Aceptar" nos mostrará el formulario:



Paramos la ejecución con el botón que vimos antes y grabamos el proyecto:



Y ya tenemos nuestra primera aplicación Windows, ahora veremos los ficheros que ha creado al generar el proyecto.

## Botón

### 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.

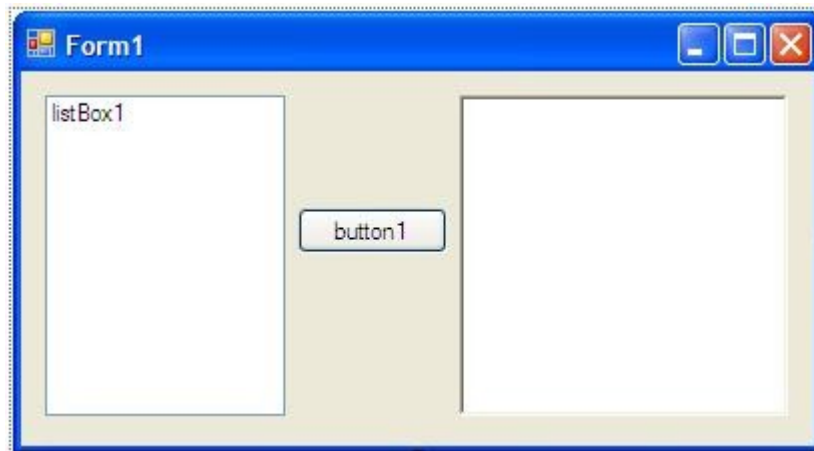
#### Eventos

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

#### Métodos

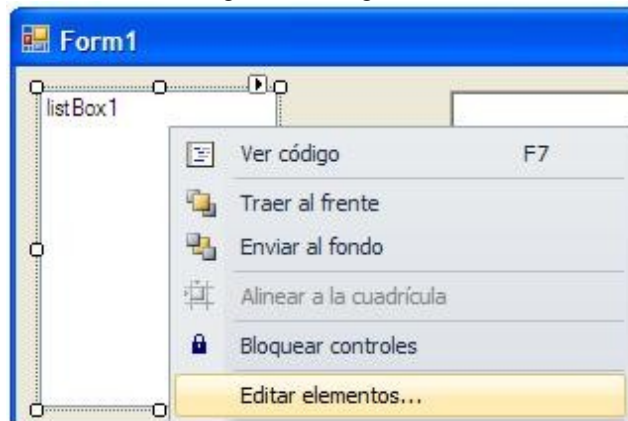
**SetFocus** Mueve el enfoque al botón.

En nuestro ejemplo vamos a dejar nuestro formulario de esta forma, trata de acomodar el tuyo también así:

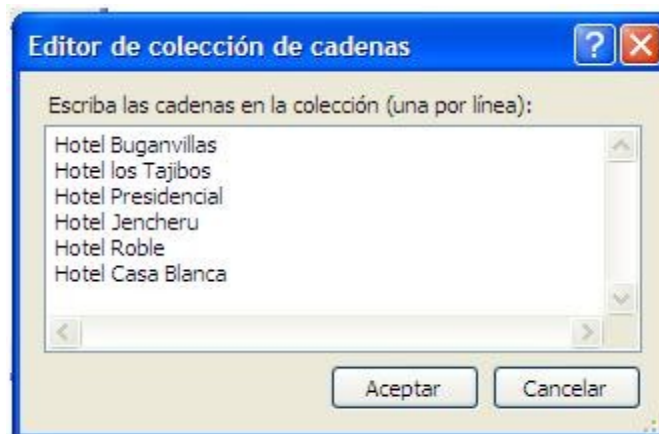


El elemento de la izquierda es un Listbox, del centro es el botón y el de la derecha es el RichText, el Listbox y el RichText son parecidos en el aspecto pero sus utilidades son diferentes.

Ahora démosle unos cuantos ítems por defecto a nuestro Listbox, es sencillo hagámosle clic derecho y después en la lista que nos desplegara escogemos haciendo clic en EDITAR ELEMENTOS... como se muestra en la siguiente imagen:

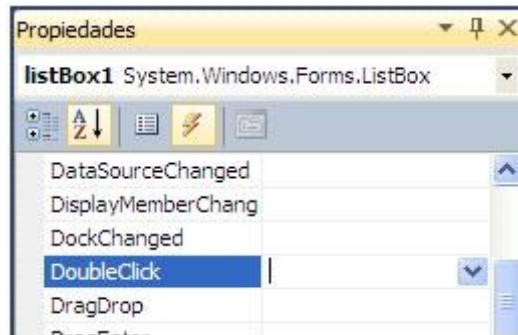


Ahora la ventana que nos aparece es un editor de texto en el cual vamos a escribir nuestros ítems de ejemplo:



Después de escribir nuestros ítems para guardarlo simplemente hacemos clic en el botón aceptar. Ahora vamos a programar para que cuando el usuario haga doble clic en un ítem del LISTBOX se pase automáticamente al RichText para eso vamos a hacer clic en el LISTBOX, después vamos a la parte derecha a las propiedades y hacemos clic en el rayo para entrar en los eventos del componente LISTBOX y buscamos el evento DOUBLECLICK y luego le hacemos doble clic para

poder escribir código en este evento, la imagen de abajo muestra el evento DOUBLECLICK donde tenemos que hacer doble clic:



Este es el editor de código que se mostrara ala entrar en el evento DOUBLECLICK:

```
private void listBox1_DoubleClick(object sender, EventArgs e)
{
}
```

Ahora escribimos el siguiente código entre las llaves:

```
//Necesitamos capturar el objeto listBox1.SelectedItem para convertirlo en String
string hotel;
hotel = System.Convert.ToString (listBox1.SelectedItem);
richTextBox1.AppendText(hotel);
```

```
private void listBox1_DoubleClick(object sender, EventArgs e)
{
    //Necesitamos capturar el objeto listBox1.SelectedItem para convertirlo en String
    string hotel;
    hotel = System.Convert.ToString (listBox1.SelectedItem);
    richTextBox1.AppendText(hotel);
}
```

Ahora hacemos doble clic en el botón del formulario para hacer la función que cargue todos los elementos del LISTBOX al RITCHBOX de una sola vez:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Cuando hagamos doble clic en el botón nos saldrá el siguiente código:

Y escribimos el siguiente código entre las llaves:

```
richTextBox1.Clear();
for (int i = 0; i < listBox1.Items.Count;i++ )
{
```

```
    richTextBox1.AppendText(System.Convert.ToString (listBox1.Items[i])+"\n");
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    richTextBox1.Clear();
    for (int i = 0; i < listBox1.Items.Count; i++)
    {
        richTextBox1.AppendText(System.Convert.ToString (listBox1.Items[i])+"\n");
    }
}
```

Listo con esto terminamos, podemos guardar y ejecutar el proyecto.

## TextBox

### 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.

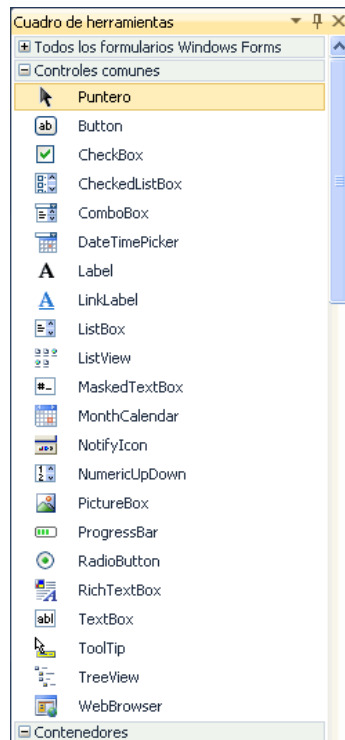
**LostFocus** Ocurre cuando el control pierde el enfoque.

#### Métodos

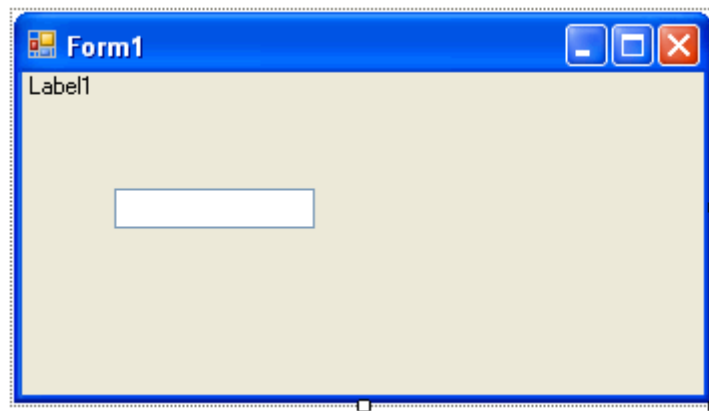
**Refresh** Actualiza el texto del control.

**SetFocus** Mueve el enfoque al control.

Los controles son los elementos que insertamos dentro de un formulario y que nos va a permitir interactuar entre el usuario y el código. Controles son: botones, cuadros de texto, etiquetas, cuadros desplegables, cuadrículas de datos,... en definitiva todos y cada uno de los elementos que vemos en los formularios de todas las aplicaciones. La lista de controles básicos disponibles la tenemos a la izquierda:



Para añadir controles al formulario utilizaremos esta barra, por ejemplo para añadir una etiqueta (Label) y una caja de texto (TextBox), simplemente haremos doble-clic sobre esos elementos de la barra de herramientas y se añadirán al formulario.



Tenemos varias formas de añadir estos controles:

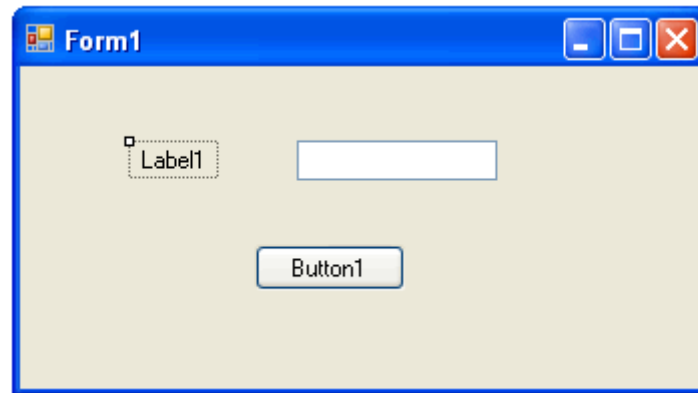
Haciendo doble clic sobre el icono del control. Se colocará un control en la parte izquierda superior del formulario.

Haciendo clic sobre el icono del control y después clic sobre el formulario. El control se insertará en el punto que hemos indicado y se extenderá hacia la derecha y abajo según su tamaño predeterminada

La tercera forma es la más utilizada. Hacemos clic sobre el control y luego en el formulario. Pero esta vez sin soltar el botón izquierdo, vamos dibujando con el ratón la posición y tamaño del control.

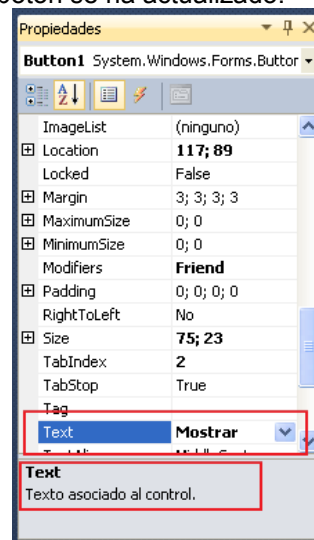
Para nuestro ejemplo añadiremos un botón (Button) y lo situaremos debajo del cuadro de texto (Textbox). Luego añadimos una etiqueta (Label) para que quede de esta forma:





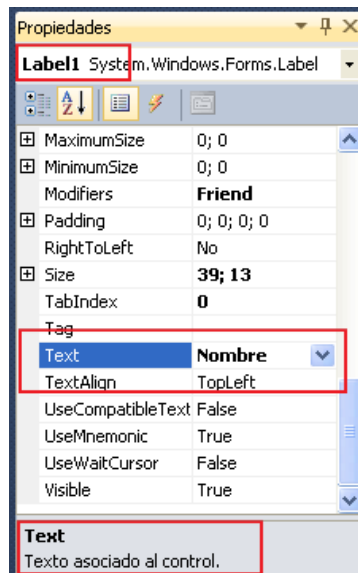
Como vemos por defecto el IDE pone unos nombres genéricos a los controles: label1, textbox1, button1, es decir utiliza el tipo de control y lo va numerando: label1, label2, label3,... es por ponerle un nombre inicial ya que siempre los controles deben tener un nombre único. En nuestros proyectos cambiaremos esos nombres de controles por otros nombres más descriptivos, por ejemplo: txt\_nombre, txt\_apellidos, txt\_telefono en lugar de los puestos por el IDE: textbox1, textbox2, textbox3, ...

Ahora vamos a cambiar el texto que contiene el botón "Button1". Para cambiarle este texto hay que utilizar la ventana de propiedades, en esta ocasión el elemento que nos interesa de esa ventana de propiedades es Text, escribimos en esta propiedad la palabra "Mostrar" y cuando pulses Intro o el tabulador veremos que el texto del botón se ha actualizado:

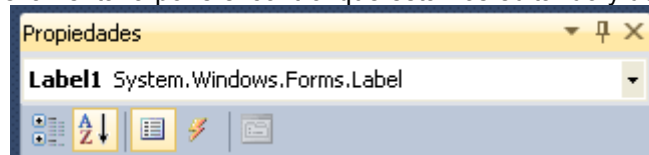


#### Anotación

Para los usuarios de versiones anteriores de VB: La propiedad Caption no existe en ningún control, ahora se llama Text. Por lo tanto hemos modificado la propiedad Text y no Caption en el botón. Hacemos lo mismo con la etiqueta, recuerda que hay que seleccionarla primero haciendo clic para que se muestren las propiedades de la etiqueta, escribimos "Nombre:" y pulsamos intro o el tabulador:



En la parte superior de la ventana pone el control que estamos editando y de qué tipo es:



En este caso es una etiqueta (label) que la ha creado a partir de la clase "System.Windows.Forms.Label"

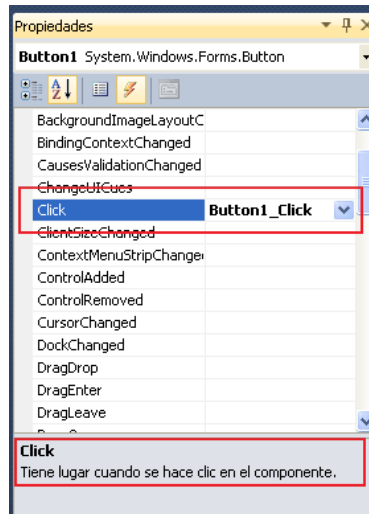
Al principio y para asegurarnos de que vamos escribiendo bien estas propiedades nos fijaremos bien que tenemos seleccionado el control adecuado. En la pantalla anterior podemos ver en el título superior el nombre del control: "Label1" así sabemos que estamos escribiendo la propiedad en el control adecuado. Es normal al principio que escribamos o modifiquemos alguna propiedad y no veamos el resultado. Esto es porque nos hemos equivocado y no estaba correctamente seleccionado el control.

Cómo escribir el programa...

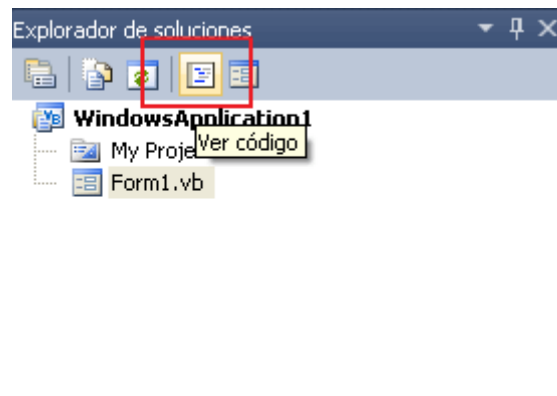
Ahora vamos a escribir código para que se ejecute cada vez que se haga clic en el botón que hemos añadido. Para ello, selecciona el botón Mostrar y hacemos doble clic en él, se mostrará una nueva ventana, en este caso la ventana de código asociada con el formulario que tenemos en nuestro proyecto. Nos mostrará:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    End Sub
End Class
```

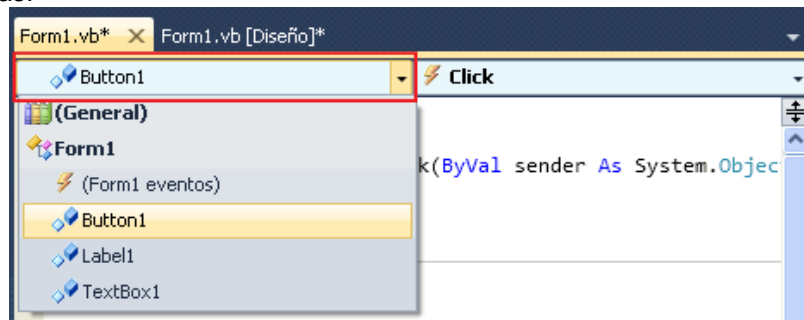
Aquí empieza la programación. Que nosotros pulsemos doble clic en el botón y que aparezca un fragmento de código significa lo siguiente. VB .NET interpreta que quieres poner código para realizar alguna acción cuando se haga clic sobre el botón, por lo tanto te muestra ya el "procedimiento" o parte de código que se va a ejecutar cuando suceda esto. A esta forma de trabajar se le llama programación orientada a eventos. Es decir, cuando se produzcan el evento de pulsar el botón (clic) ejecutas este código. Por lo tanto vemos que los controles además de tener propiedades (que modifican su aspecto) también atienden a una serie de eventos (clic, doble clic). Los eventos a los que atienden los controles los podemos ver en la ventana de propiedades, seleccionando:



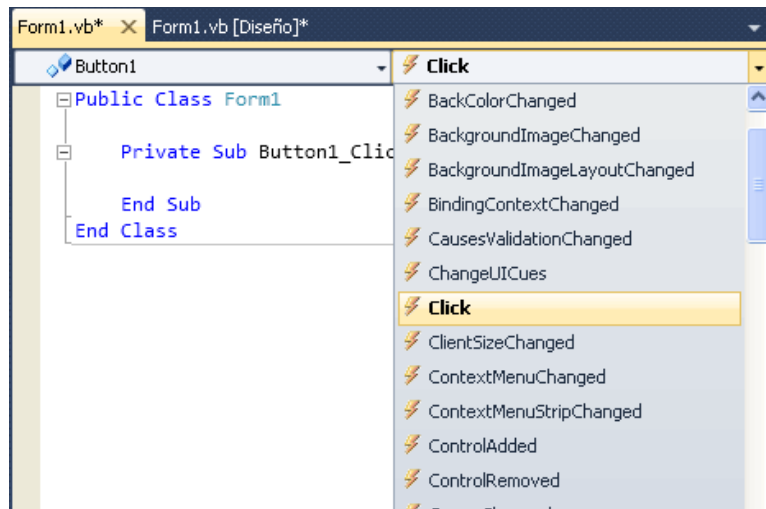
. Hay otra forma de acceder a los eventos de los controles y es esta: vete a la vista del código del formulario pulsando en:



Llegaremos otra vez a la ventana del código de antes, haz clic en el desplegable de arriba a la izquierda y verás:



Que son los objetos que hemos añadido en nuestro formulario. Selecciona en botón "Button1" y fíjate ahora en el desplegable de la derecha:



Que es la lista de eventos a los que puede atender nuestro objeto, en este caso el botón. Por tanto cuando te diga de acceder al "evento tal" del objeto iremos a la vista de código, seleccionaremos el objeto en la lista de la izquierda y el evento en la derecha.

Volvemos a la pantalla del código para el evento clic del botón:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    End Sub
End Class
```

Esta pantalla nos muestra mucha más información de lo que parece en principio. Nos está indicando en la parte superior que estamos trabajando con "Form1.vb". Debajo nos indica a la izquierda que estamos con el control llamado "Button1" y a la derecha trabajando con el evento clic. Pero ¿no le hemos dado doble clic para que aparezca esto? no sería más coherente que nos presentase el evento "doble clic" en lugar del "clic". Pues siendo estrictos es verdad pero como la propiedad más importante de los botones es el clic eso es lo que nos mostrará de forma predeterminada, (además no existe la acción del doble clic en los botones).

Sigamos con la pantalla... y este fragmento de código:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Button1.Click
```

```
End Sub
```

Recordamos que lo que queremos hacer ahora es escribir código para que se ejecute hacer clic en el botón, lo cual producirá el evento clic asociado con dicho botón. Este evento se disparará al hacer clic con el ratón, o bien porque se pulse la tecla intro o la barra espaciadora cuando el botón tenga el foco.

Anotación

El foco o enfoque es cuando un determinado control está seleccionado. Cuando pulsamos la tecla tabulador para pasar a otro control lo que están haciendo es activar el enfoque (getfocus). El anterior control lógicamente deja de estar seleccionado, luego, pierde el enfoque (lostfocus).

La nomenclatura, (forma de llamar a los elementos), para los eventos de Visual Basic siguen el siguiente esquema:

[nombre del control] [guión bajo] [nombre del evento]

button1\_click ' ejecuta el código cuando se hace clic en el elemento button1

Pero esto sólo es una sugerencia que VB.Net nos hace, en las versiones anteriores no era una sugerencia, era una imposición. Podemos dejar el nombre que nos sugiere o podemos poner el nombre que nosotros queramos. Lo importante aquí es la parte final de la línea de declaración del

procedimiento: Handles Button1.Click, con esto es con lo que el compilador/intérprete de VB sabe que este procedimiento (fragmento de programa) es un evento y que dicho evento es el evento clic del objeto Button1.

El nombre debe empezar por una letra o un guión bajo.

El nombre sólo puede contener letras, números y el guión bajo.

Es buena técnica poner las iniciales del tipo de control que es: lb\_nombre (label de un nombre), btn\_aceptar (botón de aceptar), chk\_estado (casilla de verificación de estado), txt\_nombre (cuadro de texto)

Así que, si queremos cambiar el nombre al evento que se produce cuando se hace clic en el botón, escribiremos ese nombre después de Private Sub, aunque no es necesario cambiar el nombre del evento, ya que, al menos por ahora, nos sirve tal y como está.

Lo que sí importa es lo que escribamos cuando ese evento se produzca, en este caso vamos a hacer que se muestre un cuadro de diálogo mostrándonos el nombre que previamente hemos escrito en el cuadro de texto.

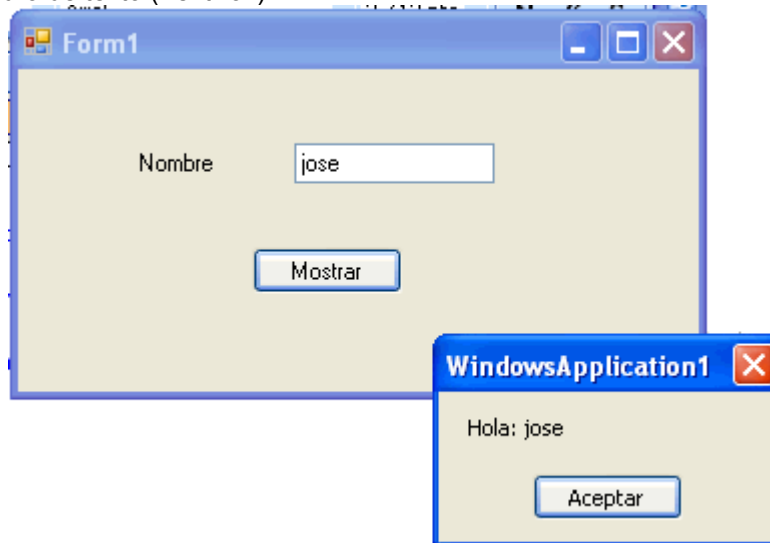
Escribimos el siguiente código en el hueco dejado por Visual Basic entre las líneas que hay entre

Private Sub... y End Sub

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MsgBox ("Hola: " & TextBox1.Text)
    End Sub
End Class
```

Antes de explicar que estamos haciendo, pulsamos F5 para que se ejecute el código que hemos escrito o pulsa en el botón "play" que está en la barra de botones.

Cuando se presente el formulario escribe algo en el cuadro de texto, pulsa en el botón Mostrar y veremos que se muestra un cuadro de diálogo diciéndote Hola y a continuación lo que hayas escrito en el cuadro de texto (TextBox):

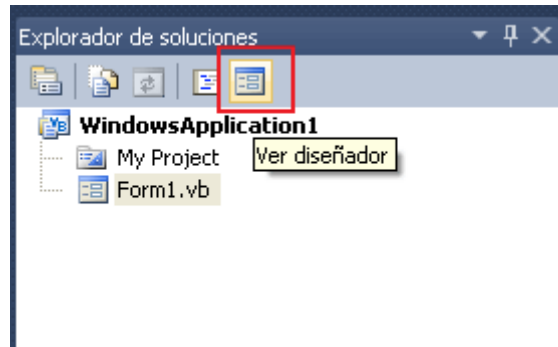


Ya tenemos otra aplicación Windows funcionando creada con Visual Basic .NET. Prueba a escribir un texto en el cuadro de texto y pulsa en Mostrar: aparecerá otro cuadro de diálogo con el contenido de lo que escribamos. Pulsamos ahora en la x del formulario para terminar la aplicación o en el botón de terminar de la barra de tareas.

Ahora vamos a añadir otro botón que llamaremos cmdCerrar con el texto "Cerrar". (el nombre es por "cmd" de comando y Cerrar de la acción: cmdCerrar). Para esto primero vamos al IDE para añadir otro botón. Ahora cambiaremos dos propiedades: Name para ponerle cmdCerrar y Text para ponerle Cerrar:

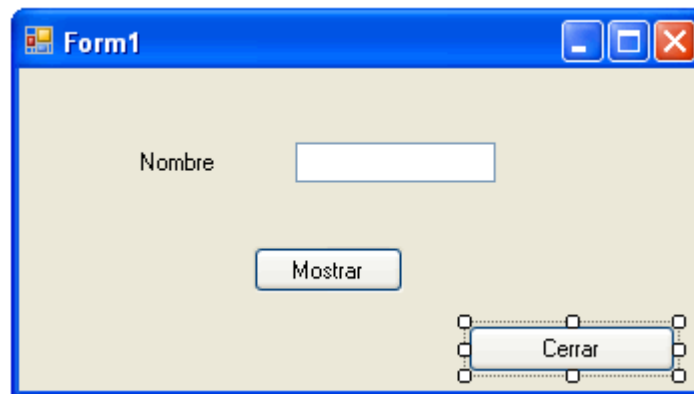
### Anotación

Si estamos con el editor de código y queremos pasar al diseñador del formulario podemos hacer de varias formas, una es haciendo doble clic en el formulario en la parte derecha en el "Explorador de soluciones". Otra y más cómoda es pulsar en el botón que aparece en el explorador de soluciones:



Que como vemos permite cambiar entre las pantallas de formulario, código y otras que veremos más adelante.

Seguimos con el ejemplo, pintamos un nuevo botón y cambiamos las dos propiedades comentadas anteriormente:



Obviamente al ponerle esta vez un nombre al control, en lugar de llamarse Button2 se llamará como le hemos indicado cmdCerrar y así lo veremos en el código. Pulsamos doble clic en él para escribir el código que queremos que ejecute:

```
Private Sub cmdCerrar_Click(ByVal sender As System.Object,
    |
End Sub
```

y escribimos la instrucción

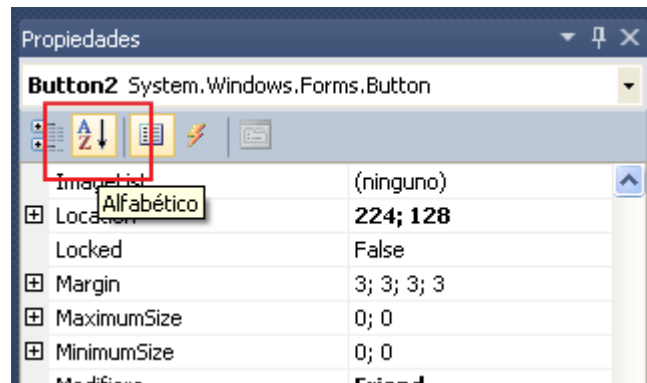
```
Me.Close()
```

Truco

o

consejo

Para buscar más fácilmente las propiedades en la ventana de propiedades, puedes hacer que se muestren por orden alfabético, simplemente pulsando en el botón AZ:



Pulsamos F5 para ejecutar el programa y la diferencia respecto al programa anterior es que ahora al pulsar este botón el programa termina, cierra el formulario y nos devuelve el entorno de desarrollo. De esta forma el programa está un poco más completo.

## Label

### 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.

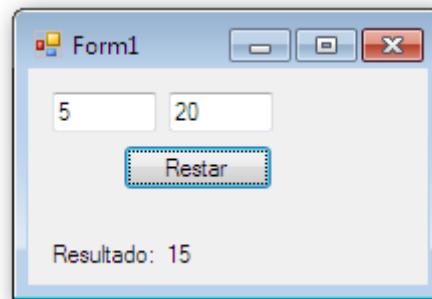
**AutoSize** True/False. Determina si el tamaño del control se ajusta automáticamente al texto que contiene.

**Caption** Texto que muestra el control.

**Name** Nombre del control.

**Font** Establece la fuente, estilo y tamaño para el texto del control.

Veamos un ejemplo en Visual Basic .NET, imaginemos un formulario con dos cajas de texto (Textbox) un botón y un label que muestra el resultado; veamos el ejemplo gráficamente y luego el código fuente:



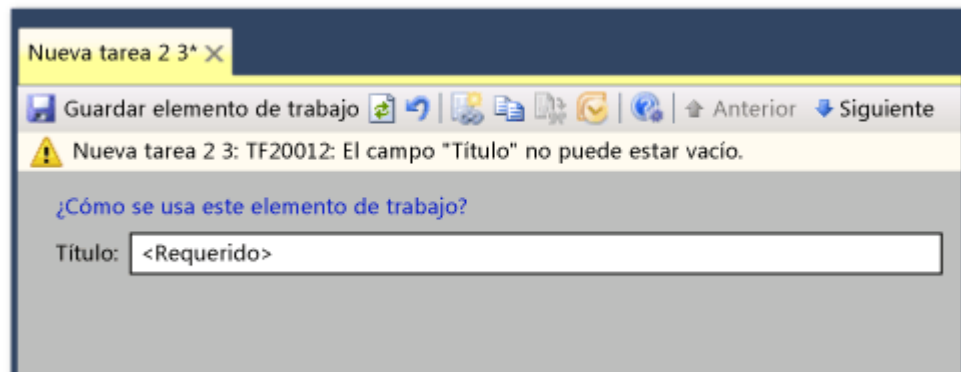
En este ejemplo crearemos una especie de programa que permita restar dos números, pero imaginemos que debemos poner un filtro que detecte cuando el primer número es menor al que vamos a restar. Veamos el código fuente:

```
Public Class Form1
    Dim resultado As Long
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        resultado = Val(TextBox1.Text) - Val(TextBox2.Text)
        If resultado < 0 Then
            resultado = resultado * -1
        End If
        Label1.Text = resultado
    End Sub
End Class
```

Lo que hicimos fue que al hacer clic, se restaran los números, pero entonces si restamos por ejemplo un número pequeño a uno grande este se convertirá en negativo, entonces agregamos un condicional que dice que si el resultado de la operación es menor 0 (osea los números negativos -1,-2,-3, etc) este será multiplicado por menos uno (-1) y así lograremos convertir nuestro resultado y convertir el número a positivo.

### LinkLabel

El control LinkLabel de formularios Windows Forms permite agregar vínculos de estilo Web a aplicaciones de Windows Forms. Puede utilizar el control LinkLabel para todo aquello para lo que pueda utilizar el control Label. También puede establecer parte del texto como un vínculo a un objeto o una página Web.



En el siguiente ejemplo se muestra cómo agregar un hipervínculo al texto mostrado en un formulario de elemento de trabajo.

```
<Group>
  <Column PercentWidth="100">
    <!-- Standalone label control 2 -->
    <Control Type="LabelControl" Label="How do I use this work item?">
      <Link UrlRoot="http://www.live.com"></Link>
    </Control>
  </Column>
</Group>
```

### CheckBox

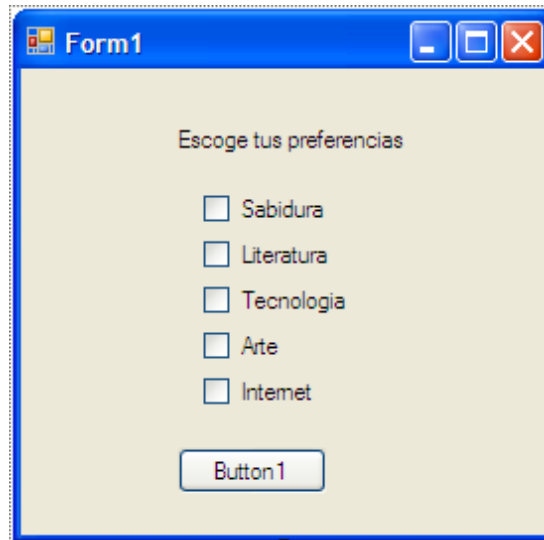
El control CheckBox proporciona una representación visual que hace que esta opción sea fácil de crear.

El control CheckBox se compone de una etiqueta de texto y un cuadro que el usuario puede seleccionar. Cuando el usuario hace clic en el cuadro, aparece una marca de verificación en él. Si se vuelve a hacer clic en el cuadro, la marca de verificación desaparece. El estado de la casilla de verificación se puede recuperar utilizando la propiedad CheckBox.Checked. Si el cuadro muestra una marca de verificación, la propiedad devuelve True. Si no se muestra ninguna comprobación, la propiedad devuelve False.

Para empezar haremos un nuevo proyecto en Visual Basic 2010:

Para esto necesitaremos tres herramientas un LABEL para el título un, cinco CHECKBOX para las cinco preferencias y un botón para aceptar las preferencias. Entonces pongamos estas herramientas a nuestro formulario dejándolo como se muestra en la siguiente imagen:





```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

Este es el código que vamos a usar es bastante sencillo de entender, escríbelo antes de END SUB:

```
DimcoleccionAsString
```

```
Dim total AsInteger
```

```
total = 0
```

```
coleccion = ""
```

```
If CheckBox1.Checked Then
```

```
coleccion = coleccion + ", " + CheckBox1.Text
```

```
total = total + 1
```

```
EndIf
```

```
If CheckBox2.Checked Then
```

```
coleccion = coleccion + ", " + CheckBox2.Text
```

```
total = total + 1
```

```
EndIf
```

```
If CheckBox3.Checked Then
```

```
coleccion = coleccion + ", " + CheckBox3.Text
```

```
total = total + 1
```

```
EndIf
```

```
If CheckBox4.Checked Then
```

```
coleccion = coleccion + ", " + CheckBox4.Text
```

```
total = total + 1
```

```
EndIf
```

```
If CheckBox5.Checked Then
```

```
coleccion = coleccion + ", " + CheckBox5.Text
```

```
total = total + 1
```

```
EndIf
```

```
MsgBox("Elegiste: " + CStr(total) + " preferencias que son:" + coleccion)
```

La siguiente imagen es una fotografía del código escrito en el editor de Visual Basic 2010:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim coleccion As String
        Dim total As Integer
        total = 0
        coleccion = ""

        If CheckBox1.Checked Then
            coleccion = coleccion + ", " + CheckBox1.Text
            total = total + 1
        End If

        If CheckBox2.Checked Then
            coleccion = coleccion + ", " + CheckBox2.Text
            total = total + 1
        End If

        If CheckBox3.Checked Then
            coleccion = coleccion + ", " + CheckBox3.Text
            total = total + 1
        End If

        If CheckBox4.Checked Then
            coleccion = coleccion + ", " + CheckBox4.Text
            total = total + 1
        End If

        If CheckBox5.Checked Then
            coleccion = coleccion + ", " + CheckBox5.Text
            total = total + 1
        End If

        MsgBox("Elegiste: " + CStr(total) + " preferencias que son:" + coleccion)

    End Sub
End Class
```

Ahora solo tienes guardar y ejecutar para ver cómo te funciona

### **RadioButton**

A diferencia de las casillas de verificación, los botones de opción siempre funcionan como parte de un grupo. Al seleccionar un botón de opción inmediatamente se borran todos los otros botones de opción en el grupo. Al definir un grupo de botones de opción, se indica al usuario que "tiene este conjunto de opciones entre las que puede elegir una y solamente una".

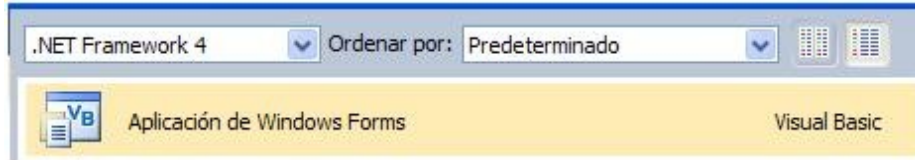
Puede utilizar grupos de controles RadioButton para permitir a los usuarios elegir entre las opciones exclusivas. Por ejemplo, puede permitir que un usuario elija salsa normal o salsa picante en la pizza, pero no ambas. Como un control CheckBox, puede recibir información sobre el estado del control RadioButton de la propiedad RadioButton.Checked.

Para utilizar botones de opción

En el Cuadro de herramientas, arrastre dos controles RadioButton al formulario.

Nosotros para ilustrar el concepto de radiobutton y su practicidad vamos a crear un proyecto bastante didáctico en el cual el usuario podrá elegir la forma de viajar que desee, lo podría hacer en avión en tren, en autobús, o en barco pero no puede viajar en ambas al mismo modo lo que significa que solo puede viajar en una de esas opciones, para ese cometido cuando el usuario elija el tipo de viaje las demás opciones deberán desactivarse automáticamente para no causar problemas al sistema.

Entonces podemos empezar creando nuestro nuevo proyecto en visual basic 2010 y solucionar el ejemplo descrito anteriormente:

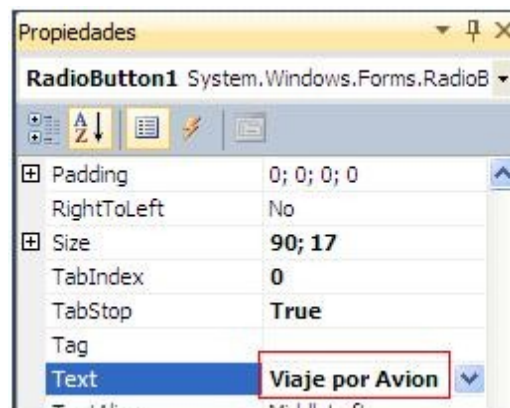


Para diseñar y programar este ejemplo solo vamos a necesitar tres herramientas que son la siguientes un groupbox para que contenga a tres radiobutton y un button

Primero agrega al formulario el groupbox y después dentro de este groupbox1 agrega los tres radiobutton para que puedan ser elegidos, y por ultimo agrega un botón, quedando de esta manera nuestro formulario:



Para cambiar el texto de lectura para el usuario de los radiobutton se logra de la siguiente forma: haz clic en cualquiera de los radiobutton y después ve a sus propiedades que se encuentra en la derecha inferior, y escribe el texto que quieras en la propiedad texto, la siguiente imagen muestra lo descrito:



Ahora otra vez volvamos al diseño del formulario y hacemos doble clic en el botón button1 y con esta acción entramos en el editor de código:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

Ahora escribimos el siguiente código dentro de la función del button1:

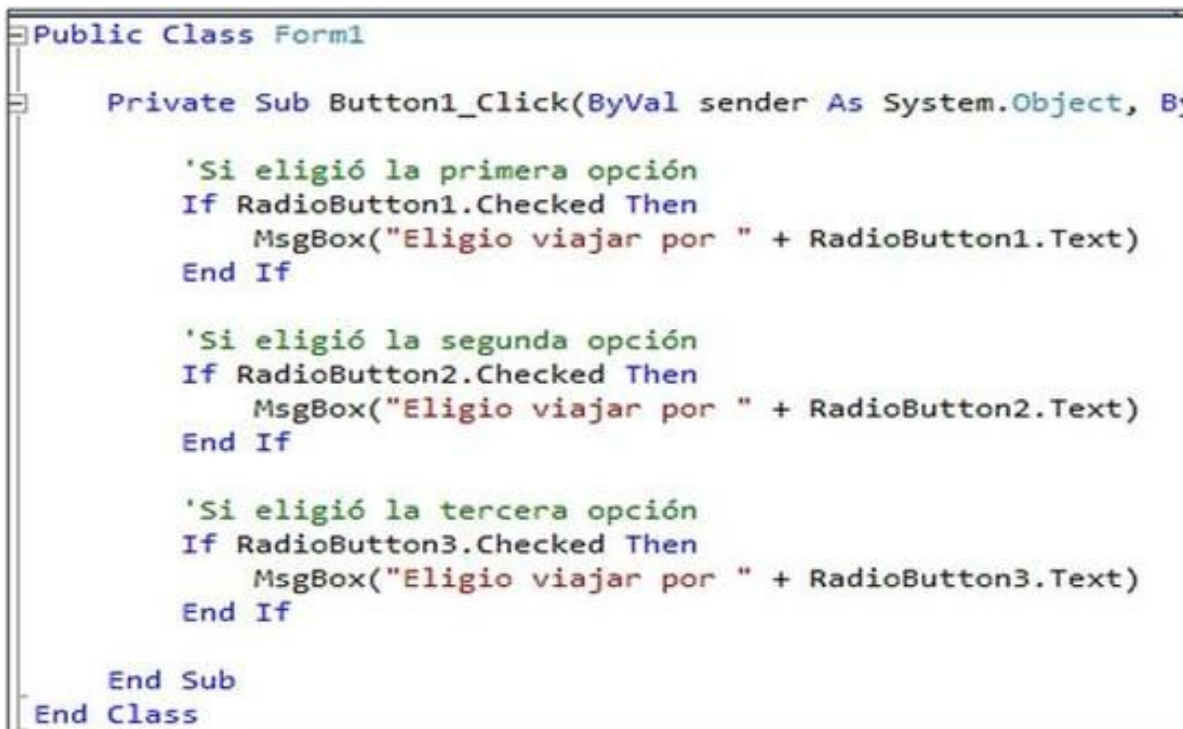
```
'Si eligió la primera opción
If radiobutton1.checked then
Msgbox("eligio viajar por " + radiobutton1.text)
```

Endif

```
'Si eligió la segunda opción  
If radiobutton2.checked then  
Msgbox("eligio viajar por " + radiobutton2.text)  
Endif
```

```
'Si eligió la tercera opción  
If radiobutton3.checked then  
Msgbox("eligio viajar por " + radiobutton3.text)  
Endif
```

El escrito se muestra en la siguiente imagen que es un pantallazo del editor de código de visual basic 2010:



```
Public Class Form1  
  
    Private Sub Button1_Click(ByVal sender As System.Object, By  
  
        'Si eligió la primera opción  
        If RadioButton1.Checked Then  
            MsgBox("Eligio viajar por " + RadioButton1.Text)  
        End If  
  
        'Si eligió la segunda opción  
        If RadioButton2.Checked Then  
            MsgBox("Eligio viajar por " + RadioButton2.Text)  
        End If  
  
        'Si eligió la tercera opción  
        If RadioButton3.Checked Then  
            MsgBox("Eligio viajar por " + RadioButton3.Text)  
        End If  
  
    End Sub  
End Class
```

Guarda y has correr para probar nuestro ejemplo con esto ya tendrás una idea de cómo usar el radiobutton.

### ***PictureBox***

Para mostrar imágenes y a mostrar una imagen como imagen de fondo en un formulario. Se dice que una imagen vale más que mil palabras y, de hecho, muchos programas las utilizan para transmitir información. Hay varias maneras de mostrar imágenes en Visual Basic: la más común es utilizando un control PictureBox.

Los controles PictureBox actúan como un contenedor para las imágenes; se elige la imagen que se va a mostrar estableciendo la propiedad Imagen. La propiedad Imagen se puede establecer en la ventana Propiedades o se puede escribir el código para decirle al programa cuál imagen se va a mostrar.

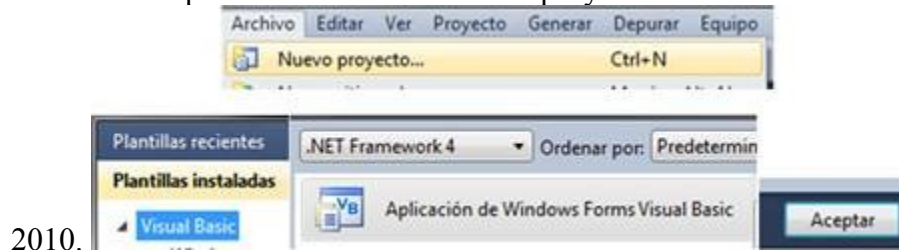
Otras propiedades útiles para el control PictureBox son la propiedad AutoSize, que determina si PictureBox se expandirá para ajustar la imagen, y la propiedad SizeMode, que se puede utilizar para expandir, centrar o ampliar la imagen dentro del control PictureBox. Antes de agregar una imagen a un control PictureBox, generalmente se agregará el archivo de imagen al proyecto como un recurso. Una vez que se agrega un recurso al proyecto, puede volver a utilizarlo cuantas veces lo desee: por ejemplo, se puede mostrar la misma imagen en varios lugares.

### ComboBox

Se usan para seleccionar un ítem u opción entre muchos ítems u opciones, un caso común es cuando nos registramos en una página web y nos dan a elegir un país entre una lista larga de todos los países del mundo.

Enseñaremos a usar las operaciones básicas del COMBOBOX en Visual Basic 2010, con un ejemplo, el cual ejecute las operaciones básicas del COMBOBOX, que son Adicionar ítems, Editar ítems y eliminar ítems.

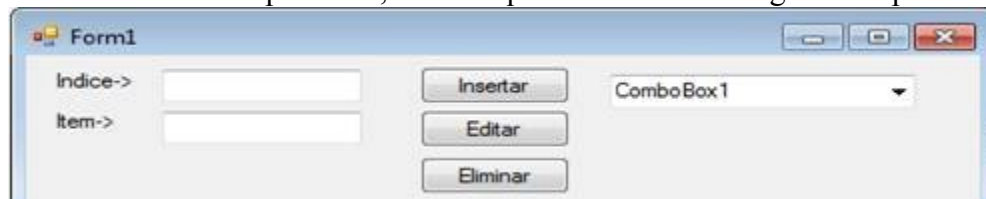
Ahora empecemos creando un nuevo proyecto de Visual Basic



Necesitamos las siguientes herramientas: 2 LABEL, 2 TEXEDIT, 3 BOTONES y un COMBOBOX.



Cuando arrastres estos componentes, trate de que tu formulario tenga este aspecto:



Primero aprenderemos a insertar un nuevo ítem, hacemos doble clic en el botón INSERTAR:



Ahora en medio de la función BUTTON1\_CLICK antes de ENDSUB escribimos el siguiente código que sirve para agregar nuevos ítems a nuestro componente COMBOBOX1



'primero verificamos que en TEXTBOX2 no esté vacío

If TextBox2.Text.Length > 0 Then

    ComboBox1.Items.Add(TextBox2.Text) 'adicionamos el item de COMBOX2

End If

En la siguiente imagen se tiene el código descrito, que es copia del editor de código de Visual Basic 2010:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    'primero verificamos que en TEXTBOX2 no este vacío
    If TextBox1.Text.Length > 0 Then
        ComboBox1.Items.Add(TextBox2.Text) 'adicionamos el item de COMBOX2
    End If
End Sub
```

Para editar un ítem de nuestra herramienta COMBOBOX1, hacemos doble clic en el botón EDITAR, el siguiente código sirve para editar un ítem.

'primero verificamos que TEXTBOX1 y TEXTBOX2 no estén vacíos

If TextBox1.Text.Length > 0 And TextBox2.Text.Length > 0 Then

    ComboBox1.Items.RemoveAt (CInt(TextBox1.Text)) ' eliminamos el ítem de la posición TEXTBOX1

    ComboBox1.Items.Insert (CInt(TextBox1.Text), TextBox2.Text) ' inserta el Nuevo elemento TEXTBOX2

End If

En la siguiente imagen se tiene el código descrito, que es copia de editor de código de Visual Basic 2010:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    'primero verificamos que TEXTBOX1 y TEXTBOX2 no esten vacío
    If TextBox1.Text.Length > 0 And TextBox2.Text.Length > 0 Then
        ComboBox1.Items.RemoveAt(CInt(TextBox1.Text)) ' eliminamos el item de la posición TEXTBOX1
        ComboBox1.Items.Insert(CInt(TextBox1.Text), TextBox2.Text) ' inserta el nuevo elemento TEXTBOX2
    End If
End Sub
```

Ahora volvemos al formulario y esta vez hacemos doble clic en el botón3 que es el botón eliminar:

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
    '
End Sub
```

Para eliminar un ítem de la herramienta COMBOBOX escribimos el siguiente código en la función de nuestro botón Eliminar:

'primero verificamos que TEXTBOX1 no este vacío

If TextBox1.Text.Length > 0 And TextBox2.Text.Length > 0 Then

    ComboBox1.Items.RemoveAt (CInt(TextBox1.Text)) ' eliminamos el ítem de la posición TEXTBOX1

End If

Si lo hemos hecho bien la escritura del código nos abra quedada de la siguiente forma como se muestra en la siguiente imagen:

```

Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
    'primero verificamos que TEXTBOX1 no este vacio
    If TextBox1.Text.Length > 0 And TextBox1.Text.Length > 0 Then
        ComboBox1.Items.RemoveAt(CInt(TextBox1.Text)) ' eliminamos el item de la poscicion TEXTBOX1
    End If
End Sub

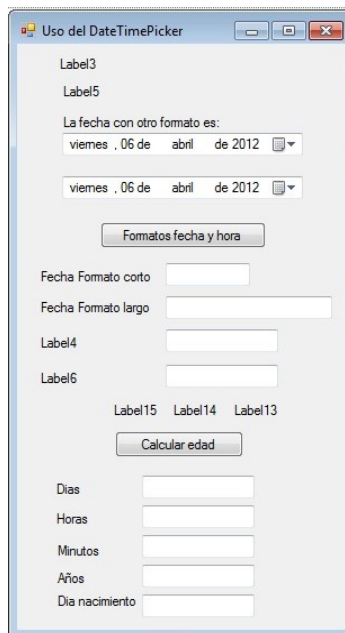
```

Ahora podemos ejecutar la aplicación sin antes guardarlo, con estas tres operaciones básicas de un COMBOBOX que son nuevo ítem, editar ítem y eliminar ítem, ya serás capaz de usar la herramienta COMBOBOX en Visual Studio 2010, y realizar tus propias utilidades.

### ***DateTimePicker***

El control DateTimePicker permite seleccionar una fecha de manera conveniente. Cuando se accede a este objeto, muestra la fecha actual y, mediante flechas de desplazamiento que pertenecen al control, muestra calendarios que pueden recorrerse mes a mes y año a año. Efectuando un clic sobre un número de día, el control se cierra, pudiéndose extraer este dato a través de la propiedad Text o la Propiedad Value, para que se pueda utilizar a posteriori.

ahora un ejemplo en VB cuyo form en vista diseño con labels, DateTimePicker, botones y textbox a continuación, el primer DateTimePicker tiene nombre datetimepicker1 y no está habilitado es solo para mostrar cómo cambiar formato, el segundo DateTimePicker si se utiliza más para cálculos con los botones correspondientes y tiene nombre DTP:

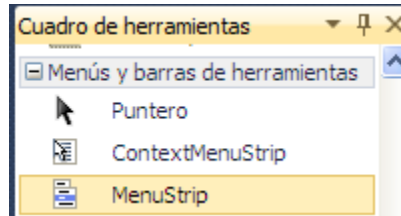


### ***MenuStrip***

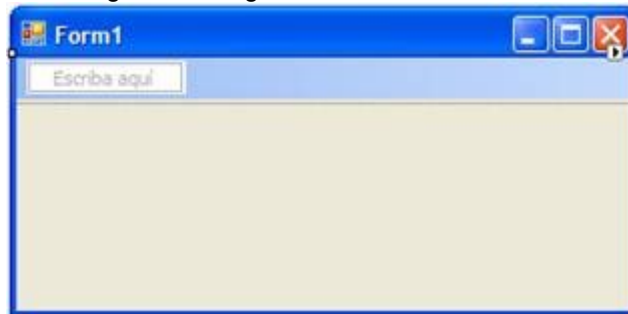
El control MenuStrip representa el contenedor para la estructura de menú de un formulario. Puede agregar objetos ToolStripMenuItem al objeto MenuStrip, que representan los comandos de menú individuales de la estructura de menú. Cada objeto ToolStripMenuItem puede ser un comando de la aplicación o un menú primario para otros elementos de submenú.

Aunque el control MenuStrip reemplaza y agrega funcionalidad al control MainMenu de versiones anteriores, se conserva MainMenu a efectos de compatibilidad con versiones anteriores y uso futuro, en su caso.

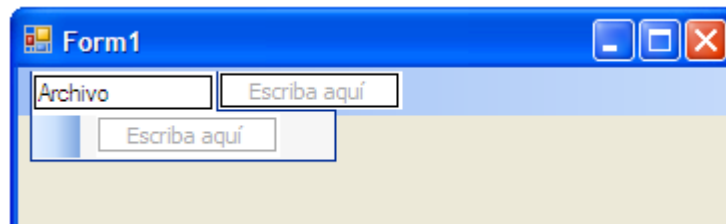
Entonces podemos empezar creando nuestro primer proyecto en VISUAL BASIC 2010, y a nuestro formulario principal le agregamos un Menu Strip de la caja de herramientas, que se ve en la siguiente imagen:



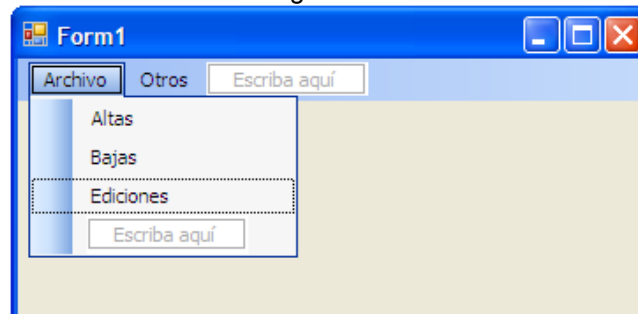
Cuando acoplemos esta herramienta a nuestro formulario principal se estacionara en la parte de arriba como se muestra en la siguiente imagen:



Donde dice escriba aquí puedes escribir el texto que te plazca en nuestro caso escribiremos ARCHIVO y veras que se expande hacia abajo y hacia la derecha para poder escribir más opciones:



Si escribes en la parte de abajo este se mostrara cuando hagas clic en el menú ARCHIVO en cambio el menú de la derecha de Archivo será otro menú independiente igual que ARCHIVO, nosotros dejaremos nuestro formulario de la siguiente manera:

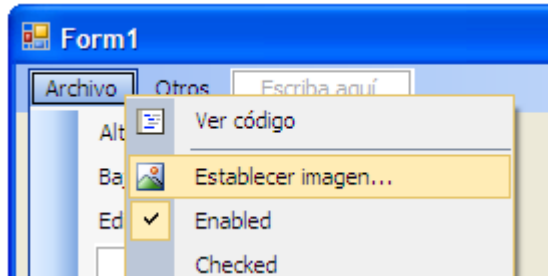




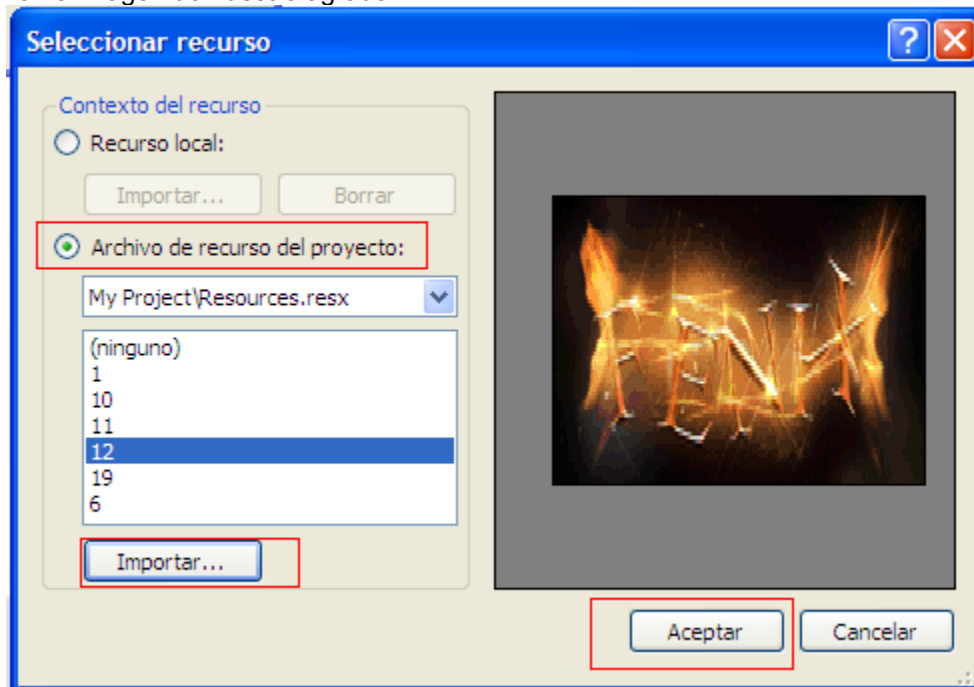
Ahora ponerle imágenes es bastante fácil podemos poner imágenes a cualquiera de las opciones ya sea en ARCHIVO, OTROS, ALTAS, BAJAS O EDICIONES; las imágenes pueden ser de tipo .GIF, .JPG, .PNG, .JPEG, la dimensión varia depende del uso.

Por ejemplo si quieres poner imagen más texto entonces sería óptimo que usaras imágenes con dimensiones de 64x64 para que se dibujen en la parte izquierda del texto en la franja azul como si fueren iconos.

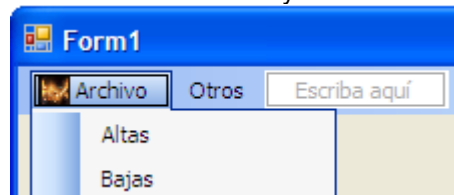
Vamos a poner entonces una imagen a la opción ARCHIVO para eso le hacemos clic derecho y luego ESTABLECER IMAGEN como en la siguiente figura:



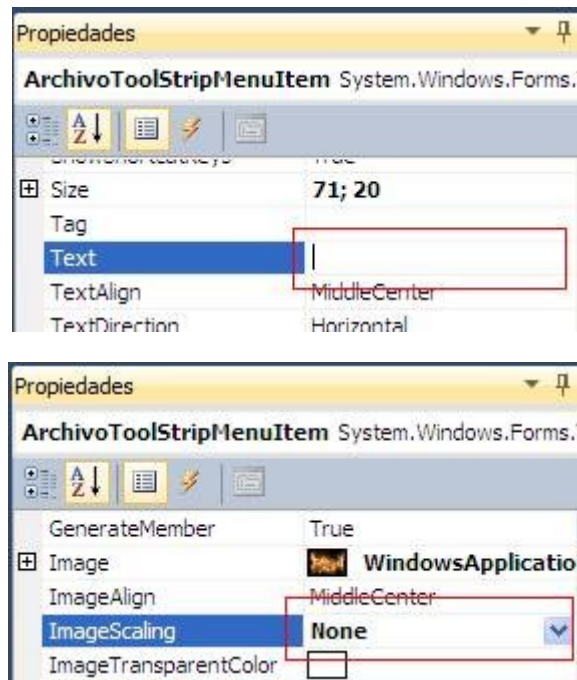
Después nos saldrá una ventana en la cual le hacemos clic en el botón IMPORTAR para seleccionar la imagen de nuestro agrado:



En nuestro caso los numero 1, 10, 11, 12, 19,6; son imágenes que importamos. Después de escoger nuestra imagen hacemos clic en ACEPTAR y nuestro Menu Strip quedara de esta forma:



Hacemos lo mismo para las demás opciones, ahora si solo queremos que se vean imágenes en nuestro menú y nada de texto hacemos lo siguiente hacemos clic en ARCHIVO nos vamos a sus propiedades borramos lo que está escrito en la propiedad TEXTO y en la propiedad IMAGESCALING escogemos NONE:



Y listo veras que solo se muestran imágenes en tu menú, para eso tus imágenes deben tener un tamaño de acuerdo a tus necesidades yo uso de 65alto X 100ancho, espero que te haya servido.

## MsgBox

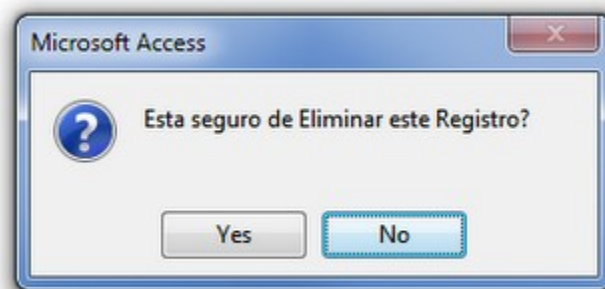
Los MsgBox son conocidos como caja de mensajes y se utilizan por lo general para dar mensajes a los usuarios a la hora de ocurrir un evento o para informarle de que ocurrió una determinada acción. La forma general de la función de un MsgBox es la siguiente:

```
respuesta=MsgBox("Texto para el usuario",tipos de botones+boton por defecto+tipos de iconos+,"Titulo")
```

ejemplo:

```
MsgBox("Esta seguro de Eliminar este Registro?",4+256+32)=7
```

es el equivalente a:



Como se puede apreciar es un MsgBox con algunos valores definidos: Mensaje, Icono, Tipo de botones + boton por defecto "que en este caso es el boton de [NO]"

para construir estos msgbox, basta con convinar (a conveniencia) los siguientes valores presentados:

#### VALORES DE RETORNO

| Valor de retorno | Constante simbólica |
|------------------|---------------------|
| 1                | vbOK                |
| 2                | vbCancel            |
| 3                | vbAbort             |
| 4                | vbRetry             |
| 5                | vbIgnore            |
| 6                | vbYes               |
| 7                | vbNo                |





#### TIPOS DE BOTONES

| Valor tipos Botones | Constante simbólica |
|---------------------|---------------------|
| 0                   | vbOKOnly            |
| 1                   | vbOKCancel          |
| 2                   | vbAbortRetryIgnore  |
| 3                   | vbYesNoCancel       |
| 4                   | vbYesNo             |
| 5                   | vbRetryCancel       |

#### LOS BOTONES POR DEFECTO

Solo son 3 valores  
 0 "para el primer boton"  
 256 "para el segundo boton"  
 512 "para el tercer boton"

#### TIPOS DE ICONOS

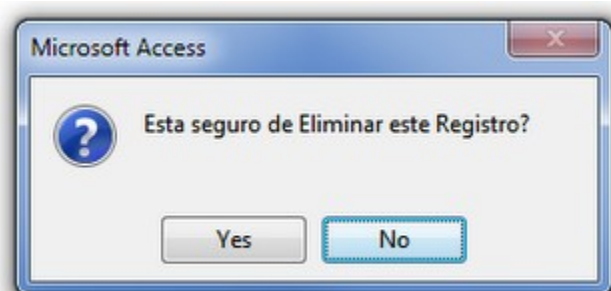
| Tabla 9 - Constantes del argumento icono  |               |       |
|---|---------------|-------|
| Icono   | Constante     | Valor |
|  | vbCritical    | 16    |
|  | vbQuestion    | 32    |
|  | vbExclamation | 48    |
|  | vbInformation | 64    |

Usando Ya estos valores dentro de un macro de Access no serviría para ejecutar o NO ejecutar el contenido de un macro, como muestra el siguiente ejemplo

```

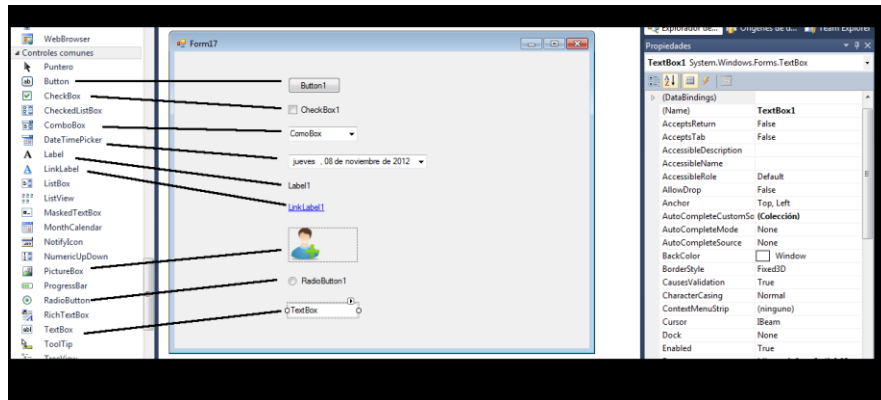
If MsgBox("Esta seguro de Eliminar este Registro?",4+256+32)=7 Then
    StopMacro
End If
  
```

que daría este resultado



Aquí se cumple la condición de la macro si se oprime el botón [NO] que nos da el valor de retorno 7 y así la condición [IF] dentro de la macro, nos daría como resultado VERDADERO y su acción a ejecutar es detener la ejecución de la macro con la acción STOPMACRO.

### *Iconos de las Herramientas*



## 4-Control de Eventos

### Manejador de Eventos

Un Manejador de Eventos contiene código que responde a eventos particulares. Un desarrollador diseña cuidadosamente sus aplicaciones determinando los controles disponibles para el usuario y los eventos apropiados asociados a estos controles, entonces, el desarrollador escribe el código para integrar los eventos consistentes con el diseño de la aplicación.

Procedimientos

Un procedimiento es un conjunto de sentencias que realizan una acción lógica. Existen tres tipos de procedimientos en Visual Basic .NET:

- Event procedures/Event handler, procedimiento que contiene código que es ejecutado en respuesta a un evento. Cuando el evento es disparado el código dentro del manejador de eventos es ejecutado.

Visual Basic .NET para los manejadores de eventos utiliza una convención estándar la cual combina el nombre del objeto seguido de un guión bajo y el nombre del evento.

Private|Public Sub objeto\_Evento(parámetros) handles Objeto.Evento

Sentencias

End Sub

Cada manejador de eventos provee dos parámetros, el primer parámetro llamado sender provee una referencia al objeto que dispara el evento, el segundo parámetro es un objeto cuyo tipo de dato depende del evento que es manejado. Ambos parámetros son pasados por valor.

Si un parámetro es declarado por referencia ByRef el parámetro apunta al argumento actual. Por default los argumentos se pasan por valor ByVal el parámetro es una copia local del argumento.

- Sub procedures, contiene código que el desarrollador crea para realizar una acción lógica.
- Function procedures, contiene código que el desarrollador crea para realizar una acción lógica y regresa un valor, el valor que una función envía de regreso al programa que lo invoca es llamado valor de regreso. Para regresar un valor se utiliza la sentencia Return.

### Eventos más Comunes.

**Click** Ocurre cuando el usuario presiona y suelta un botón del mouse sobre un objeto.

**DbClick** Ocurre cuando el usuario presiona y suelta dos veces un botón del mouse sobre un objeto.

**DragDrop** Ocurre como resultado de arrastrar y soltar con el mouse un control sobre un determinado tipo de objeto.

**DragOver** Ocurre cuando una operación de arrastrar y colocar está en curso. Puede usar este evento para controlar el puntero del mouse a medida que entra, sale o descansa directamente sobre un destino válido.

**GotFocus** Ocurre cuando un objeto recibe el { CONTROL Internet.HHCtrl.1 } { HYPERLINK "JavaScript:alink\_4.Click()" }, ya sea mediante una acción del usuario, como tabular o hacer clic en el objeto, o cambiando el enfoque en el código mediante el método SetFocus.

**LostFocus** A diferencia del evento anterior, este evento ocurre cuando el objeto pierde el enfoque, ya sea mediante tabulaciones o hacer clic sobre otro objeto.

**KeyDown** Ocurre cuando el usuario mantiene presionada una tecla.

**KeyUp** Ocurre cuando el usuario termina la operación de pulsar una tecla. Se podría decir, que este evento ocurre precisamente al terminar el evento KeyDown.

**KeyPress** Ocurre como resultado de presionar y soltar una tecla.

**MouseDown** Ocurre cuando el usuario presiona un botón del mouse, pero a diferencia del evento

**MouseDown**, permite identificar cuáles de los tres botones del mouse fue presionado y las combinaciones de tecla ALT, MAYÚS y CTRL.

**MouseUp** El evento MouseUp se produce cuando el usuario suelta el botón del mouse.

**MouseUp** es un compañero útil a los eventos MouseDown y MouseMove.

**MouseMove** Este evento ocurre mientras el usuario mueve o desplaza el puntero del mouse sobre un objeto.

**Métodos más Comunes.**

**Drag** Inicia, termina o cancela una operación de arrastre de cualquier control, excepto los controles Line, Menu, Shape, Timer o CommonDialog.

**Move** Se utiliza para mover un control o formulario, especificando sus coordenadas (Top, Left) y su tamaño (Width, Height).

**Refresh** Se utiliza para dibujar o actualizar gráficamente un control o un formulario. Se utiliza principalmente con los controles FileListBox y Data.

**SetFocus** Este método se utiliza para hacer que un objeto reciba el enfoque. Este método es uno de los más usados para los controles de Visual Basic 6.0.

**ShowWhatsThis** Permite mostrar un tema seleccionado de un archivo de Ayuda utilizando el menú emergente ¿Qué es esto? que ofrece la ayuda de Windows. Este método es muy útil para proporcionar ayuda interactiva en un menú contextual acerca de un objeto en una aplicación. Este método muestra el tema indicado por la propiedad WhatsThisHelpID del objeto especificado en la sintaxis.

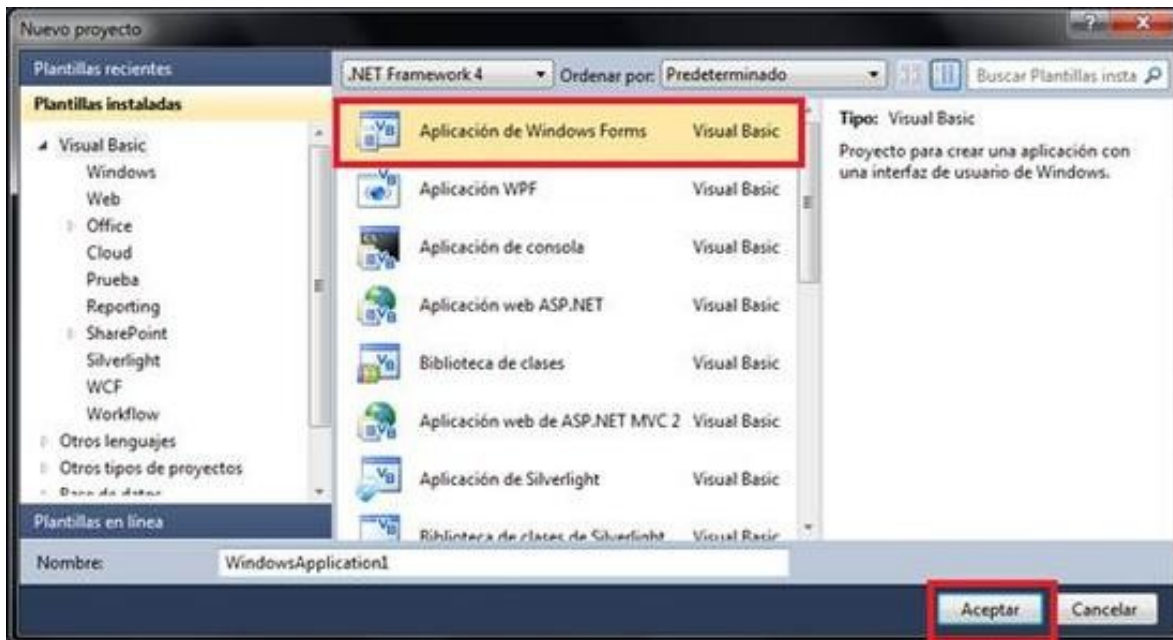
**Zorder** Se utiliza para que un control o un objeto formulario se coloque por encima o por debajo de otros objetos.

## 5-Manejo de DataRepeater y DataView

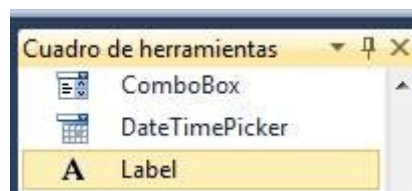
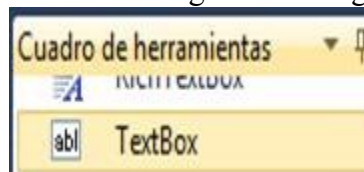
### DataView

Haremos un pequeño proyecto donde usaremos una matriz de números y en la cual los datos serán llenados por el usuario, no permitiremos que se puedan escribir letras solo números en los textbox, y tampoco dejaremos que se introduzcan datos vacíos.

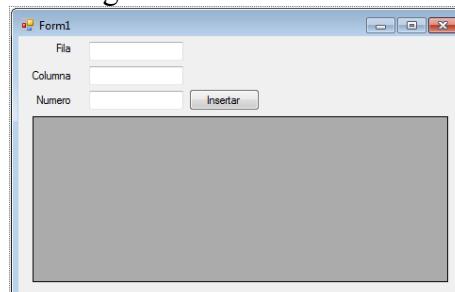
Para empezar abrimos nuestro VISUAL STUDIOS 2010 y creamos un nuevo proyecto Ahora nos saldrá una ventana en la cual escogemos el proyecto de Aplicación de Windows Forms VISUAL BASIC y hacemos clic en el botón aceptar como se muestra en la siguiente imagen:



Ahora en el formulario arrastramos un DATAGRIDVIEW tres TEXTBOX tres LABES y un botón, estos componentes se muestra en las siguientes imágenes:



Dejando nuestro formulario de la siguiente manera:



Ahora hacemos clic en el centro de DATAGRIDVIEW y la esquina derecha superior nos saldrá una flechita en la cual le hacemos clic y nos saldrá el siguiente menú:





Como en la imagen anterior le hacemos clic en AGREGAR COLUMNAS para agregar las siguientes columnas

Y hacemos clic 5 veces en el botón AGREGAR para que nos agregue 5 columnas y después hacemos clic en el botón CERRAR.

Y nuestro formulario nos quedara así:

Ahora hacemos doble clic en el botón INSERTAR y nos abrirá el editor de código que se mostrara de la siguiente manera:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    End Sub
End Class
```

Ahora escribimos el siguiente código: para poner un número entero en nuestro DATAGRID indicando la fila y la columna:

```
DataGridView1.RowCount = 5 'aumentamos el tamaño de las filas
Dim fil As Integer
Dim col As Integer
Dim num As Integer
'verificamos que los TEXTBOX no estén vacíos y todos sean números
If TextBox1.Text.Length > 0 And TextBox2.Text.Length > 0 And TextBox3.Text.Length > 0 Then
    If IsNumeric(TextBox1.Text) And IsNumeric(TextBox2.Text) And IsNumeric(TextBox3.Text) Then
        fil = CStr(TextBox1.Text) 'asignamos la fila
        col = CStr(TextBox2.Text) 'asignamos la columna
        num = CStr(TextBox3.Text) 'asignamos el numero a cargar
```

```

Iffil < 5 And col < 5 Then
DataGridView1.Item(col, fil).Value = TextBox3.Text
EndIf
EndIf
EndIf

```

Si lo hacemos bien tiene que quedarnos como se muestra en la siguiente imagen:

```

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        DataGridView1.RowCount = 5 'aumentamos el tamaño de las filas
        Dim fil As Integer
        Dim col As Integer
        Dim num As Integer

        If TextBox1.Text.Length > 0 And TextBox2.Text.Length > 0 And TextBox3.Text.Length > 0 Then
            If IsNumeric(TextBox1.Text) And IsNumeric(TextBox2.Text) And IsNumeric(TextBox3.Text) Then
                fil = CStr(TextBox1.Text) 'asignamos la fila
                col = CStr(TextBox2.Text) 'asignamos la columna
                num = CStr(TextBox3.Text) 'asignamos el numero a cargar
                If fil < 5 And col < 5 Then
                    DataGridView1.Item(col, fil).Value = TextBox3.Text
                End If
            End If
        End If
    End Sub
End Class

```

Ahora guardamos el proyecto y al ejecutar nos daremos cuenta que solo nos dejara ingresar números y que los TEXTBOX no estén vacíos, si es de esta manera no se asignara ningún valor a nuestro DATAGRIDVIEW

Puede ajustar la apariencia de la parte exterior e interior del control DataRepeater como desee.

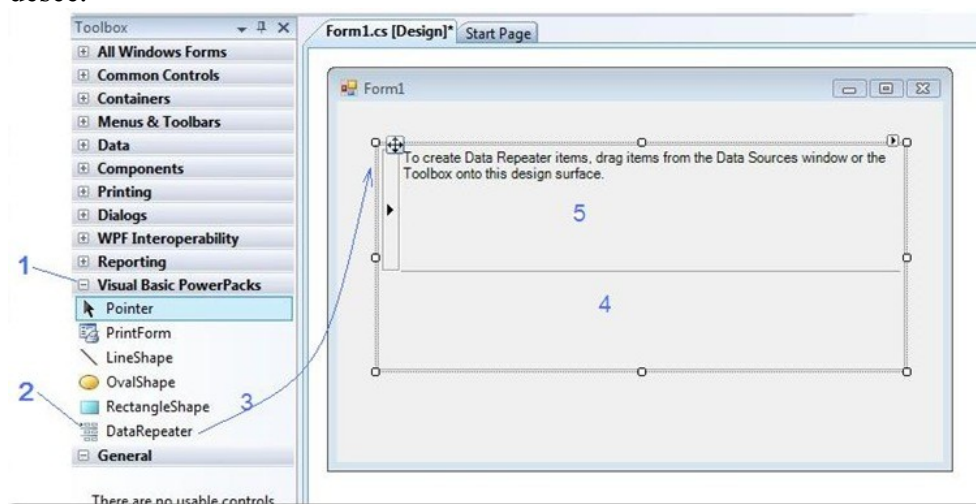


Foto 2. Arrastrar y soltar DataRepeater al Formulario

Puedo elegir mi DataRepeater haciendo clic en la sección exterior de la DataRepeater, el Browsable propiedad.

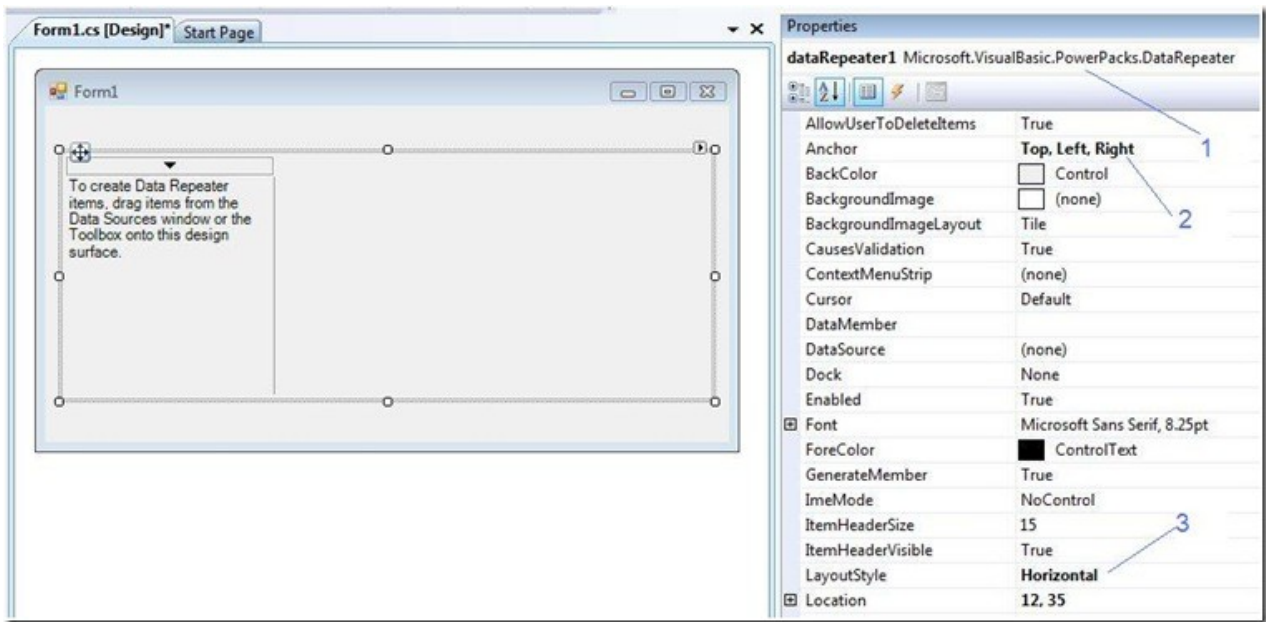
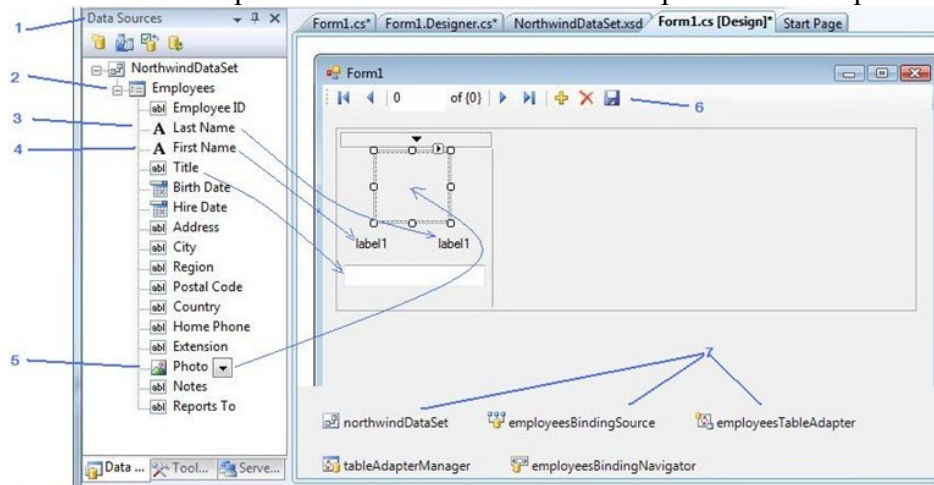


Imagen 3. DataRepeater diseño

Ahora, vamos a llenar el elemento DataRepeater sección de la plantilla con algunos controles de datos enlazados. Puedo mostrar la ventana Orígenes de datos ,cambiar la vista de la tabla de empleados al detalle , cambiar el nombre y apellido de etiquetar , deje el título por defecto TextBox , y cambiar la foto en un cuadro de imagen . Luego arrastrar y colocar estos elementos a la sección DataRepeater plantilla de elementos, como se indica en la figura y retire las etiquetas adicionales innecesarios. Tenga en cuenta que también hará que la creación automática de la barra de navegación (Fig. 4-6), conjunto de datos, BindingSource, TableAdapter, etc. La experiencia es exactamente la misma que cuando se arrastra y coloca elementos de detalle vista desde la ventana Orígenes de datos a un formulario o un control contenedor. La diferencia es que ahora los elementos de datos enlazados en la sección de plantilla de elementos ahora se repetirán en tiempo de ejecución.



Arrastrar y soltar elementos Vista detallada de ventana Orígenes de datos a la sección de plantilla de elementos

Puede desplazarse por los datos haciendo clic en la barra de desplazamiento o el botón hacia adelante y hacia atrás en la barra de navegación. Puede seleccionar un elemento, y modificar el valor de cualquier control modificable sobre el tema como la caja de texto Título. Puede agregar un nuevo ítem, eliminar un elemento. Además, podrás guardar todos los cambios (Añadir / Eliminar / actualizar) con el botón de guardar.



## 6- métodos de impresión



Pese a que vivimos en una era digital y todo (bueno, casi todo) se hace por internet y con la computadora, aún hay momentos en que el papel es necesario

### Control PrintDocument

Este control se debe agregar a cualquier proyecto en el que se quiera imprimir algo. Haz de cuenta que es la hoja en blanco donde puedes dibujar diferentes objetos, escribir texto y colocar imágenes de mapa de bits. Cuando acabes de dibujar, escribir, etc., se llama el método `Print` para que la impresora escupa la hoja. Acepta todos los métodos de dibujo que usa el objeto, es decir que si tienes que escribir texto, usas el método `DrawString`; si quieres dibujar un marco alrededor del texto, puedes usar `DrawLine` o `DrawRectangle`. Aunque el control es invisible (no aparece en la ventana ni hay forma de ir viendo conforme se va dibujando), es muy útil.

## Control PrintDialog

Este control hace aparecer la ventana default que te permite elegir y configurar la impresora que se desea emplear. Si no se usa, la impresión se hace en la impresora que se tiene definida como default con las propiedades que trae por default (o sea con lo default de lo default ). Para que aparezca, se debe usar el método `ShowDialog`.

## Control PageSetupDialog

Este control hace aparecer la ventana que te permite configurar el papel a usar: tamaño, orientación, márgenes, etc. Se debe usar su método `ShowDialog` para que aparezca, igual que con el control `PrintDialog`.

## Control PrintPreviewDialog

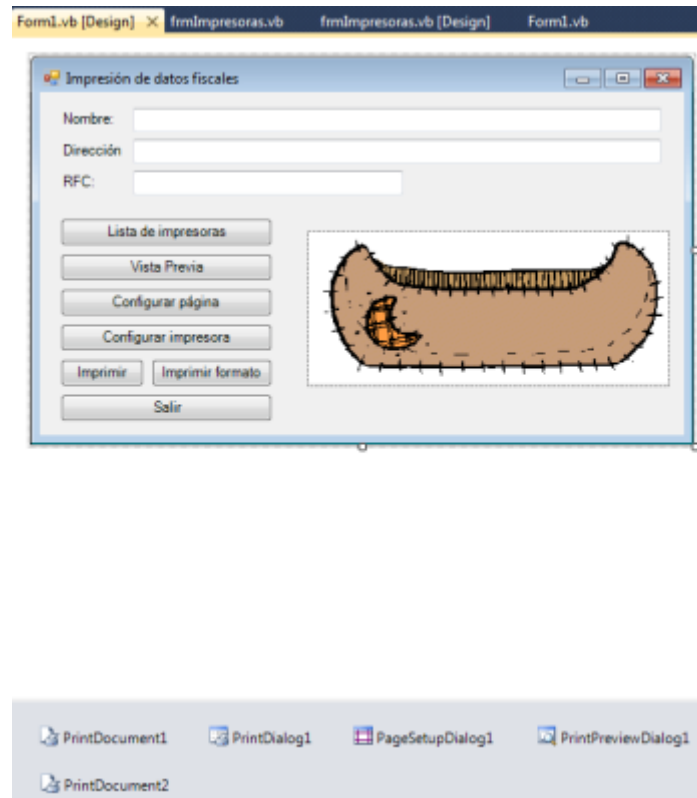
Este control hace aparecer una vista previa de lo que se va a imprimir. Como mostraré más adelante, se debe vincular este control con lo el control `PrintDocument` para que todo lo que se va a imprimir, en lugar de ir a la impresora, aparezca en esta ventana. También hay que tener cuidado: si no se tiene acceso a la impresora que está seleccionada (por ejemplo, una impresora de la red a la cual no tengo acceso en este momento o la que tengo conectada está apagada), no aparece la ventana y te marca error. Es necesario recibir este error (con un `Try...Catch`) para que no termine abruptamente el programa solo porque no halla la impresora (cosa que no hago en este post, por cierto).

## Ejemplo de su uso

Es un ejemplo muy simple que solo muestra cómo se usan los controles. Sin embargo, creo que con estas bases ya le pueden remplazar con otros contextos.



Antes que nada, les muestro el diseño de la ventana principal. Consta de 3 etiquetas, 3 cajas de texto (txtNombre, txtDir y txtRFC), un PictureBox (picCanoa) que sirve para mostrar como imprimir una imagen de mapa de bits, 7 botones cuyo código explicaré más adelante, 2 controles PrintDocument, un PrintDialog, un PageSetup y un PrintPreviewDialog. Estos 5 últimos no aparecen en la ventana: cuando los agregas a la ventana aparecen en la parte inferior.



Para que un PrintDocument funcione, es necesario editar su evento PrintPage. Si le das doble clic al icono PrintDocument1, verás este código:

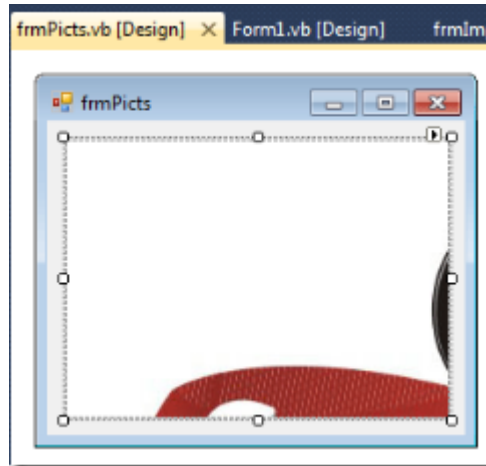
```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles Print
    e.Graphics.DrawString(txtNombre.Text, txtNombre.Font, Brushes.Black, 400, 50)
    e.Graphics.DrawString(txtDir.Text, txtDir.Font, Brushes.Black, 400, 65)
    e.Graphics.DrawString(txtRFC.Text, txtRFC.Font, Brushes.Black, 400, 80)
    e.Graphics.DrawRectangle(Pens.Black, 390, 40, 200, 90)
    e.Graphics.DrawImage(picCanoa.Image, 50, 50)
End Sub
```

Básicamente uso el parámetro e, que es “la hoja invisible” sobre la que voy a dibujar, para escribir 3 cadenas, un rectángulo y la imagen de la canoa. Como mencioné con anterioridad, son las mismas funciones que se usan para dibujar gráficos sobre la ventana .

Una vez que tienes este evento listo, puedes mandar imprimir con la función `Print`. Yo lo hago con el botón cuya etiqueta dice *Imprimir* y este es el código:

```
Private Sub btnImprime_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnImprime.Click
    PrintDocument1.Print()
End Sub
```

El `PrintDocument2` muestra una idea para imprimir formatos. Dibujé el esqueleto de un formato, lo guardé como PNG, lo importé a Visual Basic y lo puse en un `PictureBox` (llamado `PictureBox1`) en la ventana `frmPicts`. Aquí está la ventana `frmPicts` y abajo el código que lo usa (primero el botón que tiene la etiqueta *Imprimir formato*, `btnImprime2`, y luego `PrintPage`).



```
Private Sub btnImprime2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnImprime2.Click
    PrintDocument2.Print()
End Sub

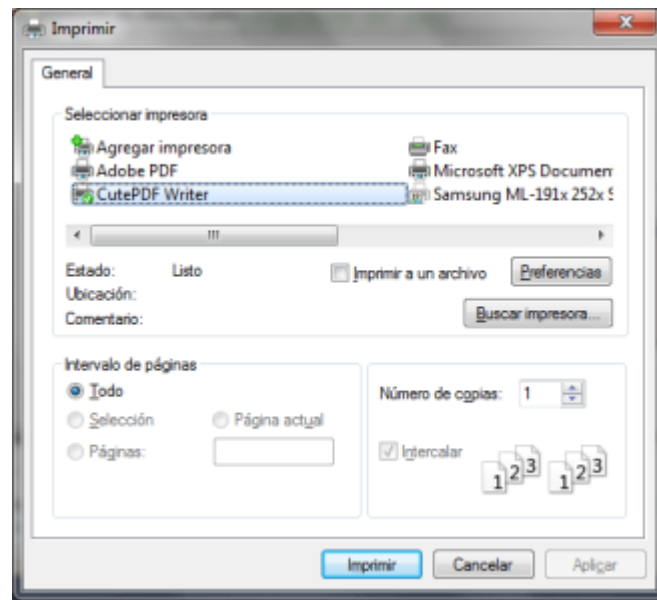
Private Sub PrintDocument2_PrintPage(ByVal sender As System.Object, ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles Print
    Dim fuente As New Font("Verdana", 12)
    e.Graphics.DrawImage(frmPicts.PictureBox1.Image, 0, 0)
    e.Graphics.DrawString("Cantidad", fuente, Brushes.Black, 82, 234)
End Sub
```

Configurar la impresora es muy simple usando el control `PrintDialog`. Este es el código del botón con la etiqueta *Configurar impresora*. Fíjate en la manera en que copio las propiedades puestas por el usuario en este cuadro de diálogo en los dos controles `PrintDocument` (en las 2 últimas instrucciones):



```
Private Sub btnConfigImpr_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnConfigImpr.Click
    ' Mostrar cuadro de diálogo de impresión
    PrintDialog1.ShowDialog()
    PrintDocument1.DefaultPageSettings.PrinterSettings = PrintDialog1.PrinterSettings
    PrintDocument2.DefaultPageSettings.PrinterSettings = PrintDialog1.PrinterSettings
End Sub
```

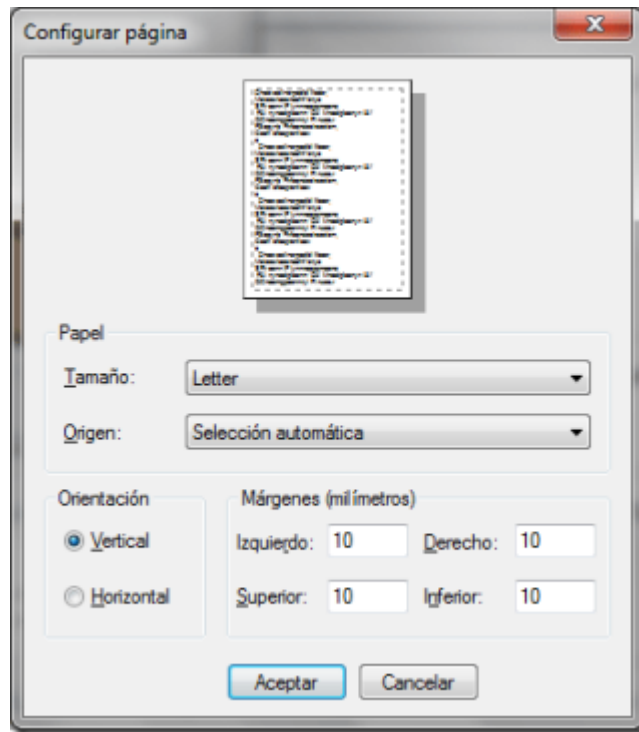
A la hora de ejecutarse, el control `PrintDialog` hace que aparezca esta ventana:



También es sencillo modificar la configuración de la página. Este es el código del botón etiquetado *Configurar página*:

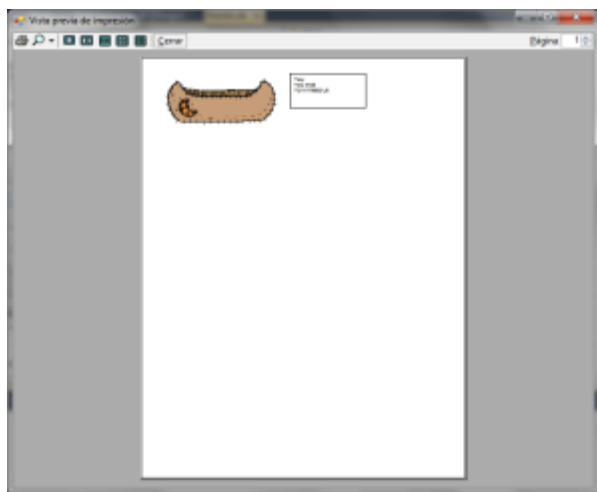
```
Private Sub btnConfPag_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnConfPag.Click
    ' Mostrar cuadro de diálogo de configuración del papel
    PageSetupDialog1.PageSettings = PrintDocument1.DefaultPageSettings
    PageSetupDialog1.ShowDialog()
    PrintDocument1.DefaultPageSettings = PageSetupDialog1.PageSettings
    PrintDocument2.DefaultPageSettings = PageSetupDialog1.PageSettings
End Sub
```

Antes del `ShowDialog` es necesario igualar sus propiedades a los que ya tiene `unPrintDocument` (en las dos instrucciones que siguen al comentario). Al correr este evento, aparece esta ventana:

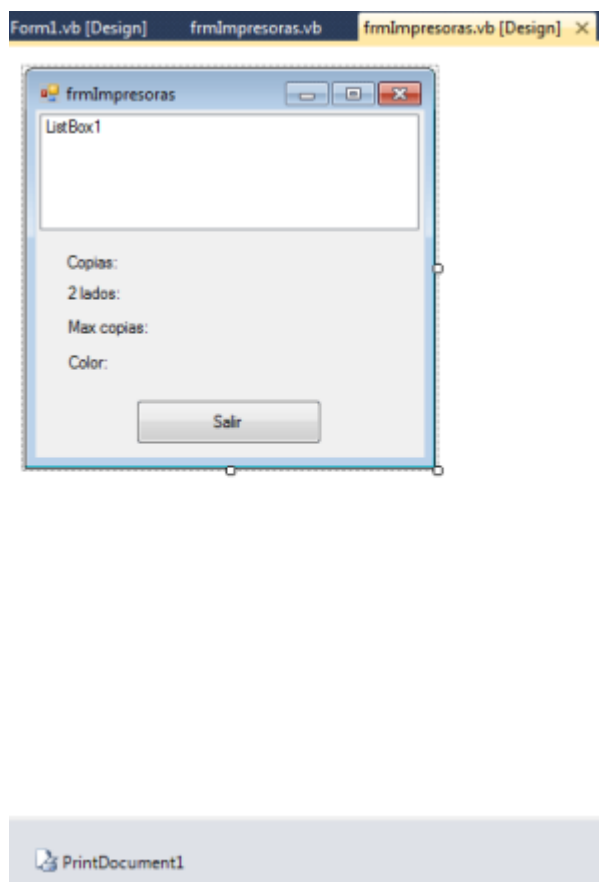


Una vista previa (o presentación preeliminar) de lo que se va a imprimir suele ser bastante útil para muchos usuarios. Hacer que aparezca es sencillísimo. Lo único que hay que hacer es hacer que, en lugar de que el evento `PrintPage` de un `PrintDocument` lo mande a la impresora, se le envía a esta caja de diálogo. Abajo pongo el código y luego la ventana que aparece que muestra la vista preeliminar:

```
Private Sub btnVistaPrev_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnVistaPrev.Click
    PrintPreviewDialog1.Document = PrintDocument1
    PrintPreviewDialog1.ShowDialog()
End Sub
```



También es posible jugar con la configuración de la impresora. Para leer algunas propiedades de las impresoras, hice el botón etiquetado *Lista de impresoras* que abre la ventana `frmImpresoras` desde donde se pueden ver la lista de impresoras instaladas y al seleccionar una de ellas, se ven algunas de sus propiedades. Esta es la ventana `frmImpresoras` en la vista diseño:



Tiene una lista donde voy a escribir la lista de impresoras (ListBox1), etiquetas que muestran los valores de las copias, si es duplex (si puede imprimir por ambas caras de la hoja), el número máximo de copias permitidas y si puede recibir documentos a color (lblCopias, lblDuplex, lblCopiasMax y lblColor, respectivamente). También es necesario un PrintDocument para poder ver estas propiedades. Pero antes que nada, veamos el código con que llena ListBox1:

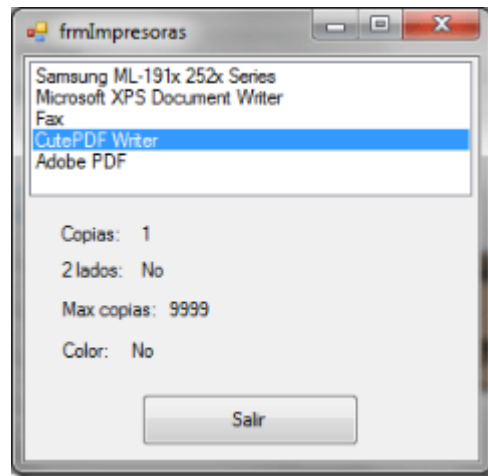
```
Private Sub frmImpresoras_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim i As Integer
    With Printing.PrinterSettings.InstalledPrinters
        For i = 0 To .Count
            ListBox1.Items.Add(.Item(i))
        Next
    End With
End Sub
```

Nótese que estoy haciendo uso de la colección `Printing.PrinterSettings.InstalledPrinters` misma que contiene la lista de todas las impresoras instaladas en el equipo donde se está ejecutando. Estoy usando la instrucción `With` para abreviar un poco y no escribir este rollo a cada rato.

No necesito escribir un evento `PrintPage` para el `PrintDocument1` porque no voy a imprimir nada: solo lo necesito para poder mostrar las propiedades. Por esta razón el otro código interesante es el que se ejecuta cuando `ListBox1` cambia de elemento seleccionado:

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
    Dim i As Integer
    i = ListBox1.SelectedIndex
    With PrintDocument1.PrinterSettings
        .PrinterName = Printing.PrinterSettings.InstalledPrinters(i)
        lblCopias.Text = .Copies
        If .CanDuplex Then
            lblDuplex.Text = "Si"
        Else
            lblDuplex.Text = "No"
        End If
        lblCopiasMax.Text = .MaximumCopies
        If .SupportsColor Then
            lblColor.Text = "Si"
        Else
            lblColor.Text = "No"
        End If
    End With
End Sub
```

Este es un ejemplo de como se ve esta ventana con una impresora seleccionada:



## 7-Conceptos de Bases de Datos

### *Definición*

Es un conjunto de información relacionada que se encuentra agrupada o estructurada. Un archivo por sí mismo no constituye una base de datos, sino más bien la forma en que está organizada la información es la que da origen a la base de datos. Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos.

### *Arquitectura Cliente/Servidor*

Los sistemas cliente/servidor están contruidos de tal modo que la base de datos puede residir en un equipo central, llamado servidor y ser compartida entre varios usuarios. Los usuarios tienen acceso al servidor a través de una aplicación de cliente o de servidor: En los sistemas cliente/servidor grandes, miles de usuarios pueden estar conectados con una instalación de SQL Server al mismo tiempo. SQL Server tiene una protección completa para dichos entornos, con barreras de seguridad que impiden problemas como tener varios usuarios intentando actualizar el mismo elemento de datos a la vez. SQL Server también asigna eficazmente los recursos disponibles entre los distintos usuarios, como la memoria, el ancho de banda de la red y la E/S de disco.

### *Sistema de gestión de base de datos (SGBD)*

Los sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre los datos, el usuario y las aplicaciones que la utilizan. El propósito general de los sistemas de gestión de base de datos es el de manejar de forma clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante, para un buen manejo de datos.

Existen distintos objetivos que deben cumplir los SGBD:

**Abstracción de la información.** Los SGBD ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario.

**Independencia.** La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.

**Consistencia.** En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea. En los SGBD existen herramientas que facilitan la programación de este tipo de condiciones.

**Seguridad.** La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura frente a usuarios malintencionados, que intenten leer información privilegiada; frente a ataques que deseen manipular o destruir la información; o simplemente ante las torpezas de algún usuario autorizado pero descuidado. Normalmente, los SGBD disponen de un complejo sistema de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.

**Integridad.** Se trata de adoptar las medidas necesarias para garantizar la validez de los datos almacenados. Es decir, se trata de proteger los datos ante fallos de hardware, datos introducidos por usuarios descuidados, o cualquier otra circunstancia capaz de corromper la información almacenada. Los SGBD proveen mecanismos para garantizar la recuperación de la base de datos hasta un estado consistente conocido en forma automática.

**Respaldo.** Los SGBD deben proporcionar una forma eficiente de realizar copias de respaldo de la información almacenada en ellos, y de restaurar a partir de estas copias los datos que se hayan podido perder.

**Control de la concurrencia.** En la mayoría de entornos, lo más habitual es que sean muchas las personas que acceden a una base de datos, bien para recuperar información, bien para almacenarla. Y es también frecuente que dichos accesos se

realicen de forma simultánea. Así pues, un SGBD debe controlar este acceso concurrente a la información, que podría derivar en inconsistencias.

**Manejo de Transacciones.** Una Transacción es un programa que se ejecuta como una sola operación. Esto quiere decir que el estado luego de una ejecución en la que se produce una falla es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.

**Tiempo de respuesta.** Lógicamente, es deseable minimizar el tiempo que el SGBD tarda en darnos la información solicitada y en almacenar los cambios realizados.

#### **Las ventajas de su uso son:**

Proveen facilidades para la manipulación de grandes volúmenes de datos. Simplificando la programación de chequeos de consistencia, manejando las políticas de respaldo adecuadas que garantizan que los cambios de la base serán siempre consistentes sin importar si hay errores en el disco, o hay muchos usuarios accediendo simultáneamente a los mismos datos, o se ejecutaron programas que no terminaron su trabajo correctamente, etc., permitiendo realizar modificaciones en la organización de los datos con un impacto mínimo en el código de los programas y permitiendo implementar un manejo centralizado de la seguridad de la información (acceso a usuarios autorizados), protección de información, de modificaciones, inclusiones, consulta.

Las facilidades anteriores bajan drásticamente los tiempos de desarrollo y aumentan la calidad del sistema desarrollado si son bien explotados por los desarrolladores.

Usualmente, proveen interfases y lenguajes de consulta que simplifican la recuperación de los datos.

#### **Las desventajas son:**

Generalmente es necesario disponer de una o más personas que administren la base de datos, en la misma forma en que suele ser necesario en instalaciones de cierto porte disponer de una o más personas que administren los sistemas operativos. Esto puede llegar a incrementar los costos de operación en una empresa. Sin embargo hay que balancear este aspecto con la calidad y confiabilidad del sistema que se obtiene.

Si se tienen muy pocos datos que son usados por un único usuario por vez y no hay que realizar consultas complejas sobre los datos, entonces es posible que sea mejor usar una planilla de cálculo.

### **Bases de Datos OLTP (On Line Transactional Processing)**

Los sistemas OLTP son bases de datos orientadas al procesamiento de transacciones. El proceso transaccional es típico de las bases de datos operacionales. El acceso a los datos está optimizado para tareas frecuentes de lectura y escritura.

### **Bases de Datos OLAP (On Line Analytical Processing)**

Los sistemas OLAP son bases de datos orientadas al procesamiento analítico. Este análisis suele implicar, generalmente, la lectura de grandes cantidades de datos para llegar a extraer algún tipo de información útil: tendencias de ventas, patrones de comportamiento de los consumidores, elaboración de informes complejos... etc. El acceso a los datos suele ser de sólo lectura. La acción más común es la consulta, con muy pocas inserciones, actualizaciones o eliminaciones.

### **Objetos de la base de datos**

- **Tablas:** En una base de datos la información se organiza en tablas, que son filas y columnas similares a las de los libros contables o a las de las hojas de cálculo. Cada fila de la tabla recibe también el nombre de registro y cada columna se denomina también campo.
- **Vistas:** Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre.
- **Índices:** Al igual que el índice de un libro, el índice de una base de datos permite encontrar rápidamente información específica en una tabla o vista indexada. Un índice contiene claves generadas a partir de una o varias columnas de la tabla o la vista y punteros que asignan la ubicación de almacenamiento de los datos.
- **Procedimientos Almacenados:** Conjunto de instrucciones escritas en lenguaje SQL para ser ejecutadas. Aceptan parámetros.
- **Funciones:** Rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor. Similares a las funciones de los lenguajes de programación.

### **Transacciones**

Una transacción es una secuencia de operaciones realizadas como una sola unidad lógica de trabajo. Una unidad lógica de trabajo debe exhibir cuatro propiedades, conocidas como propiedades de atomicidad, coherencia, aislamiento y durabilidad (ACID), para ser calificada como transacción.

**Atomicidad.** Una transacción debe ser una unidad atómica de trabajo, tanto si se realizan todas sus modificaciones en los datos, como si no se realiza ninguna de ellas.

**Coherencia.** Cuando finaliza, una transacción debe dejar todos los datos en un estado coherente. En una base de datos relacional, se deben aplicar todas las reglas a las modificaciones de la transacción para mantener la integridad de todos los datos. Todas las estructuras internas de datos, como índices de árbol B o listas doblemente vinculadas, deben estar correctas al final de la transacción.

**Aislamiento.** Las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas. Una transacción reconoce los datos en el estado en que estaban antes de que otra transacción simultánea los modificara o después de que la segunda transacción haya concluido, pero no reconoce un estado intermedio.

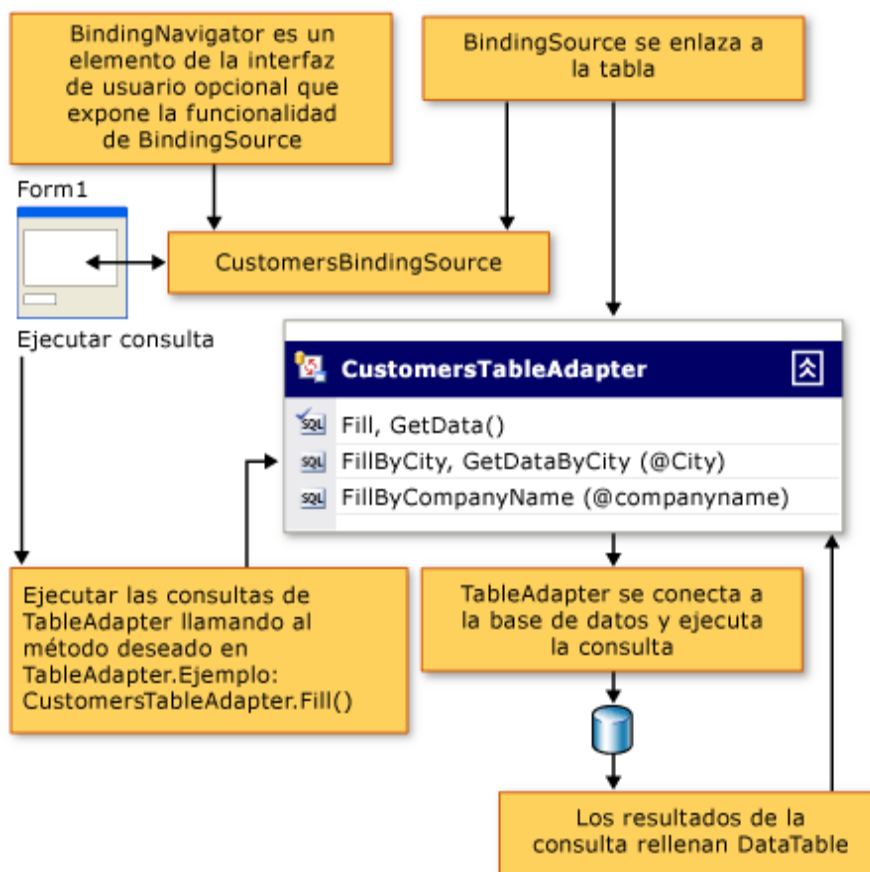


**Durabilidad.** Una vez concluida una transacción, sus efectos son permanentes en el sistema. Las modificaciones persisten aún en el caso de producirse un error del sistema.

### Conexión a Base de Datos

Visual Studio proporciona herramientas para conectar la aplicación a datos de muchos orígenes diferentes, como bases de datos, servicios Web y objetos. Si se utilizan herramientas de diseño de datos en Visual Studio, a menudo no será necesario crear de forma explícita un objeto de conexión para el formulario o componente. El objeto de conexión se crea normalmente como resultado de la finalización de uno de los asistentes de datos o al arrastrar objetos de datos al formulario. Para conectar la aplicación a los datos de una base de datos, servicio Web u objeto, ejecute el Asistente para la configuración de orígenes de datos seleccionando **Agregar nuevo origen de datos** en la Ventana Orígenes de datos.

El diagrama siguiente muestra el flujo estándar de operaciones al conectarse a datos ejecutando una consulta TableAdapter para recopilar y mostrar datos en un formulario de una aplicación para Windows.



## Crear conexiones

Al utilizar Visual Studio, las conexiones se configuran mediante el [Agregar/Modificar conexión \(Cuadro de diálogo, General\)](#). El cuadro de diálogo **Agregar conexión** aparece cuando se editan o crean conexiones dentro de uno de los asistentes de datos o del [Explorador de servidores o Explorador de base de datos](#), o cuando se editan propiedades de conexión en la ventana **Propiedades**.

Las conexiones de datos se configuran en forma automática cuando se lleva a cabo una de las acciones siguientes:

| Acción  | Descripción   |
|---|---|
| Ejecutar el <a href="#">Asistente para la configuración de orígenes de datos</a> .  | Las conexiones se configuran cuando se elige la ruta de acceso a la base de datos en el <b>Asistente para configuración de orígenes de datos</b> . Para obtener más información, vea <a href="#">Cómo: Conectarse a los datos de una base de datos</a> .  |
| Ejecutar el <a href="#">Asistente para la configuración de TableAdapter</a> .   | Las conexiones se crean dentro del <b>Asistente para la configuración de TableAdapter</b> . Para obtener más información, vea <a href="#">Cómo: Crear TableAdapters</a> .   |
| Ejecutar el <a href="#">Asistente para la configuración de consultas de TableAdapter</a> .  | Las conexiones se crean dentro del <b>Asistente para la configuración de consultas de TableAdapter</b> . Para obtener más información, vea <a href="#">Cómo: Crear consultas de TableAdapter</a> .  |
| Arrastrar elementos de la <a href="#">Ventana Orígenes de datos</a> a un formulario o al <a href="#">Diseñador de componentes</a> . | Los objetos de conexión se crean al arrastrar elementos desde la ventana <b>Orígenes de datos</b> hasta el <b>Diseñador de Windows Forms</b> o el <b>Diseñador de componentes</b> . Para obtener más información, vea <a href="#">Mostrar datos en formularios en aplicaciones para Windows</a> .               |
| Agregar nuevas conexiones de datos al <a href="#">Explorador de servidores o Explorador de base de datos</a> .                      | Las conexiones de datos en el <b>Explorador de servidores o Explorador de base de datos</b> aparecen en la lista de conexiones disponibles dentro de los asistentes de datos. Para obtener más información, vea <a href="#">Cómo: Agregar nuevas conexiones de datos al Explorador de servidores/Explorador</a> |

|  |                                    |
|--|------------------------------------|
|  | <a href="#">de bases de datos.</a> |
|--|------------------------------------|

## Cadenas de conexión

Todos los objetos de conexión exponen más o menos los mismos miembros. Sin embargo, los miembros específicos disponibles con un objeto [OleDbConnection](#) dado dependen del origen de datos al que se conecta; no todos los orígenes de datos admiten todos los miembros de la clase **OleDbConnection**.

La propiedad principal asociada a un objeto de conexión es la propiedad `ConnectionString`. Esta propiedad consta de una cadena con pares atributo/valor que proporciona la información necesaria de inicio de sesión en un servidor de bases de datos y apunta a una base de datos concreta. Una propiedad `ConnectionString` típica tendría el siguiente aspecto:

```
Provider=SQLOLEDB.1;Data Source=MySQLServer;Initial Catalog=NORTHWIND;Integrated Security=SSPI
```

Esta cadena de conexión concreta especifica que la conexión debería usar la seguridad integrada de Windows. Una cadena de conexión también puede incluir un nombre y una contraseña de usuario pero esto no es recomendable, ya que estos atributos se compilan en la aplicación y, por lo tanto, suponen una posible infracción de seguridad.

Los pares atributo/valor que OLE DB utiliza con más frecuencia están representados también, por separado, por medio de propiedades individuales tales como `DataSource` y `Database`. Cuando trabaje con un objeto de conexión, puede establecer la propiedad `ConnectionString` como una sola cadena o establecer propiedades individuales de conexión. (Si el origen de datos necesita valores de cadena de conexión que no estén representados por propiedades individuales, deberá establecer la propiedad `ConnectionString`).

## Abrir y cerrar conexiones

Los dos principales métodos para las conexiones son `Open` y `Close`. El método `Open` utiliza la información de la propiedad `ConnectionString` para ponerse en contacto con el origen de datos y establecer una conexión abierta. El

método `Close` cierra la conexión. Es esencial cerrar las conexiones, porque la mayoría de los orígenes de datos admiten sólo un número limitado de conexiones abiertas y las conexiones abiertas consumen valiosos recursos del sistema.

Si trabaja con [TableAdapters](#), [DataAdapters](#) o `DataCommands`, no tendrá que abrir ni cerrar conexiones explícitamente. Cuando se llama a un método de estos objetos (por ejemplo, a los métodos `Fill` o `Update` de un adaptador), el método comprueba si ya está abierta la conexión. Si no es así, el adaptador abre la conexión, ejecuta su lógica y cierra de nuevo la conexión.

Los métodos como `Fill` sólo abren y cierran automáticamente la conexión si no está ya abierta. Si la conexión está abierta, los métodos la utilizan pero no la cierran. Esto le proporciona la flexibilidad necesaria para abrir y cerrar comandos de datos. Puede hacer esto si tiene múltiples adaptadores que compartan una conexión. En este caso, hacer que cada adaptador abra y cierre la conexión cuando se llame a su método `Fill` no resulta eficiente. En lugar de hacerlo así, puede abrir la conexión, llamar al método `Fill` de cada adaptador y, cuando termine, cerrar la conexión.

## Agrupar conexiones

A menudo, las aplicaciones tienen diferentes usuarios ejecutando el mismo tipo de acceso a la base de datos. Por ejemplo, muchos usuarios podrían consultar la misma base de datos para obtener los mismos datos. En esos casos, el rendimiento de la aplicación puede mejorar si se hace que la aplicación comparta o *agrupe* las conexiones con el origen de datos. La sobrecarga que supone hacer que cada usuario abra y cierre una conexión separada podría tener, de lo contrario, un efecto adverso sobre el rendimiento de la aplicación.

Si utiliza la clase **`OleDbConnection`**, [OdbcConnection](#) o [OracleConnection](#), el proveedor controlará automáticamente la agrupación de conexiones, así que no tendrá que preocuparse de administrarla personalmente.

Si utiliza la clase [SqlConnection](#), la agrupación de conexiones se administra de forma implícita, pero también dispone de opciones que le permiten administrar la agrupación personalmente. Para obtener más información, vea [Uso de agrupación de conexiones](#).

## Transacciones

Los objetos de conexión admiten transacciones con un método `BeginTransaction` que crea un objeto de transacción (por ejemplo, un objeto [SqlTransaction](#)). El objeto de transacción, a su vez, admite métodos que permiten confirmar o deshacer las transacciones.

Las transacciones se administran en el código. Para obtener más información, vea [Realización de transacciones](#).

La versión 2.0 de .NET Framework incluye un nuevo marco de transacciones, accesible a través del espacio de nombres [System.Transactions](#). Este marco de trabajo expone las transacciones de tal manera que está totalmente integrado en .NET Framework, incluso en ADO.NET. Para obtener más información, vea [Aprovechamiento de System.Transactions](#).

## Información de conexión y seguridad

Dado que la apertura de una conexión implica obtener acceso a un recurso importante (una base de datos) a menudo surgen problemas de seguridad al configurar una conexión y trabajar con ella.

El modo de asegurar la aplicación y su acceso al origen de datos depende de la arquitectura del sistema. En una aplicación basada en Web, por ejemplo, los usuarios suelen obtener un acceso anónimo a Servicios de Internet Information Server (IIS) y, por lo tanto, no proporcionan credenciales de seguridad. En este caso, la aplicación mantiene su propia información de inicio de sesión y la utiliza (en lugar de cualquier información específica del usuario) para abrir la conexión y tener acceso a la base de datos.

En aplicaciones de intranet o de varios niveles, puede aprovechar la opción de seguridad integrada que proporcionan Windows, IIS y SQL Server. En este modelo, las credenciales de autenticación de un usuario para la red local se utilizan también para el acceso a los recursos de la base de datos y no se utiliza ninguna contraseña o nombre de usuario explícito en la cadena de conexión. (Habitualmente, los permisos se establecen en el equipo servidor de la base de datos por medio de grupos, de modo que no es necesario establecer permisos individuales para cada usuario que pueda tener acceso a la base de datos). En este modelo, no es necesario almacenar ninguna información de inicio de sesión para la conexión ni es necesario dar más pasos para proteger la información de la cadena de conexión.

## Conexiones en tiempo de diseño en el Explorador de servidores o Explorador de base de datos

El **Explorador de servidores o Explorador de base de datos** proporciona un medio para crear conexiones en tiempo de diseño con orígenes de datos. Permite examinar los orígenes de datos disponibles, mostrar información acerca de las tablas, columnas y otros elementos que contienen, y modificar y crear elementos de la base de datos.

Su aplicación no utiliza en forma directa las conexiones disponibles en el **Explorador de servidores o Explorador de base de datos**. Visual Studio utiliza estas conexiones para trabajar con la base de datos en tiempo de diseño. Para obtener más información, vea [Visual Database Tools](#).

Por ejemplo, en tiempo de diseño se puede utilizar el **Explorador de servidores o Explorador de base de datos** para crear una conexión a una base de datos. Más tarde, al diseñar un formulario, puede examinar la base de datos, seleccionar columnas de una tabla y arrastrarlas hacia el [Dataset Designer](#). Esto crea un [TableAdapter](#) en su conjunto de datos. También crea un nuevo objeto de conexión, el cual forma parte del TableAdapter recientemente creado.