

**TEMA 9**

# Interfaz gráfica.

## 1. Introducción

Hasta aquí hemos visto cómo funciona Visual Basic .NET, una introducción a la programación orientada a objetos y las instrucciones del lenguaje: variables, vectores, funciones, procedimientos,...

Ahora vamos a aplicar todo en programas de Windows. La base de estos programas, como ya hemos comentado, son los formularios basados en la clase:

### **System.Windows.Forms.Form**

Dentro de la clase principal System tenemos varios espacios de nombres, por ejemplo:

- Data: Consta principalmente de las clases que constituyen la arquitectura para acceso a datos ADO.NET. La arquitectura ADO.NET permite crear componentes que administran eficazmente los datos procedentes de múltiples orígenes
- Drawing: Proporciona acceso a la funcionalidad básica de gráficos de GDI+. Los espacios de nombres System.Drawing.Drawing2D, System.Drawing.Imaging y System.Drawing.Text proporcionan funcionalidades más avanzadas.
- ...
- Windows.Forms: Contiene clases para crear aplicaciones basadas en Windows y que aprovechan plenamente las características avanzadas de interfaz de usuario disponibles en este sistema operativo.

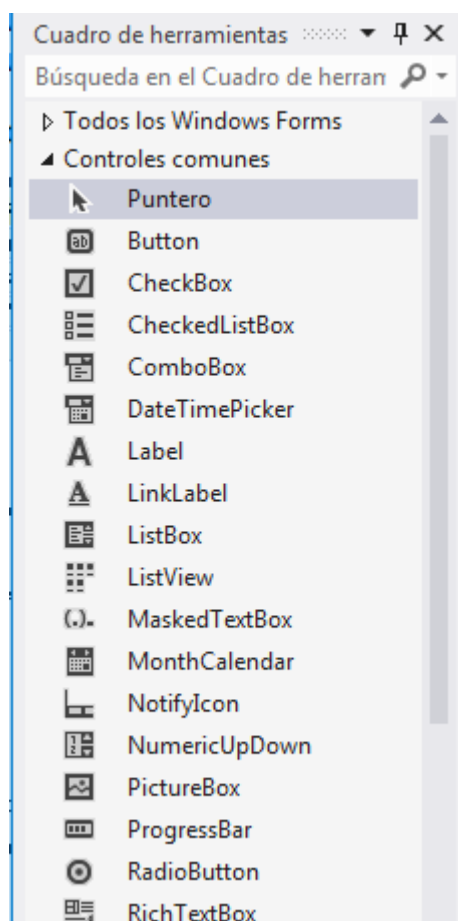
Este último es el que utilizaremos para el desarrollo de aplicaciones Windows. Las clases de este espacio de nombres se pueden agrupar en las siguientes categorías:

- **Control, User Control, and Form**. La mayoría de las clases del espacio de nombres **System.Windows.Forms** derivan de la clase Control. La clase Control proporciona la funcionalidad base de todos los controles que se muestran en un objeto Form. La clase Form representa una ventana dentro de una aplicación. Incluye cuadros de diálogo, ventanas no modales y ventanas cliente y principal de la Interfaz de documentos múltiples (MDI).

- **Controls.** El espacio de nombres **System.Windows.Forms** dispone de diferentes clases de controles que permiten crear interfaces de usuario completas. Algunos controles están diseñados para la entrada de datos en la aplicación, por ejemplo, los controles TextBox y ComboBox. Otros controles muestran datos de la aplicación, por ejemplo, Label, ListBox y ListView. El espacio de nombres también dispone de controles para invocar a comandos en la aplicación, por ejemplo, Button y ToolBar.
- **Components.** Además de los controles, el espacio de nombres **System.Windows.Forms** proporciona otras clases que no se derivan de la clase Control pero que también aportan características visuales a las aplicaciones basadas en Windows. Algunas clases, como ToolTip y ErrorProvider, amplían las capacidades o proporcionan información al usuario. Otras, como Menu, MenuItem y ContextMenu, permiten mostrar menús al usuario para que invoquen a comandos en una aplicación. Las clases Help y HelpProvider permiten mostrar información de Ayuda al usuario de las aplicaciones.
- **CommonDialog Boxes.** Windows proporciona varios cuadros de diálogo comunes que se pueden utilizar para ofrecer a la aplicación una interfaz de usuario coherente a la hora de realizar tareas como abrir y guardar archivos, manipular la fuente o el color del texto, o imprimir. Las clases OpenFileDialog y SaveFileDialog proporcionan la funcionalidad para mostrar un cuadro de diálogo que permita al usuario buscar o escribir el nombre del archivo que desea abrir o guardar. La clase FontDialog muestra un cuadro de diálogo para cambiar los elementos del objeto Font que utiliza la aplicación. Las clases PageSetupDialog, PrintPreviewDialog y PrintDialog muestran cuadros de diálogo que permiten al usuario controlar la impresión de documentos.

Los controles son los elementos que colocaremos en nuestros formularios. Cuando hablamos del IDE ya vimos dónde podíamos localizarlos y trabajamos con algunos de ellos:

Así que para ser coherentes llamaremos a las ventanas de Windows formularios o "Windows Forms".



## 2. Formularios: Systems.Windows.Forms

Windows Forms es la nueva plataforma de desarrollo de aplicaciones para Microsoft Windows, basada en .NET Framework. Este marco de trabajo proporciona un conjunto de clases claro orientado a objetos y ampliable, que permite desarrollar complejas aplicaciones para Windows.

### 2.1. ¿Qué es un formulario?

Un formulario es una representación de una ventana. La mayoría de los formularios se utilizan para mostrar controles con información que el usuario puede ver o manipular. Un formulario como hemos dicho en otras ocasiones es un objeto con propiedades que definen su aspecto, métodos que definen su comportamiento y eventos que definen su interacción con el usuario.

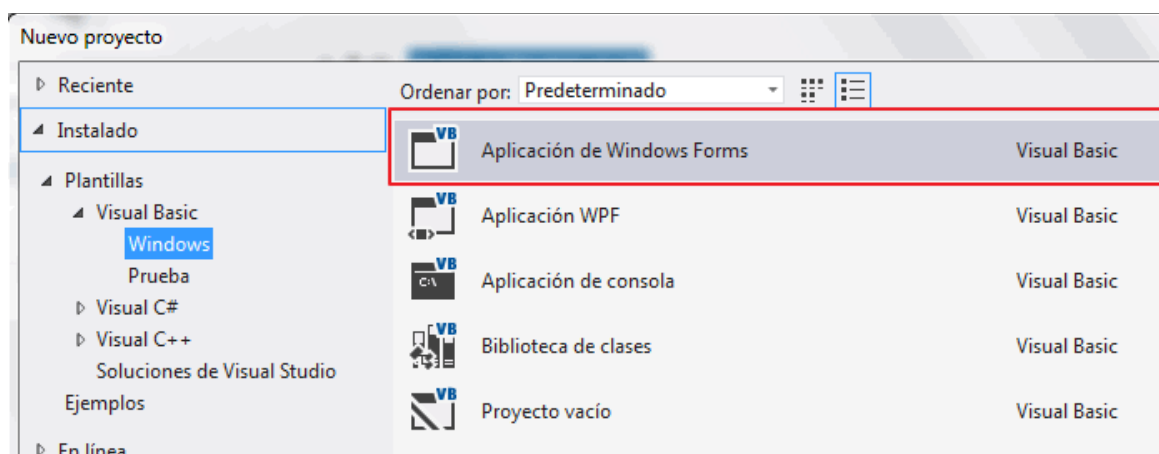
Por ejemplo, podemos modificar formularios para crear ventanas estándar, cuadros de diálogo, interfaces de múltiples ventanas (MDI) o mostrar rutinas gráficas avanzadas. .NET Framework nos permite heredar los formularios existentes para añadirle más funciones o modificar su comportamiento.

Cuando añadimos un formulario a un proyecto podemos elegir si queremos heredar un formulario de la clase Form proporcionada por .NET o si queremos heredar un formulario ya existente. .NET proporciona un entorno para los formularios común y orientado a objeto para todos los lenguajes.

Dentro de un proyecto de formularios Windows Forms, el formulario es el vehículo principal de interacción con el usuario. Podemos combinar diferentes conjuntos de controles y escribir código para obtener información del usuario y ofrecer una respuesta. También para acceder a bases de datos, ficheros de texto, sistema de archivos, dibujar,... En definitiva, todo lo que nos ofrece Windows.

### 1.2. Crear un formulario

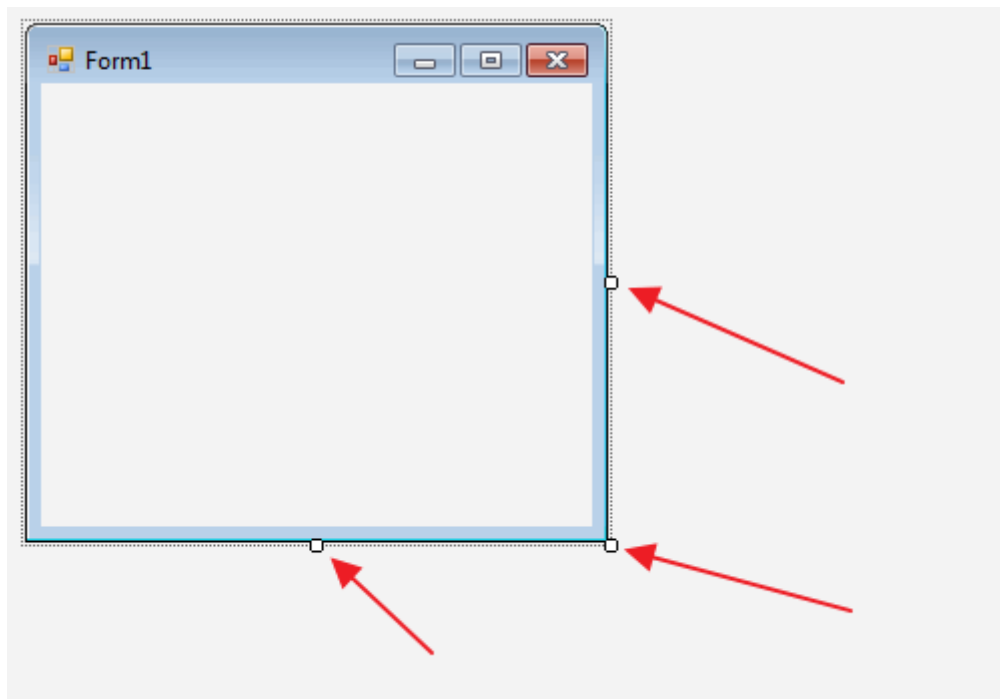
Esto ya lo vimos al principio pero vamos a realizar una sencilla aplicación para repasar su creación a través del IDE. Para empezar como siempre creamos un proyecto de Windows.



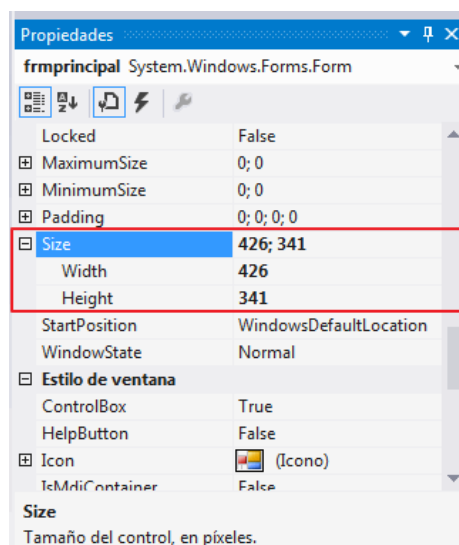
El IDE nos muestra un formulario en pantalla en el que podremos modificar algunas de sus propiedades y su forma de funcionar. En esta parte del tema profundizaremos un poco en su funcionamiento y posibilidades.

### 1.3. Cambiar el tamaño a un formulario

Muy sencillo, sólo debemos arrastrar con el ratón los bordes de la derecha inferior y ángulo:



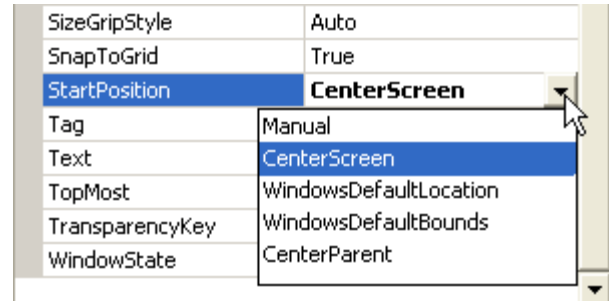
Como todas las propiedades, los valores del tamaño y posición se pueden controlar en la ventana propiedades. Podemos modificar directamente sus valores con la sección “Size”:



## 1.4. Ubicación inicial del formulario

Ya tenemos el tamaño del formulario, ahora debemos colocarlo en pantalla. En ocasiones queremos que aparezca completamente maximizado, como una aplicación normal. Otras veces la intención es que aparezca con el tamaño que lo definimos en el IDE. Por ejemplo para mostrar un mensaje, una pantalla de introducción de datos o consulta que no necesitan la pantalla completa.

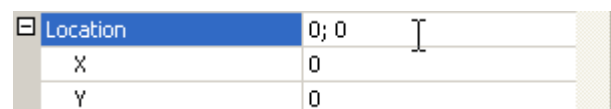
El comportamiento inicial lo va a decir la propiedad "StartPosition":



Esta propiedad permite iniciarse al formulario de cinco formas diferentes:

- Manual. Seguirá los valores de tamaño y posición que le indiquemos en la propiedad Location.
- CenterScreen. El formulario está centrado en la pantalla actual y tiene las dimensiones especificadas en el tamaño del formulario.
- WindowsDefaultLocation. El formulario se encuentra colocado en la ubicación predeterminada de Windows y tiene las dimensiones especificadas en el tamaño del formulario.
- WindowsDefaultBounds. El formulario se encuentra colocado en la ubicación predeterminada de Windows y tiene los límites establecidos por Windows de forma predeterminada
- CenterParent. El formulario está centrado en los límites de su formulario principal

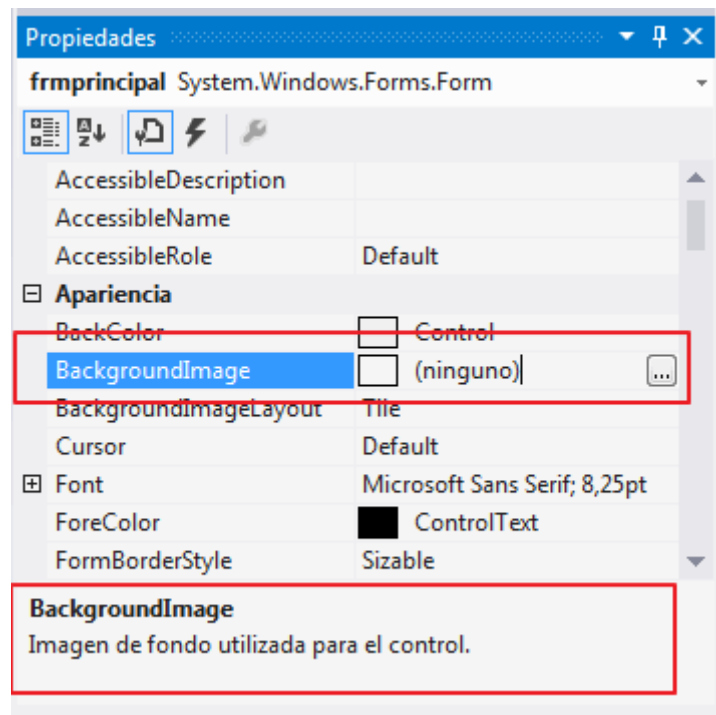
Para el caso "Manual" ajustaremos las propiedades de "Location" para indicarle la posición donde queremos que se inicie:



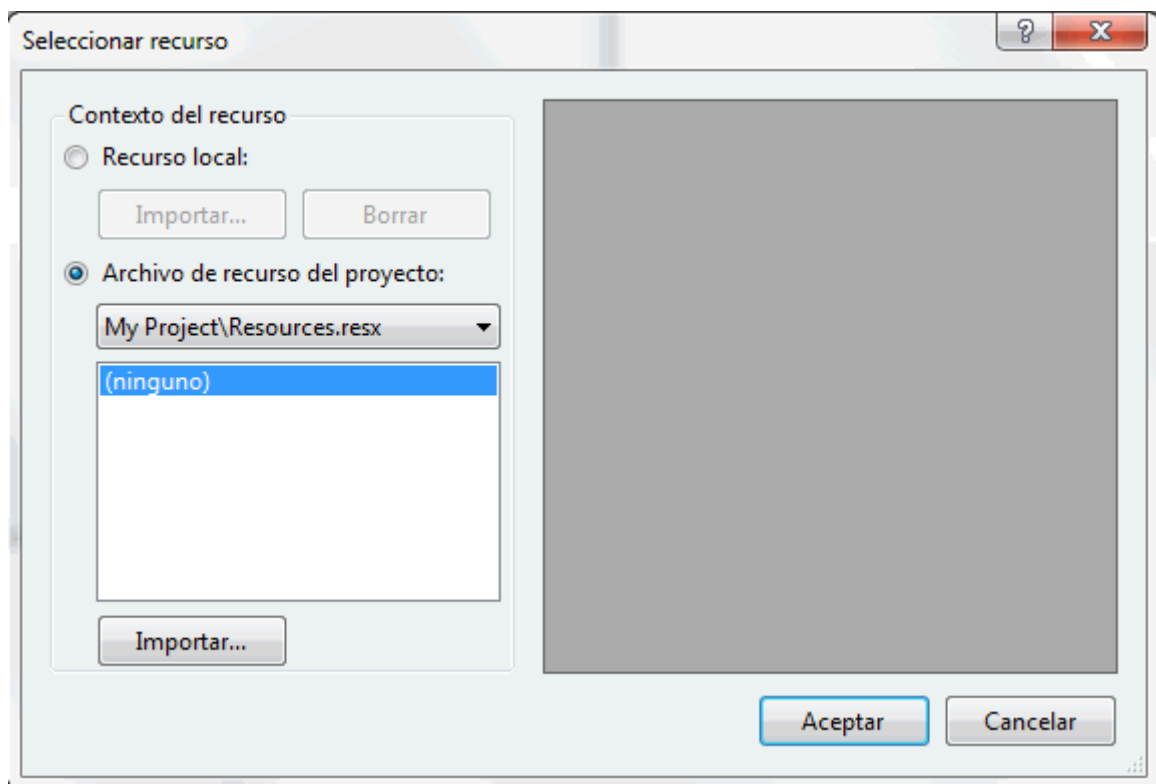
## 1.5. Poner una imagen de fondo, añadir icono.

Una propiedad que no utilizaremos muchas veces pero que ayuda a comprender el grado de personalización que podemos llegar a conseguir con estas propiedades, es la de poner una imagen de fondo en el formulario. Puede ser el caso del logotipo de nuestra empresa en la pantalla que sale al comienzo del programa (llamada en ocasiones Splash). Para poner una

imagen simplemente buscaremos la propiedad "BackgroundImage" y buscaremos la imagen de fondo que queremos:

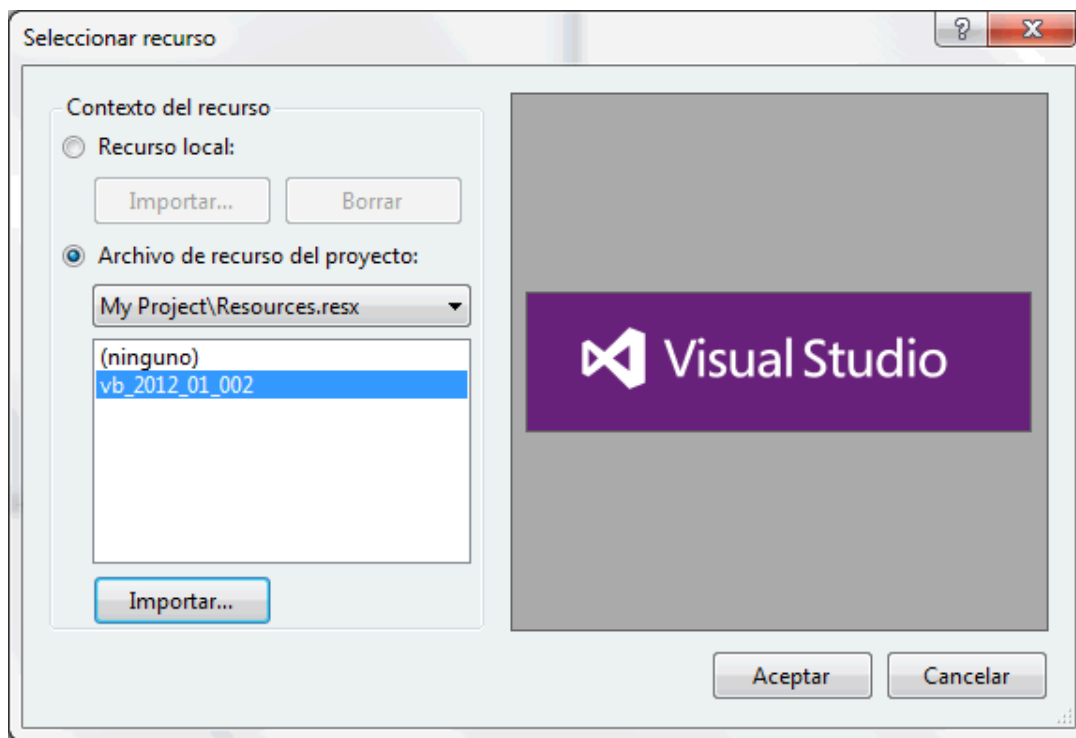


Al hacer clic nos muestra una pantalla que será habitual en varios controles:

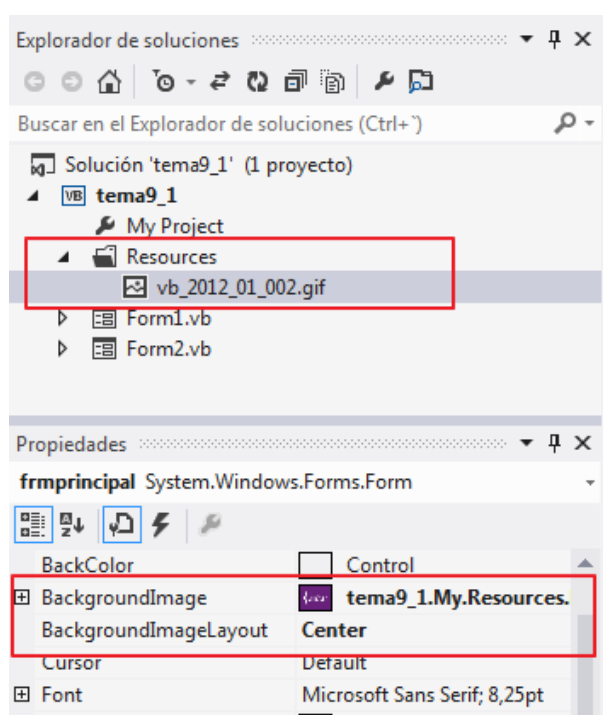


Indicaremos de dónde queremos obtener la imagen. Hay un fichero especial llamado "My project\Resources.resx" donde se almacenarán los recursos asociados a nuestro proyecto, como

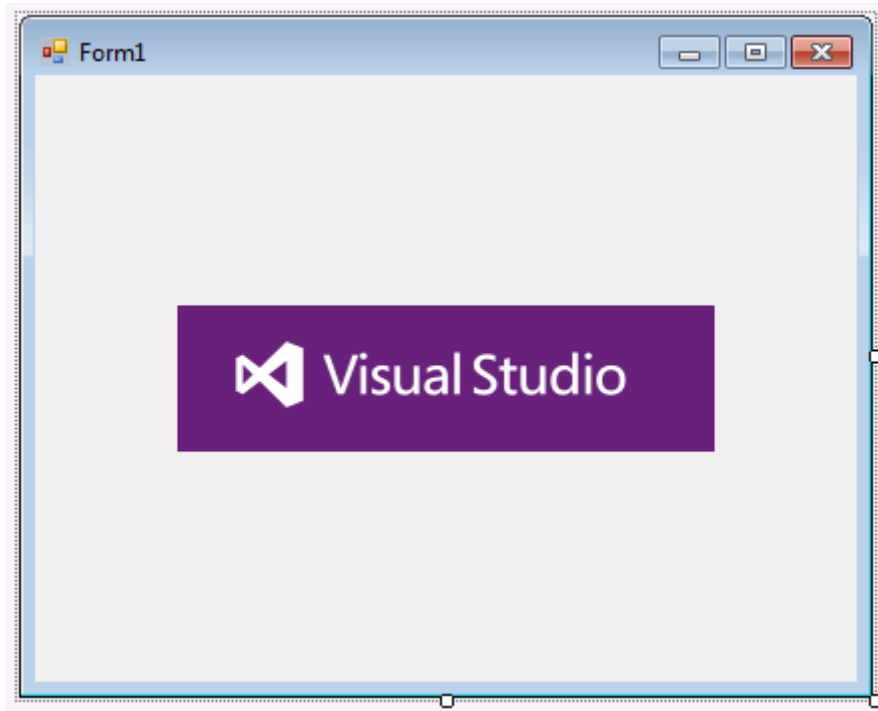
pueden ser iconos y gráficos utilizados en nuestra aplicación. Para este fondo de escritorio pulsaremos "Importar" y le indicamos el gráfico:



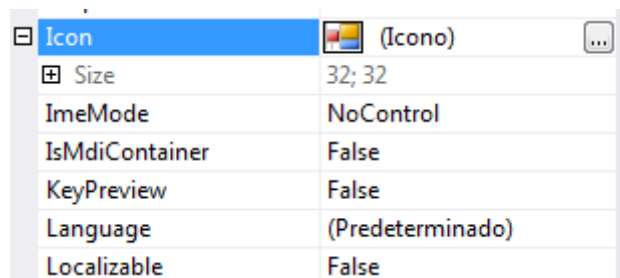
La idea es tener un "contenedor" de imágenes. Si seleccionamos la otra opción sería similar pero no se almacenaría en nuestro fichero de recursos proyecto, sino que la incluiría en la pantalla sin más. Para un uso es válido pero no si queremos invocar en otras partes del proyecto a varias imágenes distintas. Al pulsar en "Aceptar" la imagen queda incluida en nuestro proyecto tal y como te he contado:



En las propiedades nos indicará ahora que hay una imagen. Y además con la propiedad "BackgroundImageLayout" podemos hacer que simplemente quede centrada, en lugar de hacer el mosaico predeterminado:



Si se desea se puede poner un icono al formulario. Este icono será la imagen que se muestre en el cuadro de menú del sistema del formulario y cuando el formulario se minimice.

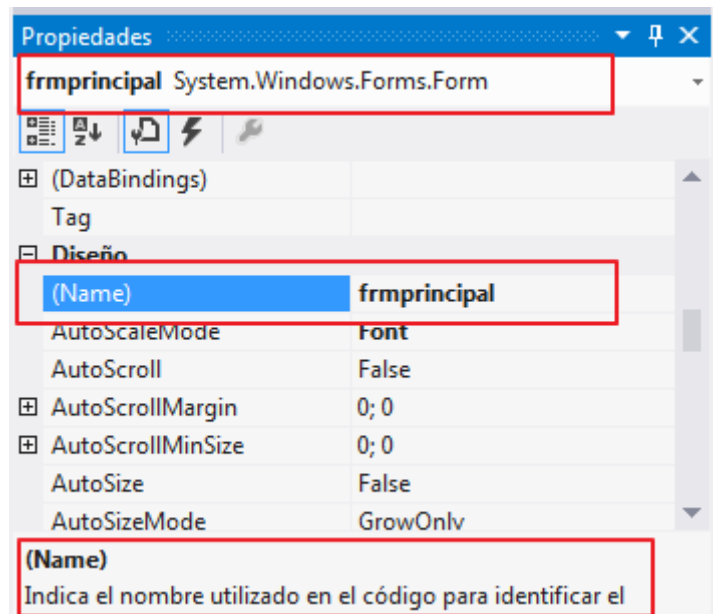


## 1.6. Nombre del formulario

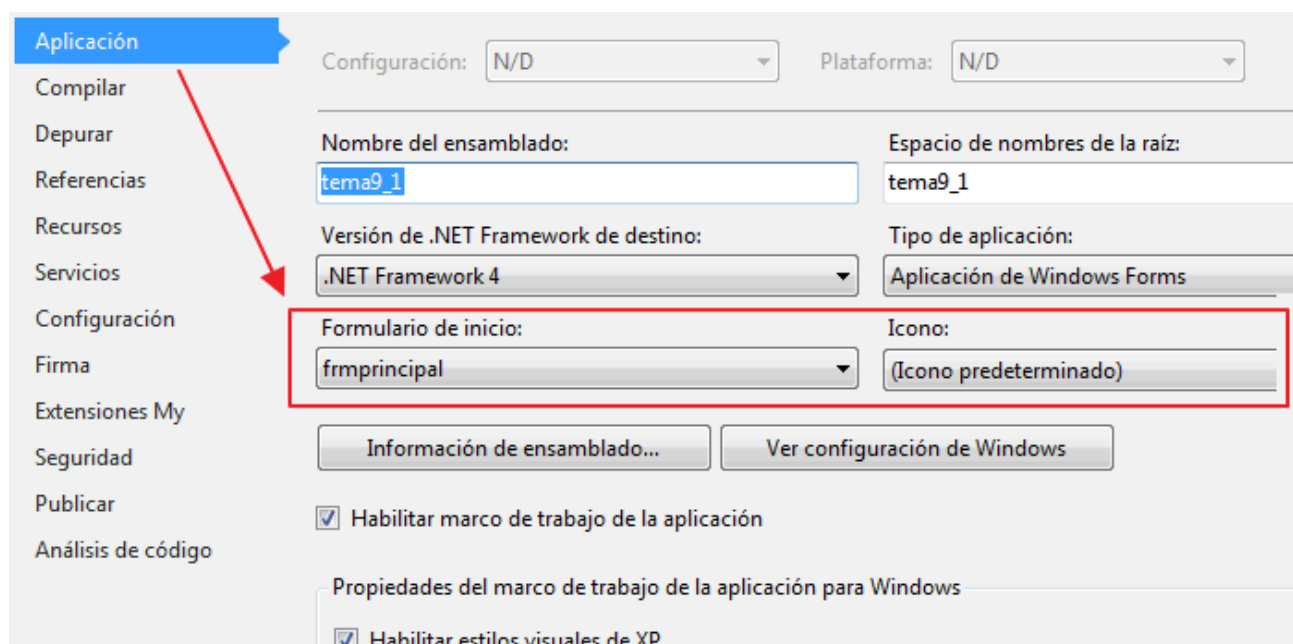
El formulario como el resto de los componentes y controles de nuestra interfaz debe tener obligatoriamente un nombre, VB.NET le pone uno genérico Form1, Form2,... que debemos cambiar para acceder a él en el código más cómodo. Para cambiarle el nombre debemos ponerlo en la propiedad Name.

**Nota:** En ocasiones nuestro formulario tiene muchos elementos y puede que en nuestra ventana de propiedades nos muestre las propiedades de otro control porque sin querer hemos hecho clic sobre otro. La mejor forma de asegurarse es comprobando en la parte superior de la ventana de propiedades que aparece la clase adecuada, en nuestro ejemplo verás frmPrincipalSystem.Windows.Forms.Form

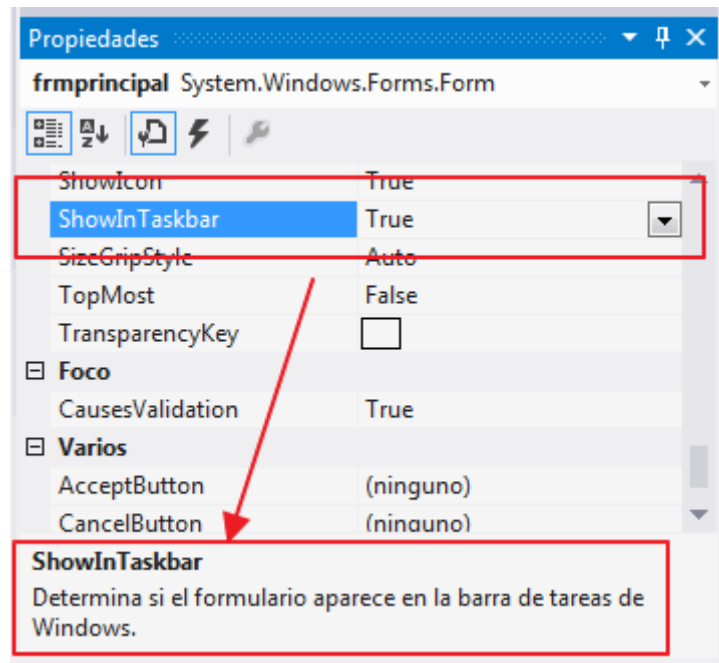




Si vamos a tener varios formularios debemos tener claro cuál va a ser el principal del programa y así lo indicaremos en las propiedades del proyecto. Es importante, porque el que indiquemos aquí será por el que comience el programa:



En la parte central indicamos el formulario de inicio, es decir, el punto de entrada a nuestro programa. Ya tenemos un pequeño avance de las propiedades del formulario, ahora toca practicar con otras para ver su funcionamiento. Para ayudarnos en estas pruebas, recuerda tener activada la ayuda que aparece en la parte inferior de la ventana de propiedades porque nos irá indicando qué hace cada propiedad:

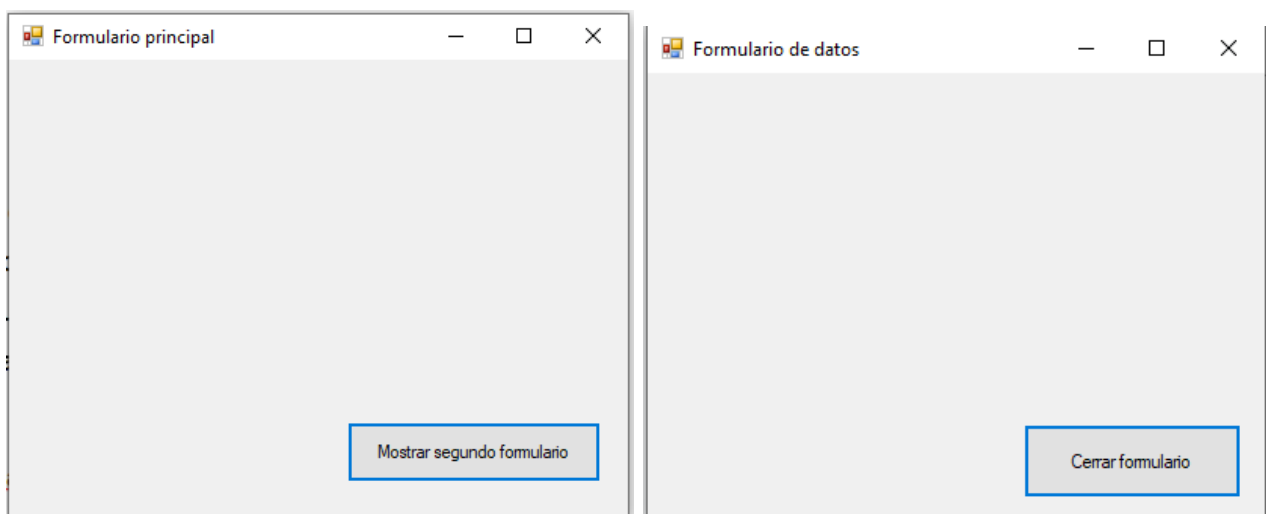


Por ejemplo, "ShowInTaskBar" hará que el formulario aparezca en la barra de tareas de Windows.

## Ejercicio: Control de formularios

Ahora que ya conocemos algunas de sus propiedades vamos a trabajar con algún método y evento para hacer un pequeño ejemplo. Este ejemplo consiste en lo siguiente: vamos a tener dos formularios. En uno de ellos pondremos un botón que ponga "Mostrar segundo formulario", y en el segundo pondremos "Cerrar formulario". De esta forma tendremos el código necesario para mostrar y ocultar un formulario.

Le pondremos como nombre al primero "frmPrincipal" y al segundo "frmDatos"



## 2. Propiedades y características de los formularios

Veamos ahora una lista de las propiedades más útiles de los formularios:

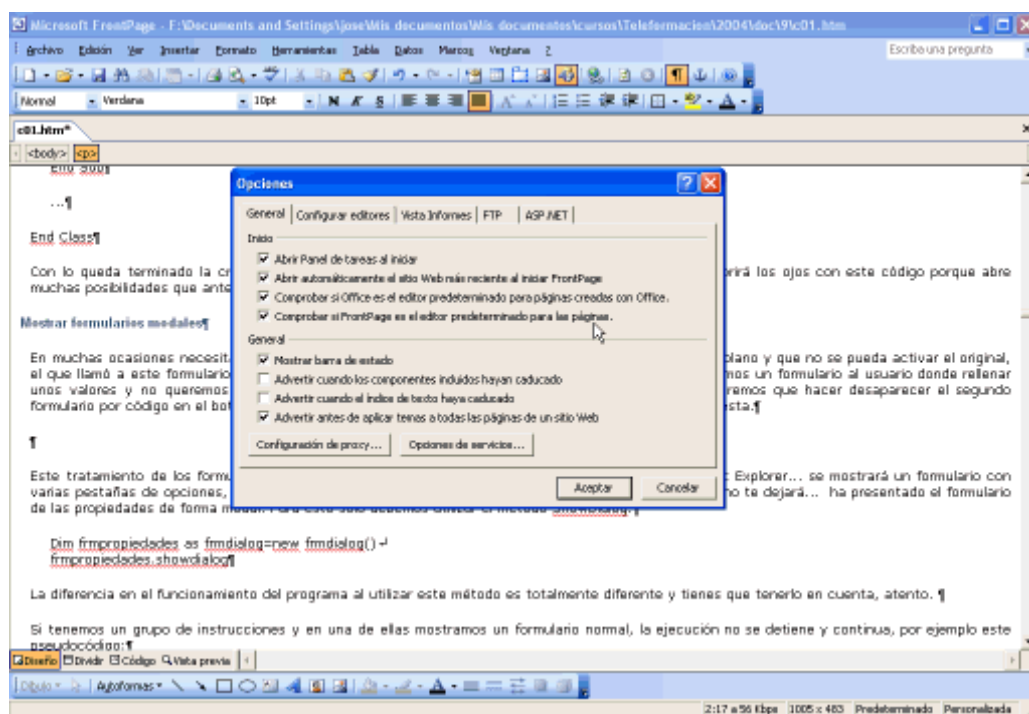
Propiedad	Descripción
<b>AcceptButton</b>	Representa el botón que se va a usar como botón Aceptar del formulario
<b>AutoScale</b>	Si está activado el formulario ajusta automáticamente la fuente de la pantalla
<b>AutoScroll</b>	Determina si aparecen automáticamente las barras de desplazamiento cuando los controles están situados fuera del área del formulario
<b>BackColor</b>	Color de fondo utilizado para mostrar texto y gráficos en el control
<b>BackgroundImage</b>	Imagen de fondo para el formulario
<b>CancelButton</b>	Obtiene o establece el control de botón que se activará cuando el usuario presione la tecla ESC
<b>CausesValidation</b>	Obtiene o establece un valor que indica si el control hace que se realice una validación de todos los controles que requieren validación cuando reciben el foco.
<b>ContextMenu</b>	Obtiene o establece el menú contextual asociado al control.
<b>ControlBox</b>	Obtiene o establece un valor que indica si se muestra un cuadro de control en la barra de título del formulario.
<b>Cursor</b>	Obtiene o establece el cursor que se muestra cuando el puntero del mouse se sitúa sobre el control.
<b>Enabled</b>	Obtiene o establece un valor que indica si el control puede responder a la interacción del usuario, es decir, si está habilitado.
<b>Font</b>	Obtiene o establece la fuente del texto que muestra el control.
<b>ForeColor</b>	Obtiene o establece el color de primer plano del control.
<b>HelpButton</b>	Obtiene o establece un valor que indica si se muestra un botón de ayuda en el cuadro de título del formulario.
<b>Icon</b>	Obtiene o establece el icono del formulario
<b>IsMdiChild</b>	Obtiene un valor que indica si el formulario es un formulario MDI (interfaz de múltiples documentos) secundario
<b>IsMdiContainer</b>	Obtiene o establece un valor que indica si el formulario es un contenedor para formularios MDI (interfaz de múltiples documentos) secundarios
<b>MaximizeBox</b>	Obtiene o establece un valor que indica si se muestra el botón Maximizar en la barra de título del formulario.
<b>Menu</b>	Obtiene o establece el MainMenu que se muestra en el formulario.
<b>MinimizeBox</b>	Obtiene o establece un valor que indica si se muestra el botón Minimizar en la barra de título del formulario
<b>Opacity</b>	Obtiene o establece el nivel de opacidad del formulario.
<b>ShowInTaskBar</b>	Obtiene o establece un valor que indica si se muestra el formulario en la barra de tareas de Windows.
<b>Size</b>	Tamaño en píxeles
<b>StartPosition</b>	Obtiene o establece la posición inicial del formulario en tiempo de ejecución.
<b>TopMost</b>	Obtiene o establece un valor que indica si el formulario debe mostrarse como formulario de nivel superior de la aplicación.

<b>TransparencyKey</b>	Obtiene o establece el color que representará las áreas transparentes del formulario.
<b>WindowState</b>	Obtiene o establece el estado de la ventana del formulario. (normal, minimizado, maximizado)

## 2.1. Mostrar formularios modales

En muchas ocasiones necesitaremos mostrar un formulario modal. Esto es, que se presente en primer plano y que no se pueda activar el original, el que llamó a este formulario. Por ejemplo, en una recogida de datos para una consulta... le mostraremos un formulario al usuario donde rellenar unos valores y no queremos que se active la pantalla de debajo si le hace clic. En este caso tendremos que hacer desaparecer el segundo formulario por código en el botón "Aceptar" o "Cancelar" pero no podremos dejar la principal delante de esta.

Por ejemplo, en la página de propiedades de FrontPage, no me deja acceder a la ventana principal hasta que no pulso en "Aceptar" o "Cancelar":



Este tratamiento de los formularios es muy utilizado. Por ejemplo, si pulsamos en las Opciones de Word se mostrará un formulario con varias pestañas laterales de opciones. Si ahora intentamos hacer clic en el programa sin cerrar esta ventana el programa no nos dejará. Esto es porque ha presentado el formulario de forma "modal". Para esto sólo debemos utilizar el método ShowDialog:

```
Dim frmPropiedades As New Form()
frmPropiedades.ShowDialog()
```

El funcionamiento del programa al utilizar este método es totalmente diferente y debemos tenerlo en cuenta.

Si tenemos un grupo de instrucciones y en una de ellas mostramos un formulario normal, la ejecución no se detiene y continúa. Por ejemplo, este pseudocódigo:

```
calcula_datos  
muestra_formulario  
graba_datos  
termina_proceso
```

En este caso después de mostrar el formulario continúa con la ejecución del código siguiente. Sin embargo, en este caso:

```
calcula_datos  
muestra_formulario_MODAL  
graba_datos  
termina_proceso
```

El programa se detiene cuando se muestra el formulario modal y no continua hasta que se ha ocultado o descargado de memoria. De ahí la importancia de utilizarlo.

## 2.2. Mostrar formularios no modales

Después del razonamiento anterior, sólo debemos saber la instrucción para mostrar un formulario no modal:

```
Dim frmToolBox As Form = New Form()  
frmToolBox.Show()
```

El ejemplo de uso de este tipo de formularios es de una barra de herramientas, es un formulario con iconos y que no hace falta cerrar para seguir trabajando, puede estar abierto a un lado de nuestra aplicación como formulario auxiliar.

## 2.3. Formularios de nivel superior (Top-Most)

Un formulario de nivel superior es aquel que se muestra siempre en primer nivel aunque no esté activo. Si en una aplicación tengo varios formularios queda siempre en primer plano aquel sobre el que hacemos clic y los demás pasan a segundo plano. En ocasiones, puedo querer que siempre haya uno en primer plano aunque ni siquiera esté activo, para esto activaré el formulario con la propiedad "TopMost":

```
frmToolBox.TopMost = True
```

Un formulario de este tipo puede ser, por ejemplo, una barra de botones, si la ponemos flotante encima de la pantalla de trabajo, por ejemplo en Word, podemos seguir escribiendo y esta ventana seguirá visible en la posición donde la hayamos colocado.

## 2.4. Bordos y tamaño de los formularios

Los bordes del formulario que se asignan con la propiedad "**FormBorderStyle**" condicionan mucho el comportamiento del formulario. Además de proporcionar un aspecto visual distinto, esta propiedad influye en cómo se escribe la barra del título y sus botones. Por otra parte, esta propiedad también afecta a la capacidad que tienen para redimensionar los formularios, podemos eliminar la pestaña inferior derecha que permite su redimensionamiento. En esta tabla tenemos los distintos valores de esta propiedad:

Valor	Descripción
<b>Ninguno</b>	Sin bordes. Utilizado en las pantalla de inicio del programa por ejemplo.
<b>Fixed3D</b>	Borde tridimensional fijo. No se puede redimensionar. Puede incluir un cuadro de control a la izquierda, barra de títulos y botones de maximizar y minimizar en la barra de título
<b>FixedDialog</b>	Borde fijo de una sola línea, utilizado para cuadros de diálogo. No se puede redimensionar. Puede incluir un cuadro de control a la izquierda, barra de títulos y botones de maximizar y minimizar en la barra de título.
<b>FixedSingle</b>	No se puede redimensionar. Puede incluir un cuadro de control a la izquierda, barra de títulos y botones de maximizar y minimizar en la barra de título.
<b>FixedToolWindow</b>	Borde de ventana de herramienta de tamaño fijo. Una ventana de herramientas no aparece en la barra de tareas ni en la ventana que aparece cuando el usuario presiona ALT+TAB. Muestra un botón de cerrar y el título con letra más pequeña.
<b>Sizable</b>	Es el valor predeterminado, se puede redimensionar. Puede incluir un cuadro de control a la izquierda, barra de títulos y botones de maximizar y minimizar en la barra de título. Se puede redimensionar utilizando el borde del formulario o los botones de maximizar/minimizar
<b>SizableToolWindow</b>	Borde de ventana de herramienta de tamaño variable. Una ventana de herramientas no aparece en la barra de tareas ni en la ventana que aparece cuando el usuario presiona ALT+TAB.. Utilizada para herramientas de Windows, muestra un botón de cerrar y el título con letra más pequeña.

Todos los bordes excepto "Ninguno" tienen el botón "Cerrar" en la parte derecha de la barra del título.

## 2.5. Cambiar el tamaño de los formularios

Hay unas propiedades que establecen el tamaño de los formularios. Éstas permiten modificar o asignar el tamaño de varias formas diferentes. El primer caso es utilizar las propiedades para asignarle la anchura y altura (Width y Height), por ejemplo para asignarle una altura de 50 píxels:

```
frmPaleta.Height = 50
```

Podemos obtener el mismo resultado si utilizamos el objeto "Size" que especifica la anchura y altura en ese orden. Este ejemplo cambia sólo la altura del formulario a 50 píxeles:

```
frmPaleta.Size = New Size(frmPaleta.Width, 50)
```

En este caso frmPaleta.width contiene la anchura actual. Por lo tanto, le sigue asignando la misma anchura y una altura de 50 píxeles. Está claro entonces que podemos cambiar los dos valores de una sola vez con size:

```
frmPaleta.Size = New Size(50, 50)
```

## 2.6. Ubicación de los formularios

Después de crear el formulario podemos especificar dónde queremos que se muestre. La propiedad "**StartPosition**" determina la posición del formulario, el valor predeterminado es "WindowsDefaultLocation" que permite que el sistema operativo ponga el formulario en la mejor posición.

Además de esta posición inicial podemos hacer como en el punto anterior para indicarle en píxeles una determinada posición con las propiedades "Left" (para la coordenada X) y "Top" (para la coordenada Y), por ejemplo, para colocarlo a 100 píxeles de la parte superior:

```
frmPaleta.Top = 100
```

Y para colocarlo a 100 píxeles de la izquierda:

```
frmPaleta.Left = 100
```

Estas propiedades eran las utilizadas en anteriores versiones y que aquí también están disponibles. Pero en .NET se ha añadido el objeto "**Location**" con las propiedades X e Y. El ejemplo anterior quedaría:

```
frmPaleta.Location = New Point(100, 100)
```

La potencia de este objeto es que podemos cambiar de forma simultánea los dos valores.

Podemos utilizar también la propiedad "**DesktopLocation**" en lugar de Location para ajustar la ubicación del formulario. Esta propiedad determina la ubicación del formulario relativa a la barra de tareas de Windows. Esto es útil si la barra de tareas no se oculta automáticamente y se ha colocado fija en la parte izquierda o superior de la pantalla que esconde el origen de coordenadas (0,0). Si establecemos la ubicación de esta forma en un formulario a (0,0) nos aseguramos que no ocultará la barra de tareas en ningún caso, por ejemplo:

```
frmPaleta.DesktopLocation = New Point(0, 0)
```

Pasemos ahora a una de las partes importantes de los formularios, y es saber de qué métodos disponemos para trabajar con ellos.

## 2.7. Métodos de los formularios

Dada la importancia de los formularios, debemos conocer algunos de sus métodos más importantes. Así como las propiedades determinan su aspecto y forma inicial, los métodos determinan su comportamiento. En el código los métodos se diferencian cuando los escribamos porque tienen el símbolo de un cubo en 3D.

Los métodos los utilizaremos durante la ejecución del programa para realizar acciones con los formularios. Por ejemplo, para ocultar un formulario utilizaremos el método "hide":

```
frmDatos.Hide()
```

Lo métodos más importantes son:

Método	Descripción
<b>Activate</b>	Activa el formulario y le proporciona el enfoque
<b>Close</b>	Cierra el formulario.
<b>Contains</b>	Obtiene un valor que indica si el control especificado es un control secundario del control.
<b>FindForm</b>	Recupera el formulario en el que se encuentra el control.
<b>Hide</b>	Oculto el formulario al establecer el valor a False
<b>LayoutMdi</b>	Organiza los formularios MDI (interfaz de múltiples documentos) secundarios del formulario MDI principal.
<b>Refresh</b>	Obliga al control a invalidar su área cliente y, acto seguido, obliga a que vuelva a dibujarse el control y sus controles secundarios.
<b>SendToBack</b>	Envía el control al final del orden Z.
<b>Show</b>	Muestra el control
<b>ShowDialog</b>	Sobrecargado. Muestra el formulario como un cuadro de diálogo modal.
<b>Update</b>	Hace que el control vuelva a dibujar las regiones no válidas en su área de cliente.

Ya hemos visto las propiedades y métodos. Para completar la descripción de los formularios nos quedarían los eventos. También son muy numerosos (aunque sólo utilizaremos un número muy reducido de ellos) pero éstos los iremos viendo más adelante.



## 2.8. Crear formularios desde el código

En ocasiones (pocas) necesitaremos crear un formulario desde el código sin utilizar el diseñador. Aunque sea una posibilidad remota, lo interesante es cómo podemos crear un objeto desde el código. Los formularios, como el resto de los objetos, pueden crearse y manipularse desde el código.

Lo tenemos bastante fácil porque el código para esto ya lo hemos visto por separado, ahora vamos a encajar las piezas... primero como en el código que introduce el IDE la definición de la clase:

```
Public Class frmMiformulario
    ...
End Class
```

Ahora le añadimos el método constructor, es decir, el código necesario para que cree el formulario y le añadimos un par de propiedades:

```
Public Class frmMiformulario
    Public Sub New()
        Me.Name = "frmMiformulario"
        Me.Text = "Formulario creado desde código"
        Me.StartPosition = FormStartPosition.CenterScreen
    End Sub
    ...
End Class
```

Con lo queda terminada la creación del formulario.

## 2.9. Eventos de formularios

Los eventos como hemos comentado en otras ocasiones suceden cuando hay cambios en los objetos o controles, por ejemplo, cuando un usuario abre, cierra, mueve o interactúa con el formulario. Estos eventos se "disparan" o activan cuando el usuario interactúa con el teclado o ratón. Existen muchos eventos pero sólo vamos a mostrar los más interesantes:

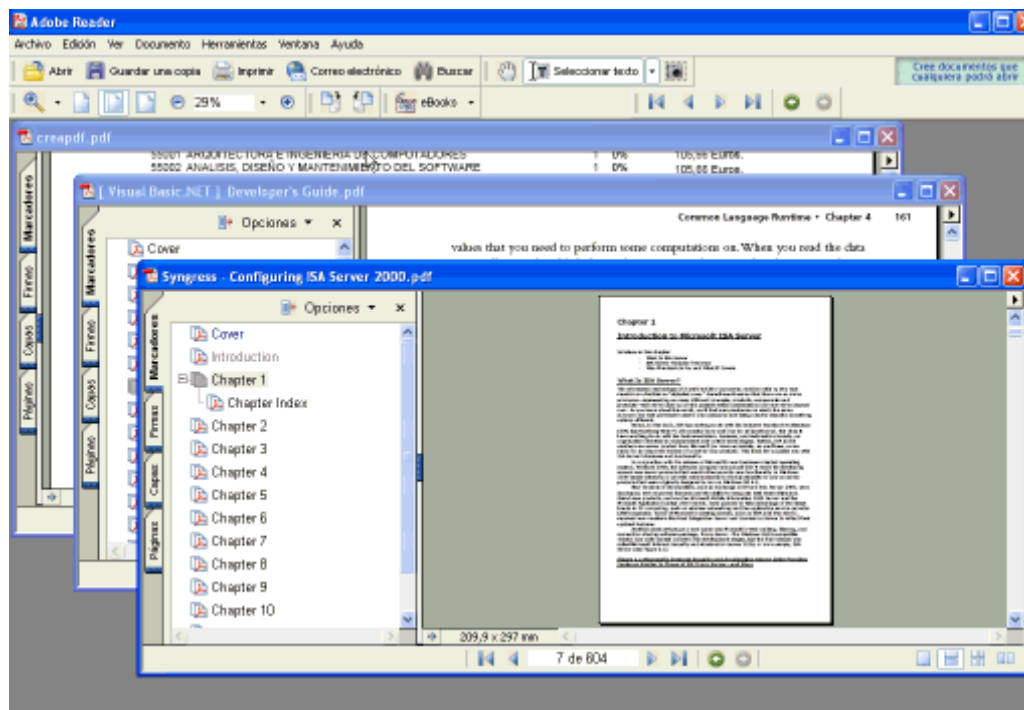
Evento	Descripción
Activated	Se produce cuando un formulario pasa a estar activo
Click	Se produce cuando se hace clic sobre el formulario
Closed	Se produce cuando se ha cerrado el formulario
Closing	Tiene lugar cuando se cierra el formulario.
Deactive	Se produce cuando el formulario pierde el enfoque y no está activo
Doubleclick	Se produce cuando se hace doble clic sobre el formulario
Gotfocus	Se produce cuando recibe el enfoque
KeyPress, KeyDown, KeyUp	Se producen cuando se pulsa una tecla, cuando se presiona y cuando se libera la tecla

LostFocus	Se produce cuando pierde el enfoque
MouseDown,MouseUp	Se produce cuando se está pulsando un botón y cuando se libera
MouseMove	Se produce cuando el ratón se mueve sobre el formulario
Move	Se produce cuando se mueve el formulario
Resize	Se produce cuando se cambia de tamaño el formulario

Ya hemos visto en alguna ocasión el evento Load(). Es muy interesante porque nos permite escribir instrucciones que se van a realizar en el proceso de carga del formulario, con lo que podremos rellenar de datos cuadros de listas y otros elementos para que cuando se muestre esté listo para el usuario.

### 3. Interfaces de múltiples documentos: MDI

Las aplicaciones MDI (interfaz de múltiples documentos) permiten mostrar varios documentos al mismo tiempo, cada uno de ellos en su propia ventana. Las aplicaciones MDI suelen tener un elemento de menú Ventana con submenús que permiten cambiar entre ventanas o documentos. Por ejemplo:



Es una imagen del Acrobat Reader que tiene abiertos tres documentos simultáneos: interfaz de múltiples documentos MDI. Lo que comentábamos arriba es que suele ser habitual incluir un menú "Ventana" para permitir al usuario desplazarse por estas ventanas y ponerlas en varias disposiciones: cascada, mosaico, horizontal...

En este tipo de programas con varios formularios distinguiremos entre un formulario principal que es el que hace de contenedor y unos hijos que son cada una de las ventanas o formularios contenidos en él.

### 3.1. Crear un formulario MDI padre

Como hemos comentado el formulario MDI padre es el alma de una aplicación MDI. Es el contenedor de múltiples documentos (formularios hijos) dentro de una aplicación MDI. Puedes utilizar la propiedad "IsMdiContainer" para crear un formulario padre.

Sigue estos pasos para verlo:

1. Crea un nuevo formulario y abre la ventana de código
2. En las propiedades del formulario le indicamos que va a ser un MDI padre con:

`Me.IsMdiContainer=True` ' es decir va a ser un contenedor MDI

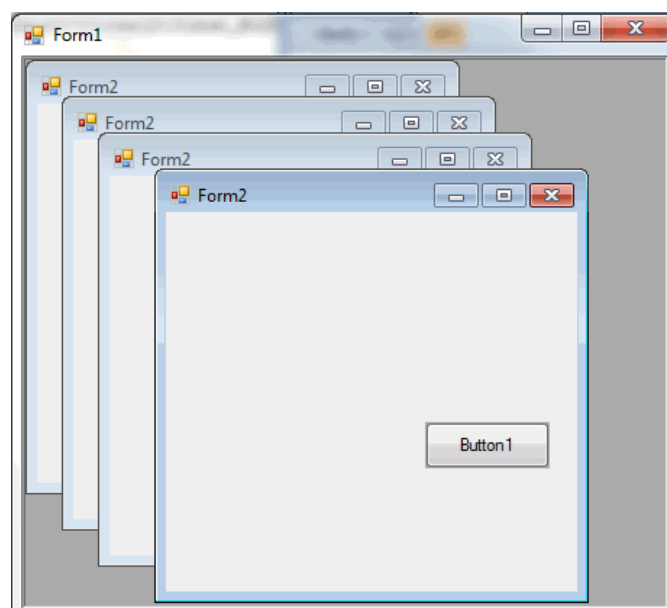
Cuando se utilizan este tipo de formularios es conveniente iniciar la aplicación a pantalla completa o maximizada. Por ejemplo, estableciendo la propiedad "WindowState" a "Maximized"

### 3.2. Crear formularios MDI hijos

Los formularios MDI hijos son los que se muestran dentro del formulario padre en las aplicaciones MDI. Vamos a realizar un ejemplo paso a paso para ver cómo podemos crear formularios hijos dentro de un padre.

### Ejercicio: Formularios MDI

Crear un formulario MDI padre y un formulario hijo. Poner un botón en el formulario y crear con él varias instancias del formulario hijo.



### 3.3. Determinar un MDI hijo activo

Debemos saber por supuesto qué formulario hijo está activo en cada momento. En una aplicación de este tipo el formulario activo es el que tiene el enfoque. Por ejemplo, si el usuario selecciona la opción Cerrar debemos cerrar el formulario activo, para esto disponemos de la propiedad "**ActiveMDIChild**" del formulario padre que nos indica el formulario activo en ese momento. Este ejemplo cierra el formulario MDI hijo activo:

```
Me.ActiveMDIChild.Close()
```

### 3.4. Organizar formularios MDI hijos

Los MDI padres suelen tener una opción de menú que organiza los formularios hijos: en cascada, horizontal, vertical... Para ofrecer esta funcionalidad podemos utilizar el método "LayoutMDI" en el formulario padre en la enumeración "**MDILayout**". Esta lista de opciones (MDILayout) tiene estos valores:

Valor	Descripción
ArrangedIcons	Muestra unos iconos que representa cada formulario hijo en la parte inferior de la ventana padre
Cascade	Muestra los iconos en cascada
TileHorizontal	Muestra los formularios acoplados horizontalmente.
TileVertical	Muestra los formularios acoplados verticalmente.

Por ejemplo, supón que ya tienes el menú creado y quieres poner el código para que se ajusten los formularios horizontalmente, el código sería:

```
me.LayoutMDI(MDILayout.TileHorizontal)
```

## 4. Menús

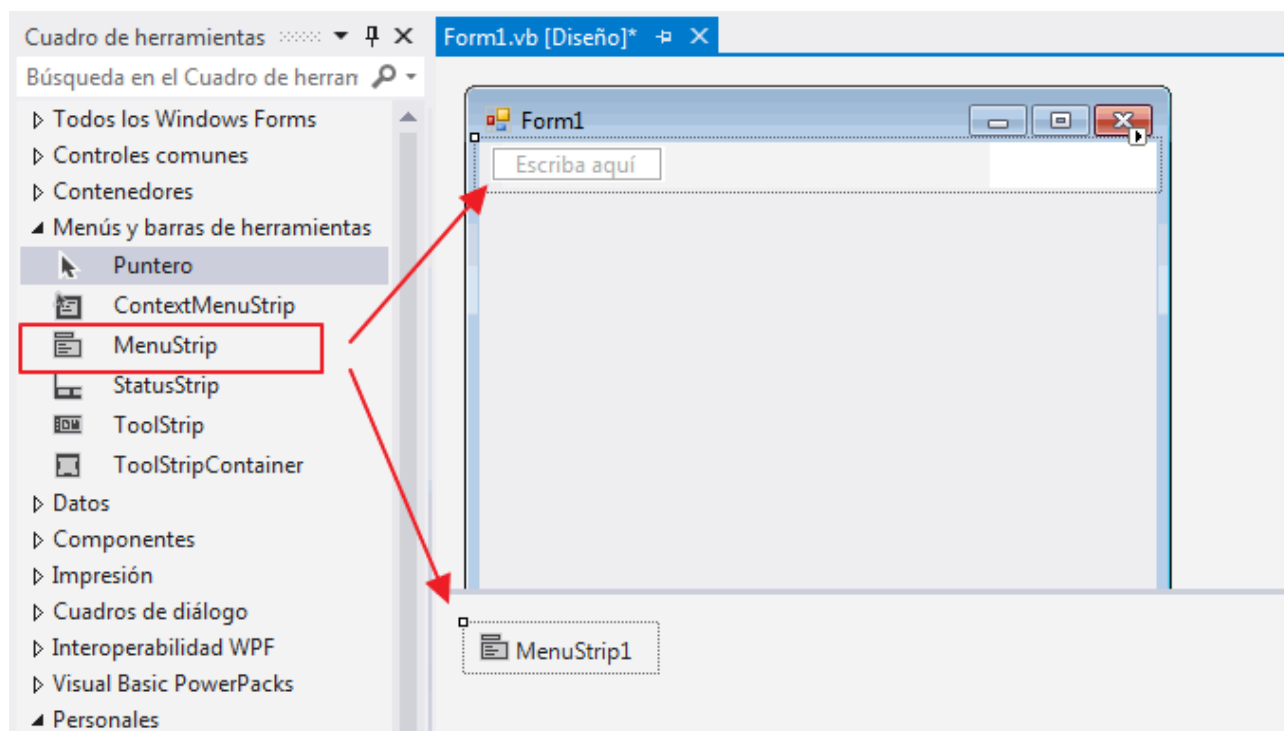
Como es lógico nuestras aplicaciones también tendrán sus menús para que el usuario seleccione las opciones disponibles. Vamos a ver los menús desde dos sitios diferentes: uno desde el diseñador que incorpora el IDE de VS.NET y otro desde código. Por un lado, crearemos un menú en tiempo de diseño y por otro con la **clase** correspondiente controlaremos y manejaremos completamente el menú existente y otros nuevos como pueden ser los contextuales. Los tipos de controles de menú son los siguientes:

- **ContextMenuStrip**: menús contextuales, que se activan con el botón derecho del ratón
- **MenuStrip**: menú principal del formulario
- **StatusStrip**: barra de estado
- **ToolStrip**: barra de herramientas con botones
- **ToolStripContainer**: contenedor de los diferentes botones que pueden componer las barras de herramientas.

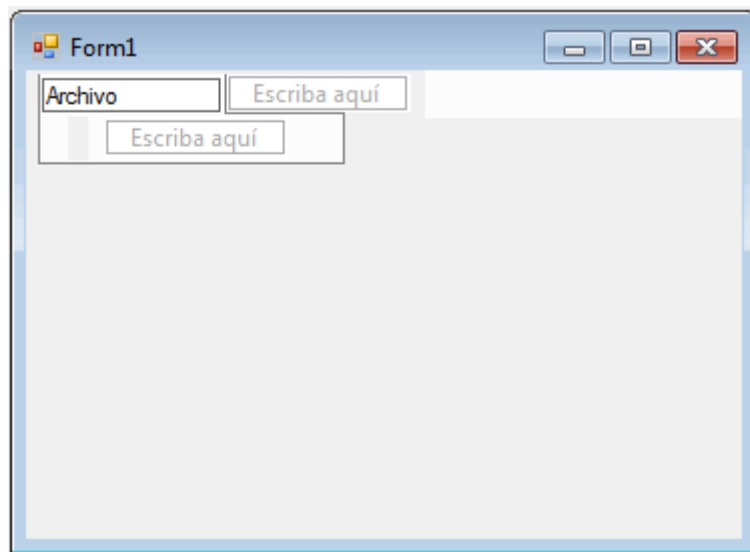
### 4.1. Creación de menús con el IDE

Activamos un formulario donde queramos insertar el menú. Seleccionamos de la lista de controles disponible el llamado "**MenuStrip**" y lo arrastramos al formulario.

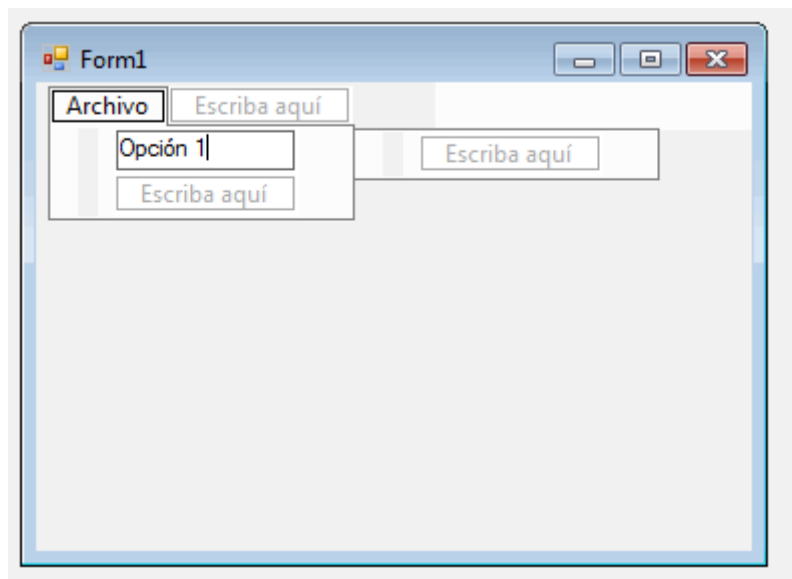
Puesto que es un control oculto no tiene interfaz de usuario y aparecerá en la parte inferior:



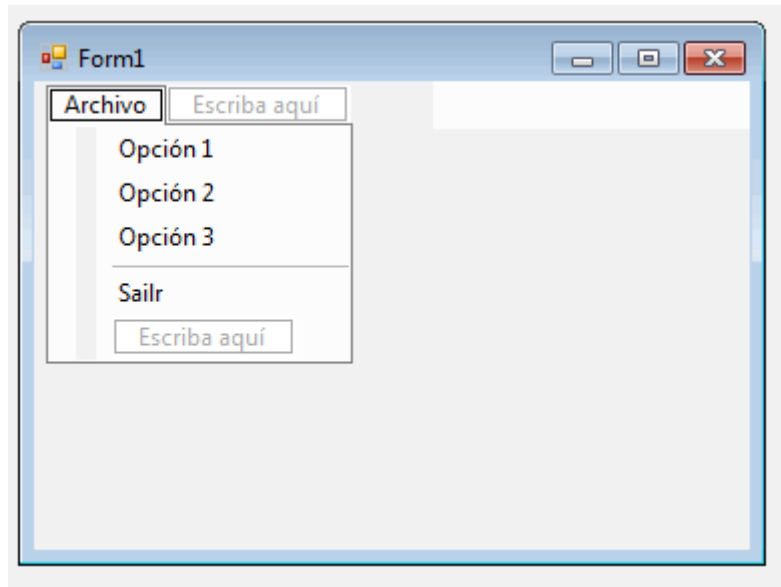
Verás en la parte superior del formulario cómo está preparado para introducir opciones de menú. Vamos a hacer clic en el mensaje que indica "Escriba aquí" y escribimos "Archivo":



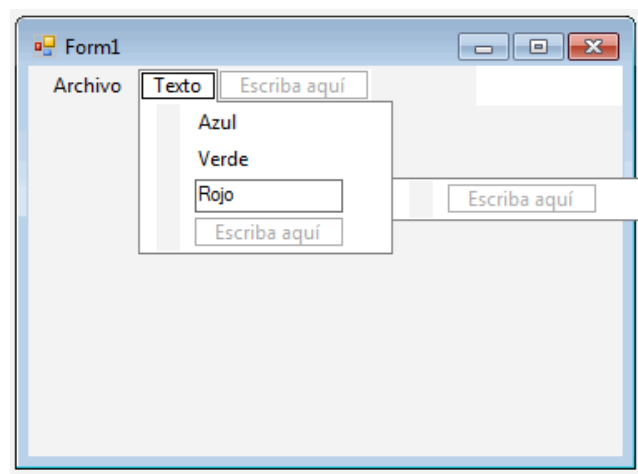
Se activan automáticamente las opciones de la derecha para poner otro menú y en la parte inferior para poner ya el primer elemento del desplegable de "Archivo". Ahora vamos a escribir en este menú una nueva opción "Opcion1":



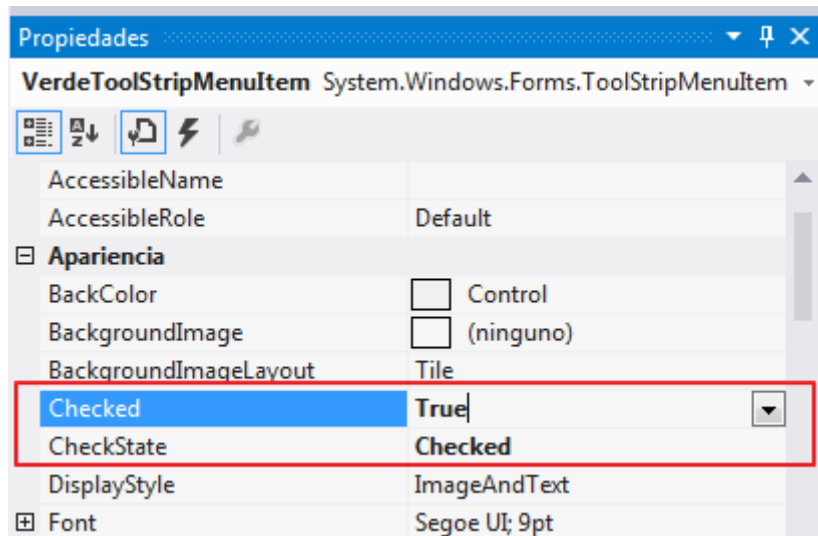
Verás además que se activa la posibilidad de desplegar un menú a la derecha de éste pero seguiremos escribiendo debajo dos opciones más: "Opcion2" y "Opcion3". Ahora quiero poner un separador entre las opciones de menú porque la última va a ser la opción "Salir" y quiero separarla de las demás. Para esto simplemente escribimos en el nombre un guión simple "-" y seguimos a la siguiente opción. Verás cómo ha dibujado una línea de separación. Finalmente introducimos la opción "Salir":



Sigamos con más opciones. Introduce un menú a la derecha que se llame Texto y que tenga tres opciones debajo que sean "Verde", "Azul" y "Rojo":



Pulsa en la opción de menú verde y fíjate en las propiedades de este elemento:



Estas son las propiedades disponibles... lo que yo quiero hacer con este menú es poner una marca a la izquierda como si estuviera activado. Y de hecho eso es lo que indica cuál de las tres opciones o colores está activo. Esto normalmente lo cambiaremos en el código pero debemos dejar una opción ya activada por defecto para que la vea el usuario. Para esto utilizaremos la propiedad "Checked" y la activaremos poniendo el valor "True". Verás como aparece la marca de activado a la izquierda de este elemento...

**Nota** Fíjate en la parte superior del cuadro de propiedades, puedes ver la clase a la que corresponde este elemento: **System.Windows.Forms.ToolStripMenuItem**

Ahora veremos las propiedades más importantes para seguir añadiendo opciones. Otra cosa es manejarlos en tiempo de ejecución que lo veremos más adelante.

- Nombre. Como todos los elementos u objetos de nuestra interfaz debe tener un nombre único. Es muy recomendable ponerle unos nombres legibles para facilitar luego el código.
- Text. Texto que muestra al usuario
- Enabled. Permite habilitar/deshabilitar la opción de menú. Si se ha deshabilitado aparece en color gris como no activa.
- Checked. Coloca una marca en la opción de menú.
- ShortcutKeys. Son los conocidos atajos de teclado. Son combinaciones de teclas que activan esa opción sin tener que utilizar el menú.
- ShowShortcutKeys. Indica si debe mostrar los atajos de teclado
- Visible. Muestra u oculta una opción de menú.
- MdiWindowListItem. Es una opción especial cuando se utilizan formularios MDI, es decir, el formulario principal hace de contenedor de varios formularios. Especifica el elemento cuyo DropDown mostrará la lista de ventanas MDI.
- DisplayStyle. Para activar que se puedan poner gráficos en las opciones del menú.

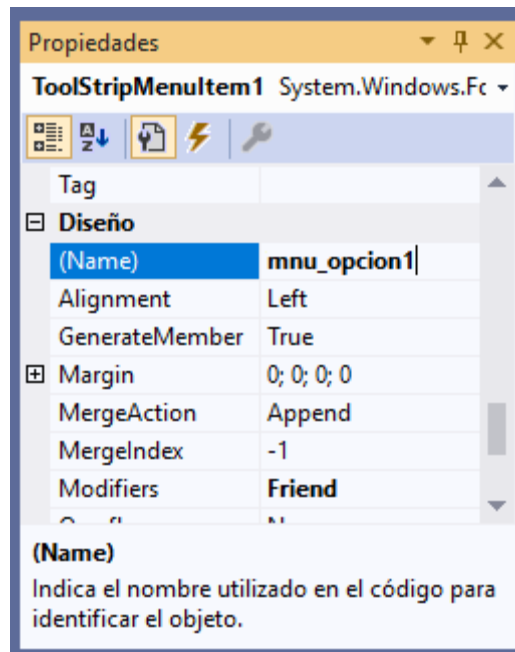
**Nota** Para dejar una letra del menú subrayada para que se active junto con la tecla "Alt" le pondremos delante el símbolo "&" en la propiedad Text. Por ejemplo "&Abrir" o "&Cerrar"

Si queremos insertar imágenes utilizaremos la propiedad "Image", como hemos visto en un ejemplo anterior.



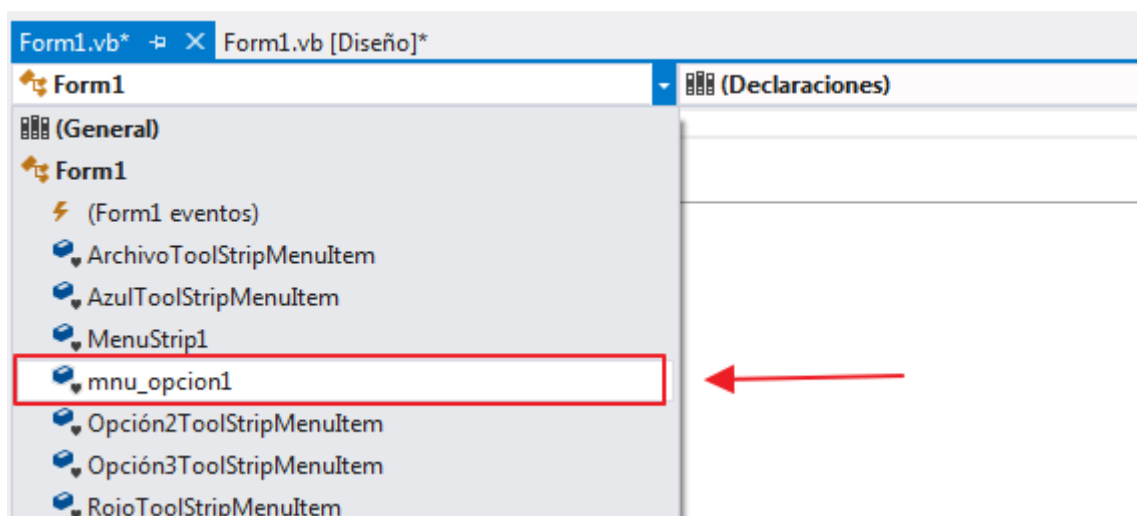
## Ejecutar el código de las opciones de menú.

Cuando ya tenemos nuestro menú con todas las opciones tenemos que decirle el código que debe ejecutar. Para esto debemos indicar en nuestro código los eventos adecuados del menú. Al añadir los menús, el IDE les ha asignado un nombre genérico "ToolStripMenuItem", muy incómodo para utilizarlo en el código. Vamos a cambiar su nombre para que sea más legible:

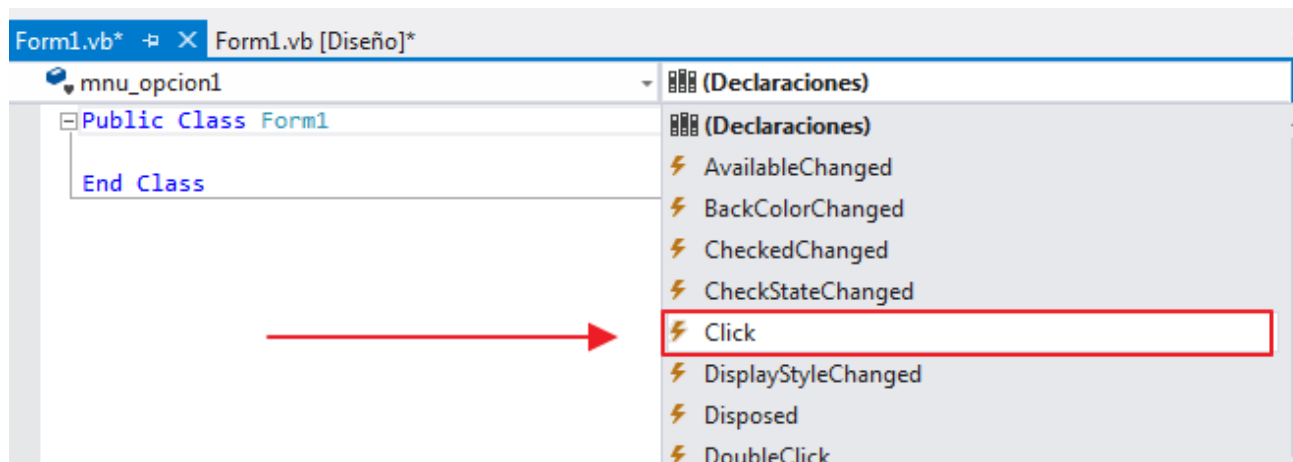


Ahora debemos localizar en nuestro código dónde está este objeto y en qué evento debo poner el código. Ya lo hemos hecho anteriormente.

Primero en el código del formulario seleccionamos el nombre del objeto en el desplegable de la izquierda.



Y luego seleccionamos el evento adecuado a la derecha, en este caso lógicamente el evento "click".



Con lo que nos escribirá el procedimiento que se ejecutará cuando se haga clic en esta opción. También podíamos haber llegado hasta aquí haciendo doble clic sobre la opción de menú deseada.

## 4.2. Manejo de los menús desde código

En muchas ocasiones tendremos que manejar dinámicamente el menú, es decir, a lo largo del programa y según en la opción en la que estemos puede que algunas opciones de menú no deban estar habilitadas. Vamos a ver cómo manejamos desde el código las propiedades que hemos visto antes.

### Habilitar una opción de menú

Ya hemos visto antes que la propiedad para esto debe ser Enabled. Luego si queremos habilitar una opción:

```
mnuEjemplo.Enabled = True
```

Y para deshabilitarla

```
mnuEjemplo.Enabled = False
```

### Marcar una opción

Utilizamos la propiedad Checked:

```
mnuEjemplo.Checked = True
```

## Mostrar u ocultar una opción de menú

```
mnuejemplo.Visible = True
```

o para ocultar

```
mnuejemplo.Visible = False
```

Si esto lo ponemos en una de las opciones del menú principal se queda visible o invisible todo el menú desplegable.

## Cambiar el texto de un menú

Muy fácil, utilizaremos la propiedad Text:

```
mnuEjemplo.Text = "Este es el nuevo texto"
```

## Añadir y eliminar elementos de menú por código

Queremos añadir y eliminar elementos de nuestro menú en tiempo de ejecución. Para esto no utilizaremos las propiedades ya que éstas sirven para modificar su aspecto y comportamiento, lo que tendremos que utilizar son los métodos disponibles en los "MenuItem"

Añadir más elementos de menú:

```
mnuprimero = New ToolStripMenuItem()  
mnuprimero.Text = "Otra opción"  
'Otro elemento  
mnusegundo = New ToolStripMenuItem()  
mnusegundo.Text = "Opción 1"  
mnusegundo.Checked = True  
'Otro elemento  
mnutercero = New ToolStripMenuItem()  
mnutercero.Text = "Opción 2"  
'Ya están creados.  
'ahora voy a enlazar los dos últimos para que dependan de mnuprimero.  
mnuprimero.DropDownItems.AddRange(New ToolStripMenuItem() {mnusegundo,  
mnutercero})  
'Ahora añadimos mnuprimero a la lista de menús de primer nivel  
Menu.Items.AddRange(New ToolStripMenuItem() {mnuprimero})
```

Eliminar elementos de menú:

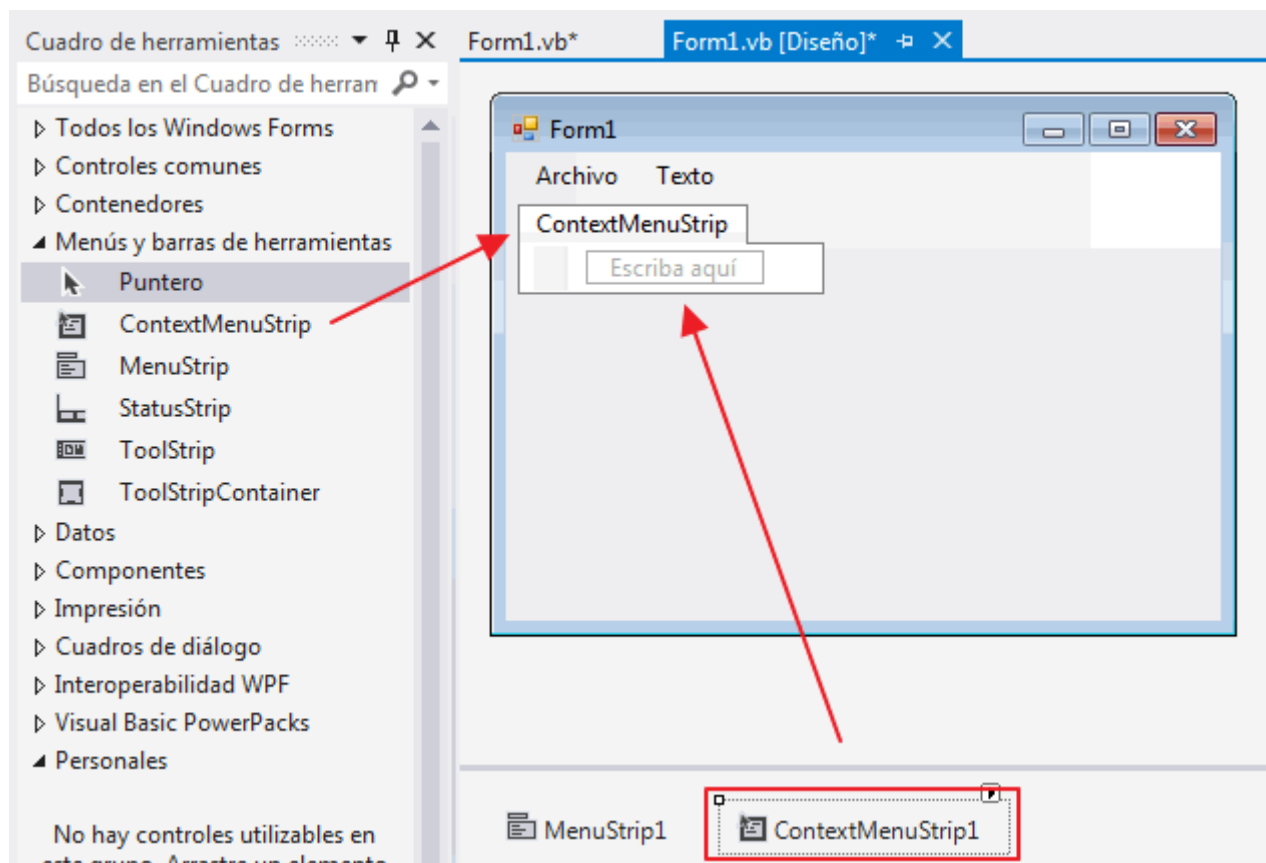
```
Menu.Items.Remove(mnusegundo)
```

### 4.3. Menús contextuales

Como ya hemos comentado son los menús que se activan con el botón derecho del ratón y dependiendo del contexto, pantalla o formulario en donde nos encontremos podemos seleccionar un menú u otro.

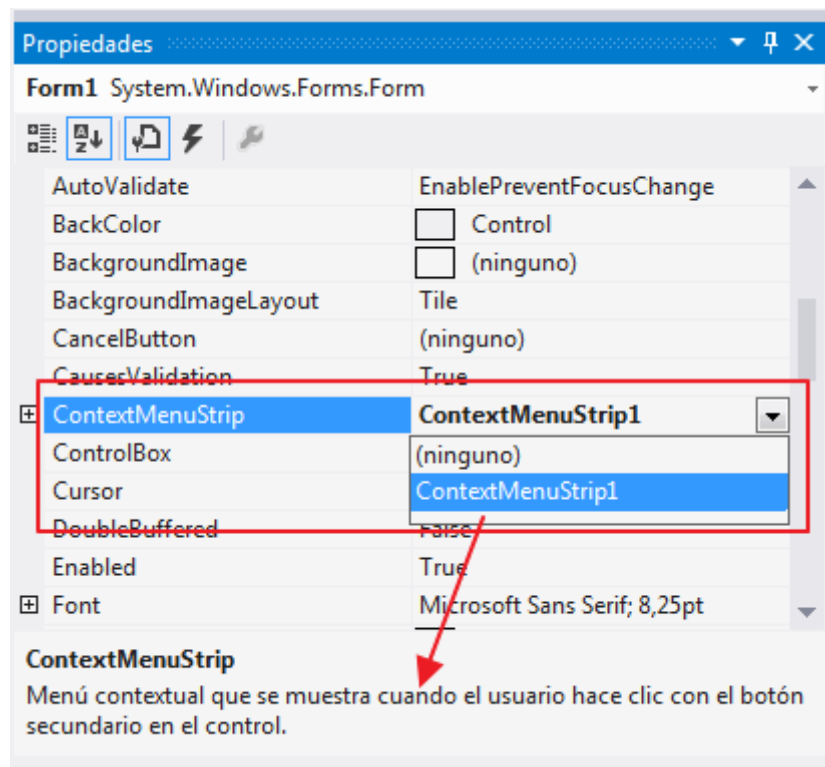
El componente **ContextMenuStrip** proporciona a los usuarios acceso a los comandos de menú de uso frecuente. Con frecuencia, los menús contextuales están asociados a un control para adecuar su menú más específicamente a las necesidades del usuario (por ejemplo, los comandos de menú **Deshacer**, **Cortar**, **Copiar**, **Pegar**, **Eliminar** y **Seleccionar todo**, que suelen formar parte del menú contextual del control **TextBox**).

Procederemos de la misma forma que con los otros menús. Añadimos un control de tipo "ContextMenuStrip" a nuestro formulario y escribimos las opciones que nos interesen:

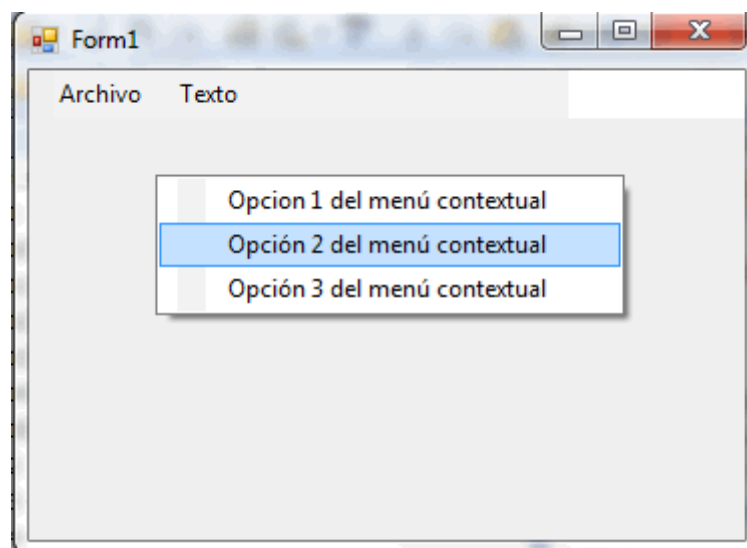


Vemos en la parte inferior cómo hemos añadido este control al formulario. Una vez creado ahora tenemos que asociarlo con un control. Podemos asociarlo a un cuadro de texto, un gráfico, un formulario,... lo que queramos. Supongamos que lo queremos asociar al formulario, es decir, cuando se haga clic con el botón derecho del ratón en el formulario queremos que se active

nuestro menú. Para esto vamos a utilizar la propiedad **ContextMenuStrip** que está disponible en muchos controles. Búscala en las propiedades del formulario:




Al desplegar la lista verás que aparece el menú contextual que hemos creado antes. Perfecto, sólo falta probarlo, ejecutamos el programa y:



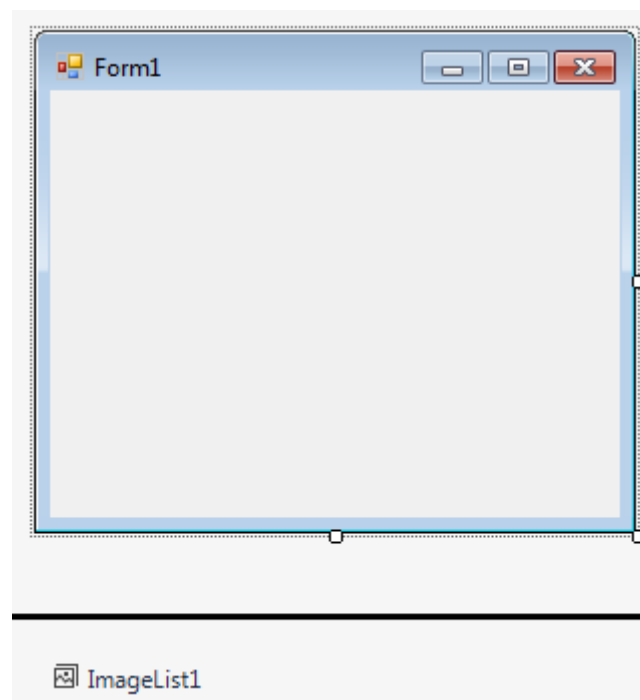
## EJERCICIO: Menús

Crear un menú dentro de un formulario que cambie el tamaño de la letra y el color de un texto escrito dentro de una etiqueta.

## 5. Control de lista de imágenes. ImageList

 **ImageList** Aunque todavía no hemos empezado a trabajar con los controles vamos a adelantarnos con la definición de uno de ellos, el control de imágenes o ImageList.

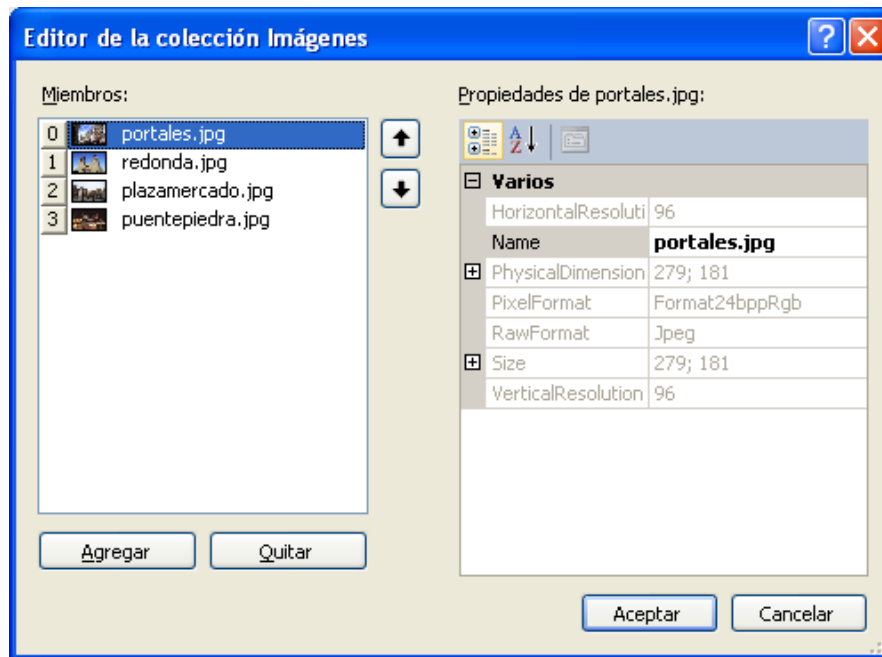
Este control es muy sencillo ya que simplemente es un contenedor de imágenes, es decir, almacena una colección de imágenes. Al añadirlo a nuestro proyecto aparecerá en la parte inferior como control no visible. Es decir, no muestra nada al usuario pero es utilizado por otros controles para utilizar las imágenes que tiene dentro almacenadas:



Tiene varias propiedades aunque la más importante es "Images" que es la colección de las imágenes. Las otras propiedades son:

Propiedad	Descripción
ColorDepth	Número de colores utilizado para procesar las imágenes
Images	Colección de imágenes
ImageSize	Tamaño de las imágenes individuales en la lista de imágenes
TransparentColor	Color tratado como transparente

La primera propiedad indica la profundidad de color o números de colores, puede ser de 8 bits para iconos o logotipos o de hasta 32 bits para fotografías. Pero veamos la propiedad "Images", al pulsarla tenemos como en otras colecciones la lista de miembros y sus propiedades:



En la lista de la izquierda iremos añadiendo las imágenes y a la derecha podremos ver las propiedades de cada una de ellas. También podemos añadir más elementos mediante:

```
Me.ImageList1.Images.Add(New Bitmap("ejemplo.gif"))
```

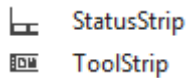
Pero lo utilizaremos pocas veces ya que es en tiempo de diseño cuando preparamos las imágenes para los iconos de las barras u otros elementos de nuestra interfaz.

## EJERCICIO: Lista de imágenes

En un Windows Form, utilizando un control ImageList haz que se muestren en un control PictureBox una colección de imágenes de Logroño. Para ello utilizarás un botón que a cada pulsación del mismo hará que se muestre una.

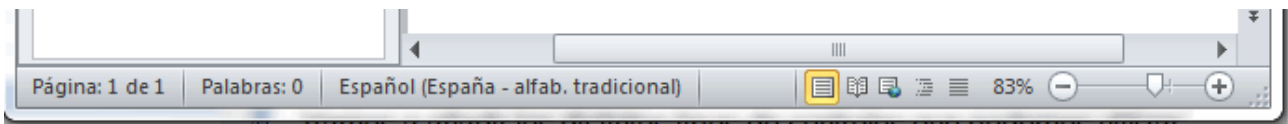


## 6. Barras de estado y herramientas



Hasta esta versión, las barras de herramientas y de estado eran muy distintas y cada uno tenía su particular colección de objetos y programación. En esta versión todo es mucho más coherente y vamos a tener muchos elementos en común, facilitándonos su programación.

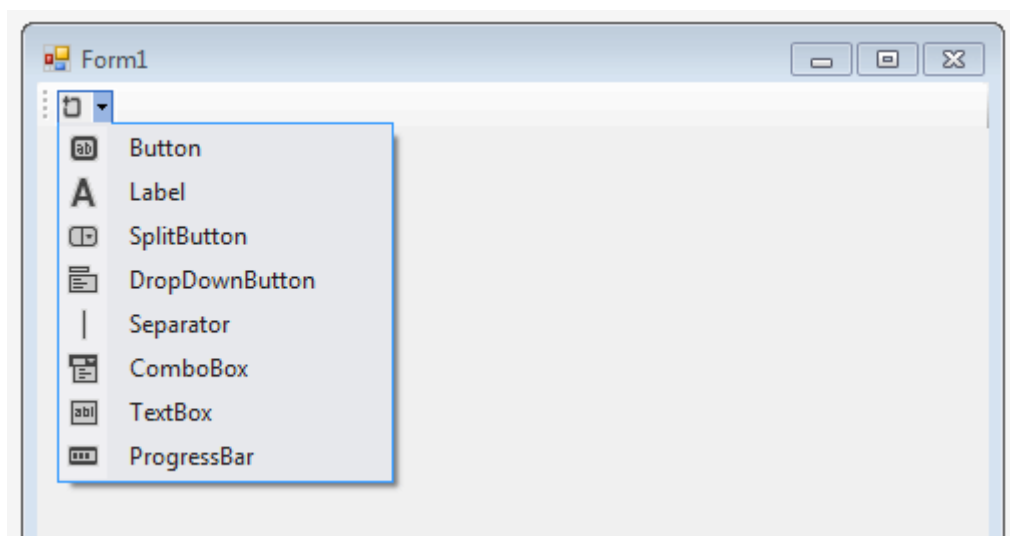
Una barra de herramientas proporciona botones en la parte superior de la ventana para acceder más rápidamente a determinadas opciones del programa. Por otro lado, la barra de estado se sitúa en la parte inferior de la ventana y muestra distintos tipos de opciones:



Las posiciones de estos dos controles la podemos cambiar para colocarlos donde queramos. Habitualmente las barras de botones se colocan en la parte superior y la barra de estado en la inferior. Vamos a añadir a un proyecto nuevo estos dos controles:



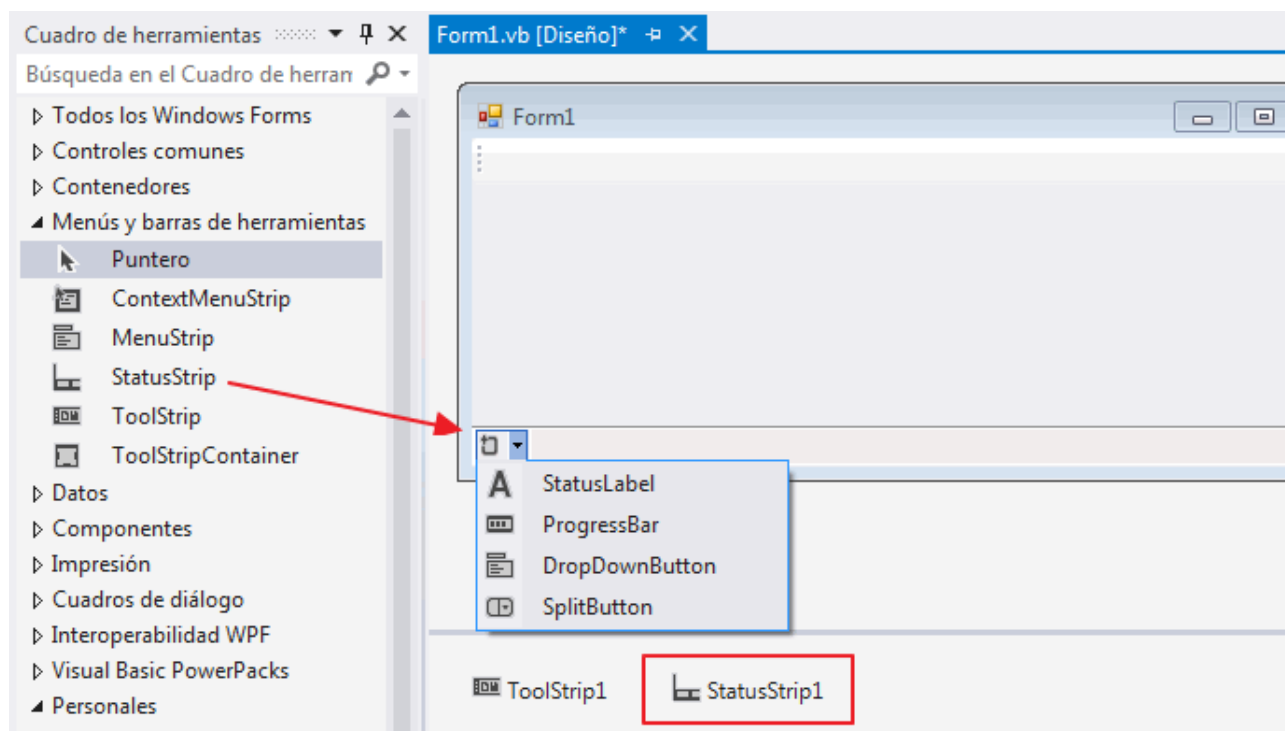
Vamos a añadir los distintos tipos de controles que podemos utilizar:



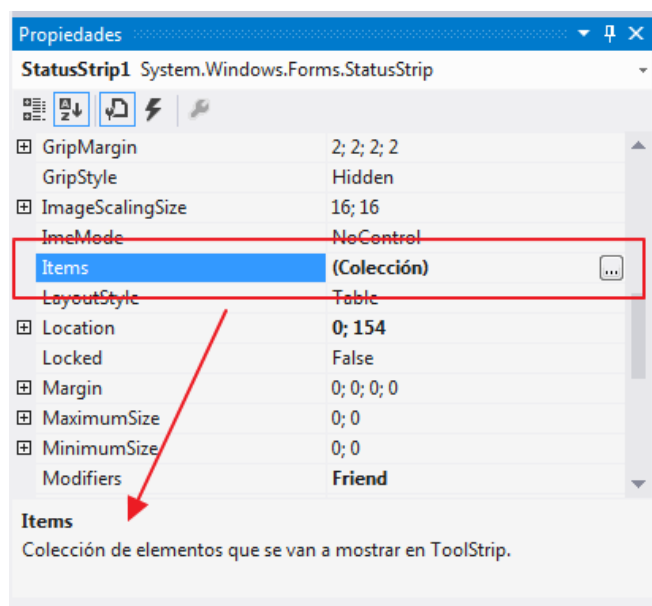


Este control ha evolucionado muchísimo y las opciones ahora son muy extensas. Sólo hay que ver la lista de objetos que podemos poner en estas barras. Las barras de botones han ido evolucionando mucho, ya no solo hay botones sino que además hay cuadros desplegables para los tipos de letra, por ejemplo. Hay cuadros de texto y hasta barras de progreso.

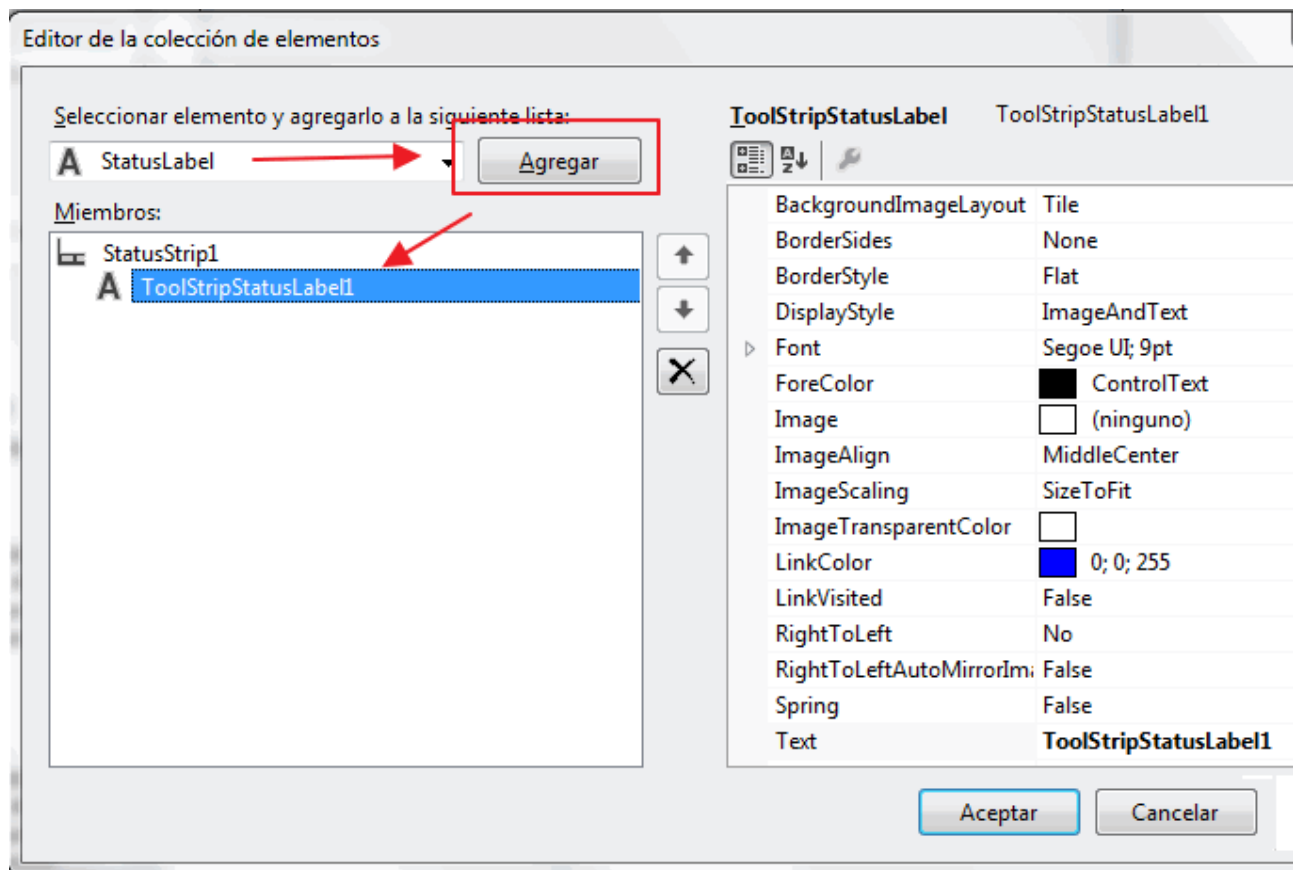
Ahora hacemos lo mismo con la barra de estado. Añadimos una, veremos que el funcionamiento es muy similar pero con algún control menos:



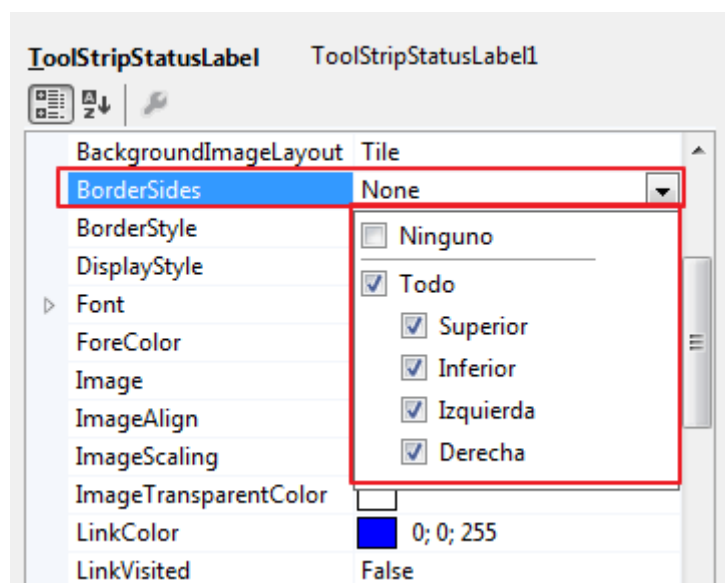
Para añadir elementos, podemos escribir directamente, como cuando hicimos los menús anteriores, o acceder a la propiedad "items"



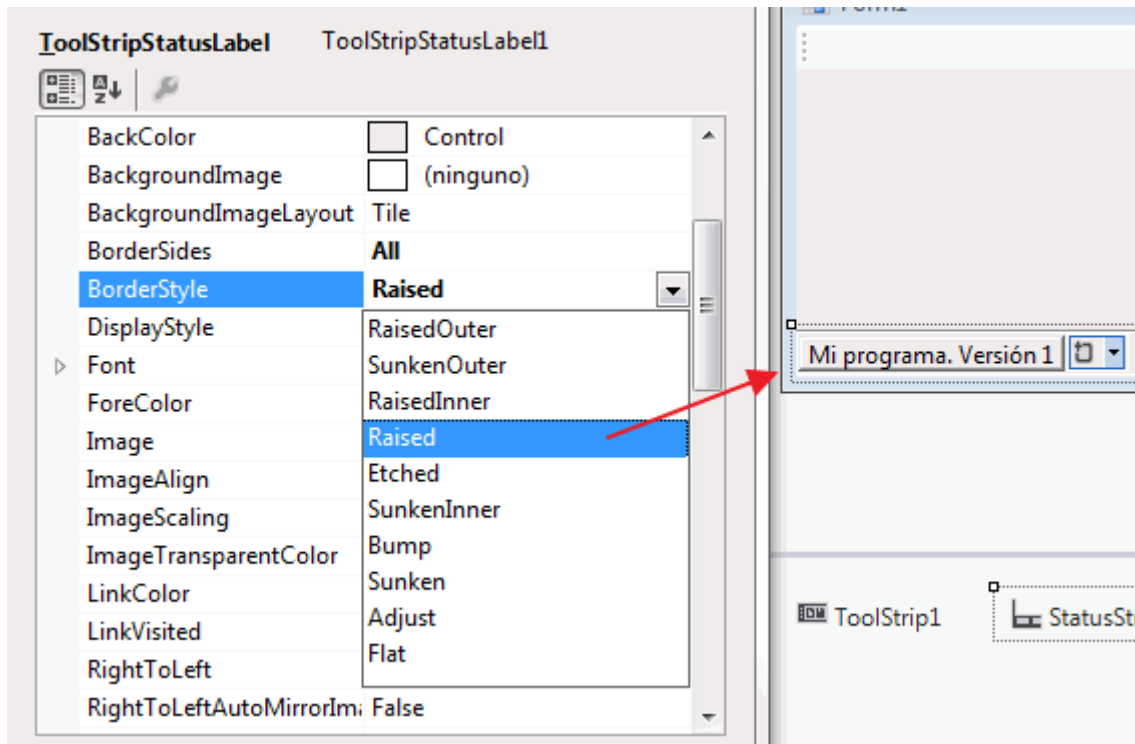
Que nos permitirá editar los elementos de una forma más ordenada:



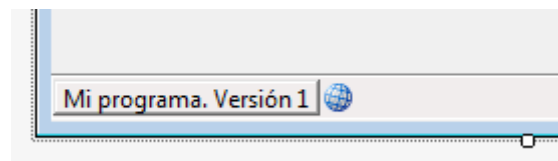
Vamos a añadir varios paneles de ejemplo de tipo texto y les ponemos algún borde:



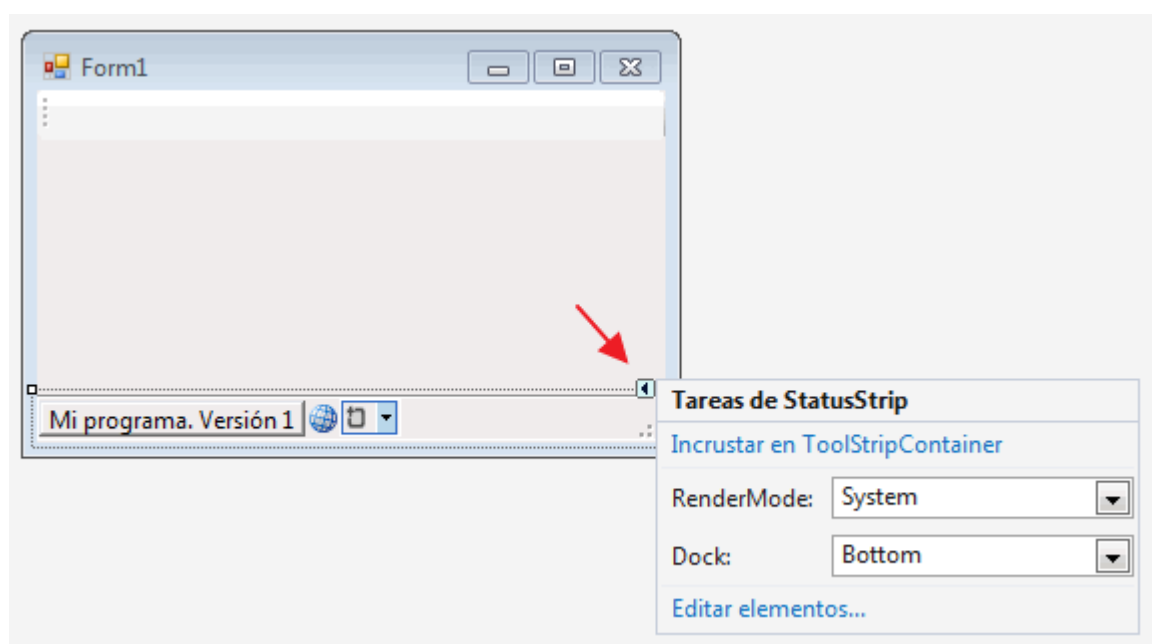
También en el estilo del contorno "borderStyle" podemos ponerle más estilos:



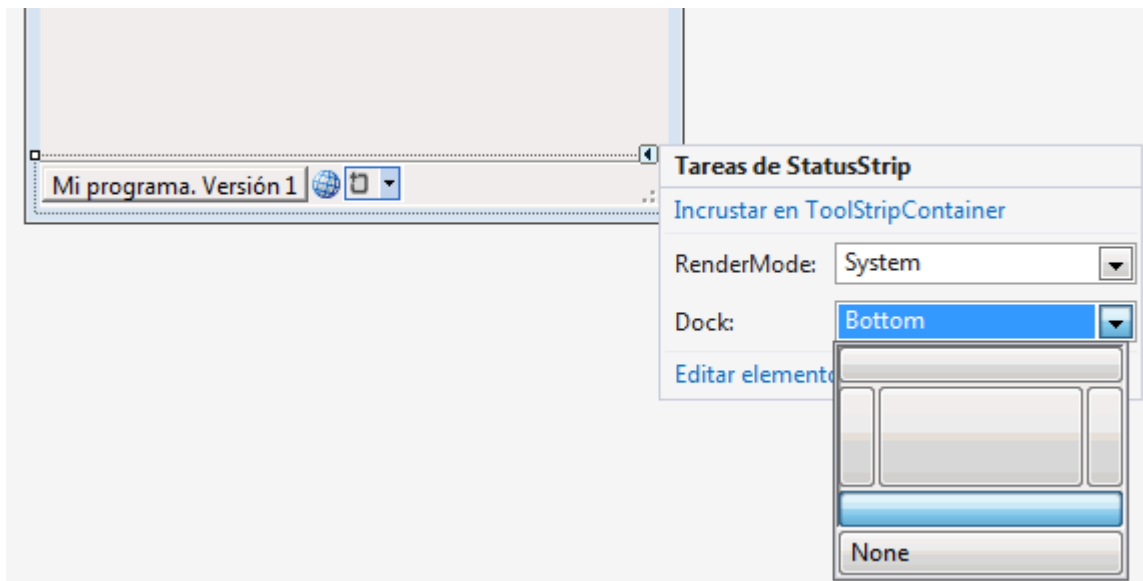
E incluso añadir imágenes:



Poco a poco vamos componiendo la barra de estado. Ahora nos fijamos en una pequeña flecha que aparece en la parte derecha y que nos muestra algunas propiedades:



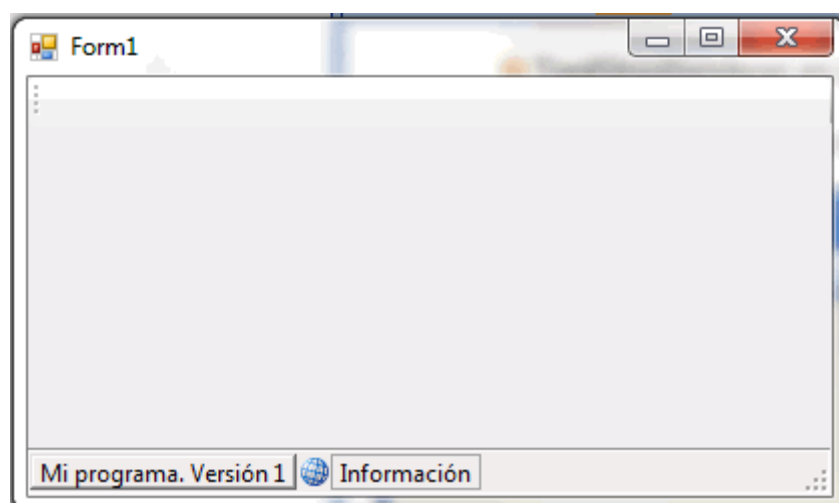
Se trata de opciones generales del control, donde podremos, por ejemplo, anclar el control en la parte de la pantalla que queramos:



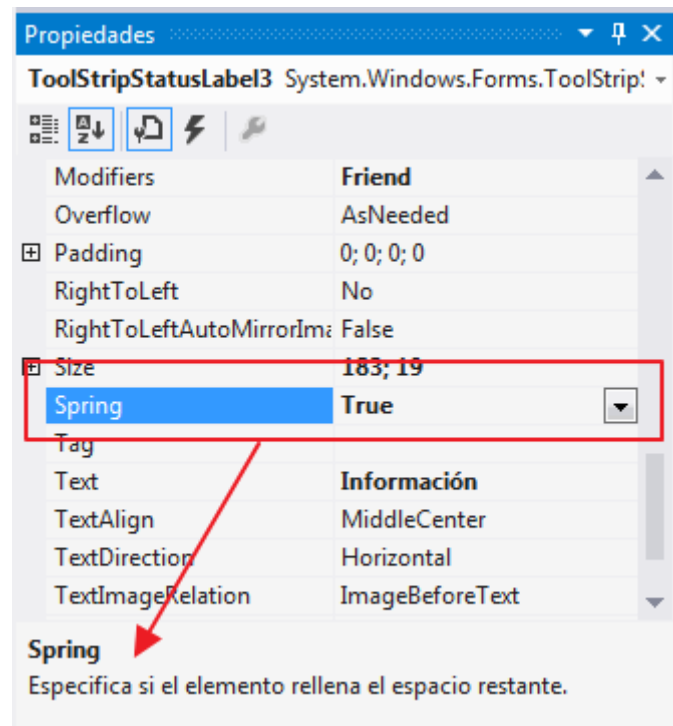
La primera opción nos coloca este control en otro del tipo "ToolStripContainer" que no vamos a ver de momento. La opción de "RenderMode" permite:

- ToolStripSystemRenderer proporciona el aspecto y estilo de su sistema operativo.
- ToolStripProfessionalRenderer proporciona el aspecto y estilo de Microsoft Office.
- ToolStripRenderer es la clase base abstracta para las otras dos clases de representación.

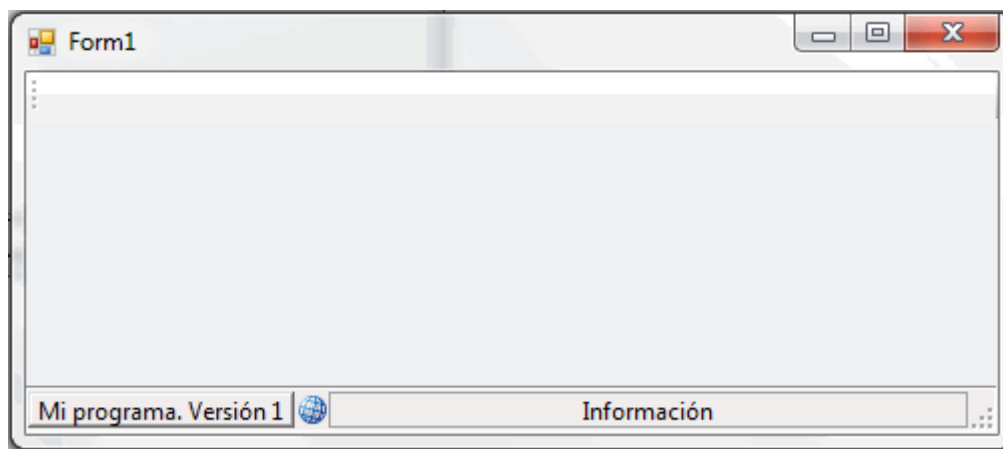
Ahora nos fijaremos en el comportamiento del ajuste de los controles. Si ejecutamos este ejemplo, que tiene dos paneles de texto, y lo ampliamos vemos que su tamaño es fijo:



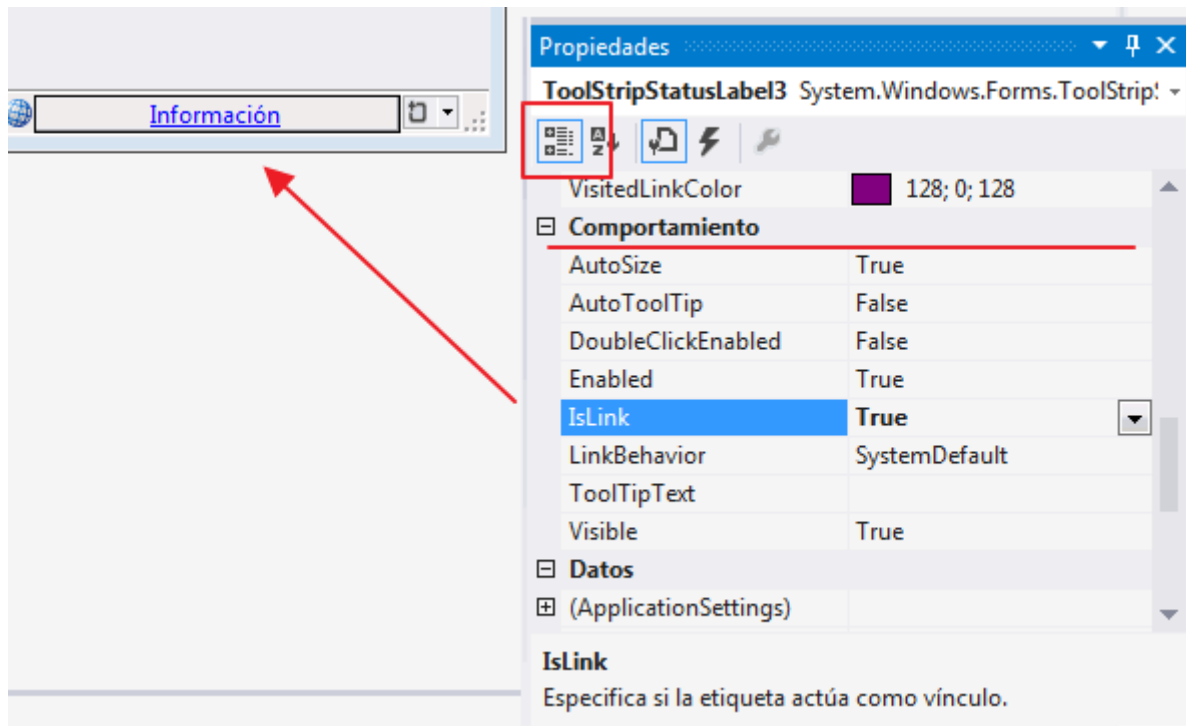
Pero queremos que ese segundo control se ajuste al total del ancho de la pantalla. Para esto vamos a cambiar la propiedad "Spring":



Ejecutamos ahora el programa. Podremos comprobar que ese segundo control ahora tiene su tamaño dinámico:

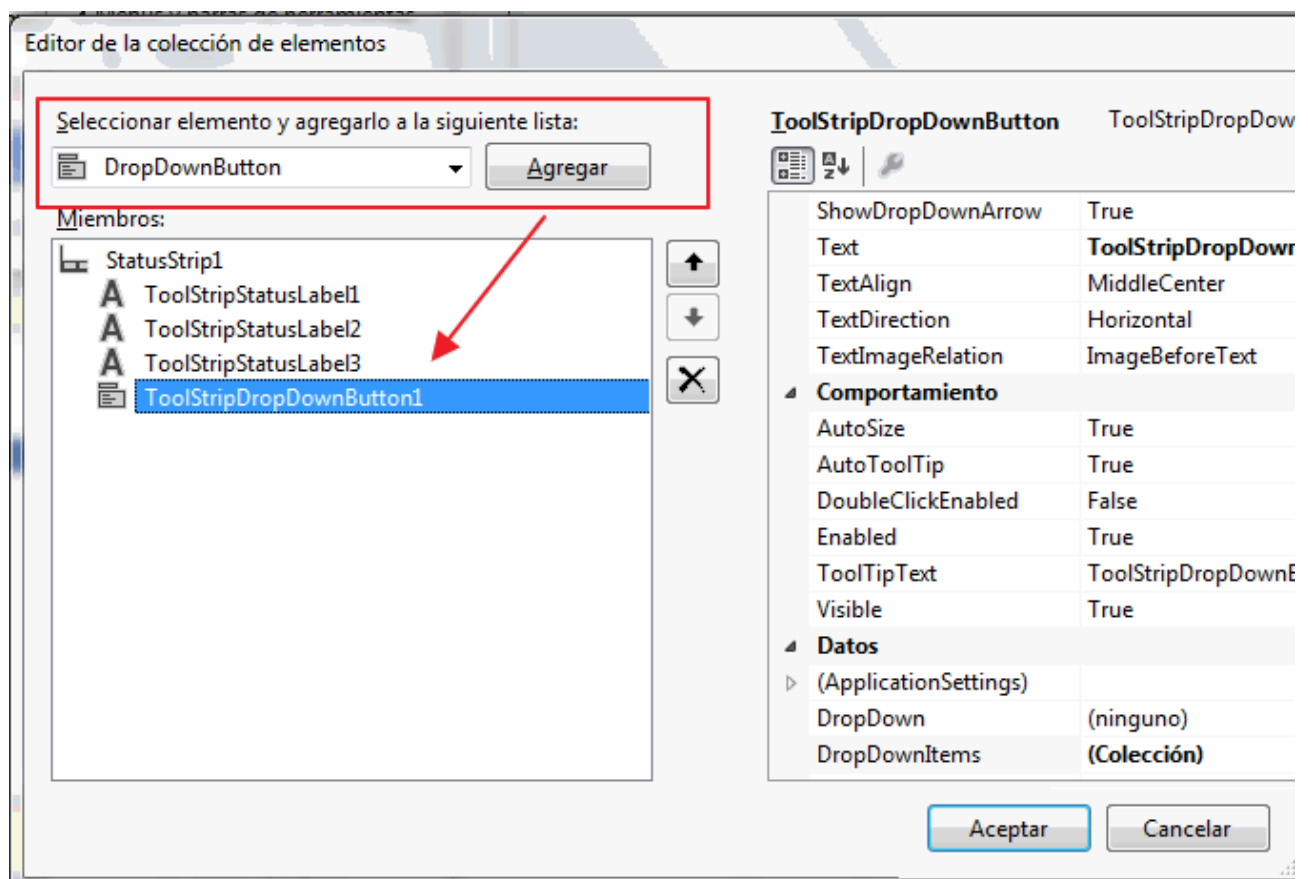


Si agrupamos las propiedades por categorías, podemos ver las que afectan al comportamiento. Por ejemplo, incluso si queremos que el texto se comporte como un hipervínculo:

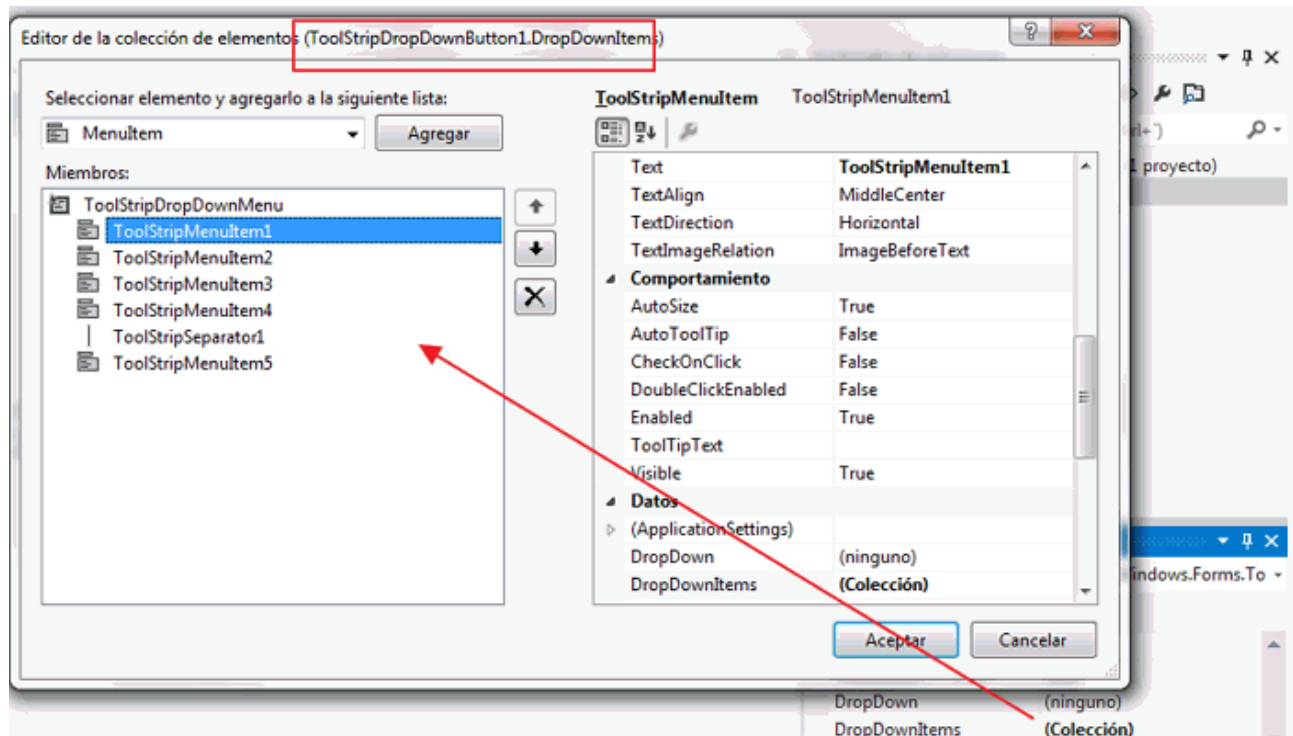


Explorando las propiedades, podemos ver incluso que puede tener un menú contextual. Las posibilidades son ilimitadas.

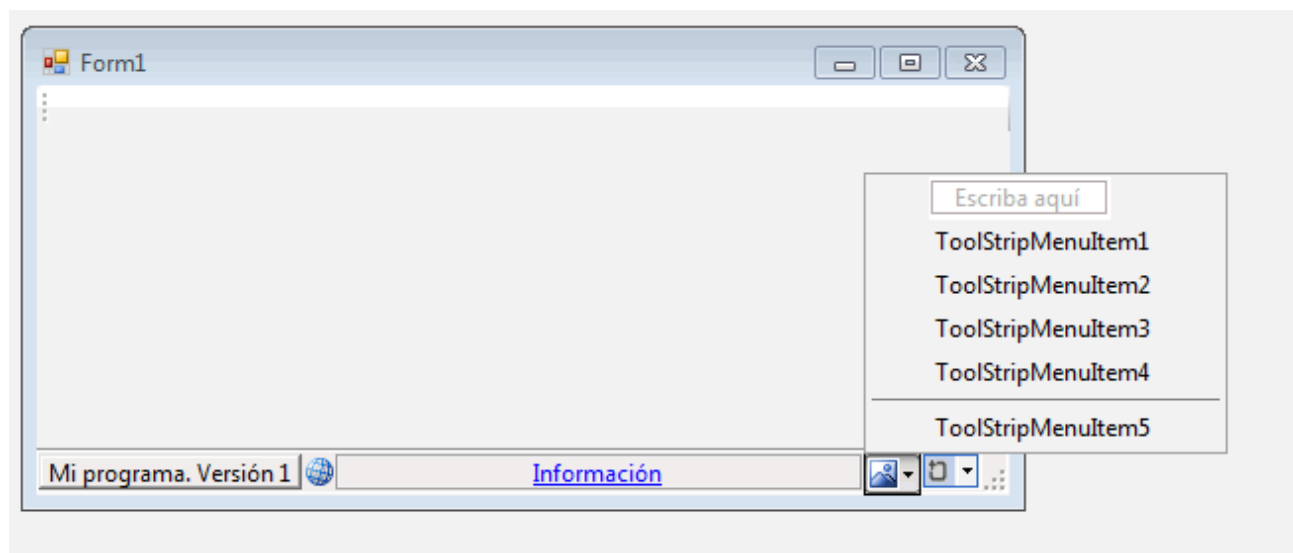
Veamos ahora el control de tipo "DropDownButton".



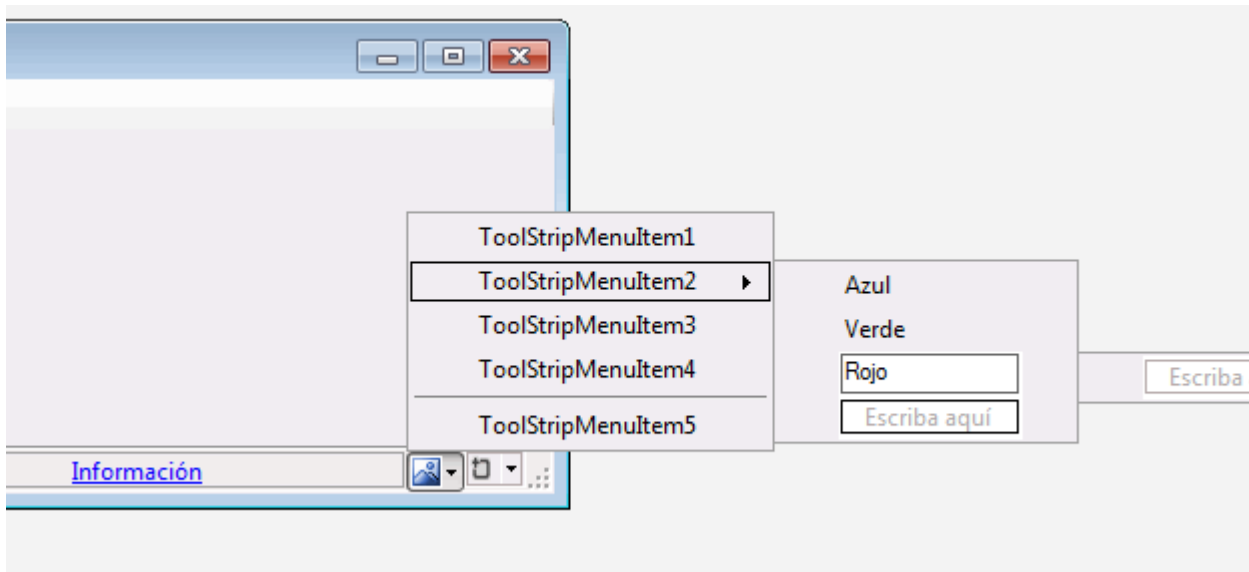
Este objeto se puede personalizar como el anterior. Si lo marcamos podemos ir a la ventana de propiedades para ver los "DropDownItems":



Como vemos en la pantalla, podemos añadir esta otra colección de objetos a este desplegable de la barra de estado. Tener la posibilidad de añadir elementos como si fuera un menú que al ejecutar la página:

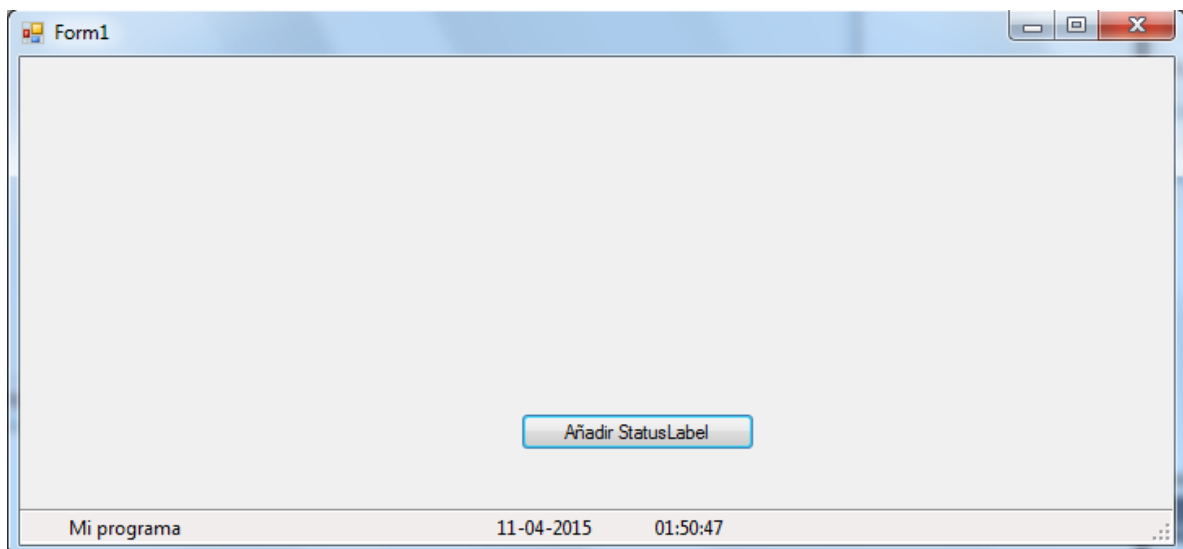


Podemos insertar controles similares a los del menú:




## EJERCICIO: Añadir StatusLabel a una barra de estado

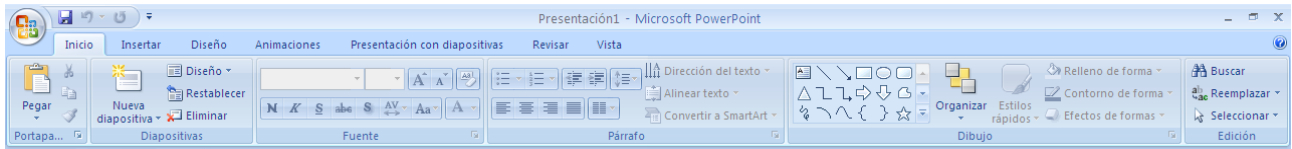
Amplía el ejemplo anterior de la barra de estado introduciendo un botón en el formulario de tal forma que al pulsarlo añada un nuevo StatusLabel a la StatusStrip.





## 7. Barras de herramientas ToolStrip

 **ToolStrip** El otro elemento habitual en los formularios principales de las aplicaciones es la barra de herramientas. Por ejemplo, en PowerPoint:



Las barras de herramientas se componen de botones e iconos que luego despliegan opciones, cuadros de texto, cuadros desplegables, botones que se excluyen entre sí (justificación de textos), separadores, ... Y todo además en unos contenedores que podemos mover y cambiar de sitio.

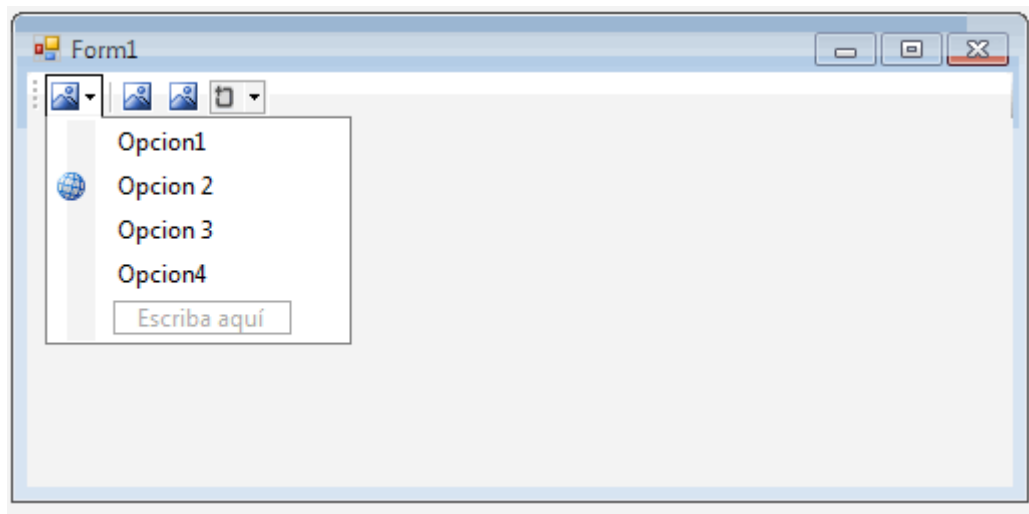
Los elementos que tenemos para utilizar en esta barra de botones son:

- ToolStripButton. Crea un botón de barra de herramientas que admite texto e imágenes.
- ToolStripLabel. Representa una clase ToolStripItem no seleccionable que representa texto e imágenes y opcionalmente con hipervínculos
- ToolStripSplitButton. Representa una combinación de un botón estándar situado a la izquierda y un botón de lista desplegable situado a la derecha
- ToolStripDropDownButton. Es como el ToolStripButton, pero muestra un área desplegable cuando el usuario hace clic en él.
- ToolStripSeparator. Representa una línea utilizada para agrupar elementos
- ToolStripComboBox. Representa un cuadro combinado.
- ToolStripTextBox. Representa un cuadro de texto
- ToolStripProgressBar. Representa una barra de progreso.

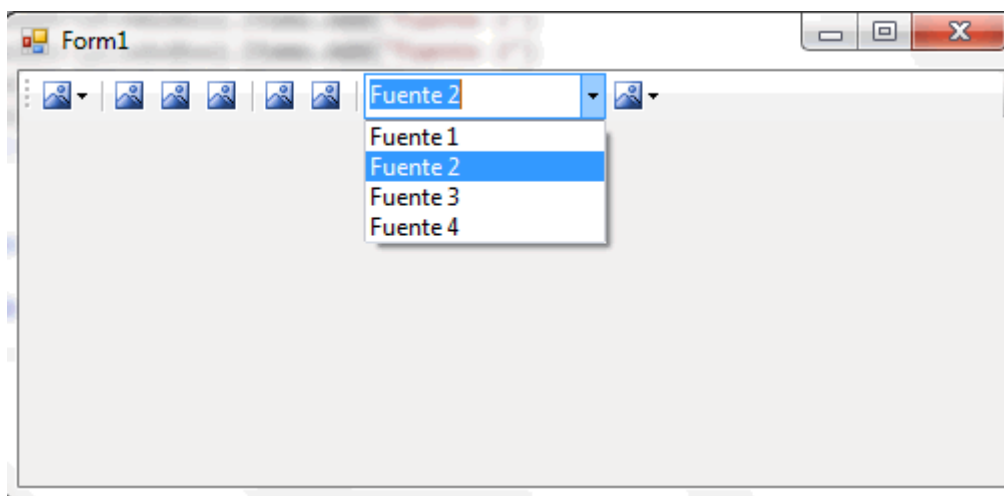
Coinciden con las opciones gráficas que ya hemos visto anteriormente:



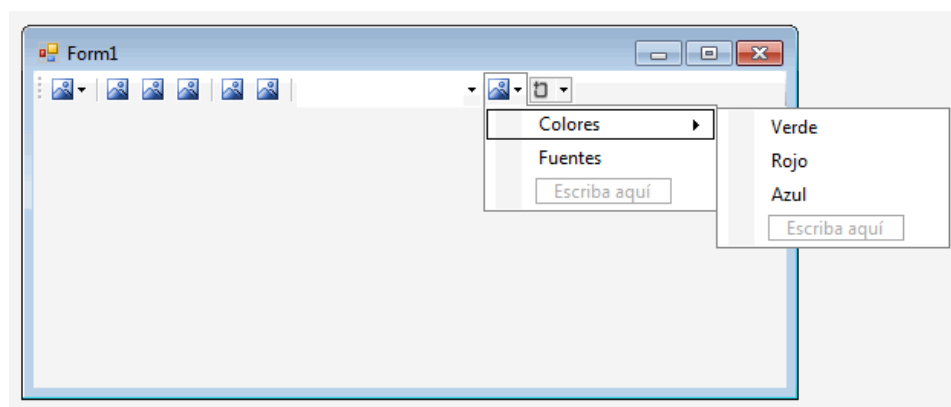
Como ejemplo vamos a intentar reproducir una barra completa de un programa, empezaremos con la opción de un icono que despliega un menú:



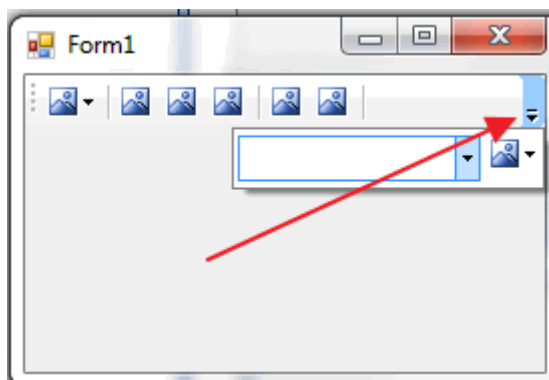
Lo que despliega es un objeto del mismo tipo que los menús, así que las opciones son las mismas. A continuación hay unos botones con iconos y con unos separadores para dejarlos agrupados. Después tenemos un cuadro desplegable:



Donde podríamos seleccionar tipos de letra, por ejemplo. Luego tenemos un control del tipo "splitbutton" que permite menús más complejos como:



Si ahora encojemos la ventana para provocar que no quepan los controles:



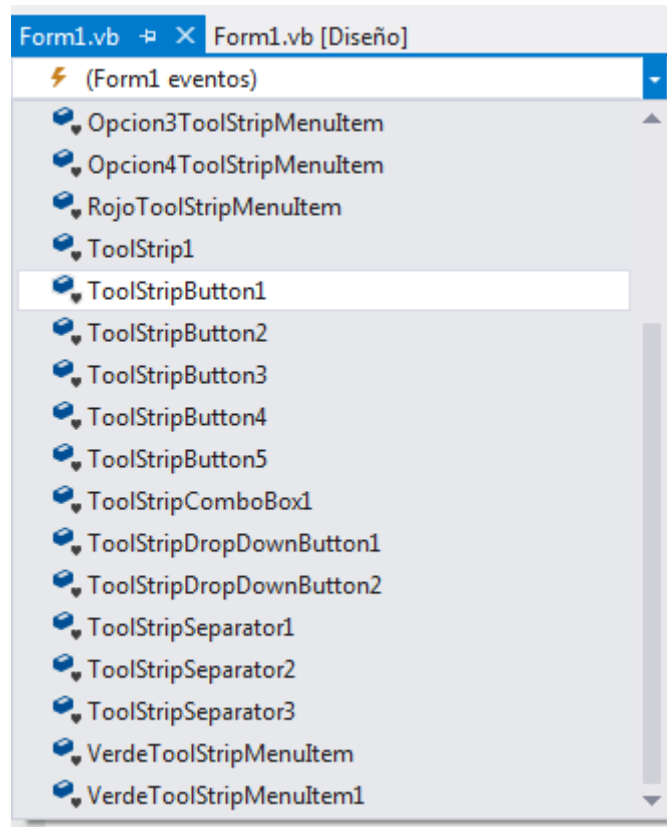
Vemos que entra en acción la propiedad que, de forma predeterminada está activa para controlar el overflow, o desbordamiento. En la parte derecha ha puesto una flecha para poder continuar dibujando el menú debajo.

Como vemos las posibilidades son ilimitadas y además del diseño podemos tener en cuenta las propiedades más importantes de ToolStrip que son:

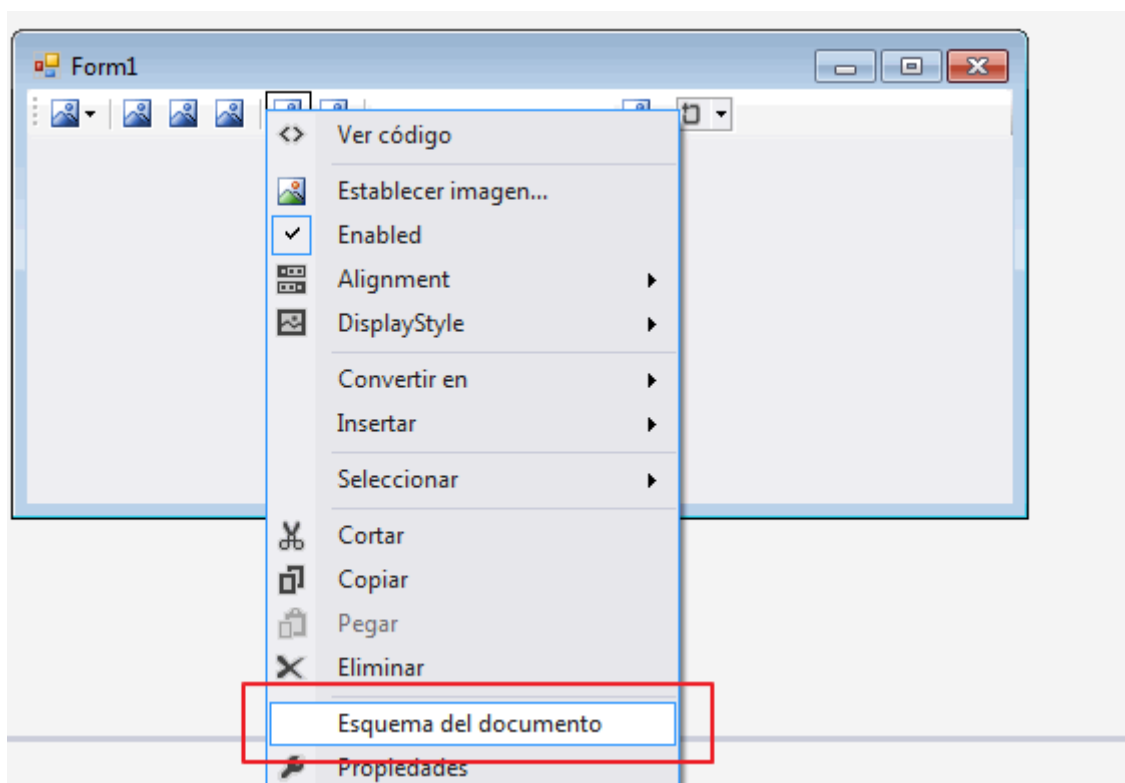
Nombre	Descripción
<b>AllowItemReorder</b>	Obtiene o establece un valor que indica si la clase ToolStrip controla de forma privada las operaciones de arrastrar y colocar, y la reordenación de elementos.
<b>AllowMerge</b>	Obtiene o establece un valor que indica si se pueden combinar varios ToolStrip, ToolStripDropDownMenu, ToolStripMenuItem y otros tipos.
<b>BackColor</b>	Obtiene o establece el color de fondo del ToolStrip.
<b>BackgroundImage</b>	Obtiene o establece la imagen de fondo que se muestra en el control.(Se hereda de Control).
<b>BackgroundImageLayout</b>	Obtiene o establece el diseño de la imagen de fondo tal como se define en la enumeración ImageLayout.(Se hereda de Control).
<b>CanOverflow</b>	Obtiene o establece un valor que indica si pueden enviar elementos de ToolStrip a un menú de desbordamiento.
<b>ContextMenu</b>	Obtiene o establece el menú contextual asociado al control.(Se hereda de Control).
<b>ContextMenuStrip</b>	Obtiene o establece el ContextMenuStrip asociado a este control.(Se hereda de Control).
<b>DefaultBackColor</b>	Obtiene el color de fondo predeterminado del control.(Se hereda de Control).
<b>DefaultDropDownDirection</b>	Obtiene o establece un valor que representa la dirección predeterminada en la que se muestra un control

	ToolStripDropDown con respecto a ToolStrip.
<b>DefaultFont</b>	Obtiene la fuente predeterminada del control.(Se hereda de Control).
<b>DefaultForeColor</b>	Obtiene el color de primer plano predeterminado del control.(Se hereda de Control).
<b>DockPadding</b>	Obtiene la configuración de relleno de acople para todos los bordes del control.(Se hereda de ScrollableControl).
<b>ForeColor</b>	Obtiene o establece el color de primer plano del control ToolStrip.
<b>GripDisplayStyle</b>	Obtiene la orientación del controlador de movimiento de ToolStrip.
<b>GripMargin</b>	Obtiene o establece el espacio que hay alrededor del controlador de movimiento de ToolStrip.
<b>GripRectangle</b>	Obtiene los límites del controlador de movimiento de ToolStrip.
<b>GripStyle</b>	Obtiene o establece si el controlador de movimiento de ToolStrip es visible o está oculto.
<b>ImageList</b>	Obtiene o establece la lista de imágenes que contiene la imagen mostrada en un elemento de ToolStrip.
<b>Items</b>	Obtiene todos los elementos que pertenecen a ToolStrip.
<b>Orientation</b>	Obtiene la orientación de ToolStripPanel.
<b>OverflowButton</b>	Obtiene el ToolStripItem que es el botón de desbordamiento para un ToolStrip con desbordamiento habilitado.
<b>Padding</b>	Obtiene o establece el relleno dentro del control.(Se hereda de Control).
<b>ShowItemToolTips</b>	Obtiene o establece un valor que indica si se mostrará información sobre herramientas en los elementos de ToolStrip.
<b>Stretch</b>	Obtiene o establece un valor que indica si ToolStrip se estira de extremo a extremo en ToolStripContainer.
<b>TextDirection</b>	Obtiene o establece la dirección en la que se dibujará el texto en ToolStrip.

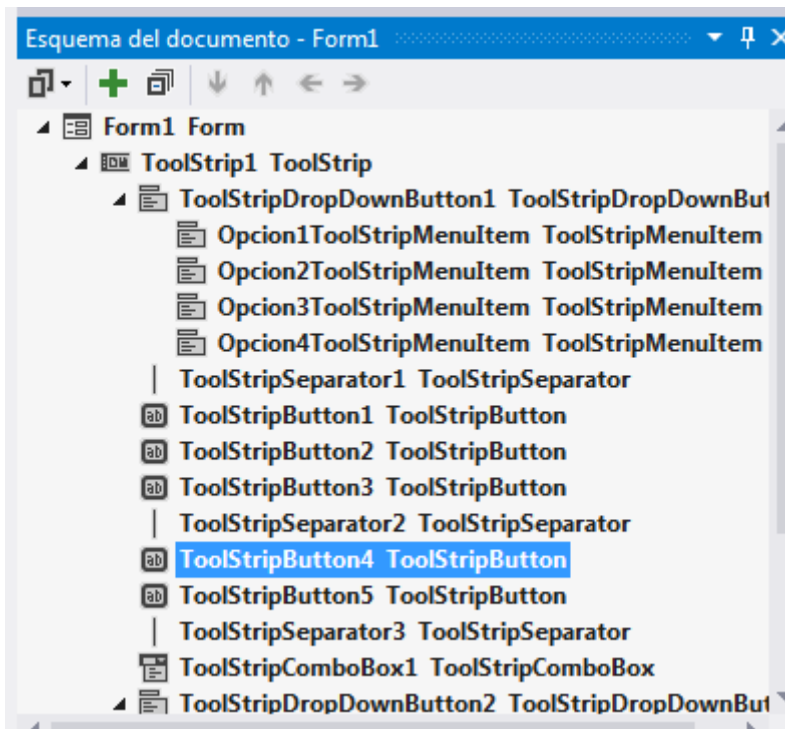
Hay un tema importante que debemos tener en cuenta y es poner nombres legibles a toda la cantidad de elementos que podemos añadir. De esta forma será mucho más fácil acceder a ellos. Sino al dejar los nombres genéricos tendremos un buen lío de nombres:



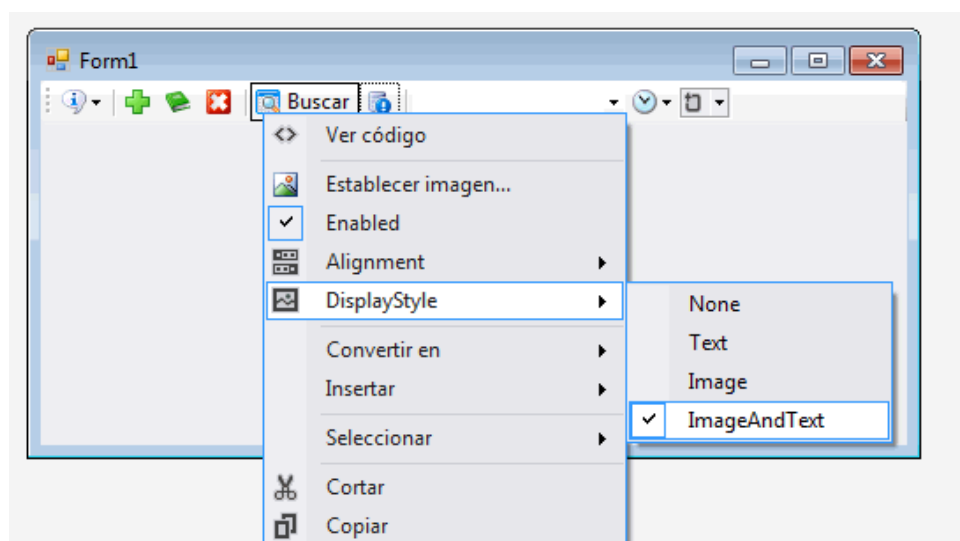
Sería muy difícil de depurar así que es mejor invertir un poco de tiempo en poner los nombres. En esta ocasión puede ser interesante la vista jerárquica del formulario. Así podemos ver la dependencia de unos controles y otros. Y, si tenemos controles anidados, es decir, unos dentro de otros, podemos verlos jerárquicamente:



Nos muestra en uno de los laterales, el esquema jerárquico de los controles del formulario:



En la personalización, podemos indicar que nos muestre el texto junto a los iconos. Por ejemplo:



En el botón se ha incluido el texto "Buscar", indicando en sus propiedades que sea del tipo "ImageAndText": imagen y texto.

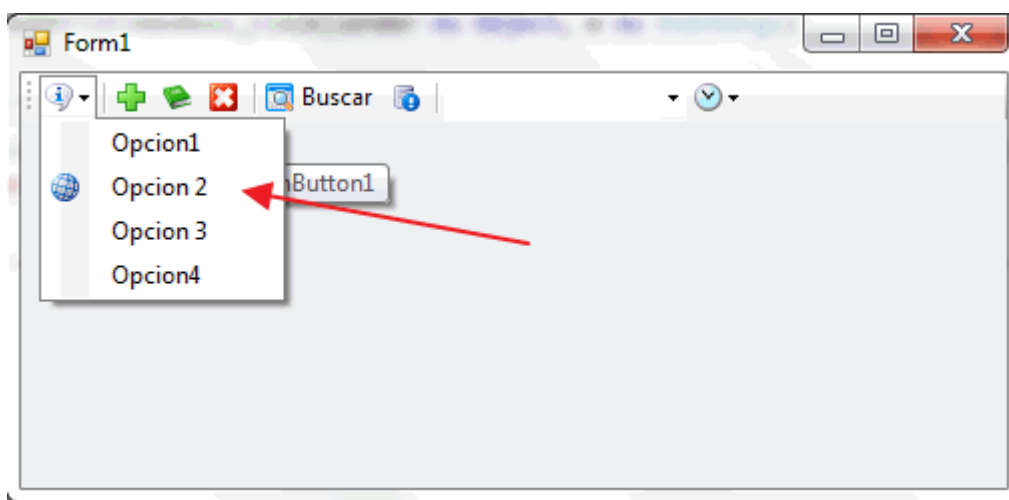
## 7.1. Detectar las opciones en el código

Obviamente una vez definida nuestra barra de herramientas debemos detectar las opciones del usuario en nuestro código para así poder realizar las acciones necesarias. Para esto debemos introducir el código necesario en los distintos eventos.

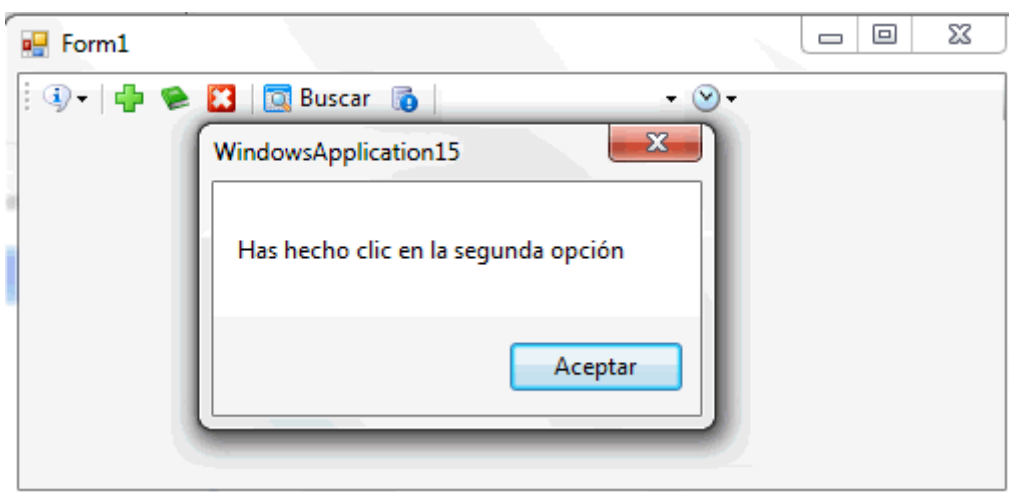
En el ejemplo anterior es fácil detectar los eventos porque cada elemento es un objeto de los que vemos enumerados en el gráfico anterior. Por tanto basta con seleccionar el elemento y luego el evento. Por ejemplo un clic en la segunda opción del menú:

```
Private Sub Opcion2ToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles Opcion2ToolStripMenuItem.Click
    MsgBox("Has hecho clic en la segunda opción")
End Sub
```

Al hacer clic en la opción:



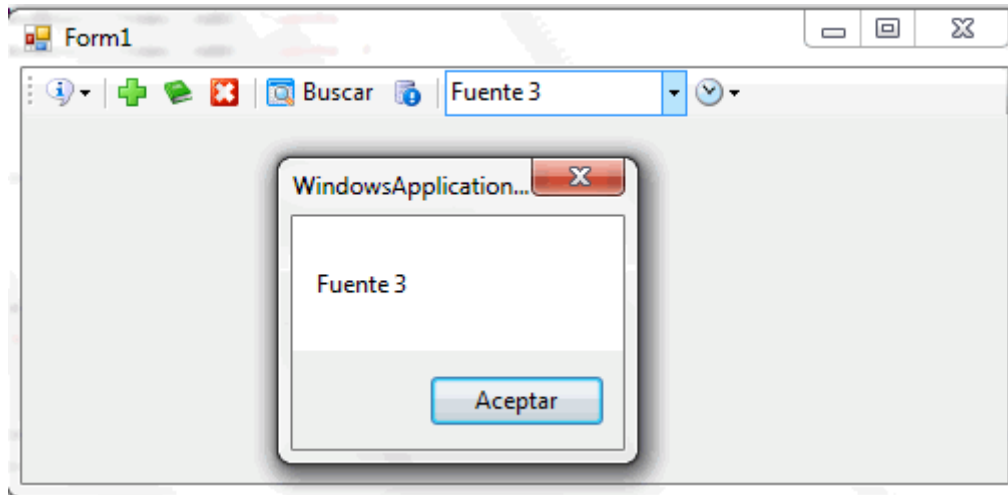
Se dispara el evento y podremos hacer la operación que queramos:



En el cuadro combinado debemos coger el valor seleccionado en su propiedad "value" utilizando el evento que ya conocemos para detectar que se ha seleccionado un elemento: `SelectedIndexChanged`. Por tanto seleccionamos el objeto del cuadro combinado y luego este evento para escribir el valor actual que corresponde con la selección:

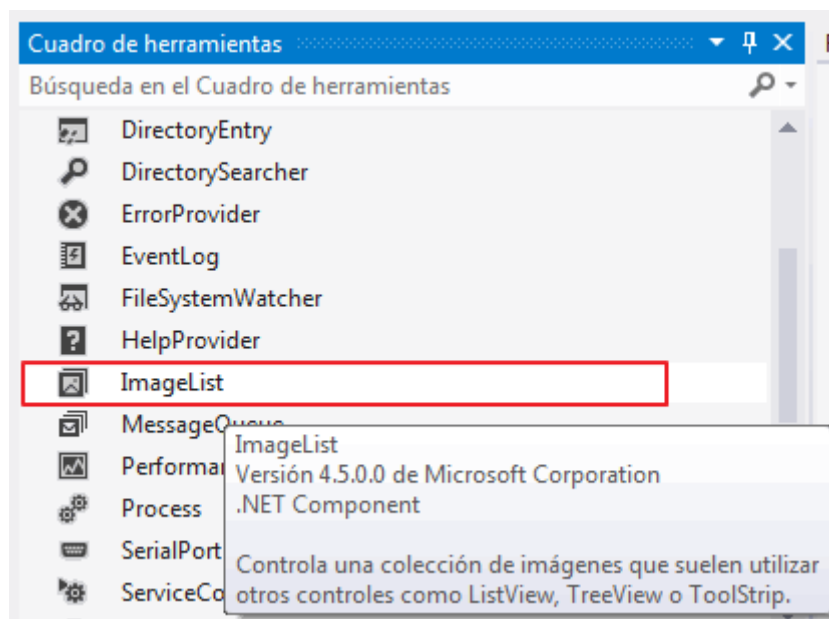
```
Private Sub ToolStripComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)  
    MsgBox(ToolStripComboBox1.SelectedItem)  
End Sub
```

Se dispara el evento y podemos capturar el valor seleccionado:



Esta técnica común de los menús, barras de botones y de estado ha simplificado enormemente la programación de todos estos elementos y, sobre todo, hacer que sean verdaderamente intuitivos.

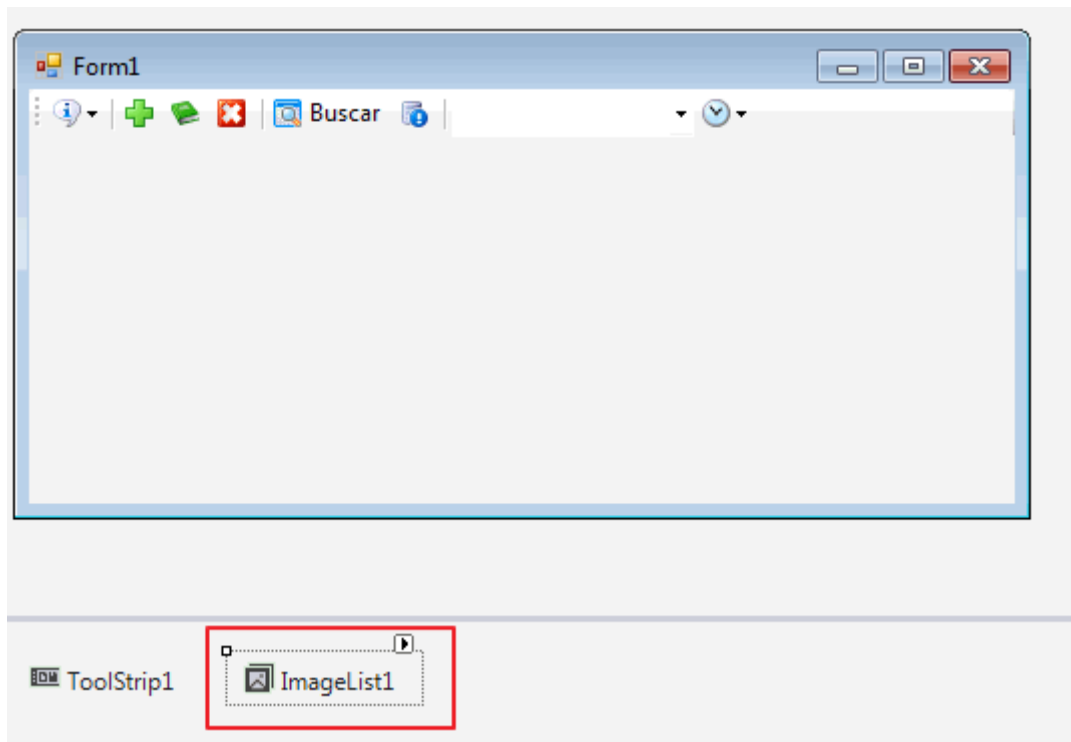
## 8. Control de lista de imágenes. ImageList



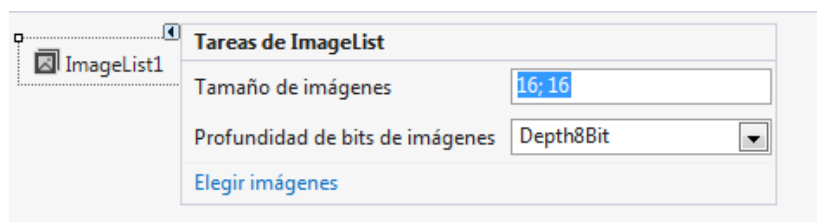
Aunque todavía no hemos empezado a trabajar con los controles vamos a adelantarnos con la definición de uno de ellos: el control de imágenes. Ya que lo necesitaremos para otros controles.



Este control es muy sencillo y su función es simplemente la de ser un contenedor de imágenes, es decir, almacena una colección de imágenes. Al añadirlo a nuestro proyecto aparecerá en la parte inferior como control no visible. Es decir, no muestra nada al usuario pero es utilizado por otros controles para utilizar las imágenes que tiene dentro almacenadas:



Aunque en esta versión de .NET tenemos la nueva forma de añadir recursos gráficos que hemos visto antes en los menús, todavía hay muchos controles que utilizan esta opción para añadir imágenes. Vamos primero a hacer clic en la fecha que tiene justo al lado del control que nos permitirá acceder a sus propiedades más importantes:

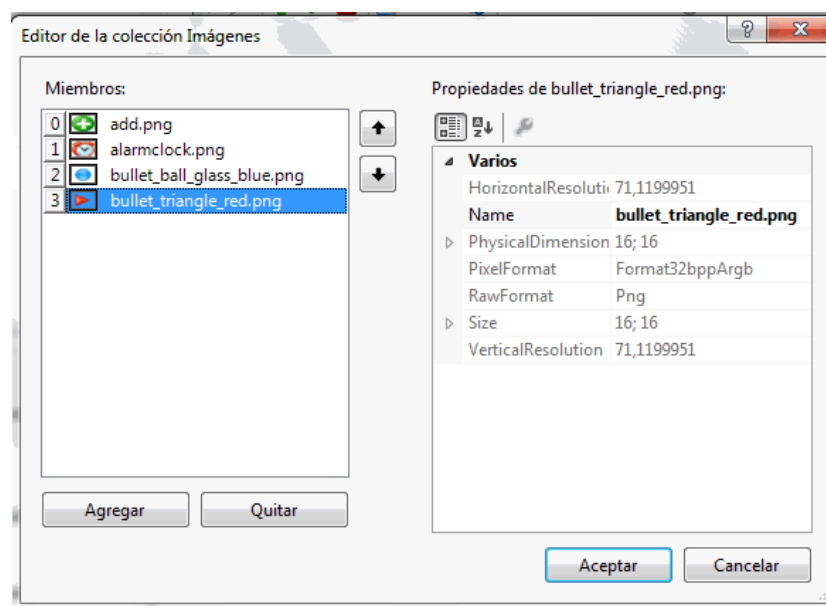


Configuraremos el tamaño de la imagen, la profundidad del color y la opción de elegir las imágenes. La profundidad de color de 8 bits, que es la predeterminada es para imágenes muy sencillas con pocos colores, que para los iconos nos basta. Si queremos más calidad porque van a ser de tipo fotográfico lo ponemos en esa opción.

Propiedad	Descripción
<b>ColorDepth</b>	Número de colores utilizado para procesar las imágenes

<b>Images</b>	Colección de imágenes
<b>ImageSize</b>	Tamaño de las imágenes individuales en la lista de imágenes
<b>TransparentColor</b>	Color tratado como transparente

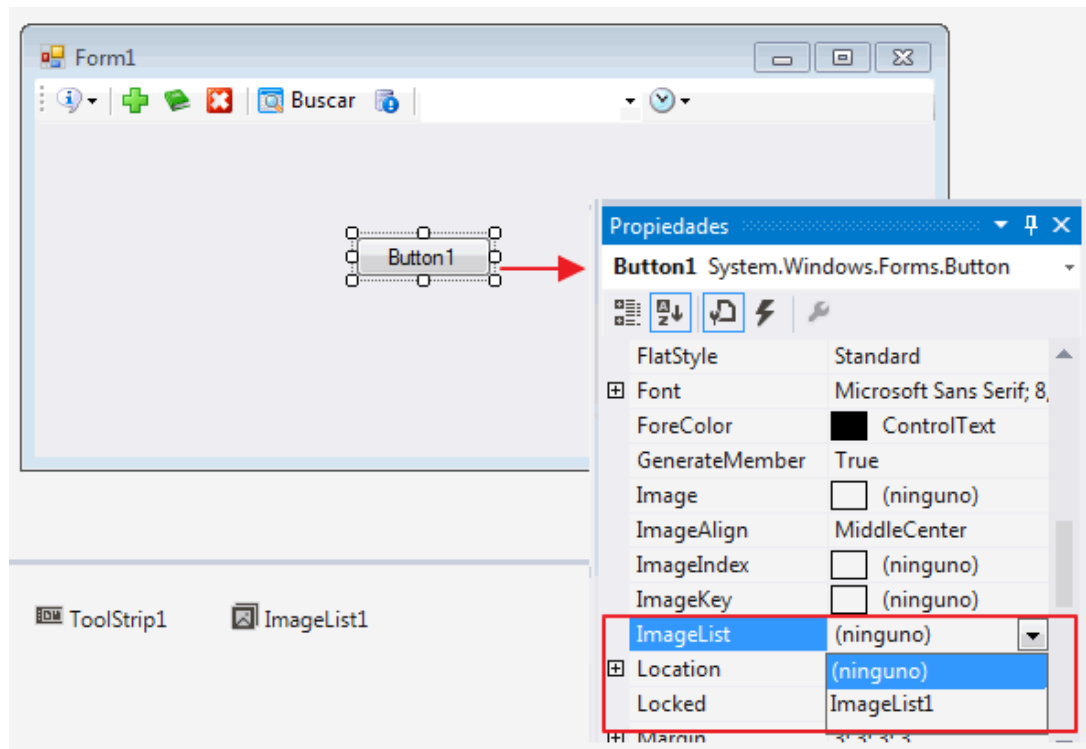
La primera propiedad como hemos dicho indica la profundidad de color o números de colores, puede ser de 8 bits para iconos o logotipos o de hasta 32 bits para fotografías. Pero veamos la propiedad "Images", al pulsarla tenemos como en otras colecciones la lista de miembros y sus propiedades:



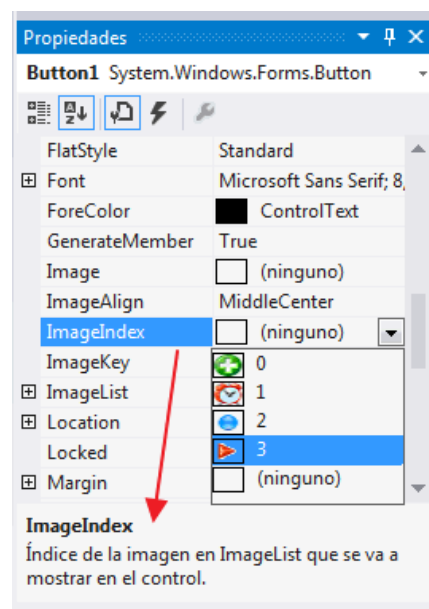
En la lista de la izquierda iremos añadiendo las imágenes y a la derecha podremos ver las propiedades de cada una de ellas. También podemos añadir más elementos mediante programa:

```
Me.listaimagenes.Images.Add(New Bitmap("ejemplo.gif"))
```

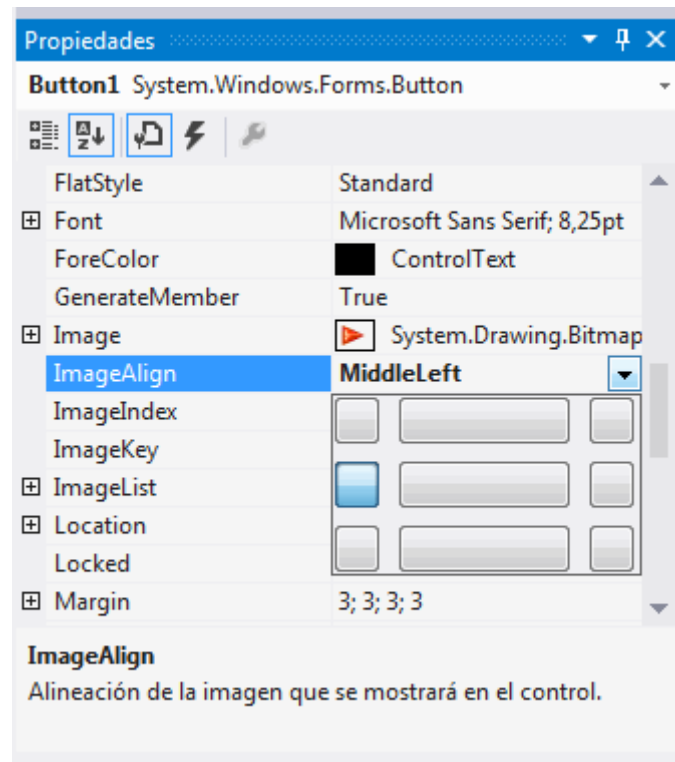
Veamos cómo se asocia a un control. Colocamos un botón en el formulario e indicamos la propiedad que apunta a la lista de imágenes que acabamos de crear:



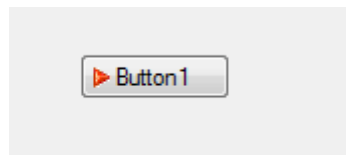
Le estamos indicando que se asocie con un control de este tipo que ya existe y que actúa como contenedor de imágenes. Así que ahora debemos elegir la imagen de este contenedor mediante la propiedad "imageindex". Al pulsar en el desplegable nos muestra los iconos directamente:



Y luego le indicamos la posición con la propiedad "ImageAlign":



Para darnos como resultado el botón así:

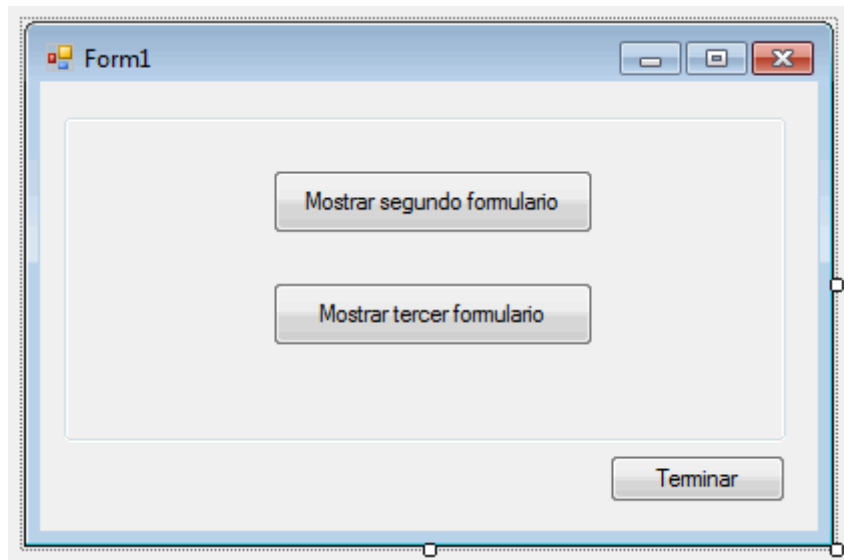


Y como te decía antes también puede alimentarse de imágenes mediante la propiedad "Image" que accede a la sección de recursos del proyecto.

## Ejercicios

### Ejercicio 1

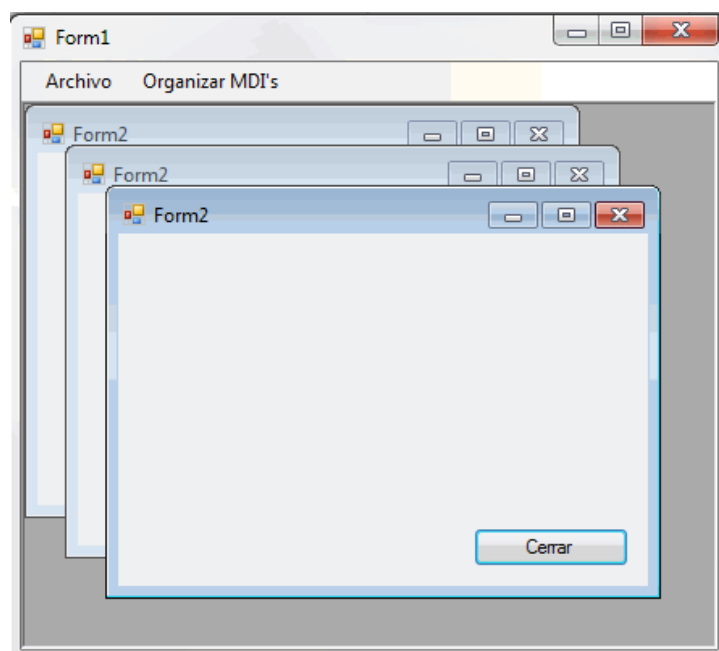
Crea un programa con tres formularios. El principal debe tener tres botones: dos para mostrar los otros dos formularios y un tercero para terminar la aplicación. Los otros dos formularios deben tener dos botones: uno para cerrarse y otro para terminar la aplicación.



Nota: La instrucción que hace que finalice un programa es "End"

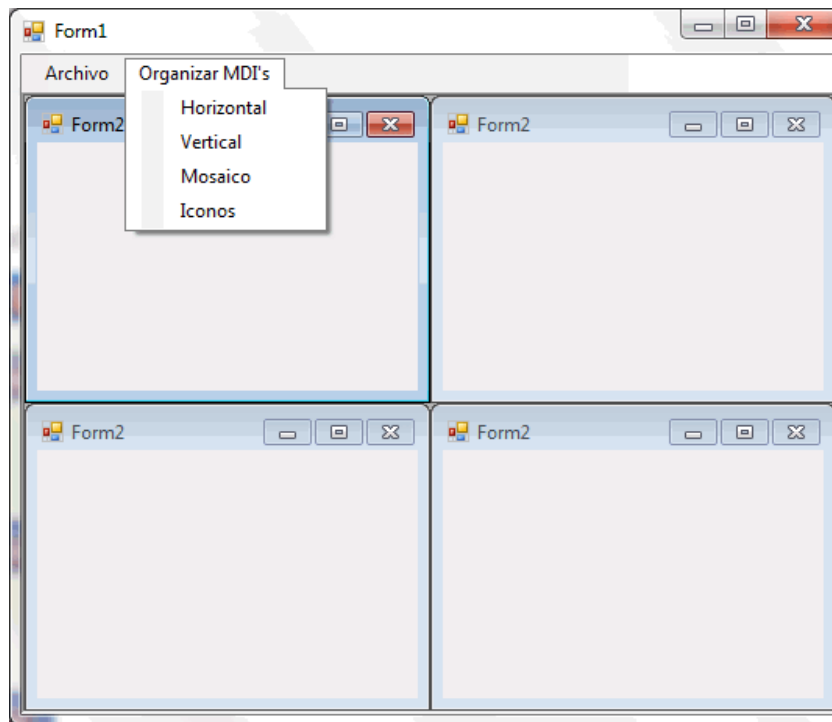
### Ejercicio 2

Crea un programa que tenga formularios MDI y un menú. En uno de los menús debe permitir crear nuevos formularios mdi y salir del programa. Cada formulario MDI debe tener la opción de cerrar el formulario.



## Ejercicio 3

Añade otro menú que permitirá cambiar la disposición de los formularios MDI abiertos.



## Ejercicio 4

Crea un programa que tenga:

- Un menú principal que permita terminar el programa
- Una barra de estado con dos paneles. El de la izquierda con el mensaje "Mi programa" y que tenga ajuste automático. El segundo panel que tenga ajuste fijo y se actualice con el color elegido en el menú que se describe a continuación.
- Un menú contextual con las palabras: "Rojo, Azul, Verde, Amarillo, Negro, Marrón"
- Poner la selección de este menú en el segundo panel de la barra de estado.
- Crea una barra de herramientas con tres opciones. La primera de ellas que sea para terminar el programa, el segundo tendrá un menú desplegable que será el mismo que el contextual y entre los botones habrá un separador.

