Visual Basic .NET : Windows Forms

Introducción

Un formulario Windows representa la conocida ventana utilizada en los Sistemas Operativos de tipo Windows.

Por otra parte, un control es aquel elemento situado dentro de una ventana o formulario y que permite al usuario de la aplicación Windows interactuar con la misma, para introducir datos o recuperar información.

Dentro de .NET, las ventanas clásicas Windows reciben la denominación de Windows Forms o WinForms.

System.Windows.Forms

Es el espacio de nombres que contiene todos los tipos del entorno, a través de los cuales podremos desarrollar aplicaciones compuestas por los formularios Windows, junto a los correspondientes controles que permiten interactuar al usuario con la aplicación desarrollada.

El conjunto de clases, estructuras, enumeraciones, . . ., del espacio *System.Windows.Forms* permiten la creación de aplicaciones del tipo Windows.

La Clase Form

Es la clase que contiene todos los elementos necesarios para la creación y manipulación de formularios (ventanas).

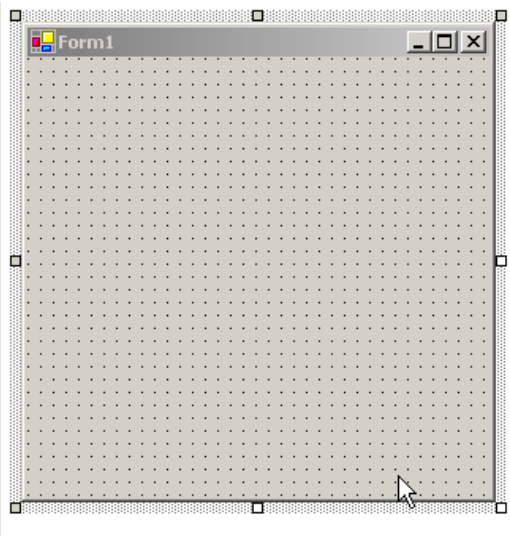
Tras instanciar un objeto de esta clase, dispondremos en nuestro programa de un formulario o ventana que podremos configurar según nuestras necesidades (Propiedades del formulario), y el cual podrá a su vez contener todo tipo de controles (cajas de texto, botones, ListBox, . . .) de forma que permita la interactuación necesaria con el usuario.

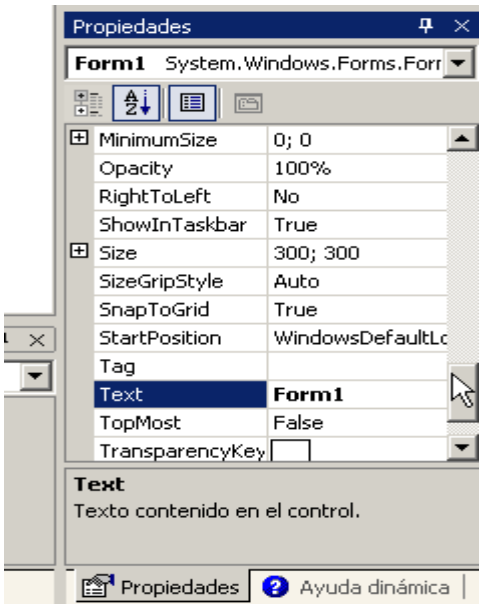
Creación de una aplicación Windows Form

La aplicación Windows Form más sencilla que se puede crear es la de un simple

formulario. Para ello deberemos escoger la creación de un *Nuevo Proyecto* y como tipo de proyecto elegir *Aplicación para Windows*. En ese momento, Visual Studio se encargará de configurar el nuevo proyecto para que sea el inicio de una futura aplicación de ventanas, llegando a crear y visualizar el primero de los formularios de dicha aplicación de forma totalmente automática.

Una vez creado el primer formulario, podremos configurarlo a través de la ventana *Propiedades*, la cual toma los valores del elemento que se encuentra seleccionado (en este caso nuestro formulario).



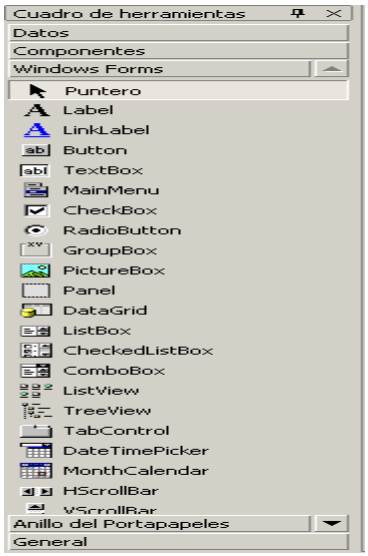


Controles

Como ya se ha comentado anteriormente, los controles proporcionan el único medio por el cual un usuario puede interactuar con una aplicación Windows Forms, con nuestro formulario en este caso.

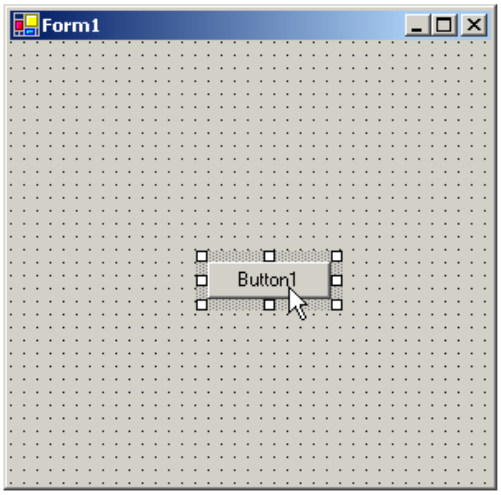
El Cuadro de Herramientas

El *Cuadro de Herramientas* es el componente de Visual Basic .NET a través del cual podremos crear e insertar todo tipo de controles en nuestra aplicación de una forma rápida y sencilla.



Insertar un control

Para insertar un control, simplemente tendremos que escoger el tipo de control que deseemos del *Cuadro de Herramientas* y elegir la posición que deseamos que ocupe dentro del formulario.



Propiedades de los controles

Todos los controles, de la misma forma que ocurre con los formularios, disponen de multitud de parámetros configurables que permiten darle una forma o comportamiento determinado y personalizado a cada uno de ellos. Todo eso se hace a través de la ventana *Propiedades* que nos informa acerca de esas propiedades para el control o formulario que hayamos preseleccionado con el ratón. De esa manera podremos consultar y modificar las

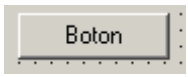
propiedades de cada uno de los controles para adaptarlos a nuestras necesidades.

Controles más comunes

A continuación, se van a comentar brevemente los controles más habituales en aplicaciones Windows Forms.

- Button
- Label
- TextBox
- ListBox
- ComboBox
- CheckBox
- RadioButton
- GroupBox

Button



Representa un botón. Las propiedades más importantes de este control son las siguientes:

- **Text** : Almacena el título del botón.
- **TextAlign** : Indica la alineación del texto dentro del botón.
- **BackColor** : Indica el color de fondo para el botón.
- **Font** : Cambia el tipo de letra para el título del botón.
- **Enabled** : Permite habilitar/deshabilitar el botón.

Eventos sobre controles

Los eventos son sucesos provocados por los controles cuando un usuario actúa sobre ellos, y que pueden ser capturados para asociar código a los mismos, de manera que cada vez que se produzca una acción determinada del usuario (pulsar un botón) se producirá un evento (Click, en este caso) y el programador podrá escribir código asociado a ese evento. Lo que se consigue es que cuando un usuario pulse sobre un botón se lleven a cabo una serie de acciones en el programa (mostrar un texto en el caso más sencillo).

Ahora, se va a mostrar un ejemplo de implementación de eventos sobre controles, en concreto sobre un control de tipo *Button*.

Implementaremos un evento de manera que cuando pulsemos sobre un botón, éste actúe mostrandonos un mensaje de texto sobre la pantalla a través del método *MessageBox.Show().*

Una forma sencilla de codificar el evento sobre un control es hacer doble click sobre ese control desde la vista *Diseño* de Visual Studio. De esta manera, se nos implementa la cabecera del evento más común de ese control. Así, bastará simplemente con escribir el código que queramos que se ejecute cada vez que se produzca ese evento.

En el caso que se ha expuesto, quedaría como sigue:

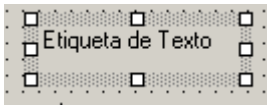
```
Private Sub boton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles boton.Click

      MessageBox.Show("Acabas de pulsar este botón")

End Sub
```

Eventos sobre formularios

También es posible, asociar código a eventos sobre formularios. Por ejemplo, al evento que lanza un formulario cuando el usuario cierra el mismo. Este evento se llama *Closing* y puede ser capturado y programado para realizar alguna acción, por ejemplo, la de confirmar el cierre del formulario (o aplicación):

```
Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
            System.ComponentModel.CancelEventArgs) Handles MyBase.Closing

     If MessageBox.Show("¿Cerrar la ventana?", "Atención",
     MessageBoxButtons.YesNo, MessageBoxIcon.Hand) =
     DialogResult.No Then
           e.Cancel = True
     End If

End Sub
```

Label



Muestra un texto informativo al usuario. Su utilización puede ser conjunta con otro

control. Por ejemplo, podremos colocar un *Label* cerca de una *TextBox* que informe acerca del contenido de éste control.

Se trata de un control estático por lo que el usuario no podrá interactuar con él ya que simplemente se limita a mostrar un determinado texto.

TextBox

Muestra una caja capaz de almacenar un texto introducido por el usuario y cuyo contenido puede cambiar a lo largo del programa.

Las propiedades más importantes de este control son las siguientes:

- **Text** : Almacena el texto del control.

- **Multiline** : Permite indicar si queremos permitir almacenar una sola línea o varias en el mismo control.

- **Enabled** : Permite habilitar/deshabilitar la caja de texto.

- **ReadOnly** : Permite indicar que el contenido de la caja de texto sea de sólo lectura por lo que el usuario no podrá alterar su contenido.

- **MaxLength** : Permite indicar el máximo de caracteres que será capaz de almacenar la caja de texto.

- **TextLength** : Devuelve la longitud del texto que actualmente está almacenado en este control.

Eventos sobre TextBox

Al igual que ocurria con el control de tipo *Button* o los formularios, también es posible codificar eventos sobre un control de tipo *TextBox*. El evento más importante de los controles *TextBox* se llama *TextChanged* y tiene lugar cada vez que se altera el contenido del control (el texto que éste contiene):

```
Private  Sub cajaTexto_TextChanged(ByVal sender As Object, ByVal e As
      System.EventArgs) Handles cajaTexto.TextChanged)

      ' Cuando cambie el texto, cambiará su longitud
      Me.longitudTexto.Text = Me.cajaTexto.TextLength

End Sub
```

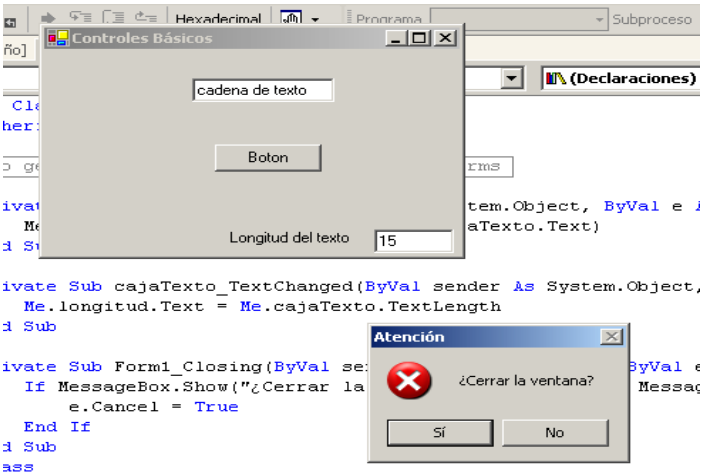
Ejemplo : Controles Básicos

Vamos a ver ahora una aplicación muy sencilla con los controles vistos hasta ahora: Button, Label y TextBox y los eventos más importantes asociados a éstos. Tambien se mostrará un ejemplo del evento Closing asociado al cierre de un formulario:

El código de la aplicación es el siguiente:

```
Private Sub boton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles boton.Click
        MessageBox.Show("Has escrito " & Me.cajaTexto.Text)
    End Sub

    Private Sub cajaTexto_TextChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles cajaTexto.TextChanged
        Me.longitud.Text = Me.cajaTexto.TextLength
    End Sub

    Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        If MessageBox.Show("¿Cerrar la ventana?", "Atención",
MessageBoxButtons.YesNo, MessageBoxIcon.Hand) = DialogResult.No Then
            e.Cancel = True
        End If
    End Sub
```

Y aquí podemos ver la apariencia de la aplicación en funcionamiento



CheckBox

Este control muestra una casilla de verificación, que podemos marcar o no para activar alguna opción de nuestro programa.

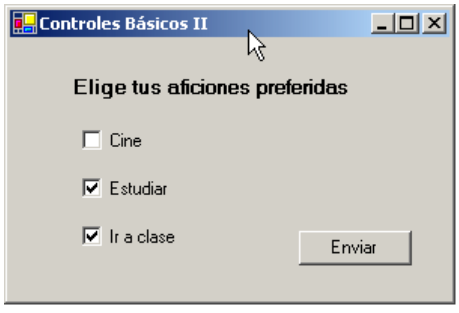
Internamente este control funciona como un Boolean ya que su estado podrá ser Verdadero (ha sido seleccionado) o Falso (no está seleccionado).

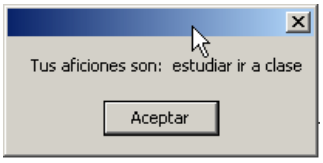
Las propiedades más importantes de este control son las siguientes:

- **Checked** : Indica True o False dependiendo de si el control ha sido seleccionado o no, respectivamente.

Ejemplo : CheckBox

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

        If Not checkCine.Checked And Not checkEstudiar.Checked And Not
checkClase.Checked Then
            MessageBox.Show("¡No tienes aficiones!")
        Else
            If checkCine.Checked Then
                aficiones += "cine"
            End If

            If checkEstudiar.Checked Then
                aficiones += " estudiar"
            End If

            If checkClase.Checked Then
                aficiones += " ir a clase"
            End If

            MessageBox.Show("Tus aficiones son: " & aficiones)
        End If
    End Sub
```



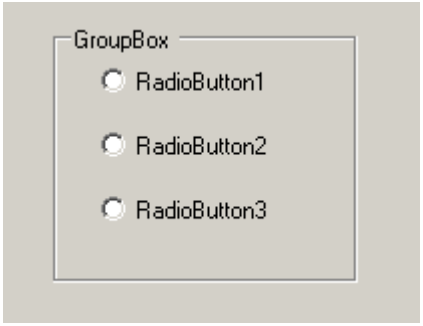


RadioButton

Un control *RadioButton* permite definir conjuntos de opciones autoexcluyentes. Es decir, se mostrará por ejemplo un grupo de 3 RadioButton de manera que sólo se pueda seleccionar uno de ellos e indique asi la opción escogida por el usuario.

Para poder detectar cuando ha sido seleccionada la opción de un determinado *RadioButton* disponemos del evento *CheckedChange*, el cual podremos implementar convenientemente de manera que cada vez que se pulse dicho RadioButton se ejecuten las acciones que correspondan.

GroupBox

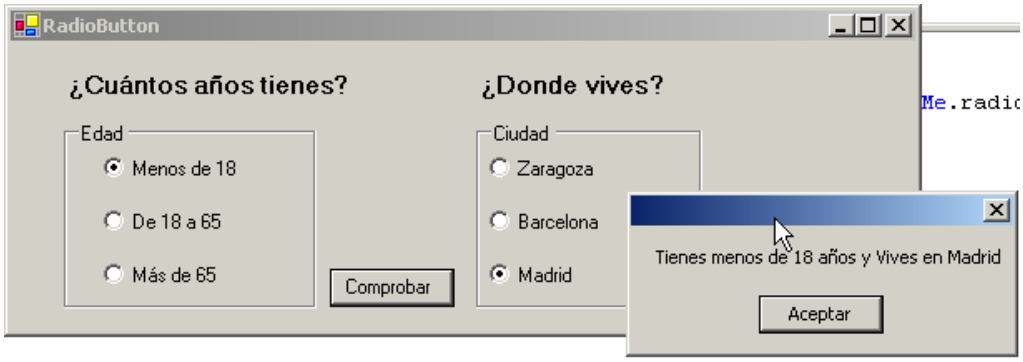


Este control permite agrupar controles en su interior de manera que, en cierta manera, permanezcan aislados del resto de controles del formulario. Este control nos será útil para crear más de un grupo de RadioButton ya que sino los separasemos con un control de este tipo se relacionarían entre ellos y representarían la misma opción.

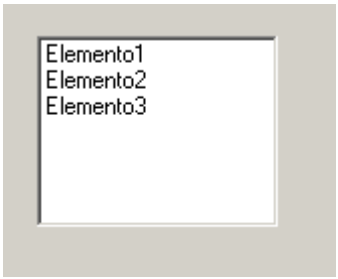
Ejemplo : RadioButton y GroupBox

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

        Dim edad, ciudad As String

        If Me.radio1.Checked Then
            edad = "Tienes menos de 18 años"
        ElseIf Me.radio2.Checked Then
            edad = "Tienes de 18 a 65 años"
        ElseIf Me.radio3.Checked Then
```

```
            edad = "Tienes más de 65 años"
        End If

        If Me.radio4.Checked Then
            ciudad = "Vives en Zaragoza"
        ElseIf Me.radio5.Checked Then
            ciudad = "Vives en Barcelona"
        ElseIf Me.radio6.Checked Then
            ciudad = "Vives en Madrid"
        End If

        MessageBox.Show(edad & " y " & ciudad)

    End Sub
```



ListBox



Este control es capaz de almacenar una lista de valores, y a la vez permite al usuario seleccionar uno o varios de estos valores. Las propiedades más importantes de este control son:

- **Items** : Contiene la lista de valores que almacena el *ListBox*. Se trata de un objeto Collection, el cual proporciona una serie de métodos para poder trabajar sobre ese conjunto de valores:

    - **Add(item)** : Permite añadir un nuevo elemento a la lista.

- **Sorted** : Si almacena el valor True, se ordenarán todos los elementos de la lista (tanto los que ya estaban almacenados como los que se inserten posteriormente) y se le asigna el valor False ya no se ordenarán los nuevos elementos que se inserten (aunque los que ya estaban insertados mantendrán el orden).

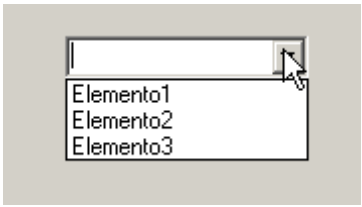
- **SelectedItem** : Devuelve el elemento de la lista que se encuentra seleccionado por el usuario.

Ejemplo : ListBox

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

        If Me.cajaTexto.Text.Length > 0 Then
            Me.lista.Items.Add(Me.cajaTexto.Text)
            Me.cajaTexto.Clear()
        End If

    End Sub

    Private Sub check_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles check.CheckedChanged

        If Me.check.Checked Then
            Me.lista.Sorted = True
        Else
            Me.lista.Sorted = False
        End If

    End Sub

    Private Sub botonEliminar_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles botonEliminar.Click
        Me.lista.Items.Remove(Me.lista.SelectedItem)
    End Sub

    Private Sub botonLimpiar_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles botonLimpiar.Click
        Me.lista.Items.Clear()
    End Sub

    Private Sub checkInsertar_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
checkInsertar.CheckedChanged

        If Me.checkInsertar.Checked Then
            Me.cajaTexto.Enabled = False
            Me.botonEliminar.Enabled = False
            Me.botonInsertar.Enabled = False
        Else
```

```
            Me.cajaTexto.Enabled = True
            Me.botonEliminar.Enabled = True
            Me.botonInsertar.Enabled = True
        End If

    End Sub
```

ComboBox



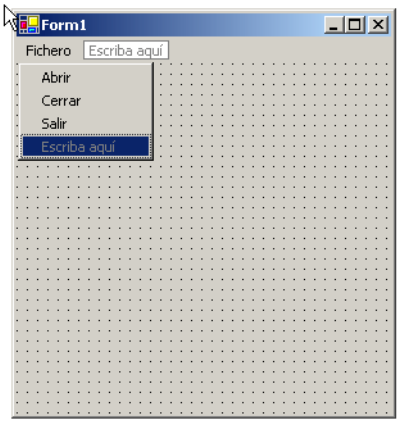
Se trata de un control mezcla de otros dos que ya hemos visto, el *TextBox* y el ListBox, ya que realmente se trata de un *ListBox* pero con apariencia de *TextBox* en el sentido de que es capaz de almacenar una lista de valores (ListBox) pero visualmente parece ser un *TextBox*. A diferencia de este último, si pulsamos sobre él desplegará la lista de los valores que almacena.

Ejemplo : ComboBox

*** Ver Ejemplo : ListBox**

Menús

El menú es un control muy utilizado en todo tipo de aplicaciones y también es posible crear e implementar este tipo de controles en una aplicación con Visual Basic .NET. A continuación se verá como diseñar un menú muy sencillo con alguna funcionalidad típica de éstos.



Una vez diseñado el menú desde la vista Diseño de Visual Studio, cada uno de las opciones de cada menú se puede ver como un botón a la hora de programar ya que lo que más nos interesará será codificar el evento Click de cada uno de ellos. Esto se hará de forma similar a como ocurría con los botones, haremos doble click en ellos y nos colocaremos dentro del método adecuado, sólo nos faltará escribir el código de lo que queremos que ocurra al seleccionar esa opción.